ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές Συλλογής Σκουπιδιών στη Διαχείριση Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Ζούλιας

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές Συλλογής Σκουπιδιών στη Διαχείριση Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Ζούλιας

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Λέκτορας Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25η Ιουλίου 2006.

.....	.....	.....
Νικόλαος Παπασπύρου	Ευστάθιος Ζάχος	Αριστείδης Παγουρτζής
Λέκτορας Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2006

Περίληψη

Στόχος της παρούσας έρευνας είναι να εισαγάγει τον αναγνώστη στους βασικούς όρους της συλλογής σκουπιδιών, δηλαδή της αυτόματης διαχείρισης, από το υπολογιστικό σύστημα, των τμημάτων της μνήμης που δεν χρησιμοποιούνται πλέον. Το ζήτημα έχει ιδιαίτερο προγραμματιστικό ενδιαφέρον και απασχολεί τους ειδικούς από τα τέλη της δεκαετίας του 1950.

Γίνεται μια σύντομη ιστορική ανασκόπηση στο θέμα και εξετάζεται το ζήτημα αν η διαχείριση μνήμης θα πρέπει να ελέγχεται αποκλειστικά από τον προγραμματιστή ή αυτόματα από το σύστημα εκτέλεσης των προγραμμάτων. Επίσης, επιχειρείται μια σύντομη αναφορά στις κλασικότερες μεθόδους συλλογής σκουπιδιών (καταμέτρηση αναφορών, εκκαθάριση με σήμανση και συλλογή με αντιγραφή) και γίνεται μια σύγκριση μεταξύ τους.

Ο αναγνώστης θα γνωρίσει τα θεμελιώδη ζητήματα που αφορούν το θέμα και θα κατατοπιστεί για τα βασικότερα κριτήρια με τα οποία μπορεί κάποιος να αποφανθεί αν η επίλυση ενός δεδομένου προβλήματος απαιτεί τη χρήση συλλέκτη σκουπιδιών και ποια τεχνική ταιριάζει καλύτερα στην περίπτωση.

Λέξεις Κλειδιά

Συλλογή σκουπιδιών, αυτόματη διαχείριση μνήμης, καταμέτρηση αναφορών, εκκαθάριση με σήμανση, συλλογή με αντιγραφή.

Abstract

This survey intends to introduce the reader in the basic concepts of garbage collection, that is, the automated management by a computer system of memory locations that are no longer in use. This issue is of vital importance for programmers and is subject to extensive research since the late 1950s.

After a brief historic survey on the topic, we discuss the issue whether memory management should be the full responsibility of the programmer, without any garbage collection support, or whether it should be handled automatically by the run-time system. Furthermore, there is an introduction to the classic garbage collection techniques (reference counting, mark-sweep and copying collection) and a comparison among them.

The reader will get to know the fundamentals of garbage collection and will be introduced to the basic criteria, based on which one can determine whether garbage collection is needed for a specific problem and which technique fits best the occasion.

Keywords

Garbage collection, automatic memory management, reference counting, mark-sweep, copying collection.

Ευχαριστίες

Ιδιαίτερες ευχαριστίες οφείλω στον επιβλέποντα καθηγητή της διπλωματικής μου κ. Νικόλαο Παπασπύρου για την προτροπή του να ασχοληθώ με το παρόν ενδιαφέρον θέμα, καθώς και για την καθοδήγησή του κατά τη διάρκεια της εκπόνησης της εργασίας και την επιείκεια του στα σφάλματα και τις παραλείψεις μου.

Επιπλέον, θα ήθελα να ευχαριστήσω τους γονείς μου κατά πρώτο λόγο, και όλους τους υπολοίπους, είτε μεμονωμένα άτομα, είτε οργανωμένους φορείς, οι οποίοι, από την αρχή της ζωής μου έως και σήμερα, συνέβαλαν στην παιδεία μου και τη γενικότερη διαμόρφωση του χαρακτήρα μου. Σε όλους αυτούς χρωστάω το «ευ ζην», ας δεχτούν λοιπόν αυτή την μικρή ευχαριστία ως ελάχιστο φόρο τιμής.

Ιωάννης Α. Ζούλιας,

Αθήνα, 23η Ιουλίου 2006.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-4-06, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2006.

URL: http://www.softlab.ntua.gr/techrep
FTP: ftp://www.softlab.ntua.gr/pub/techrep

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1. Σκοπός της εργασίας

Σκοπός της παρούσας εργασίας είναι να παρουσιάσει στον αναγνώστη τις βασικές αρχές της συλλογής σκουπιδιών. Ως συλλογή σκουπιδιών εννοείται στη συνέχεια η αυτόματη διαχείριση από το σύστημα εκτέλεσης των προγραμμάτων των τμημάτων της μνήμης και η αποδέσμευση αυτών που δεν χρησιμοποιούνται πλέον από το πρόγραμμα. Γίνεται μια σύντομη ιστορική ανασκόπηση στο θέμα και εξετάζεται το ζήτημα αν η διαχείριση μνήμης θα πρέπει να ελέγχεται αποκλειστικά από τον προγραμματιστή ή αυτόματα από το σύστημα εκτέλεσης των προγραμμάτων. Επιχειρείται μια σύντομη αναφορά στις κλασικότερες μεθόδους συλλογής σκουπιδιών και γίνεται μια σύγκριση μεταξύ τους.

1.2. Σύνοψη της εργασίας

Στο Κεφάλαιο 2 γίνεται μια πρώτη αναφορά στο τί είναι η συλλογή σκουπιδιών και εισάγονται οι βασικές έννοιες που σχετίζονται με την διαχείριση μνήμης.

Στα Κεφάλαια 3, 4 και 5 παρουσιάζονται συνοπτικά οι αλγόριθμοι των τριών κλασικών τεχνικών συλλογής σκουπιδιών, δηλαδή της καταμέτρησης αναφορών, της εκκαθάρισης με σήμανση και της συλλογής με αντιγραφή αντίστοιχα και σχολιάζονται τα βασικά χαρακτηριστικά τους, τα συγκριτικά πλεονεκτήματα και μειονεκτήματά τους.

Στο Κεφάλαιο 6 γίνεται μια ανασκόπηση των κριτηρίων επιλογής μεταξύ των διάφορων αλγορίθμων συλλογής σκουπιδιών.

Στο Κεφάλαιο 7 αναφέρονται δύο βελτιωτικές τεχνικές, η συλλογή σήμανσης με συγκέντρωση και οι γενεαλογικοί αλγόριθμοι που μπορούν να συνδυαστούν με τους έμμεσους συλλέκτες σκουπιδιών.

Τέλος, στο Κεφάλαιο 8 δίνεται ένας μικρός επίλογος.

2. Συλλογή Σκουπιδιών

Συλλογή σκουπιδιών στην επιστήμη των υπολογιστών καλείται η αυτόματη ανάκληση κομματιών της μνήμης του υπολογιστή τα οποία δεν χρησιμοποιούνται πλέον. Τις τελευταίες δεκαετίες, η συλλογή σκουπιδιών ωρίμασε. Ενώ αρχικά αποτελούσε μέρος του βασιλείου της Lisp και του συναρτησιακού προγραμματισμού, σήμερα η τεχνολογία αυτή αποτελεί σημαντικό κομμάτι του συστήματος διαχείρισης μνήμης πολλών σύγχρονων γλωσσών, τόσο προστακτικών όσο και δηλωτικών. Παρόλο που αρχικά η συλλογή σκουπιδιών είχε τη φήμη ότι προκαλεί καθυστερήσεις και εμποδίζει τη σωστή λειτουργία αλληλεπιδραστικών εφαρμογών, σύγχρονες τεχνικές υλοποίησης έχουν μειώσει σημαντικά το υπολογιστικό κόστος της, σε σημείο που να αποτελεί ρεαλιστική επιλογή ακόμα και για παραδοσιακές γλώσσες, όπως η C.

Παρά τη ραγδαία αύξηση στα μεγέθη της μνήμης ακόμα και των πιο ταπεινών υπολογιστών, η παροχή χώρου αποθήκευσης σε καμιά περίπτωση δεν πρέπει να θεωρηθεί ανεξάντλητη. Όπως κάθε περιορισμένος πόρος απαιτεί προσεχτική διαχείριση και ανακύκλωση. Πολλές σύγχρονες γλώσσες προγραμματισμού επιτρέπουν στον προγραμματιστή δυναμική εκχώρηση μνήμης. Η διαχείρισή της μπορεί να γίνει με κλήσεις συναρτήσεων, οι οποίες είτε είναι ενσωματωμένες στο πρόγραμμα, είτε βρίσκονται σε βιβλιοθήκες, και δεσμεύουν χώρο στη μνήμη ή τον απελευθερώνουν όταν δεν είναι χρήσιμος πλέον.

Όταν αυτή η λειτουργία ελέγχεται από τον άνθρωπο, συχνά τα αποτελέσματα δεν είναι ικανοποιητικά. Η εναλλακτική λύση είναι να αποδοθεί αυτή η ευθύνη στο σύστημα ελέγχου της ροής των προγραμμάτων. Σε αυτή την περίπτωση ο προγραμματιστής και πάλι θα πρέπει

να ζητάει μόνος του τη δυναμική δέσμευση μνήμης, αλλά θα απαλλάσσεται απ' την ανάγκη να την αποδεσμεύσει μέσω εντολών όπως η «free» και η «dispose». Η συλλογή σκουπιδιών είναι ακριβώς αυτή η διαδικασία, η αυτόματη διαχείριση του δυναμικά δεσμευμένου χώρου στη μνήμη. Μερικοί συγγραφείς κάνουν διάκριση μεταξύ των άμεσων τεχνικών, όπως είναι η καταμέτρηση αναφορών (reference counting), και των έμμεσων, που αναζητούν τα σκουπίδια. Παρόλα αυτά, ο όρος «συλλογή σκουπιδιών» χρησιμοποιείται ευρύτατα όταν αναφερόμαστε σε κάθε μορφή αυτόματης διαχείρισης του δυναμικά δεσμευμένου χώρου, ασχέτως του σε ποια από τις παραπάνω υποκατηγορίες ανήκει. Θα πρέπει πάντως να διακρίνουμε τον συλλέκτη σκουπιδιών από το κομμάτι του προγράμματος που κάνει την «πραγματική» δουλειά. Ακολουθώντας την ορολογία του Dijkstra, θα καλούμε το κυρίως πρόγραμμα «μεταλλάκτη», καθώς ο μοναδικός σκοπός του, απ' τη σκοπιά του συλλέκτη σκουπιδιών, είναι να μεταλλάσσει τις συνδέσεις στο γράφο των ενεργών εγγραφών στη μνήμη.

Στη συνέχεια αυτής της εισαγωγής θα μας απασχολήσουν τρία ερωτήματα. Ποια προβλήματα επιλύει η συλλογή σκουπιδιών; Πόσο δαπανηρή είναι; Μέσω ποιων παραμέτρων είναι εφικτή η σύγκριση διαφορετικών αλγορίθμων υλοποίησης; Επίσης θα αναφερθούμε σε μια πρώτη ταξινόμηση των διάφορων τεχνικών. Πρώτα απ' όλα όμως ας κάνουμε μια σύντομη ιστορική αναδρομή στις γλώσσες προγραμματισμού και συγκεκριμένα στον τρόπο με τον οποίο υλοποιούσαν την διαχείριση μνήμης, απ' το 1940 ως και σήμερα.

2.1. Η ιστορία της διαχείρισης μνήμης

Η ιστορική εξέλιξη των γλωσσών προγραμματισμού θα μπορούσε να θεωρηθεί ως μία συνεχή προσπάθεια να υποστηριχθούν όσο το δυνατόν περισσότερο οι αφηρημένες έννοιες (abstraction) και παράλληλα να αυτοματοποιηθούν ενέργειες τις οποίες αρχικά διεκπεραίωνε ο προγραμματιστής.

Τα πρώτα χρόνια των υπολογιστών κάθε επικοινωνία μεταξύ ανθρώπου και μηχανής γινόταν σε επίπεδο bit, με απλούς διακόπτες ως είσοδο. Σύντομα εισήχθησαν απλές συσκευές εισόδου και εξόδου που χρησιμοποιούσαν το δεκαεξαδικό σύστημα αρίθμησης, διευκολύνοντας την επικοινωνία. Το επόμενο βήμα ήταν να δοθεί η δυνατότητα χρήσης μνημονικών κωδικών απ' την πλευρά του προγραμματιστή, οι οποίοι μπορούσαν να μεταφραστούν αυτόματα σε δυαδικές εντολές. Σε κάθε περίπτωση πάντως, ο χρήστης είχε την απόλυτη ευθύνη για κάθε λεπτομέρεια στη λειτουργία του προγράμματος. Για παράδειγμα, ιδιαίτερη φροντίδα χρειαζόταν στην καταμέτρηση των λέξεων του προγράμματος ούτως ώστε να μπορεί ο προγραμματιστής να αποφανθεί αν η μνήμη ήταν αρκετή για την εκτέλεση του, ή να υπολογίζεται σε κάθε εντολή η ακριβής διεύθυνσή της στη μνήμη για να ορίζονται σωστά οι προορισμοί στις εντολές «jump».

Στα τέλη τις δεκαετίας του 40 και στις αρχές της δεκαετίας του 50, αυτή η υποχρέωση μεταφέρθηκε στους κώδικες μακροεντολών και τις γλώσσες assembly. Τα συμβολικά προγράμματα που καθιερώθηκαν τότε ήταν πολύ πιο εύκολα τόσο στη γραφή όσο και στην ανάγνωση από τη γλώσσα μηχανής, κυρίως λόγω της αντικατάστασης των αριθμητικών κωδικών για διευθύνσεις και τελεστές με συμβολισμούς που είχαν περισσότερο νόημα. Και πάλι όμως, ο προγραμματιστής θα έπρεπε τελικά να γνωρίζει πολύ καλά τις συγκεκριμένες λειτουργίες του υπολογιστή και το πώς και πού γινόταν η παράσταση των δεδομένων στη μηχανή. Ο μεγάλος αριθμός μικρών λεπτομερειών που διαφέρουν από μηχανή σε μηχανή κάνει ακόμα και σήμερα την χρήση της γλώσσας assembly εξοντωτική.

Για να αντιμετωπιστούν αυτά τα προβλήματα, στα μέσα και στα τέλη της δεκαετίας του 40 άρχισαν να εμφανίζονται οι πρώτες ιδέες για γλώσσες υψηλού επιπέδου, με στόχο τη

διευκόλυνση των προγραμματιστών. Το 1952 εμφανίστηκαν οι πρώτοι μεταγλωττιστές και στις αρχές του 1957 κατασκευάστηκε ο πρώτος μεταγλωττιστής για τη Fortran. Ένας μεταγλωττιστής υψηλού επιπέδου θα πρέπει να κατανέμει τους πόρους της μηχανής ώστε να αναπαριστούν τα δεδομένα που επεξεργάζεται το πρόγραμμα του χρήστη. Υπάρχουν τρεις τρόποι με τους οποίους είναι εφικτή αυτή η κατανομή:

2.1.1. Στατική Κατανομή

Η στατική κατανομή είναι η απλούστερη μέθοδος. Όλα τα ονόματα στο πρόγραμμα συνδέονται με θέσεις μνήμης κατά τη διαδικασία της μεταγλώττισης και δεν γίνονται αλλαγές κατά την εκτέλεση του προγράμματος. Αυτό υπονοεί ότι οι τοπικές μεταβλητές μίας διαδικασίας καταγράφονται στην ίδια θέση μνήμης κάθε φορά που καλείται η διαδικασία. Η στατική κατανομή αρχικά υλοποιήθηκε από τη Fortran και χρησιμοποιείται ακόμα, για παράδειγμα. στη Fortran77. Η συγκεκριμένη μέθοδος έχει τρεις περιορισμούς:

- Το μέγεθος κάθε δεδομένου θα πρέπει να είναι γνωστό κατά τη διάρκεια της μεταγλώττισης.
- Δεν είναι δυνατή η αναδρομική κλήση διαδικασιών, καθώς όλες οι κλήσεις τους μοιράζονται τον ίδιο χώρο μνήμης για τις τοπικές μεταβλητές τους.
- Δεν είναι εφικτή η δημιουργία δυναμικών δομών δεδομένων.

Παρόλα αυτά, η στατική κατανομή μνήμης έχει δύο σημαντικά πλεονεκτήματα. Οι υλοποιήσεις της συνήθως δημιουργεί γρήγορες γλώσσες, καθώς δεν υπάρχει ανάγκη για δημιουργία ή αποδέσμευση δομών δεδομένων. Καθώς η θέση κάθε δεδομένου είναι γνωστή στον μεταγλωττιστή, υπάρχει πρόσβαση σε όλα τα δεδομένα άμεσα και όχι έμμεσα. Επιπλέον, η στατική κατανομή δίνει μια εγγυημένη ασφάλεια: το πρόγραμμα δεν πρόκειται να αποτύχει κατά την εκτέλεση λόγω έλλειψης μνήμης, καθώς εξ αρχής γνωρίζουμε τις απαιτήσεις σε μνήμη.

2.1.2. Κατανομή Στοίβας

Οι πρώτες γλώσσες αυτού του είδους εμφανίστηκαν το 1958 με την Algol-58 και την Atlas Autocode. Αποφεύγουν κάποια από τα μειονεκτήματα των στατικών γλωσσών κατανέμοντας μνήμη με μορφή στοίβας. Μια *τιμή ενεργοποίησης* τοποθετείται στη στοίβα του συστήματος όταν καλείται μία διαδικασία και επιστρέφεται όταν η διαδικασία ολοκληρωθεί. Η κατανομή στοίβας έχει πέντε συνέπειες:

- Διαφορετικές κλήσεις μιας διαδικασίας δεν μοιράζονται τις ίδιες θέσεις για τις τοπικές μεταβλητές. Οι αναδρομικές κλήσεις είναι εφικτές, αυξάνοντας την εκφραστική ικανότητα της γλώσσας.
- Το μέγεθος τοπικών δομών δεδομένων, όπως ένας πίνακας, μπορεί να εξαρτάται από μια παράμετρο που περνιέται στη διαδικασία.
- Οι τιμές των τοπικών μεταβλητών που έχουν αποθηκευτεί στη στοίβα δεν μπορεί να διατηρηθεί μεταξύ των κλήσεων της διαδικασίας, αν και αυτό θα μπορούσε να είναι χρήσιμο σε ορισμένες περιπτώσεις.
- Μια δομή δεδομένων δεν μπορεί να διαρκέσει περισσότερο από τη διεργασία που την δημιούργησε.
- Μόνο αντικείμενα των οποίων το μέγεθος είναι γνωστό κατά τη μεταγλώττιση μπορούν να επιστραφούν ως αποτελέσματα διεργασιών.

2.1.3. Κατανομή Σωρού

Στην περίπτωση της στοίβας υπάρχει ο περιορισμός ότι το αντικείμενο που μπήκε τελευταίο θα πρέπει να βγει πρώτο. Αυτό δεν ισχύει στο σωρό, όπου τα αντικείμενα μπορούν να ανακληθούν με οποιαδήποτε σειρά. Επομένως τα δυναμικά εκχωρημένα δεδομένα μπορούν να διαρκέσουν περισσότερο από τις διαδικασίες που τα δημιούργησαν. Η κατανομή σωρού έχει αρκετά πλεονεκτήματα:

Η σχεδίαση απαιτεί τη δημιουργία αφηρημένων εννοιών που μοντελοποιούν πραγματικά προβλήματα και πολλά από αυτά έχουν μια φυσική ιεραρχία. Τυπικά παραδείγματα είναι οι λίστες και τα δένδρα. Η κατανομή σωρού επιτρέπει η στερεή αναπαράσταση τέτοιων εννοιών να είναι αναδρομική.

Το μέγεθος των δομών δεδομένων δεν χρειάζεται πλέον να είναι καθορισμένο, καθώς μπορεί να αλλάξει δυναμικά. Η υπέρβαση των ορίων του μεγέθους μιας δομής δεδομένων, για παράδειγμα ενός πίνακα, είναι ένα από τα πιο συνήθη αίτια προβλημάτων.

Αντικείμενα με δυναμικά ρυθμιζόμενο μέγεθος μπορούν να επιστραφούν ως αποτέλεσμα μιας διαδικασίας.

Πολλές σύγχρονες γλώσσες προγραμματισμού επιτρέπουν την επιστροφή μιας διαδικασίας ως αποτέλεσμα μιας άλλης διαδικασίας. Αυτό μπορεί να επιτευχθεί και με κατανομή στοίβας, αρκεί να αποφεύγονται οι φωλιασμένες διαδικασίες, επιστρέφοντας τη στατική διεύθυνση της διαδικασίας. Όμως οι συναρτησιακές και οι προστακτικές γλώσσες υψηλής τάξης μπορεί να επιτρέψουν το αποτέλεσμα μιας συνάρτησης να είναι μία άλλη συνάρτηση. Αυτές οι τιμές θα πρέπει λοιπόν να διαρκέσουν περισσότερο από τη συνάρτηση που τις δημιούργησε.

Σήμερα πολλές, αν όχι οι περισσότερες, γλώσσες προγραμματισμού υψηλού επιπέδου είναι ικανές να κατανείμουν μνήμη και σε στοίβα και σε σωρό. Πολλές γλώσσες, όπως η Pascal ή η C, παραδοσιακά διαχειρίζονται όλα τα δεδομένα στο σωρό με ρητές εντολές από τον προγραμματιστή. Η C++ είναι μια σύγχρονη γλώσσα που παραμένει προσκολλημένη σ' αυτή την παράδοση. Οι συναρτησιακές, οι λογικές και οι περισσότερες αντικειμενοστραφείς γλώσσες χρησιμοποιούν συλλογή σκουπιδιών για να διαχειριστούν τον σωρό αυτόματα. Παραδείγματα αυτού του είδους περιλαμβάνουν τις Scheme, Dylan, ML, Haskell, Miranda, Prolog, Smalltalk, Eiffel, Java και Oberon. Άλλες γλώσσες, με πιο αξιοσημείωτη την Modula-3, προσφέρουν τόσο ρητή, όσο και αυτόματη διαχείριση του σωρού.

2.2. Καταστάσεις, «ζωντάνια» και προσβασιμότητα από δείκτες

Οι τιμές τις οποίες ένα πρόγραμμα μπορεί να επεξεργαστεί άμεσα είναι αυτές που βρίσκονται στους καταχωρητές του επεξεργαστή, αυτές που είναι στη στοίβα του προγράμματος (συμπεριλαμβάνονται οι τοπικές και οι προσωρινές μεταβλητές), και αυτές που είναι αποθηκευμένες σε ολικές (global) μεταβλητές. Τέτοιες περιοχές που περιέχουν αναφορές σε δεδομένα στο σωρό αποτελούν τις *ρίζες* του υπολογισμού. Η αυτόματη διαχείριση της μνήμης του σωρού απαιτεί από τον προγραμματιστή να ακολουθεί κάποιους κανόνες. Τα δυναμικά εκχωρημένα δεδομένα θα πρέπει να είναι προσβάσιμα από το πρόγραμμα του χρήστη μόνο μέσω των ριζών ή ακολουθώντας κάποια σειρά από δείκτες που ξεκινά από τις ρίζες. Συγκεκριμένα, το πρόγραμμα δε θα πρέπει να έχει πρόσβαση σε τυχαίες θέσεις μνήμης ακόμα κι αν βρίσκονται μέσα στο κομμάτι της μνήμης που είναι στη δικαιοδοσία του. Κάτι τέτοιο θα μπορούσε να γίνει για παράδειγμα αν το πρόγραμμα διάβαζε σε μια θέση που απέχει μια αυθαίρετη απόσταση από τη βάση του σωρού. Αυτή η

απαγόρευση δεν οφείλεται μόνο στη συλλογή σκουπιδιών. Επιβάλλεται από τις γλώσσες με ισχυρό σύστημα τύπων, όπως είναι η Pascal. Η ασφαλής χρήση των εντολών malloc/free στη C επίσης επιβάλλει τη μη χρησιμοποίηση θέσεων μνήμης που δεν έχουν κατοχυρωθεί.

Ένα μεμονωμένο κατοχυρωμένο κομμάτι δεδομένων στο σωρό θα λέγεται στο εξής *κόμβος, κελί* ή *αντικείμενο* (ο τελευταίος όρος δε χρησιμοποιείται με την αντικειμενοστραφή του έννοια). Οι παραπάνω κανόνες υπονοούν ότι απ' την πλευρά του μηχανισμού αποθήκευσης η «ζωντάνια» στο γράφο των αντικειμένων στο σωρού ορίζεται από την *προσβασιμότητα από δείκτες*. Ένα αντικείμενο στο σωρό είναι ζωντανό αν η διεύθυνσή του βρίσκεται σε κάποια ρίζα ή αν υπάρχει δείκτης σ' αυτό από άλλο ζωντανό αντικείμενο. Σε ποιο επίσημη διατύπωση, ας ορίσουμε το βέλος προς τα δεξιά «$\rightarrow$» ως τη σχέση «δείκτης προς»: Για κάθε κόμβο ή ρίζα M και για κάθε κόμβο στο σωρό N, $M \rightarrow N$ αν και μόνο αν το M έχει αναφορά στο N. Το σύνολο των ζωντανών κόμβων στο σωρό είναι το ελάχιστο σύνολο *live* για το οποίο:

$$live = \{N \in Nodes \mid (\exists\, r \in Roots.r \rightarrow N) \vee (\exists\, M \in live.M \rightarrow N)\}$$

Μέχρι τώρα παρατηρούμε ότι αυτή η εκτίμηση του συνόλου των ζωντανών κελιών στο σωρό είναι απλώς μία συντηρητική προσέγγιση του πραγματικού συνόλου των κελιών που είναι δυνητικά προσβάσιμα στο πρόγραμμα. Μπορεί να περιέχονται και κελιά τα οποία μετά από ανάλυση του κώδικα του προγράμματος ή της ροής δεδομένων να αποδεικνύονταν νεκρά από έναν μεταγλωττιστή που κάνει βελτιώσεις. Τυπικό παράδειγμα θα μπορούσε να είναι μια τοπική μεταβλητή μετά την τελευταία της κλήση στη διαδικασία, ή θέσεις σε ένα πλαίσιο στοίβας που δεν έχουν αρχικοποιηθεί ακόμα, ή ένας παρωχημένος δείκτης που έχει διατηρηθεί σε ένα μητρώο (για να αποφευχθεί το κόστος της διαγραφής του). Θα επιστρέψουμε σ' αυτό το ερώτημα αργότερα.

Η ζωντάνια ενός κόμβου μπορεί να αποσαφηνιστεί είτε άμεσα, είτε έμμεσα. Οι άμεσες μέθοδοι απαιτούν να υπάρχει μια καταγραφή για κάθε κόμβο στο σωρό όλων των αναφορών στον κόμβο από άλλους κόμβους του σωρού ή ρίζες. Στην πιο κοινή περίπτωση αρκεί ο αριθμός αυτών των αναφορών, η *καταμέτρηση των αναφορών* του, που αποθηκεύεται στο ίδιο το κελί. Άμεσοι αλγόριθμοι για κατανεμημένα συστήματα μπορεί, αντίθετα, να διατηρούν λίστες των μακρινών επεξεργαστών που περιέχουν αναφορές για κάθε αντικείμενο. Σε κάθε περίπτωση, αυτές οι καταχωρήσεις θα πρέπει να διατηρούνται ενήμερες καθώς ο μεταλλάκτης αλλάζει τις συνδέσεις του γράφου στο σωρό.

Οι έμμεσοι συλλέκτες τυπικά παράγουν ένα σύνολο ζωντανών κόμβων όποτε μια αίτηση για μνήμη από το πρόγραμμα αποτυγχάνει. Ο συλλέκτης ξεκινά απ' τις ρίζες και, ακολουθώντας τους δείκτες, επισκέπτεται κάθε προσιτό κόμβο. Αυτοί είναι οι κόμβοι που θεωρούνται ζωντανοί και όλη η μνήμη που καταλαμβάνεται από άλλους κόμβους γίνεται διαθέσιμη για ανακύκλωση. Αν απελευθερωθεί αρκετή μνήμη, η αίτηση του προγράμματος ικανοποιείται και συνεχίζεται η εκτέλεσή του.

2.3. Ρητή ανάθεση μνήμης στο σωρό

Ένα απλό παράδειγμα:

Παραδοσιακά, οι περισσότερες προστακτικές γλώσσες τοποθέτησαν στον προγραμματιστή την ευθύνη για την ανάθεση και αποδέσμευση μνήμης αντικειμένων στο σωρό. Στην Pascal η μνήμη ανατίθεται στο σωρό μέσω της εντολής new. Αν p είναι ένας δείκτης, η εντολή new(p) επιστρέφει στον δείκτη την διεύθυνση μιας νέας δεσμευμένης θέσης μνήμης

με μέγεθος ίσο με το μέγεθος του τύπου στον οποίο ο p είναι ορισμένος να δείχνει. Το αντικείμενο αποδεσμεύεται με την εντολή dispose(p). Το παρακάτω κομμάτι προγράμματος δημιουργεί τη λίστα [1, 2, 3].

```
program pointer (input, output);
type ptr = ^cell;
        cell = record
          value: integer;
          next  : ptr
     end;
var myList: ptr;

function Insert (item: integer; list: ptr): ptr;
var temp: ptr;
begin
     new(temp);
     temp^.value:= item;
     temp^.next:= list;
     Insert:= temp
end;

begin
     myList:= Insert (1, Insert(2, Insert(3, nil)))
end.
```

Σκουπίδια:

Η δυναμικά εκχωρημένη μνήμη μπορεί να γίνει απρόσιτη. Τα αντικείμενα που ούτε είναι ζωντανά, ούτε αποδεσμεύονται ονομάζονται *σκουπίδια*. Η ρητή αποδέσμευση μνήμης από τον προγραμματιστή δεν μπορεί να τα ανακτήσει, αφού δεν υπάρχει δείκτης προς αυτά. Ο χώρος που καταλαμβάνουν θεωρείται ότι έχει *διαρρεύσει*. Για παράδειγμα, θα μπορούσαμε να δημιουργήσουμε μια διαρροή στο παραπάνω πρόγραμμα αν προσθέταμε τη γραμμή

```
myList^.next:= nil;
```

αμέσως μετά τη δημιουργία της λίστας.

Τώρα μόνο το πρώτο στοιχείο είναι προσιτό στο πρόγραμμα. Ο χώρος της μνήμης που περιέχει τα στοιχεία 2 και 3 δεν βρίσκεται στην εμβέλεια του, και δεν μπορεί ούτε να χρησιμοποιηθεί, ούτε να ανακληθεί. Η αυτόματη διαχείριση μνήμης μπορεί να ανακαλέσει τέτοια απρόσιτα τμήματα της μνήμης. Αυτό είναι το αντικείμενο της παρούσας έρευνας.

Μετέωρες αναφορές:

Κομμάτια της μνήμης είναι δυνατόν να αποδεσμευτούν ενώ υπάρχουν ακόμα δείκτες προς αυτά. Αν υποθέσουμε ότι αντί της γραμμής που αναφέρεται στην προηγούμενη παράγραφο, τοποθετούσαμε στον αρχικό αλγόριθμο στην ίδια θέση τη γραμμή:

```
dispose (myList^.next);
```

επιστρέφοντας το αντικείμενο 2 στο διαχειριστή του σωρού. Και πάλι το 3 γίνεται σκουπίδι, αυτή η μικρή διαρροή όμως δεν θα επηρεάσει πολύ το πρόγραμμά μας. Το ουσιαστικότερο πρόβλημα είναι ότι το πεδίο next του αντικειμένου 1 αναφέρεται πλέον σε κομμάτι της μνήμης το οποίο έχει αποδεσμευτεί. Έχει δημιουργηθεί, λοιπόν, ένας μετέωρος δείκτης.

Το πρόγραμμα δεν ελέγχει τον τρόπο με τον οποίο θα χρησιμοποιηθεί η αποδεσμευμένη μνήμη. Μπορεί να καθαριστεί, να χρησιμοποιηθεί για αποθήκευση πληροφοριών ή να ανακυκλωθεί από τον διαχειριστή του σωρού. Αν το πρόγραμμα χρησιμοποιήσει τη μετέωρη αναφορά το καλύτερο στο οποίο μπορούμε να ελπίζουμε είναι ότι θα διακοπεί αμέσως η λειτουργία του. Αν ο διαχειριστής του σωρού είχε ήδη αναθέσει την αποδεσμευμένη μνήμη σε κάποια άλλη δομή δεδομένων του προγράμματος, τότε μία μόνο θέση μνήμης θα αντιστοιχούσε σε δύο διαφορετικά αντικείμενα. Θα πρέπει να θεωρούμε τους εαυτούς μας τυχερούς αν το πρόγραμμα κολλήσει αρκετά σύντομα. Διαφορετικά, θα συνεχίσει παράγοντας λαθεμένα αποτελέσματα.

Μοίρασμα:

Τα σκουπίδια και οι μετέωρες αναφορές είναι οι δύο όψεις του ίδιου νομίσματος, δηλαδή της ρητής ανάθεσης μνήμης από τον προγραμματιστή. Σκουπίδια δημιουργούνται αν καταστραφεί η τελευταία αναφορά σε ένα αντικείμενο, προτού αυτό αποδεσμευτεί. Μετέωρες αναφορές δημιουργούνται αν αποδεσμευτεί ένα αντικείμενο και παραμείνουν οι αναφορές σ' αυτό. Φαινομενικά, η λύση και στα δύο προβλήματα είναι απλή αν οι δύο πράξεις, δηλαδή η καταστροφή της τελευταίας αναφοράς και η αποδέσμευση της μνήμης συνδυαστούν. Μια τέτοια όμως διαδικασία δεν είναι καθόλου εύκολη στην περίπτωση του μοιράσματος.

Έστω ότι δύο λίστες μοιράζονται την ίδια κατάληξη. Μια σωστή ρουτίνα για τη διαγραφή της μίας λίστας θα φρόντιζε να σβήσει αναδρομικά κάθε κόμβο που ανήκει σ' αυτήν, όταν θα σβηνόταν ο δείκτης στην κεφαλή της λίστας αυτής. Αυτό όμως θα είχε καταστροφικές συνέπειες για την άλλη, καθώς ο τελευταίος μη κοινός κόμβος της θα γινόταν μετέωρη αναφορά. Αυτό ακριβώς το πρόβλημα οδήγησε στο ενδιαφέρον για τεχνικές αυτόματης ανάκλησης μνήμης στα τέλη της δεκαετίας του 50.

Σφάλματα:

Η δυναμική εκχώρηση μνήμης είναι δύσκολο να επιτευχθεί χειροκίνητα σε πολύπλοκα προγράμματα, και υπάρχουν πολλά παραδείγματα αποτυχημένων προγραμμάτων. Προγράμματα που διακόπτονται αναπάντεχα ή εξυπηρετητές (servers) που ξεμένουν από μνήμη χωρίς προφανή αιτία. Τα αποτελέσματα τέτοιων προβληματικών λαθών δεν είναι δυνατόν να προσδιοριστούν, ιδίως σε πολυνηματικά (multi-threaded) περιβάλλοντα. Οι μετέωρες αναφορές μπορεί να μην έχουν δυσάρεστες συνέπειες αν ο διαχειριστής του σωρού δεν αναθέσει τη μνήμη σε άλλο αντικείμενο. Οι διαρροές μπορεί να μην εμφανιστούν κατά τη διάρκεια του ελέγχου ή υπό κανονικές συνθήκες χρήσης. Τα σημαντικά σφάλματα συνήθως κάνουν την εμφάνισή τους όταν το πρόγραμμα πιέζεται ή χρησιμοποιείται για μεγάλα χρονικά διαστήματα. Για παράδειγμα, μπορεί η είσοδος ενός μεταγλωττιστή να είναι ένα πρόγραμμα δημιουργημένο από μηχανή και να παραβιάζει τις υποθέσεις για το πώς θα είναι ο κώδικας τον οποίο λογικά αναμένεται να γράψει ένας προγραμματιστής. Οι διαρροές στη μνήμη μπορεί να μη γίνουν ποτέ αντιληπτές στον υπολογιστή όπου γράφεται το πρόγραμμα. Όταν, όμως, το πρόγραμμα εκτελείται σε μηχάνημα με λιγότερη μνήμη ή σε έναν εξυπηρετητή για μεγάλα χρονικά διαστήματα, η διαρροή είναι δυνατόν να εξαντλήσει τη μνήμη. Η διόρθωση των σφαλμάτων σε τέτοιες συνθήκες είναι ιδιαίτερα δύσκολη, καθώς τα σφάλματα συνήθως δεν επαναλαμβάνονται με τον ίδιο τρόπο.

2.4. Γιατί να κάνουμε συλλογή σκουπιδιών;

Οι απαιτήσεις της γλώσσας:

Η συλλογή σκουπιδιών μπορεί να είναι απαραίτητη ή απλώς ιδιαίτερα επιθυμητή. Ίσως να είναι μια απαίτηση της γλώσσας: Η διαχείριση του σωρού απαιτείται όταν οι δομές δεδομένων ζουν περισσότερο από τις διαδικασίες που τις δημιούργησαν. Αν αυτές οι δομές μετά περαστούν ως παράμετροι σε άλλες διαδικασίες ή συναρτήσεις, τότε ίσως ο προγραμματιστής ή ο μεταγλωττιστής να μην είναι σε θέση να κρίνει πότε μπορεί να τις αποδεσμεύσει από τη μνήμη. Η επικράτηση του μοιράσματος και των αναβολών στην εκτέλεση στις συναρτησιακές γλώσσες συχνά οδηγεί σε απρόβλεπτη σειρά στην εκτέλεση του προγράμματος. Σε τέτοιες περιπτώσεις η συλλογή σκουπιδιών είναι υποχρεωτική.

Οι απαιτήσεις του προβλήματος:

Η συλλογή σκουπιδιών μπορεί να απαιτείται από το πρόβλημα. Έστω ότι ένας γενικός τύπος στοίβας πρόκειται να υλοποιηθεί σε C σαν μια συνδεδεμένη λίστα. Κάθε κόμβος στη στοίβα περιέχει δύο πεδία: τα δεδομένα data και τον δείκτη στον επόμενο κόμβο next. Χρησιμοποιούμε την εντολή pop για να αφαιρέσουμε την κορυφή της στοίβας, το στοιχείο first, και να επιστρέψουμε ένα δείκτη για την υπόλοιπη στοίβα. Θα πρέπει η pop να αποδεσμεύσει την πληροφορία $\text{first} \rightarrow \text{data}$; Αν τα δεδομένα έχουν εκχωρηθεί στατικά τότε η απάντηση είναι «όχι». Διαφορετικά, αν αυτή είναι η τελευταία αναφορά σε αυτό το στοιχείο, η απάντηση είναι «ναι». Αν το δεδομένο μπορεί να έχει καταχωρηθεί σε περισσότερες από μία στοίβες, η απάντηση είναι «ίσως». Κάποια σύμβαση είναι απαραίτητη για την αποδέσμευση μνήμης ακόμα και σε αυτή την απλή αφαίρεση ενός στοιχείου. Κάτι τέτοιο είτε θα έκανε ιδιαίτερα πολύπλοκες τις συναρτήσεις που επεξεργάζονται τη στοίβα, είτε θα μείωνε την εφαρμοσιμότητά της στοίβας, είτε θα ανάγκαζε τον προγραμματιστή να διατηρεί αντίγραφα, οπότε οι αποφάσεις για αποδέσμευση μνήμης να παίρνονται τοπικά. Η «καθαρότερη» λύση όμως για αυτή την περίπτωση είναι η συλλογή σκουπιδιών.

Θέματα μηχανικής λογισμικού:

Η μηχανική λογισμικού περιγράφεται πιο εύστοχα ως η διαχείριση της πολυπλοκότητας συστημάτων λογισμικού μεγάλης κλίμακας. Τα δύο σημαντικότερα εργαλεία στα χέρια του μηχανικού είναι η αφαιρετική ικανότητα και ο δομημένος σχεδιασμός. Πιστεύουμε ισχυρά ότι η ρητή διαχείριση μνήμης από τον προγραμματιστή έρχεται σε πλήρη αντίθεση με τις δύο αυτές αρχές. Η αυτόματη διαχείριση μνήμης προσθέτει στο αφαιρετικό σχέδιο του προγραμματιστή. Το μοντέλο της διαχείρισης μνήμης είναι λιγότερο χαμηλού επιπέδου και οι προγραμματιστές απαλλάσσονται από την υποχρέωση να διατηρούν τα πάντα με ακρίβεια. Έτσι έχουν περισσότερο χρόνο να ασχοληθούν με θέματα σχεδιασμού υψηλότερου επιπέδου και με την προγραμματιστική υλοποίηση του προβλήματος που πρέπει να επιλύσουν. Η διαχείριση μνήμης από το σύστημα εκτέλεσης υιοθετείται από όλες τις γλώσσες υψηλού επιπέδου για δεδομένα που εκχωρούνται στατικά ή σε στοίβα. Η αφαίρεση αυτών των χαμηλού επιπέδου λεπτομερειών από τη σχεδίαση είναι γενική αρχή όλων των σχεδιαστών σε γλώσσες υψηλού επιπέδου ώστε να επιτυγχάνουν δεδομένα ολικά (global) και με σωστή λεκτική εμβέλεια. Οι προγραμματιστές δεν χρειάζεται να ανησυχούν για το που θα τοποθετήσουν ολικές μεταβλητές ή για το πως θα στήσουν πλαίσια ενεργοποίησης διαδικασιών στη στοίβα. Πιστεύουμε ότι αυτή η άποψη έχει εξίσου σημαντική βαρύτητα και στην περίπτωση δεδομένων αποθηκευμένων σε σωρό στην περίπτωση σύνθετων προβλημάτων.

Αξιόπιστος κώδικας είναι ο ευανάγνωστος κώδικας. Σε επίπεδο υποπρογραμμάτων αυτό σημαίνει ότι η συμπεριφορά του κάθε υποπρογράμματος γίνεται σαφής μέσω του ίδιου του υποπρογράμματος ή, στη χειρότερη περίπτωση, μέσω μερικών γειτονικών υποπρογραμμάτων. Δεν θα πρέπει να είναι απαραίτητη η κατανόηση ολόκληρου του υπόλοιπου προγράμματος προτού αναπτυχθεί ένα νέο υποπρόγραμμα. Αυτό είναι εντελώς απαραίτητο σε συστήματα μεγάλης κλίμακας, όπου εμπλέκονται πολλές ομάδες προγραμματιστών. Στην περίπτωση της ρητής διαχείρισης μνήμης από τον προγραμματιστή είναι δυνατό ένα υποπρόγραμμα να προκαλεί δυσλειτουργία των υπολοίπων δημιουργώντας διαρροές ή πρόωρη αποδέσμευση μνήμης. Η συμπεριφορά των υποπρογραμμάτων πλέον δεν είναι ανεξάρτητη από το υπόλοιπο πρόγραμμα.

Ο πολυπόθητος στόχος να καταφέρουν τα κομμάτια του λογισμικού να συνεργαστούν αρμονικά σαν τα εξαρτήματα του hardware απαιτεί την συγγραφή απλών και καλά ορισμένων διασυνδέσεων (interfaces) μεταξύ τους. Τα επεκτάσιμα υποπρογράμματα μπορούν να συνεργαστούν καλύτερα με άλλα υποπρογράμματα, καθώς το κάθε υποπρόγραμμα μπορεί να λειτουργήσει ανάλογα με τα συμφραζόμενα που του δίνονται. Η αύξηση της σταθερότητας των υποπρογραμμάτων διευκολύνει και τη συντήρηση του συστήματος. Κάθε υποπρόγραμμα θα πρέπει να επικοινωνεί με όσο το δυνατόν λιγότερα υποπρογράμματα και όταν επικοινωνεί με αυτά θα πρέπει ο όγκος των πληροφοριών που ανταλλάσσουν να διατηρείται στο ελάχιστο.

Η ζωντάνια των αντικειμένων είναι μια ενιαία ιδιότητα για ολόκληρο το πρόγραμμα. Το να προσθέτουμε λεπτομερή διαχείριση μνήμης στα υποπρογράμματα αποδυναμώνει την επεκτασιμότητα τους. Επιπλέον, αλλαγές σε ένα υποπρόγραμμα ενδεχομένως να απαιτούν και μεταβολές στη διαχείριση μνήμης του. Καθώς όμως η ζωντάνια των δεδομένων δεν είναι τοπικό ζήτημα, μεταβολές στις λεπτομέρειες του κώδικα ενός υποπρογράμματος μπορεί να δυναμιτίσουν τη σωστή λειτουργία των υπόλοιπων υποπρογραμμάτων.

Παρόλο που η ρητή διαχείριση μνήμης από των προγραμματιστή μπορεί να είναι αποτελεσματική και να ταιριάζει σε μονολιθικά συστήματα χτισμένα από ιεραρχικές σχεδιάσεις όπου η εκχώρηση της μνήμης μπορεί να γίνει σε βήματα, αυτή όμως η σχεδιαστική προσέγγιση δε φαίνεται να έχει κοινά σημεία με τον αντικειμενοστραφή προγραμματισμό. Συγκρούεται με την αρχή της ελάχιστης επικοινωνίας μεταξύ των υποπρογραμμάτων και καταστρέφει τις διασυνδέσεις μεταξύ τους. Αν κάποιο αντικείμενο πρόκειται να ξαναχρησιμοποιηθεί με διαφορετικό περιεχόμενο, το νέο περιεχόμενο θα πρέπει να είναι συμβατό με το παλαιό, γεγονός που περιορίζει την ελευθερία στη σύνθεση αντικειμένων. Κάποιοι ισχυρίζονται ότι το πρόβλημα της διαχείρισης μνήμης σε σύνθετα συστήματα μπορεί να λυθεί χωρίς συλλογή σκουπιδιών μόνο αν η σωστή διαχείριση της μνήμης αποτελέσει κύριο στόχο του προγραμματιστή. Η συλλογή σκουπιδιών ξεχωρίζει το πρόβλημα της διαχείρισης της μνήμης από τις συναρτήσεις της κλάσης, αντί να το διασκορπίζει μέσα στον κώδικα του προγράμματος. Γι' αυτό αποτελεί θεμελιώδες στοιχείο πολλών αντικειμενοστραφών γλωσσών.

Μια ακόμα ένδειξη για την έκταση του προβλήματος είναι και η εκτενής συλλογή από εργαλεία που κυκλοφορούν για να βοηθήσουν στον έλεγχο της σωστής χρήσης του σωρού. Τα πιο γνωστά παραδείγματα περιλαμβάνουν το CenterLine και το Purify. Και μόνο η ύπαρξη τέτοιων προγραμμάτων καταδεικνύει τη σημασία της σωστής διαχείρισης της μνήμης και τη δυσκολία του εγχειρήματος αυτού. Εντούτοις, αυτά τα εργαλεία πρακτικά χρησιμεύουν μόνο κατά τη διόρθωση των λαθών, καθώς προσθέτουν σημαντικά στον υπολογιστικό φόρτο του προγράμματος.

Παρόλο που αυτά τα εργαλεία είναι συνήθως πολύ χρήσιμα στο να εντοπίζουν προγραμματιστικά λάθη, δεν εντοπίζουν την καρδιά του προβλήματος. Τα εργαλεία διόρθωσης δεν κάνουν τίποτα για να βελτιώσουν τις διασυνδέσεις των πολύπλοκων

συστημάτων, ούτε για να αυξήσουν την επαναχρησιμοποίηση των λογισμικών εξαρτημάτων. Σημαντική προσπάθεια πρέπει να καταβληθεί για να διορθωθεί η υλοποίηση ή, ακόμα χειρότερα, η σχεδίαση αν βρεθεί κάποια διαρροή μνήμης ή κάποια μετέωρη αναφορά. Τα εργαλεία διόρθωσης αντιμετωπίζουν τα συμπτώματα, αλλά όχι την ίδια την ασθένεια. Η συλλογή σκουπιδιών, απ' την άλλη, είναι ένα αποτελεσματικό εργαλείο για τον μηχανικό λογισμικού, επειδή ανακουφίζει τον προγραμματιστή από την υποχρέωση να ανακαλύπτει λάθη στη διαχείριση μνήμης προσφέροντας εγγύηση ότι δεν πρόκειται ποτέ να εμφανιστούν.

Έρευνες αποδεικνύουν ότι ένα σημαντικό μέρος του χρόνου ανάπτυξης μιας εφαρμογής θα αφιερωθεί στην αντιμετώπιση προβλημάτων με τη διαχείριση μνήμης. Εκτιμάται ότι 40% του χρόνου ανάπτυξης του συστήματος Mesa δαπανήθηκε γι' αυτό το σκοπό. Σήμερα, οι αντικειμενοστραφείς γλώσσες προγραμματισμού χρησιμοποιούνται όλο και συχνότερα. Τα προγράμματα που είναι γραμμένα σ' αυτές τις γλώσσες τυπικά εκχωρούν μεγαλύτερη μερίδα των δεδομένων τους στο σωρό από ότι τα συμβατικά αντίστοιχά τους. Οι δομές δεδομένων που δημιουργούνται και τα προβλήματα που χρειάζεται να αντιμετωπιστούν είναι συνήθως πιο πολύπλοκα. Αυτοί οι παράγοντες απλώς αυξάνουν την πολυπλοκότητα υλοποίησης ρητής διαχείρισης μνήμης.

Οι σχεδιαστές και οι προγραμματιστές μπαίνουν στον πειρασμό να είναι υπερβολικά αμυντικοί προκειμένου να αποφύγουν τις δυσκολίες της ρητής δυναμικής διαχείρισης μνήμης. Τα δεδομένα εκχωρούνται στατικά ή αντιγράφονται μεταξύ των υποπρογραμμάτων αντί να μοιράζονται. Έτσι, το κάθε υποπρόγραμμα μπορεί ελεύθερα να διαγράψει το τοπικό του αντίγραφο. Η ενιαία έννοια της ζωντάνιας διασπάται σε τοπικές. Αυτή όμως η συνήθως αναίτια αντιγραφή και στατική εκχώρηση είναι, στην καλύτερη περίπτωση, σπατάλη χώρου καθώς πρέπει να γίνει υπερεκτίμηση των αναγκών για μνήμη. Αν χρησιμοποιηθεί σε μεγαλύτερα προβλήματα, πάντως, οι στατικοί περιορισμοί μπορεί να αποδειχτούν ανεπαρκείς και τα προγράμματα θα αποτύχουν.

Μια συνήθη εναλλακτική τακτική είναι η χρήση domain-specific συλλεκτών σκουπιδιών. Οι συγκεκριμένοι συλλέκτες όμως συχνά δεν καταφέρνουν να εκμεταλλευτούν την πρόοδο στις ευρύτερες τεχνικές συλλογής σκουπιδιών. Επίσης, εξ ορισμού οι εφαρμογές τους είναι περιορισμένες, οπότε το κόστος κατασκευής τους δεν μπορεί να αποσβεστεί μέσω της χρήσης σε ευρύτερο σύνολο προγραμμάτων. Για τον ίδιο λόγο πιθανότατα θα είναι περιορισμένος και ο έλεγχος για τη σωστή λειτουργία τους. Ίσως η ύπαρξη και μόνο τόσο αδύναμων συλλεκτών να καταδεικνύει την ανάγκη για συλλογή σκουπιδιών. Η λύση είναι η συλλογή σκουπιδιών να ενσωματωθεί στο σύστημα και να μην είναι απλώς μία προσθήκη σε αυτό.

Σίγουρα όχι η απόλυτη λύση:

Δεν υποστηρίζουμε ότι η χρήση συλλογής σκουπιδιών είναι υποχρεωτική για την επίλυση οποιουδήποτε προβλήματος σε οποιαδήποτε γλώσσα προγραμματισμού. Προγράμματα με ξεκάθαρες απαιτήσεις σε διαχείριση μνήμης μπορούν να επιλυθούν με μικρότερο υπολογιστικό κόστος αν χρησιμοποιηθεί γι' αυτά ρητή διαχείριση μνήμης. Προσοχή, όμως, χρειάζεται σε απλά προβλήματα τα οποία μπορεί να χρησιμοποιηθούν ξανά σε πιο σύνθετα προγράμματα. Το βραχυπρόθεσμο κέρδος μπορεί να στοιχίσει μακροπρόθεσμα. Ακόμα, οι ανάγκες ενός προγράμματος μπορεί να μην καλύπτονται μέσω τις συλλογής σκουπιδιών. Τα απαιτητικά συστήματα πραγματικού χρόνου θέλουν εγγυήσεις ότι η διαχείριση μνήμης δε θα τα εμποδίσει να προλάβουν τις αυστηρές χρονικές προθεσμίες τους. Το πρόβλημα της συλλογής σκουπιδιών σε τέτοια συστήματα δεν έχει επιλυθεί ακόμα χωρίς τη χρήση ειδικού hardware.

Ούτε υποστηρίζουμε ότι η συλλογή σκουπιδιών είναι πανάκια για κάθε πρόβλημα στη διαχείριση μνήμης. Η συλλογή σκουπιδιών έχει το κόστος της, τόσο σε χρόνο όσο και σε χώρο, και θα μιλήσουμε για αυτά στη συνέχεια. Επιπλέον, ενώ η συλλογή σκουπιδιών απαλείφει τα δύο σημαντικότερα κλασικά προβλήματα μνήμης, τους μετέωρους δείκτες και τις διαρροές μνήμης, είναι ευάλωτη στα υπόλοιπα σφάλματα, και επιπροσθέτως χρειάζεται και το δικό της έλεγχο για προβλήματα.

Η συλλογή σκουπιδιών δεν προσφέρει λύση στην περίπτωση δομών δεδομένων που αυξάνονται ανεξέλεγκτα. Τέτοιες δομές είναι εκπληκτικά συνήθης, με τυπικό παράδειγμα την προσωρινή αποθήκευση ενδιάμεσων αποτελεσμάτων ώστε να αποφευχθούν οι διπλές πράξεις. Αυτή η αύξηση συνήθως είναι αμελητέα σε προγράμματα που δοκιμάζονται για μικρά χρονικά διαστήματα, όπου το πρόγραμμα συνήθως τερματίζεται εγκαίρως προτού να εξαντληθεί η μνήμη. Αν όμως το μέγεθος του προγράμματος αυξηθεί ή αποτελέσει τμήμα ενός εξυπηρετητή (server) που λειτουργεί για πολλές ώρες, το πρόγραμμα μπορεί να κολλήσει.

Τονίσαμε νωρίτερα ότι ένα από τα κυριότερα πλεονεκτήματα της συλλογής σκουπιδιών είναι η υποστήριξη αφαιρετικών μορφών που οδηγεί σε απλούστερες διασυνδέσεις μεταξύ των εξαρτημάτων του λογισμικού. Δυστυχώς αυτές οι αφαιρετικές μορφές μπορεί να κρύβουν μια άλλη πηγή λαθών αν η συμπαγής αναπαράσταση ενός αντικειμένου αναφέρεται σε δεδομένα στο σωρό, ενώ η αφαιρετική υπόστασή του δεν το κάνει αυτό. Το πιο κοινό παράδειγμα τέτοιας συμπεριφοράς είναι μία στοίβα με αναφορές σε στοιχεία του σωρού, η οποία έχει υλοποιηθεί με πίνακα. Τι θα πρέπει να κάνει η εντολή pop; Απ' τη σκοπιά της αφαιρετικής αναπαράστασης της στοίβας θα πρέπει να επιστρέψει την αναφορά στο σωρό για το στοιχείο στο οποίο δείχνει η κορυφή της στοίβας, και μετά να μειώσει κατά ένα τον δείκτη της κορυφής της στοίβας. Αυτό όμως αφήνει το στοιχείο του σωρού να είναι ακόμα προσβάσιμο από τον πίνακα, ο οποίος παραμένει στη μνήμη. Η ασφαλής λύση είναι μέσω της pop να γραφτεί η τιμή null στην κορυφή της στοίβας προτού μειωθεί η τιμή του δείκτη.

Οι έμμεσες τεχνικές συλλογής σκουπιδιών ξεχωρίζουν τα ζωντανά δεδομένα ακολουθώντας τους δείκτες από τις ρίζες που υπάρχουν στο πρόγραμμα, συμπεριλαμβανόμενης και της στοίβας του προγράμματος. Δυστυχώς η στοίβα μπορεί να μολυνθεί με ανενεργούς δείκτες, τους οποίους αν ακολουθήσει ο συλλέκτης σκουπιδιών θα θεωρήσει ζωντανούς τους προορισμούς τους και θα προκληθεί διαρροή μνήμης. Ένας τρόπος με τον οποίο μπορούν να παραμείνουν ανενεργοί δείκτες στη στοίβα είναι αν παραληφθεί η μετατροπή των τοπικών μεταβλητών σε null μετά την τελευταία χρήση τους. Όμως, ένα πλαίσιο της στοίβας μπορεί να κληρονομήσει ανενεργούς δείκτες από ένα άλλο πλαίσιο όταν αυτό τερματιστεί. Έστω η διαδικασία Α που αρχικά καλεί την διαδικασία Β και μετά τη Γ, και η Β αποθηκεύει έναν δείκτη x προς δεδομένα στο σωρό. Αν η Β ολοκληρωθεί χωρίς να καθαρίσει το χώρο της - και επειδή αυτό είναι δαπανηρό σε χρόνο συχνά δε γίνεται – και η Γ μετά δεσμεύσει το χώρο όπου βρισκόταν το x και πάλι χωρίς να καθαρίσει αρχικά, τότε το αντικείμενο του σωρού όπου έδειχνε ο x θα γίνει και πάλι προσβάσιμο – είναι σαν να αναστήθηκε! Παρόλο που το πρόβλημα αυτό είναι γνωστό στους κατασκευαστές συλλεκτών σκουπιδιών, είναι αρκετά επικίνδυνο καθώς ο x φαίνεται να είναι ένας καθ' όλα έγκυρος δείκτης. Υπό φυσιολογικές συνθήκες το πρόβλημα αυτό δεν είναι σοβαρό. Ο χώρος στον οποίο βρίσκεται ο x πιθανότατα θα χρησιμοποιηθεί και θα αλλάξει τιμή πριν η συλλογή σκουπιδιών εκτελεστεί, έτσι τα δεδομένα στο σωρό θα απομακρυνθούν κανονικά ως σκουπίδια. Τα πολυνηματικά όμως συστήματα είναι ιδιαίτερα ευπαθή σε τέτοιου είδους διαρροές. Στο προηγούμενο παράδειγμα το νήμα που εκτελεί τη Γ μπορεί να μπλοκαριστεί, και να συμβούν αρκετές συλλογές σκουπιδιών μέχρι τελικά το x να αλλάξει τιμή.

Έχουν δημιουργηθεί εργαλεία για την καταπολέμηση τέτοιων προβλημάτων σε προγράμματα στη γλώσσα Modula-3. Η εν λόγω γλώσσα έχει ισχυρό σύστημα τύπων και κάθε αντικείμενο στο σωρό έχει μια ετικέτα με τον τύπο του. Τα εργαλεία αυτά επιτρέπουν κατά την ανάθεση μνήμης στο σωρό να ελέγχεται ο τύπος των δεδομένων, και κατά τη χρήση να ελέγχεται και ο τύπος και η συνάρτηση κλίσης (καθώς μερικοί τύποι είναι πανταχού παρόντες). Τα εργαλεία αυτά επίσης επιτρέπουν στον προγραμματιστή να αναγνωρίσει κάθε αντικείμενο που είναι προσβάσιμο από μια συγκεκριμένη ρίζα και να ελέγξει αν ένα αντικείμενο είναι απρόσιτο. Αν ο έλεγχος αποδειχθεί ψευδής τότε το εργαλείο θα τυπώσει μια διαδρομή δεικτών από μια ρίζα προς το αντικείμενο.

2.5. Πόσο δαπανηρή είναι η συλλογή σκουπιδιών;

Η συλλογή σκουπιδιών έχει τη φήμη ότι επιβαρύνει σημαντικά την εκτέλεση των προγραμμάτων. Κατά το παρελθόν, αυτή η κατηγορία ήταν βάσιμη για ορισμένες εφαρμογές παρόλο που το κόστος εξαρτάται σε μεγάλο βαθμό από το σύστημα υλοποίησης. Για παράδειγμα, έρευνες από τη δεκαετία του 70 και τις αρχές της δεκαετίας του 80 έδειξαν ότι μεγάλα προγράμματα σε Lisp τυπικά δαπανούσαν το 40 τοις εκατό του χρόνου εκτέλεσης τους σε συλλογή σκουπιδιών. Σε περιπτώσεις όπου μπορούσαν να γίνουν συγκρίσεις, προγράμματα γραμμένα σε γλώσσες με ενσωματωμένη υποστήριξη συλλογής σκουπιδιών έτρεχαν πιο αργά από τα αντίστοιχα με συμβατικές γλώσσες. Η συλλογή σκουπιδιών αποτελούσε έναν εμφανή αποδιοπομπαίο τράγο. Όμως οι υλοποιήσεις σε τέτοιες γλώσσες συχνά έτρεχαν αργά για άλλους λόγους, όπως ο λιγότερο αποτελεσματικός μηχανισμός για το πέρασμα παραμέτρων ή η υποστήριξη για συναρτήσεις υψηλότερης τάξης ή η καθυστέρηση στην αποτίμηση των εκφράσεων.

Οι σύγχρονες τεχνικές συλλογής σκουπιδιών έχουν μειώσει δραματικά την επιβάρυνση, σε τέτοιο βαθμό που ακόμα και γλώσσες για προγραμματισμό συστημάτων, όπως οι Modula-2+ και η Modula-3 υποστηρίζουν συλλογή σκουπιδιών. Το κόστος της αυτόματης διαχείρισης της μνήμης εξαρτάται έντονα από την εφαρμογή και τη γλώσσα, γι' αυτό δεν μπορούν να δοθούν συγκεκριμένες εκτιμήσεις για την επιβάρυνση που επιφέρει. Για παράδειγμα, η συλλογή σκουπιδιών θα είναι κατά πολύ ελαφρύτερη σε μια γλώσσα προγραμματισμού που χρησιμοποιεί μεταφραστή, από ότι στην ίδια γλώσσα αν υλοποιηθεί με τη χρήση ενός μεταγλωττιστή που κάνει πολλές βελτιστοποιήσεις στον κώδικα του προγραμματιστή. Το ύφος των δοκιμαστικών προγραμμάτων που χρησιμοποιούνται (για παράδειγμα, μπορεί να είναι γραμμένα με συναρτησιακή μορφή) και οι λεπτομέρειες της υλοποίησης της γλώσσας (για παράδειγμα, αν οι καταγραφές ενεργοποίησης των διαδικασιών γίνονται σε στοίβα ή σε σωρό) θα έχουν επίσης προφανές αντίκτυπο. Το κόστος της συλλογής σκουπιδιών επίσης θα εξαρτηθεί από στατιστικά στοιχεία των αντικειμένων, όπως η διασπορά της διάρκειας ζωής τους και του μεγέθους τους. Τέλος, είναι συνήθως δυνατόν να θυσιαστεί μνήμη για ταχύτητα. Σίγουρα η συχνότητα της συλλογής σκουπιδιών θα μπορεί να γίνει μικρότερη αν αυξηθεί το μέγεθος της περιοχής που καθαρίζεται κάθε φορά.

Συνοψίζοντας όλα τα παραπάνω, η συνολική διάρκεια εκτέλεσης της συλλογής σκουπιδιών τυπικά βρίσκεται κάτω από το 20 τοις εκατό της εκτέλεσης του προγράμματος. Ένα ποσοστό της τάξης του 10 τοις εκατό δε θα πρέπει να θεωρείται παράλογο για ένα καλά υλοποιημένο σύστημα. Πάντως, οι μονοψήφιες υποσχέσεις για συλλογή σκουπιδιών θα πρέπει να αντιμετωπίζονται με σκεπτικισμό για το κατά πόσο είναι ρεαλιστικές.

2.6. Συγκρίνοντας αλγορίθμους συλλογής σκουπιδιών

Είναι δύσκολο να γίνει σύγκριση μεταξύ διαφορετικών αλγορίθμων συλλογής σκουπιδιών, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο. Μπορούν βέβαια να υπολογιστούν συναρτήσεις για την πολυπλοκότητα, οι σταθερές όμως που εμφανίζονται σ' αυτές και οι λεπτομέρειες της υλοποίησης συχνά έχουν τεράστια επίδραση στην πραγματική απόδοση. Στη συνέχεια, θα προσπαθήσουμε να μελετήσουμε τις σημαντικότερες τεχνικές. Το πιο εμφανές κόστος που θα πρέπει να λάβουμε υπ' όψιν, από την άποψη τόσο χρόνου όσο και χώρου που απαιτεί ο συλλέκτης, είναι η ανάκτηση κελιών. Δεν είναι όμως αυτός ο μοναδικός παράγοντας. Το κόστος κατά την ανάθεση είναι εξίσου σημαντικό. Ένας αποτελεσματικός αλγόριθμος συλλογής με υψηλό όμως κόστος κατά την ανάθεση νέων κελιών είναι απίθανο να θεωρηθεί αποδοτικός. Κάποιοι αλγόριθμοι επιπλέον επιβαρύνουν και άλλες προγραμματιστικές λειτουργίες, όπως την ανάγνωση ή εγγραφή δεικτών (με κύριο παράδειγμα τις μεθόδους καταγραφής αναφορών). Φυσικά και αυτό είναι κάτι που πρέπει να μην παραμεληθεί. Επίσης, το πρόγραμμα μπορεί να χρειαστεί να διακοπεί όσο τρέχει ο συλλέκτης, ένα κόστος το οποίο σε κάποια είδη εφαρμογών θα θεωρηθεί δυσβάστακτο.

Δυστυχώς, όλες αυτές οι παράμετροι δεν είναι ανεξάρτητες μεταξύ τους. Επιπλέον, τα αποτελέσματα που παρουσιάζονται κατά καιρούς στη σχετική βιβλιογραφία για την αξιολόγηση των διαφόρων μεθόδων συνήθως προέρχονται από διάφορα μηχανήματα με διαφορετικούς επεξεργαστές και λειτουργικά συστήματα. Ο τρόπος υλοποίησης των αλγορίθμων μπορεί να κρύβει ύπουλες και απρόβλεπτες συνέπειες στη συνολική επίδοση. Ο χρόνος εκτέλεσης ενός κύκλου συλλογής μπορεί να εξαρτάται εν μέρει στην τοπολογία και τον όγκο των ζωντανών δεδομένων στο σωρό. Ακόμα και απλά ζητήματα, όπως μικρές μετατροπές στο μέγεθος του σωρού ή στη διάταξη των αντικειμένων, μπορεί να προκαλέσουν την εκτέλεση συλλογών με διαφορετικούς γράφους ζωντανών αντικειμένων. Διαφορετικά σχήματα πρόσβασης δεδομένων αλληλεπιδρούν διαφορετικά με τα υποσυστήματα της μνήμης, δηλαδή τον δίσκο, την κύρια και τη λανθάνουσα μνήμη. Η σειρά με την οποία διασχίζεται ή αντιγράφεται ένας γράφος μπορεί να επηρεάσει τη συμπεριφορά της εικονικής μνήμης του προγράμματος. Θα ήταν ιδανικό να μπορούσαμε να συζητήσουμε τις συνέπειες μίας μόνο επιλογής μας, με «όλα τα άλλα θέματα σταθερά», αλλά αυτό σπάνια επιτυγχάνεται στην πράξη.

Πάντως, μπορούμε να εκτιμήσουμε τις αρχές και τους παράγοντες που θα πρέπει να λάβουμε υπ' όψιν όταν επιλέγουμε έναν αλγόριθμο για συλλογή δεδομένων. Η συλλογή δεδομένων θα πρέπει να είναι *ασφαλής*. Σε καμιά περίπτωση δε θα πρέπει να αποδεσμεύει ζωντανά δεδομένα. Υπάρχει πάντως το ενδεχόμενο κάποιοι συλλέκτες να οδηγηθούν σε συμβιβασμούς ενώπιον επιθετικών μεταγλωττιστών οι οποίοι, στα πλαίσια των βελτιώσεων που θα κάνουν στο τελικό πρόγραμμα, θα αγνοήσουν τις εντολές για τον έλεγχο στην προσβασιμότητα των δεικτών.

Οι συλλέκτες σκουπιδιών θα πρέπει να είναι αποτελεσματικοί. Δεν θα πρέπει να αφήνουν σκουπίδια στο σωρό. Εν τούτοις, η προσέγγιση των διαφόρων συλλεκτών σχετικά με το τι είναι αποτελεσματικότητα διαφέρει. Οι περισσότεροι συλλέκτες μέτρησης αναφορών δεν μπορούν να ανακαλέσουν κυκλικά συνδεδεμένες δομές σκουπιδιών. Άλλοι συλλέκτες, αντί να καθαρίζουν ολόκληρο το σωρό επικεντρώνουν την προσπάθειά τους, κατά τη διάρκεια ενός κύκλου συλλογής, σε κάποιο συγκεκριμένη περιοχή του σωρού. Λογικό είναι να προβληματιζόμαστε πότε θα καθαριστεί και ο υπόλοιπος σωρός και με ποιο κόστος. Εναλλακτικά, ένας κύκλος συλλογής μπορεί να μοιράζεται χρόνο εκτέλεσης με το πρόγραμμα του προγραμματιστή. Η πλέον αποτελεσματική πολιτική συλλογής εγγυάται ότι κάθε δεδομένο που έγινε σκουπίδι πριν την ολοκλήρωση της συλλογής θα ανακτηθεί στον

τρέχοντα κύκλο. Πάντως μια τέτοια πολιτική θα είναι στις περισσότερες περιπτώσεις ιδιαίτερα δαπανηρή στην υλοποίησή της. Ο συλλέκτης θα πρέπει να έχει πιο χαλαρή συμπεριφορά και να συλλέγει ορισμένα αντικείμενα στους επόμενους κύκλους.

Ο προγραμματιστής θα πρέπει να συνυπολογίσει στο χρόνο εκτέλεσης του προγράμματος και το κόστος της συλλογής σκουπιδιών. Ένας παράγοντας είναι ο συνολικός χρόνος που αφιερώνει το πρόγραμμα εκτελώντας τον ίδιο τον συλλέκτη. Στα αλληλεπιδραστικά προγράμματα είναι επίσης σημαντικό το ενδεχόμενο το πρόγραμμα να διακόπτεται κατά τη διάρκεια της συλλογής και, αν συμβαίνει αυτό, ποια είναι η διάρκεια αυτών των παύσεων. Αν ο συλλέκτης μπορεί να ανακαλέσει το σωρό κατά περιοχές, η διάρκεια κάθε παύσης θα είναι σημαντικά μικρότερη, απ' όσο αν καθάριζε ολόκληρος ο σωρός. Η σχετική συχνότητα αυτών των μικρών συλλογών σε σχέση με τις πλήρεις μπορεί να είναι σημαντική.

Οι προσθετικοί (incremental) συλλέκτες δεν διακόπτουν τη λειτουργία του προγράμματος κατά τη διάρκεια της συλλογής. Προκαλούν, όμως, σύντομες παύσεις κατά την έναρξη κάθε κύκλου, καθώς ο συλλέκτης αρχικοποιείται. Για παράδειγμα μπορεί να χρειαστεί η λήψη ενός «στιγμιοτύπου» της κατάστασης του προγράμματος εξετάζοντας τις ρίζες. Οι ασύγχρονοι προσθετικοί (non-concurrent incremental) συλλέκτες επίσης θα διακόψουν για λίγο το πρόγραμμα σε κάθε βήμα του αλγόριθμου συλλογής καθώς θα εκτελείται ένα μέρος της δουλειάς από τον συλλέκτη. Αυτό μπορεί να ποικίλει σε μέγεθος από έναν μόνο κόμβο μέχρι τη σάρωση μιας ολόκληρης σελίδας εικονικής μνήμης. Ένας επιπλέον παράγοντας στην προσθετική συλλογή είναι το κόστος ελέγχου αν ο κύκλος της συλλογής έχει ολοκληρωθεί. Και αυτό επίσης μπορεί να προκαλέσει αναβολή του προγράμματος.

Συνολικά ο χρόνος της συλλογής σκουπιδιών και η διάρκεια των παύσεων δεν είναι οι μόνοι χρονικοί παράγοντες που πρέπει να συνεκτιμηθούν. Για καλή αλληλεπιδραστική απόκριση ή σε εφαρμογές πραγματικού χρόνου, δεν είναι αρκετή απλώς η μείωση αυτών των χρόνων. Είναι εξίσου απαραίτητο να τοποθετηθούν όρια για τον χρόνο που δαπανάται από τον συλλέκτη κάθε χρονική στιγμή, ούτως ώστε να είναι εφικτό να προχωράει και το πρόγραμμα σε έναν ικανοποιητικό ρυθμό.

Το κόστος εκχώρησης νέων δεδομένων στο σωρό είναι τόσο σημαντικό όσο και ο χρόνος που δαπανάται κατά τη συλλογή σκουπιδιών. Σε γενικές γραμμές, είναι πιο δαπανηρό να εκχωρήσεις μνήμη σε έναν κατακερματισμένο σωρό, καθώς θα πρέπει να γίνει αναζήτηση για μια συνεχή περιοχή αρκετά μεγάλου μεγέθους για το νέο αντικείμενο. Η αναζήτηση αυτή θα είναι ευκολότερη αν όλα τα αντικείμενα είναι όμοια και έχουν το ίδιο μέγεθος, παρά αν δεσμεύονται δεδομένα διαφορετικού μεγέθους. Το πρόβλημα της ανάθεσης χώρου σε δεδομένα διαφόρων μεγεθών σε έναν κατακερματισμένο σωρό δεν είναι μοναδικό για τη συλλογή σκουπιδιών αλλά είναι κοινό σε όλα τα συστήματα διαχείρισης μνήμης, τόσο στα χειροκίνητα όσο και στα αυτόματα.

Η αυτόματη διαχείριση μνήμης μπορεί να επιβάλει μια άμεση επιβάρυνση στις λειτουργίες του προγράμματος, όπως είναι οι εγγραφές σε δείκτες. Τα απλούστερα συστήματα καταμέτρησης αναφορών απαιτούν ανανέωση της μέτρησης όποτε ένας δείκτης στο σωρό δημιουργείται ή σβήνεται. Πιο καλοδουλεμένοι συλλέκτες αυτού του είδους μπορεί να αντιμετωπίσουν πιο χαλαρά αυτή την καταμέτρηση ώστε να μειώσουν την επιβάρυνση προς το πρόγραμμα. Οι προσθετικοί και οι γενεαλογικοί (generational) συλλέκτες καθαρίζουν ένα μέρος του σωρού κάθε φορά. Οι προσθετικοί συλλέκτες συνήθως εγγυώνται τον καθαρισμό κάθε σκουπιδιού που δημιουργήθηκε πριν την εκτέλεση του τρέχοντος κύκλου συλλογής. Οι γενεαλογικοί συλλέκτες καθαρίζουν μόνο μια περιοχή του σωρού, μία *γενιά*, σε κάθε κύκλο. Και τα δύο αυτά είδη προσθέτουν μια χρονική επιβάρυνση στο πρόγραμμα για

να αναφέρουν κάθε αλλαγή που συμβαίνει στις συνδέσεις του γράφου καθώς ο συλλέκτης εκτελείται. Οι γενεαλογικοί συλλέκτες απαιτούν από το πρόγραμμα να καταγράφει ένα αρχείο των αναφορών σε κελιά μιας γενιάς σε κελιά μιας άλλης (συνήθως παλαιότερης) γενιάς.

Ο ρόλος του συλλέκτη είναι τυπικά να ανακτά μνήμη όταν το πρόγραμμα εξαντλεί το σωρό. Όμως, ο συλλέκτης μπορεί να απαιτεί επιπλέον μνήμη για τις ανάγκες του και αυτή η επιβάρυνση σε χώρο πρέπει να ληφθεί σοβαρά υπ' όψιν. Οι συλλέκτες μπορεί να απαιτούν χώρο στο σωρό σε κάθε κελί για να καταγράφουν τις αναφορές σ' αυτό, bit ελέγχου ότι το κελί είναι ζωντανό, ή τη νέα διεύθυνση του κελιού (αν ο συλλέκτης πρόκειται να κάνει μετακινήσεις). Ένας συλλέκτης μπορεί επίσης να διατηρεί πληροφορίες σε κάθε κελί για τους δείκτες που είναι αποθηκευμένοι σε αυτό (αν και συνήθως και το πρόγραμμα διατηρεί εκ των πραγμάτων αυτές τις πληροφορίες και δεν πρέπει να θεωρηθούν επιπλέον επιβάρυνση εξαιτίας του συλλέκτη).

Ο συλλέκτης μπορεί επίσης να εισάγει δικούς του βοηθητικούς τύπους δεδομένων, όπως μια στοίβα για την αναδρομική διάσχιση του σωρού. Οι συλλέκτες αντιγραφής χρειάζονται επιπλέον χώρο στη μνήμη, σε σχέση με τους συλλέκτες που δεν κάνουν μετακινήσεις, καθώς μεταφέρουν όλα τα ζωντανά δεδομένα από την περιοχή που καταλαμβάνουν και τα αντιγράφουν σε μια νέα περιοχή της μνήμης. Ανάλογα με τη διάταξη του σωρού και τη στρατηγική του συλλέκτη, οι συλλέκτες αντιγραφής μπορεί να απαιτούν ακόμα και διπλάσιο χώρο στη μνήμη από ότι τα υπόλοιπα είδη συλλεκτών.

Το κόστος ενός συγκεκριμένου αλγόριθμου συλλογής δεν μπορεί να υπολογιστεί με απλές αναλύσεις πολυπλοκότητας, όπως ότι είναι ανάλογη του μεγέθους του σωρού ή του όγκου των ζωντανών δεδομένων. Οι σταθερές στις συναρτήσεις της πολυπλοκότητας έχουν επίσης μεγάλη σημασία. Επίσης, η καταμέτρηση του αριθμού των εντολών που εκτελούνται κατά την ανάθεση μνήμης και κατά τη συλλογή σκουπιδιών δεν παρέχει μια πλήρη απάντηση. Οι συνέπειες τις τοπικότητας των αναφορών ενός προγράμματος είναι επίσης σημαντική. Πρόσφατες έρευνες έχουν δείξει ακόμα ότι διαφορετικά είδη συλλεκτών σκουπιδιών έχουν διαφορετικές επιδόσεις τόσο σε επίπεδο εικονικής μνήμης, όσο και στην λανθάνουσα μνήμη. Ίσως να είναι δυνατόν να αλλάξουμε τη συμπεριφορά του συλλέκτη ώστε να τα βελτιώσουμε και τα δύο. Συγκεκριμένα, αξίζει τον κόπο να δαπανηθεί χρόνος από τον επεξεργαστή για να μειωθεί ο κατακερματισμός σε ένα περιβάλλον εικονικής μνήμης. Μπορεί να είναι δυνατόν να χρησιμοποιηθεί ο συλλέκτης για τη βελτίωση της τοπικότητας των αναφορών του προγράμματος και με αυτό τον τρόπο να αυξηθεί η απόδοση.

Η *πληρότητα* του σωρού σε ένα πρόγραμμα είναι απίθανο να παραμείνει σταθερή. Οι αλγόριθμοι συλλογής μπορεί να επηρεάζονται από την πληρότητα αυτή. Στους μετρητές αναφορών η πληρότητα δεν είναι πρόβλημα, αλλά οι έμμεσοι συλλέκτες θα παρεμβαίνουν πιο συχνά διακόπτοντας τη ροή του προγράμματος αν η πληρότητα του σωρού είναι υψηλή. Είναι σημαντικό να γνωρίζουμε πόσο ευνοϊκά μεταβάλλονται οι επιδόσεις της αυτόματης διαχείρισης μνήμης σε συνάρτηση με την πληρότητα του σωρού του προγράμματός μας.

Τέλος, οι αλγόριθμοι συλλογής σκουπιδιών μπορεί να είναι γενικής χρήσεως, ή μπορεί η εφαρμογή τους να περιορίζεται σε συγκεκριμένα είδη προγραμματιστικών γλωσσών (για παράδειγμα σε καθαρά συναρτησιακές γλώσσες ή σε λογικές γλώσσες), ή να περιορίζεται σε συγκεκριμένα γλωσσικά ιδιώματα (για παράδειγμα να διευκρινίζεται ο τρόπος με τον οποίο δημιουργούνται ή χρησιμοποιούνται οι κυκλικές δομές δεδομένων).

Πολλοί από αυτούς τους παράγοντες έρχονται σε σύγκρουση μεταξύ τους. Η επιλογή μεταξύ χρόνου και χώρου είναι συνηθισμένη στην επιστήμη των υπολογιστών και η κάθε εφαρμογή καλείται να θέσει τις προτεραιότητές της. Για παράδειγμα, μια αλληλεπιδραστική εφαρμογή θα δώσει έμφαση στο να είναι μικροί οι χρόνοι παύσης, ενώ ο συνολικός χρόνος

εκτέλεσης θα είναι πιο σημαντικός σε ένα μη αλληλεπιδραστικό πρόγραμμα. Μια εφαρμογή πραγματικού χρόνου θα απαιτεί χαμηλά ανώτατα όρια τόσο στις παύσεις της συλλογής σκουπιδιών όσο και στο ποσοστό του υπολογιστικού χρόνου που δαπανά ο συλλέκτης ανά πάσα στιγμή. Η σωστή συμπεριφορά σελιδοποίησης είναι σημαντική για ένα πρόγραμμα που εκτελείται σε ένα σταθμό εργασίας (workstation) σε εικονική μνήμη, ενώ η χαμηλή επιβάρυνση σε χώρο μπορεί να είναι η πρώτη προτεραιότητα ενός προγράμματος που τρέχει σε έναν μικρό προσωπικό ηλεκτρονικό υπολογιστή ή σε ένα ενσωματωμένο σύστημα.

Η μεταφερσιμότητα του συλλέκτη μεταξύ διαφορετικών αρχιτεκτονικών, μεταγλωττιστών ή εφαρμογών μπορεί επίσης να είναι σημαντικό μέλημα. Επίσης, και η ευκολία συντήρησης θα πρέπει να έχει βαρύτητα, όπως και σε κάθε είδος σχεδίασης. Οι συλλέκτες που μετρούν αναφορές θα είναι πιο στενά συνδεδεμένοι με τον μεταγλωττιστή και η συντήρησή τους θα έχει μεγαλύτερες δυσκολίες από άλλους με ευκολότερη διασύνδεση, όπως είναι οι έμμεσοι συλλέκτες.

Αναπόφευκτα, θα κλιθούμε να πάρουμε αποφάσεις και να θυσιάσουμε κάποια από τα παραπάνω για να αποκτήσουμε τα υπόλοιπα. Στη συνέχεια απλώς θα συγκρίνουμε κάποιες μεθόδους που έχουν προταθεί. Δεν θα δώσουμε «απόλυτη» λύση στην ερώτηση ποια στρατηγική συλλογής να χρησιμοποιείτε, αλλά ελπίζουμε να εγείρουμε σωστά ερωτήματα και να προτείνουμε ορθές προσεγγίσεις ώστε να μπορείτε να διερευνήσετε το θέμα για την περίπτωση που αντιμετωπίζετε. Το σύνθημα της συλλογής σκουπιδιών θα πρέπει να είναι «μάθε τις απαιτήσεις του συστήματος και κατανόησε τα δημογραφικά στοιχεία – τον όγκο, τον τύπο, την τοπολογία και τη διάρκεια ζωής – των δεδομένων που θα παράγονται». Δεν πρόκειται για μια πρόταση αποκλειστικά για τη συλλογή σκουπιδιών, εφαρμόζεται εξίσου και στη χειροκίνητη διαχείριση μνήμης. Είναι φανερό ότι στα προγράμματα που χρησιμοποιείτε, οι επιδόσεις μπορεί να βελτιωθούν με καλύτερη κατανόηση τόσο της συμπεριφοράς του προγράμματος, όσο και της συμπεριφοράς του μηχανισμού ανάθεσης μνήμης.

Συνοψίζοντας, τονίζουμε ότι η συλλογή σκουπιδιών είναι ένα χρήσιμο εργαλείο στα χέρια του μηχανικού λογισμικού. Πιστεύουμε ότι είναι απαραίτητη για συγκεκριμένα προβλήματα και είδη προγραμματισμού και τουλάχιστον ρεαλιστική εναλλακτική πρόταση για τα υπόλοιπα. Η εμπειρία αποδεικνύει ότι η χρήση συστημάτων με συλλογή σκουπιδιών οδηγεί σε ταχύτερους χρόνους ανάπτυξης. Στη χειρότερη περίπτωση αξίζει τον κόπο να εκτιμηθούν ως εναλλακτικές λύσεις στη χειροκίνητη διαχείριση μνήμης, όπου ο χρόνος που θα δαπανιόταν για την αναζήτηση και αντιμετώπιση προβλημάτων στην μνήμη μπορεί να αξιοποιηθεί καλύτερα σε άλλα θέματα βελτίωσης της απόδοσης ή της λειτουργικότητας.

Στη συνέχεια θα παρουσιάσουμε τις κυρίαρχες προγραμματιστικές μεθόδους ανάκτησης μνήμης, ξεκινώντας με τις τρεις κλασικές μεθόδους: την καταμέτρηση αναφορών, την εκκαθάριση με σήμανση (mark-sweep) και την αντιγραφή. Καθώς οι τεχνικές και οι ιδέες πίσω από αυτούς τους αλγορίθμους αποτελούν τη βάση και των υπόλοιπων μεθόδων, η κατανόησή τους για όποιον ασχοληθεί σοβαρά με την αυτόματη διαχείριση μνήμης είναι απαραίτητη. Ακολουθεί ένα κεφάλαιο με σχόλια και συγκρίσεις για τις τρεις αυτές τεχνικές, που συνοψίζει όλους τους παράγοντες που θα πρέπει να συνυπολογίσουμε όταν αποφασίζουμε ποια μέθοδος ταιριάζει καλύτερα στις ανάγκες μας. Ολοκληρώνουμε με μια ενότητα που αναφέρει εν συντομία μερικές από τις τεχνικές που έχουν παρουσιαστεί για την περαιτέρω βελτίωση των κλασικών αλγορίθμων.

3. Αλγόριθμοι Καταμέτρησης Αναφορών

Ο πρώτος αλγόριθμος με τον οποίο καταπιανόμαστε είναι μία άμεση μέθοδος η οποία βασίζεται στην καταμέτρηση των αναφορών που υπάρχουν για κάθε κελί, από όλα τα υπόλοιπα ενεργά κελιά και τις ρίζες: την μέθοδο καταμέτρησης αναφορών. Η πρώτη αναφορά στην τεχνική αυτή έγινε το 1960 από τους H. Gelenter, J.R. Hansen και C.L. Gerberich, αλλά ο αλγόριθμος που παρουσίασαν ήταν αναποτελεσματικός και επιρρεπής σε λάθη. Το Δεκέμβριο της ίδιας χρονιάς ο George Collins παρουσίασε τον στάνταρ αλγόριθμο της μεθόδου και θεωρείται ο εισηγητής της. Ο υπολογιστής που χρησιμοποιούσε ο Collins ήταν ένας CDC 1604 με διευθύνσεις των 15 bit και λέξεις των 48 bit. Η αρχιτεκτονική αυτή ήταν ευνοϊκή για την ανάπτυξη της καταμέτρησης αναφορών (βλέπε και αντίστοιχη ιστορική αναφορά για τη μέθοδο εκκαθάρισης με σήμανση στην αρχή του τρίτου κεφαλαίου). Στα αξιοσημείωτα αναφέρουμε ότι ο Collins θεώρησε τον αλγόριθμο της εκκαθάρισης με σήμανση (που προηγήθηκε χρονικά, αλλά θα τον εξετάσουμε στο επόμενο κεφάλαιο) ως «κομψό αλλά αναποτελεσματικό» καθώς σε μετρήσεις που είχε κάνει αποδείκνυε ότι η νέα τεχνική που εισήγαγε ήταν τρεις φορές αποτελεσματικότερη (!) όταν ο σωρός του προγράμματος είναι περίπου μισογεμάτος, όπως τυπικά συμβαίνει. Αυτή η δήλωση προκαλεί έκπληξη και δεν δικαιώνεται, όπως θα δούμε παρακάτω. Θα πρέπει πάντως να θεωρηθεί ως ένα πρώτο παράδειγμα του πόσο στενά συνδεδεμένη είναι η απόδοση μιας τεχνικής με τον υπολογιστή στον οποίο υλοποιείται.

Η αρχή της καταμέτρησης αναφορών βασίζεται στον συνεχή έλεγχο αν ένα κελί είναι σε χρήση ή όχι. Πρόκειται για μία εκ φύσεως προσθετική μέθοδο, η οποία κατανέμει ομοιόμορφα την χρονική επιβάρυνση σε ολόκληρο το πρόγραμμα. Αλγόριθμοι που

στηρίζονται σ' αυτή την τεχνική έχουν αναπτυχθεί για πολλές γλώσσες και εφαρμογές. Για παράδειγμα, οι πρώτες εκδόσεις της αντικειμενοστραφούς γλώσσας Smalltalk, η InterLisp, η Modula-2+ και το δημοφιλές πρόγραμμα Adobe Photoshop. Χρησιμοποιείται επίσης από λειτουργικά συστήματα, όπως το Unix, για να ελεγχθεί αν μπορεί ένα αρχείο να διαγραφεί.

Η καταγραφή αναφορών λειτουργεί διαφορετικά από τους αλγόριθμους έμμεσης συλλογής σκουπιδιών. Κάθε κελί οφείλει να έχει ένα ξεχωριστό πεδίο, τον *αριθμό αναφορών*. Ο διαχειριστής της μνήμης θα πρέπει να διατηρεί αυτά τα πεδία ενήμερα, ώστε ανά πάσα στιγμή να υποδηλώνουν τον αριθμό των δεικτών από τις ρίζες ή τα άλλα κελιά με προορισμό το κελί στο οποίο ανήκει το πεδίο. Αρχικά, όλα τα κελιά τοποθετούνται σε μια δομή ελεύθερων κελιών, η οποία συνήθως είναι μια συνδεδεμένη λίστα – μια αλυσίδα κελιών που συνδέονται με πεδία δεικτών τα οποία θα ονομάζουμε next – καθώς και ένας ελεύθερος δείκτης list στην αρχή της λίστας. Το πεδίο next δεν χρειάζεται να είναι ξεχωριστό. Τυπικά μπορεί να είναι το ίδιο που θα χρησιμοποιηθεί και στην καταμέτρηση αναφορών αργότερα. Άλλωστε για τα ελεύθερα κελιά δεν χρειάζεται να γίνεται καταμέτρηση αναφορών. Εναλλακτικά πάντως, μπορεί να χρησιμοποιηθεί οποιοδήποτε άλλο πεδίο του κελιού το οποίο προορίζεται μετά για χρήση από τον προγραμματιστή.

3.1. Ο αλγόριθμος

Τα ελεύθερα κελιά έχουν μετρητή αναφορών μηδέν, αφού κανένας δείκτης δεν κατευθύνεται σ' αυτά. Όταν ένα νέο κελί εκχωρηθεί από τη λίστα, ο μετρητής του γίνεται αμέσως ένα. Κάθε φορά που ένας δείκτης κατευθύνεται να δείχνει σ' αυτό το κελί, ο μετρητής του αυξάνει κατά ένα, ενώ κάθε φορά που η αναφορά διαγράφεται, ο μετρητής μειώνει κατά ένα. Αν ο μετρητής μηδενιστεί, αυτό υπονοεί ότι δεν υπάρχουν πλέον δείκτες προς αυτό το κελί, οπότε η πληροφορία του δεν είναι πλέον προσβάσιμη από το πρόγραμμα. Επομένως, το κελί δεν είναι πια απαραίτητο και μπορεί να επιστραφεί στη λίστα των ελεύθερων κελιών.

Παρακάτω παρουσιάζουμε πως μπορεί να γίνει εκχώρηση μιας θέσης μνήμης σε σύστημα με καταγραφή αναφορών:

```
allocate() =
    newcell = free_list
    free_list = next (free_list)
    return newcell

New() =
    if free_list == nil
        abort "Memory exhausted"
    newcell = allocate()
    RC(newcell) = 1
    return newcell
```

Ας παρατηρήσουμε τον αλγόριθμο αυτό. Η συνάρτηση allocate είναι ο γενικής χρήσεως μηχανισμός ανάθεσης μνήμης στο πρόγραμμα. Σε αυτή την απλοϊκή περίπτωση απλώς επιστρέφει το πρώτο στοιχείο της ελεύθερης λίστας. Σε άλλους αλγορίθμους αυτή η υλοποίηση θα διαφέρει. Η New επιστρέφει το νέο κελί αφού θέσει το μετρητή του στην τιμή ένα. Αν η λίστα είναι άδεια, η διαδικασία τερματίζεται - θα μπορούσε βεβαίως σε αυτή την περίπτωση να επεκταθεί ο όγκος του σωρού – διαφορετικά η allocate αφαιρεί το πρώτο στοιχείο της λίστας και το επιστρέφει στη New. Για περισσότερη ασφάλεια τα πεδία δεικτών του νέου κελιού θα έπρεπε να σβηστούν, αν και αυτή η επιβάρυνση μπορεί να αποφευχθεί αν

ο προγραμματιστής φροντίσει να αρχικοποιηθεί το κελί αμέσως μετά τη δέσμευσή του. Η ίδια η New θα μπορούσε να συμβάλλει στην αρχικοποίηση του κελιού μεταφέροντας δεδομένα από τη στοίβα του προγράμματος στα πεδία τιμών του κελιού. Για απλούστευση, μέχρι τώρα θεωρούμε ότι τα κελιά έχουν το ίδιο σταθερό μέγεθος.

Στα συστήματα με καταμέτρηση αναφορών η ανανέωση της τιμής ενός δείκτη μπορεί να γίνει με τον τρόπο που φαίνεται παρακάτω:

```
free(N) =
    next(N) = free_list
    free_list = N

delete(T) =
    RC(T) = RC(T) – 1
    if RC(T) == 0
        for U in Children(T)
            delete(*U)
        free(T)

Update(R, S) =
    RC(S) = RC(S) + 1
    delete (*R)
    *R = S
```

Η συνάρτηση Update αντιγράφει την πρώτη παράμετρό της R, που είναι μια λέξη στο σωρό, με τη δεύτερη παράμετρο S, την οποία θεωρούμε δείκτη. Ο μετρητής της S αυξάνεται για να καταμετρήσει αυτή τη νέα αναφορά. Αντίστοιχα, η ανανέωση αφαίρεσε τον αρχικό δείκτη του R προς το *R, οπότε πρέπει να μειωθεί ο μετρητής του *R. Μεριμνώντας να γίνει πρώτα η αύξηση του μετρητή της νέας αναφοράς και μετά η μείωση της παλιάς, καλύπτουμε και την περίπτωση στην οποία οι δύο αναφορές έχουν τον ίδιο προορισμό. Έστω ότι η R αρχικά αναφέρεται στον κόμβο T. Αν αυτός ο δείκτης είναι η τελευταία αναφορά στο T, η delete θα πρέπει να τον επιστρέψει στην ελεύθερη λίστα. Αλλά προτού γίνει αυτό, θα πρέπει να σβήσει αναδρομικά και τα παιδιά του. Σε μία λίγο αποδοτικότερη υλοποίηση η delete θα άφηνε τον μετρητή στο ένα, ώστε να είναι έτοιμος όταν το κελί δεσμευτεί πάλι, και να γλιτώσουμε δύο εντολές.

3.2. Πλεονεκτήματα και μειονεκτήματα της καταμέτρησης αναφορών

Στα θετικά της καταμέτρησης αναφορών συγκαταλέγεται ότι η επιβάρυνση από την αυτόματη διαχείριση μνήμης κατανέμεται στο πρόγραμμα. Η διαχείριση των ενεργών κελιών και των σκουπιδιών αναμιγνύεται με την εκτέλεση του προγράμματος. Αυτό δεν συμβαίνει στους (μη προσθετικούς) έμμεσους αλγορίθμους, όπως είναι η εκκαθάριση με σήμανση, όπου η λειτουργία του συλλέκτη σκουπιδιών αναβάλει την εκτέλεση του προγράμματος. Η καταμέτρηση αναφορών λοιπόν μπορεί να είναι ιδανική στην περίπτωση που απαιτείται ομαλή απόκριση, όπως σε ένα αλληλεπιδραστικό πρόγραμμα ή ένα σύστημα πραγματικού χρόνου. Όμως, ο προαναφερθείς απλός αλγόριθμος καταμέτρησης αναφορών κατανέμει την επιβάρυνση ακαθόριστα. Το κόστος διαγραφής του τελευταίου δείκτη ενός υπογράφου εξαρτάται από το μέγεθός του. Πιο προσεγμένοι όμως αλγόριθμοι μπορούν να διορθώσουν αυτό το πρόβλημα.

Ένα άλλο σχετικό πλεονέκτημα της καταγραφής αναφορών σε σχέση με άλλους αλγόριθμους συλλογής σκουπιδιών είναι ότι η χωρική τοπικότητα αναφορών πιθανότατα δε

θα είναι χειρότερη από αυτήν του προγράμματος. Ένα κελί του οποίου ο μετρητής αναφορών μηδενίζεται, μπορεί να αποδεσμευτεί χωρίς να χρειάζεται πρόσβαση σε κελιά σε άλλες σελίδες του σωρού (εκτός ίσως από τους απογόνους του κελιού). Αυτό δεν συμβαίνει στους έμμεσους αλγόριθμους όπου τυπικά απαιτείται να επισκεφτούμε όλα τα ζωντανά κελιά προτού αποδεσμεύσουμε τα σκουπίδια. Πάντως, θα πρέπει να σημειωθεί ότι η συνάρτηση Update μεταβάλλει τόσο τον μετρητή του παλιού, όσο και του νέου προορισμού του δείκτη όταν το πεδίο του ανανεώνεται. Αν κάποιος από τους δύο μετρητές βρίσκεται εκτός της σελίδας (σε μηχάνημα με εικονική μνήμη) ή δε βρίσκεται στη λανθάνουσα μνήμη, τότε θα συμβεί σφάλμα σελίδας (page fault) ή αστοχία της λανθάνουσας μνήμης (cache miss) αντίστοιχα.

Τρίτον, παρόλο που οι εμπειρικές μελέτες εξαρτώνται και από την υλοποίηση, και από την προγραμματιστική γλώσσα που χρησιμοποιείται, ένα μεγάλο εύρος μελετών υποδεικνύει ότι ελάχιστα κελιά μοιράζονται και πολλά είναι βραχύβια. Ακόμα και η απλούστερη μέθοδος καταγραφής αναφορών επιτρέπει την επαναχρησιμοποίηση αυτών των κελιών αμέσως μετά την αποδέσμευσή τους, σε μια μορφή παρόμοια με στοίβα. Στις έμμεσες μεθόδους τα σκουπίδια παραμένουν αναξιοποίητα μέχρι να εξαντληθεί ο σωρός, οπότε παρεμβαίνει ο συλλέκτης. Η άμεση επαναχρησιμοποίηση των κελιών προκαλεί λιγότερα σφάλματα σελίδας στα συστήματα με εικονική μνήμη, και πιθανότατα καλύτερη αξιοποίηση της λανθάνουσας μνήμης, σε σχέση με τη χρήση εμμέσων μεθόδων συλλογής σκουπιδιών που χρησιμοποιούν νέα κελιά από το σωρό, εκτός αν η κύρια μνήμη ή η λανθάνουσα είναι αρκετά μεγάλη ώστε να χωράει ολόκληρο το σωρό.

Η άμεση γνώση πότε ένα κελί μπορεί να αποδεσμευτεί αποφέρει και άλλα πλεονεκτήματα. Αν χρειάζεται να δημιουργηθεί ένα αντίγραφο ενός αντικειμένου στο οποίο δεν υπάρχουν άλλες αναφορές, το κελί μπορεί να αντιγραφεί με απλή χρήση ενός δείκτη προς αυτό και να ανανεωθούν τα περιεχόμενά του επί τόπου, αντί να χρειάζεται η δέσμευση ενός νέου κελιού, η αντιγραφή των δεδομένων λέξη προς λέξη και τελικά η αποδέσμευση του παλιού κελιού. Αυτή η βελτιστοποίηση έχει ιδιαίτερη σημασία σε μεταγλωττιστές για συναρτησιακές γλώσσες προγραμματισμού. Η καταμέτρηση αναφορών μπορεί επίσης να απλοποιήσει τις διαδικασίες τερματισμού, όπως για παράδειγμα είναι το κλείσιμο αρχείων, καλώντας την αντίστοιχη συνάρτηση αμέσως όταν ένα αντικείμενο δεν χρησιμοποιείται πλέον.

Στον αντίποδα, η καταμέτρηση αναφορών υποφέρει από μια σειρά από μειονεκτήματα τα οποία συνεπάγονται ότι πολλοί δεν τη χρησιμοποιούν στις υλοποιήσεις τους επειδή δεν τη θεωρούν αποδοτική μέθοδο για τη διαχείριση μνήμης. Το σοβαρότερο μειονέκτημα είναι το υψηλό κόστος σε υπολογιστική ισχύ που δαπανάται για την ανανέωση των μετρητών ώστε να είναι πάντα ενημερωμένοι. Κάθε φορά που μεταβάλλεται η τιμή ενός δείκτη, οι μετρητές και του παλιού και του νέου προορισμού θα πρέπει να τροποποιηθούν. Αντίθετα, οι μεταβολές στους δείκτες δεν προκαλούν επιβάρυνση σε συστήματα με έμμεσους αλγορίθμους αυτόματης διαχείρισης μνήμης.

Η καταμέτρηση αναφορών συνδέεται στενά με το πρόγραμμα που εξυπηρετεί ή τον μεταγλωττιστή. Κάθε φορά που αλλάζει ή αντιγράφεται η τιμή ενός δείκτη, πρέπει να ανανεωθούν και οι μετρητές. Για παράδειγμα, σε μια απλή υλοποίηση, αυτό σημαίνει ότι ο μετρητής πρέπει να αυξηθεί όταν ο αντίστοιχος δείκτης σταλεί σε μια υπορουτίνα και να μειωθεί πάλι κατά την έξοδό της. Μία και μόνο παράληψη μπορεί να προκαλέσει καταστροφή. Αυτή η αστάθεια των συστημάτων με καταμέτρηση αναφορών τα κάνει πιο δύσκολα στη συντήρηση από άλλα συστήματα με μεθόδους διαχείρισης μνήμης που δεν συνδέονται τόσο στενά με το πρόγραμμα που τα φιλοξενεί.

Οι τεχνικές καταμέτρησης αναφορών θα πρέπει επιπλέον να χρησιμοποιήσουν για κάθε κελί τον αντίστοιχο μετρητή. Στη χειρότερη περίπτωση, αυτό το πεδίο θα πρέπει να είναι αρκετά μεγάλο ώστε να χωράει τον αριθμό όλων των δεικτών που βρίσκονται στο σωρό και στις ρίζες. Θα πρέπει λοιπόν να έχει το μέγεθος δείκτη. Στην πράξη οι μετρητές δεν πρόκειται ποτέ να γίνουν τόσο μεγάλοι και ένα μικρότερο πεδίο (πιθανόν ακόμα και ένα bit) είναι αρκετό αν χρησιμοποιηθεί σε συνδυασμό με μια τακτική για αντιμετώπιση υπερχειλίσεων.

3.3. Κυκλικές δομές δεδομένων

Πάντως, το μεγαλύτερο μειονέκτημα των απλών αλγορίθμων καταμέτρησης αναφορών είναι η αδυναμία τους να ανακτήσουν κυκλικές δομές. Αυτή η δυνατότητα είναι απαραίτητη σε πολλά συστήματα, καθώς οι κυκλικές δομές δεδομένων χρησιμοποιούνται πιο συχνά από ό,τι φαίνεται με μια πρώτη ματιά. Συνήθη παραδείγματα είναι οι διπλά συνδεδεμένες λίστες και τα «δένδρα» στα οποία υπάρχουν αναφορές από τα φύλλα πίσω προς τη ρίζα. Επίσης, πολλές υλοποιήσεις νωχελικών (lazy) συναρτησιακών γλωσσών, που στηρίζονται σε γράφους, χρησιμοποιούν κύκλους για την αντιμετώπιση των αναδρομών. Ως παράδειγμα, έστω μια διπλά συνδεδεμένη λίστα, στο κεφάλι της οποίας δείχνει μόνο μία αναφορά. Αν η αναφορά αυτή σβηστεί, η συνάρτηση delete θα ελέγξει το κεφάλι της λίστας αν ο μετρητής του είναι μηδέν. Αυτό όμως δε συμβαίνει επειδή στο κεφάλι δείχνει και το δεύτερο στοιχείο της λίστας. Επομένως, η λίστα δε θα σβηστεί, εξαιτίας των εσωτερικών δεικτών που υπάρχουν σ' αυτή, τους οποίους όμως δεν μπορεί να χρησιμοποιήσει το πρόγραμμα για να έχει πρόσβαση στα στοιχεία της. Έτσι, στο γράφο των κελιών δημιουργήθηκε μια απομονωμένη νησίδα, η οποία δεν πρόκειται να χρησιμοποιηθεί στο μέλλον από το πρόγραμμα, αλλά ούτε και ο συλλέκτης δεν μπόρεσε να την καθαρίσει. Ουσιαστικά δηλαδή έχουμε διαρροή μνήμης, ένα τυπικό δείγμα προβληματικής συμπεριφοράς της καταμέτρησης αναφορών στην αντιμετώπιση κυκλικών δομών δεδομένων.

Ευτυχώς, οι υπόλοιπες τεχνικές συλλογής σκουπιδιών δεν έχουν κανένα πρόβλημα με τις κυκλικές δομές δεδομένων, και αρκετοί συγγραφείς συνιστούν το συνδυασμό της καταμέτρησης αναφορών με άλλους έμμεσους συλλέκτες. Η καταμέτρηση αναφορών μπορεί να χρησιμοποιείται μέχρι να εξαντληθεί ο σωρός, οπότε επεμβαίνει ο έμμεσος συλλέκτης. Ο συλλέκτης αυτός ξεκινά μηδενίζοντας πάλι όλους τους μετρητές σε όλα τα κελιά. Ο μετρητής κάθε ενεργού κελιού μετά αποκαθίσταται καθώς ο συλλέκτης αυξάνει κατά ένα την τιμή του κάθε φορά που επισκέπτεται το κελί ξεκινώντας από τις ρίζες, Αφού η καταμέτρηση επισκέπτεται το κάθε κελί ακριβώς μία φορά για κάθε δείκτη προς αυτό από άλλο ενεργό κελί, στο τέλος της διαδικασίας αυτής ο μετρητής κάθε κελιού θα δείχνει τον αριθμό των αναφορών προς αυτό από τα υπόλοιπα ενεργά κελιά, το οποίο είναι αυτό ακριβώς που απαιτεί η καταμέτρηση αναφορών ώστε να είναι έγκυρη. Αυτή η μέθοδος προσφέρει δύο πλεονεκτήματα. Πρώτον, ο έμμεσος συλλέκτης μπορεί εύκολα να ανακυκλώσει τις κυκλικές δομές που δεν χρησιμοποιούνται πλέον. Δεύτερον, μπορούν να χρησιμοποιηθούν μικρά πεδία για τους μετρητές αναφορών, ώστε να μειωθούν οι απαιτήσεις σε μνήμη. Οι μετρητές που φτάνουν στη μέγιστη τιμή που μπορεί να αποθηκευτεί στο μικρό αυτό πεδίο, δεν θα υπόκεινται πλέον σε τροποποιήσεις από τη συνάρτηση Update. Η ευθύνη γι' αυτά τα κελιά θα μεταφέρεται στον έμμεσο συλλέκτη.

4. Αλγόριθμοι Εκκαθάρισης με Σήμανση (Mark – Sweep)

Ο πρώτος, χρονικά, αλγόριθμος που χρησιμοποιήθηκε για αυτόματη ανάκτηση μνήμης ήταν μία έμμεση τεχνική συλλογής σκουπιδιών: η μέθοδος εκκαθάρισης με σήμανση. Ο αλγόριθμος της εκκαθάρισης με σήμανση, όπως και αυτός της καταμέτρησης αναφορών, αναπτύχθηκε για να εξυπηρετήσει την υλοποίηση της προγραμματιστικής γλώσσας Lisp. Μάλιστα παρουσιάστηκαν για πρώτη φορά την ίδια χρονιά, το 1960. Τον Απρίλιο ο John McCarthy παρουσίασε την εκκαθάριση με σήμανση, ενώ στο τέλος της ίδιας χρονιάς, όπως είπαμε και στην προηγούμενη ενότητα, ο George Collins παρουσίασε την καταμέτρηση αναφορών. Οι πρώτες εκδόσεις της Lisp απαιτούσαν από τον προγραμματιστή να διαγράφει χειροκίνητα της λίστες με μια ενσωματωμένη συνάρτηση με το όνομα eralis. Σημειώνουμε ότι η Lisp αναπτύχθηκε στον υπολογιστή IBM 704, ένα μηχάνημα με καταχωρητές 15 bit και λέξεις των 36 bit, εκ των οποίων τα 15 αποτελούσαν τη διεύθυνση.

Η αρχιτεκτονική του συγκεκριμένου υπολογιστή δεν ήταν κατάλληλη για την χρησιμοποίηση καταμέτρησης αναφορών. Επιπλέον η εισαγωγή της μεθόδου αυτής, αντί της εκκαθάρισης με σήμανση, θα απαιτούσε να ξαναγραφτούν από την αρχή τα προγράμματα, εξαιτίας της έντονης σύνδεσης του συλλέκτη με το κυρίως πρόγραμμα. Είναι ενδιαφέρον να αναφέρουμε ότι ο McCarthy υλοποίησε την αναδρομή, που χρειάζεται στην εκκαθάριση με σήμανση, χρησιμοποιώντας ετικέτες (labels), δηλαδή νέα αντίγραφα στους ορισμούς των συναρτήσεων, γεγονός που δεν δημιουργούσε κύκλους. Έτσι, ουσιαστικά απέτρεψε τη δημιουργία κυκλικών δομών δεδομένων αν και, όπως θα δούμε και στη συνέχεια, η εκκαθάριση με σήμανση δεν παρουσιάζει κανένα πρόβλημα στην ανάκτησή τους.

Σύμφωνα με αυτή τη σχεδίαση, τα κελιά δεν ανακτώνται αμέσως όταν γίνονται σκουπίδια, αλλά παραμένουν απρόσιτα από το πρόγραμμα και δεν γίνονται αντιληπτά μέχρι να εξαντληθεί όλος ο διαθέσιμος χώρος. Όταν γίνει μια αίτηση για νέο κελί και δεν υπάρχει άλλη διαθέσιμη μνήμη, η λειτουργία του προγράμματος διακόπτεται προσωρινά και παρεμβαίνει η ρουτίνα του συλλέκτη σκουπιδιών για να καθαρίσει το σωρό από τα άχρηστα κελιά, τα οποία επιστρέφει στη λίστα ελεύθερων κελιών. Η μέθοδος στηρίζεται στη διάσχιση όλων των ζωντανών αντικειμένων για να αποφανθεί ποια κελιά δεν χρησιμοποιούνται και θα πρέπει να ανακυκλωθούν. Αυτή η διάσχιση, που ξεκινάει από τις ρίζες, υποδεικνύει ποια κελιά είναι προσβάσιμα και, εξ ορισμού, ενεργά. Όλοι οι υπόλοιποι κόμβοι είναι σκουπίδια και θα πρέπει να επιστραφούν στη λίστα των ελεύθερων κελιών. Αν ο συλλέκτης σκουπιδιών επιτύχει να αποδεσμεύσει αρκετή μνήμη, μπορεί πλέον να ικανοποιηθεί η αίτηση του προγράμματος για νέο χώρο και η εκτέλεσή του συνεχίζεται.

4.1. Ο αλγόριθμος

Ακολουθεί ένας απλός αλγόριθμος βασισμένος στη μέθοδο εκκαθάρισης με σήμανση:

```
New() =
    if free_pool is empty
        mark_sweep()
    newcell = allocate()
    return newcell
```

Ας προσέξουμε λίγο τον παραπάνω αλγόριθμο. Η συνάρτηση New δεσμεύει ένα νέο κελί και επιστρέφει γι' αυτό έναν δείκτη. Και πάλι, δεν προσδιορίζουμε ακριβώς τον τρόπο λειτουργίας της allocate, αλλά χρησιμοποιούμε τον αφηρημένο όρο free_pool για να περιγράψουμε το σύνολο των ελεύθερων κελιών. Μια πιθανή υλοποίηση είναι να συνδέονται τα ελεύθερα κελιά σε μια συνδεδεμένη λίστα ακριβώς όπως το περιγράψαμε και στην προηγούμενη ενότητα για τους αλγόριθμους καταμέτρησης αναφορών, αλλά υπάρχουν και πιο αποδοτικές εναλλακτικές λύσεις.

Στη μέθοδο εκκαθάρισης με σήμανση, όταν μεταβάλλεται η τιμή ενός δείκτη δεν υπάρχει επιπλέον επιβάρυνση στην εντολή write. Αυτή είναι μια έντονη διαφορά με την μέθοδο καταμέτρησης αναφορών όπου απαιτούνται αρκετές επιπλέον εντολές για τη διαχείριση των μετρητών. Το κόστος του αλγορίθμου μπορεί να αυξηθεί περισσότερο αν προκληθούν αστοχίες στη λανθάνουσα μνήμη ή ακόμα χειρότερα, αν κάποια από τις σελίδες που περιέχει τους μετρητές βρίσκεται τη δεδομένη στιγμή εκτός σελιδοποίησης (paged out).

Η μέθοδος εκκαθάρισης με σήμανση εκτελείται σε δύο φάσεις, όπως φαίνεται στον παρακάτω αλγόριθμο:

```
mark_sweep () =
    for R in Roots
        mark (R)
    sweep ()
    if free_pool is empty
        abort "Memory exhausted"
```

Στην πρώτη φάση, γνωστή ως *σήμανση (marking)*, αναγνωρίζονται όλα τα ενεργά κελιά. Στη δεύτερη φάση, την *εκκαθάριση (sweep)*, επιστρέφονται τα σκουπίδια στο σύνολο των ελεύθερων κελιών. Αν η φάση της εκκαθάρισης αποτύχει να αποδεσμεύσει αρκετό χώρο,

απαιτείται επέκταση του σωρού, διαφορετικά η εκτέλεση του προγράμματος είναι ανέφικτή και θα πρέπει να τερματιστεί.

Ένα bit που συνδέεται με κάθε κελί δεσμεύεται για χρήση από τον συλλέκτη σκουπιδιών. Αυτό το *bit σήμανσης* χρησιμοποιείται για να καταγραφεί αν το κελί είναι προσβάσιμο από τις ρίζες. Καθώς η σήμανση διασχίζει όλα τα ζωντανά κελιά, ξεκινώντας από τις ρίζες, το bit σήμανσης κάθε κελιού ορίζεται στην τιμή που δηλώνει ότι έγινε επίσκεψη. Χάριν απλούστευσης, παρουσιάζουμε τον παρακάτω αλγόριθμο σε μια απλοϊκή αναδρομική μορφή. Η εργασία αυτή μπορεί να γίνει με πολύ αποδοτικότερους τρόπους:

```
mark (N) =
    if mark_bit (N) = = unmarked
        mark_bit (N) = marked
        for M in Children (N)
            mark (*M)
```

Η φάση σήμανσης τερματίζεται καθώς η αναδρομή διακόπτεται στα κελιά που έχουμε ήδη επισκεφτεί. Όταν η φάση ολοκληρωθεί, όλα τα κελιά που είναι προσβάσιμα από τις ρίζες θα έχουν σημαδεμένο το bit σήμανσης τους. Από αυτή την άποψη, η διαδικασία σήμανσης είναι μια απλή ερμηνεία του ορισμού των ζωντανών αντικειμένων ως το σύνολο των κελιών που είναι προσβάσιμα.

Μετά το πέρας της πρώτης φάσης, αναλαμβάνει η εκκαθάριση, με σκοπό να αποδεσμεύσει τα κελιά των οποίων το bit σήμανσης δεν έχει μεταβληθεί:

```
sweep () =
    N = Heap_bottom
    while N < Heap_top
        if mark_bit (N) = = unmarked
            free (N)
        else mark_bit (N) = unmarked
        N = N +size (N)
```

Ο συλλέκτης εκκαθαρίζει τον σωρό γραμμικά, ξεκινώντας από τη βάση και φτάνοντας στην κορυφή. Κατά τη διαδικασία αυτή επιστρέφει τα σκουπίδια στο σύνολο των ελεύθερων αντικειμένων και αρχικοποιεί πάλι το bit σήμανσης των ενεργών κελιών, ώστε να είναι έτοιμα για το νέο κύκλο συλλογής. Και πάλι, δεν ορίζουμε ακριβώς τι σημαίνει αποδέσμευση, απλώς λέμε ότι το αντικείμενο επιστρέφεται στο ελεύθερο σύνολο για να επαναχρησιμοποιηθεί στο μέλλον. Για παράδειγμα, αν το ελεύθερο σύνολο υλοποιηθεί σαν μια απλή συνδεδεμένη λίστα, τότε ο κώδικας που δόθηκε και στην προηγούμενη ενότητα (αλγόριθμοι μέτρησης αναφορών) για τη συνάρτηση free μπορεί να χρησιμοποιηθεί και τώρα:

```
free (N) =
    next (N) = free_list
    free_list = N
```

4.2. Πλεονεκτήματα και μειονεκτήματα της μεθόδου εκκαθάρισης με σήμανση

Οι αλγόριθμοι συλλογής σκουπιδιών αυτής της μορφής έχουν δύο πλεονεκτήματα σε σχέση με τους αντίστοιχους καταμέτρησης αναφορών. Το γεγονός αυτό οδήγησε στην υιοθέτηση της μεθόδου αυτής από αρκετά συστήματα, όπως η συναρτησιακή γλώσσα Μιράντα, αλλά και από συντηρητικούς συλλέκτες σκουπιδιών. Πρώτον, οι κυκλικές δομές δεδομένων αντιμετωπίζονται φυσιολογικά, δεν απαιτούν ιδιαίτερη μεταχείριση. Δεύτερον, το σύστημα δεν επιβαρύνεται με επιπλέον υπολογισμούς κατά τη διαχείριση των τιμών δεικτών. Στον αντίποδα, η εκτέλεση του αλγόριθμου εκκαθάρισης με σήμανση απαιτεί τη διακοπή της εκτέλεσης του κυρίως προγράμματος. Οι παύσεις αυτές που προκαλούνται από το συλλέκτη μπορεί να είναι σημαντικές. Για παράδειγμα, έρευνες στις αρχές της δεκαετίας του ’80 έδειξαν ότι, καθώς το μέγεθος της μνήμης αυξανόταν ταχύτερα από την υπολογιστική ισχύ των επεξεργαστών, ορισμένα μεγάλου μεγέθους προγράμματα σε γλώσσα Lisp δαπανούσαν από 25 έως 40 τοις εκατό του χρόνου τους σε σήμανση και εκκαθάριση σκουπιδιών. Σύμφωνα με τις ίδιες μελέτες, οι χρήστες κατά μέσο όρο περίμεναν 4,5 δευτερόλεπτα κάθε 79 δευτερόλεπτα για τη λειτουργία του συλλέκτη. Σίγουρα, οι αλγόριθμοι εκκαθάρισης με σήμανση που κάνουν ολική διάσχιση χωρίς δυνατότητα διακοπής δεν έχουν πρακτική χρήση σε πραγματικού χρόνου, αλληλεπιδραστικά ή κατανεμημένα συστήματα. Θα ήταν τελείως απαράδεκτο ένα σύστημα ασφαλείας πραγματικού χρόνου ή ακόμα και ένα ηλεκτρονικό παιχνίδι να διακόπτονται για μεγάλες περιόδους κατά τη διάρκεια της συλλογής σκουπιδιών. Μια λύση ενδεχομένως να είναι η απενεργοποίηση του συλλέκτη σε κρίσιμα τμήματα της εκτέλεσης του προγράμματος. Άλλες μέθοδοι για να μειωθεί ο χρόνος παύσης περιλαμβάνουν γενεαλογικές και αυξητικές τεχνικές.

Πάντως, αν ο χρόνος απόκρισης δεν είναι σημαντικό μέλημα, οι αλγόριθμοι εκκαθάρισης με σήμανση όντως προσφέρουν καλύτερες επιδόσεις από τους αυξητικούς αλγορίθμους, όπως είναι η καταμέτρηση αναφορών. Σε κάθε περίπτωση, το κόστος της συλλογής σκουπιδιών είναι υψηλό. Η φάση σήμανσης επισκέπτεται κάθε ενεργό κελί, και όλα τα κελιά ελέγχονται κατά την εκκαθάριση. Επομένως, η πολυπλοκότητα του αλγορίθμου είναι ανάλογη του μεγέθους ολόκληρου του σωρού και όχι, για παράδειγμα, μόνο του πλήθους των ενεργών κελιών.

Επιπλέον, ο απλός αλγόριθμος που παρουσιάστηκε νωρίτερα έχει την τάση να κατακερματίζει τη μνήμη, σκορπίζοντας τα κελιά σε κάθε σημείο του σωρού. Σε ένα σύστημα με πραγματική μνήμη το πλήγμα στις επιδόσεις δεν θα είναι και τόσο σημαντικό, αν και μπορεί να χάνεται το πλεονέκτημα της λανθάνουσας μνήμης. Σε ένα σύστημα με εικονική μνήμη, όμως, αυτός ο κατακερματισμός μπορεί να οδηγήσει σε απώλεια τοπικότητας μεταξύ συνδεδεμένων κελιών μιας δομής δεδομένων, με τραγικές συνέπειες, όπως συνεχείς εναλλαγές (swapping) σελίδων μεταξύ της πρωτεύουσας και δευτερεύουσας μνήμης. Σε κάθε περίπτωση, ο κατακερματισμός δυσκολεύει τη διαδικασία της ανάθεσης νέου χώρου στη μνήμη καθώς θα πρέπει να αναζητηθούν μέσα στο σωρό κενά κατάλληλου μεγέθους για να φιλοξενήσουν τα νέα αντικείμενα.

Επίσης, η έμμεση συλλογή σκουπιδιών απαιτεί ελεύθερο χώρο στο σωρό για να είναι αποδοτική. Έστω ότι ο ρυθμός ανάθεσης νέου χώρου στη μνήμη είναι συνεχής και ότι δεν μας αφορά ο κατακερματισμός. Ο ενδιάμεσος χρόνος μεταξύ των κύκλων συλλογής εξαρτάται από τον όγκο σκουπιδιών που αποδεσμεύεται κάθε φορά. Η συλλογή σκουπιδιών θα είναι λοιπόν πιο συχνή αν αυξηθεί ο όγκος ζωντανών κελιών στο σωρό, με άμεσο αποτέλεσμα τη μείωση του μεριδίου του χρόνου που αφιερώνεται στην εκτέλεση του προγράμματος. Με άλλα λόγια, ο συλλέκτης θα κολλήσει. Στις μεθόδους με καταμέτρηση

αναφορών, αντίθετα, η πληρότητα του σωρού δεν αποτελεί πρόβλημα, αν και εμμέσως μπορεί και εδώ να επηρεάσει τη λειτουργία ανάθεσης μνήμης.

5. Αλγόριθμοι Συλλογής με Αντιγραφή

Το 1963 έγιναν δύο σημαντικά γεγονότα στο χώρο της συλλογής σκουπιδιών. Πρώτον, ο Harold McBeth ήταν ο πρώτος που παρατήρησε ότι οι αλγόριθμοι καταμέτρησης αναφορών παρουσίαζαν αδυναμία στην ανάκτηση κυκλικών δομών και δεύτερον, ο Marvin Minsky παρουσίασε την τρίτη από τις κλασικές μεθόδους συλλογής σκουπιδιών, την μέθοδο της αντιγραφής, την οποία θα εξετάσουμε σε αυτή την ενότητα. Πολλοί ήταν αυτοί που ασχολήθηκαν στη συνέχεια με την τεχνική αυτή, προσπαθώντας να βελτιώσουν τον αλγόριθμο του Minsky. Κυριότερη από αυτές τις προσπάθειες ήταν του C.J. Chimney ο οποίος το 1970 παρουσίασε τον μέχρι και σήμερα πιο διαδεδομένο αλγόριθμο για την μέθοδο της αντιγραφής.

Οι συλλέκτες αντιγραφής χωρίζουν το σωρό σε δύο ίσα υποσύνολα. Το ένα περιέχει ενημερωμένα δεδομένα και το άλλο παρωχημένα. Όταν ξεκινάει τη λειτουργία του ο συλλέκτης αντιστρέφει τους ρόλους των δύο υποσυνόλων. Στη συνέχεια ο συλλέκτης διασχίζει τα ενεργά δεδομένα στο παλιό υποσύνολο, το *αρχικό σύνολο*, αντιγράφοντας κάθε ζωντανό κελί στο νέο υποσύνολο, το *τελικό σύνολο*, όταν επισκέπτεται το κελί για πρώτη φορά. Στο τέλος της διαδικασίας αυτής, όταν όλα τα ενεργά κελιά του αρχικού συνόλου έχουν εντοπιστεί, στο τελικό σύνολο έχει δημιουργηθεί ένα αντίγραφο της δομής των ζωντανών δεδομένων και συνεχίζεται η λειτουργία του προγράμματος του χρήστη. Καθώς τα σκουπίδια απλώς εγκαταλείπονται στο αρχικό σύνολο, οι συλλέκτες αντιγραφής συχνά χαρακτηρίζονται ως *«ρακοσυλλέκτες»* - αναζητούν και παίρνουν χρήσιμα αντικείμενα ανάμεσα στα σκουπίδια.

Μια φυσική και ευνοϊκή παρενέργεια της συλλογής σκουπιδιών με αντιγραφή είναι ότι τα ενεργά δεδομένα συγκεντρώνονται στην αρχή του τελικού συνόλου. Αυτό επιτρέπει στη συνέχεια να γίνεται πολύ πιο εύκολα η ανάθεση νέου χώρου στη μνήμη, αφού τα δεδομένα δεν είναι κατακερματισμένα. Το μόνο που θα πρέπει να κάνει η συνάρτηση New είναι να ελέγξει αν υπάρχει επαρκής χώρος και μετά να αυξήσει το δείκτη του ελεύθερου χώρου, τον οποίο στο εξής θα ονομάζουμε free. Καθώς τα ενεργά δεδομένα είναι συμπιεσμένα στο τελικό σύνολο, ο έλεγχος για επάρκεια χώρου είναι απλώς μια σύγκριση δεικτών, όπως φαίνεται και στον παρακάτω αλγόριθμο. Οι αλγόριθμοι αντιγραφής διαχειρίζονται άνετα δεδομένα διαφορετικού μεγέθους, οπότε μπορούμε να στέλνουμε στη New ως παράμετρο n το μέγεθος του κελιού το οποίο θέλουμε να δεσμευτεί. Όπως και στον αλγόριθμο εκκαθάρισης με σήμανση, η συλλογή σκουπιδιών με αντιγραφή δεν επιβαρύνει με επιπλέον εντολές τις λειτουργίες του προγράμματος όπως την αλλαγή τιμών των δεικτών.

```
init () =
    Tospace = Heap_bottom
    space_size = Heap_size / 2
    top_of_space = Tospace + space_size
    Fromspace = top_of_space + 1
    free = Tospace

New (n) =
    if free + n > top_of_space
        flip ()
    if free + n > top_of_space
        abort “Memory exhausted”
    newcell = free
    free = free +n
    return newcell

flip () =
    Fromspace, Tospace = Tospace, Fromspace
    top_of_space = Tospace + space_size
    free = Tospace
    for R in Roots
    R = copy (R)
```

Σημείωση: Ο P δείχνει σε μία λέξη, όχι ένα κελί

```
copy (P)
    if atomic (P) or P = = nil        - ο P δεν είναι δείκτης
        return P
    if not forwarded (P)
        n = size (P)
        P’ = free                     - δέσμευση χώρου στο τελικό σύνολο
        free = free + n
        temp = P [0]                  - το πεδίο 0 κρατάει τη νέα διεύθυνση
        forwarding_address (P) = P’
        P’ [0] = copy (temp)
        for i = 1 to n-1              - αντιγραφή κάθε πεδίου του P στο P’
            P’ [i] = copy (P[i])
    return forwarding_address (P)
```

5.1. Ο αλγόριθμος

Αρχικά, οι ρόλοι του αρχικού και του τελικού συνόλου αντιστρέφονται μέσω της συνάρτησης flip, η οποία μεταβάλλει τις τιμές των μεταβλητών Tospace, Fromspace και top_of_space, όπως φαίνεται παραπάνω. Μετά, κάθε κελί που είναι προσβάσιμο από μία ρίζα αντιγράφεται από το αρχικό στο τελικό σύνολο. Χάριν απλούστευσης, χρησιμοποιούμε έναν απλό αναδρομικό αλγόριθμο. Υπάρχουν φυσικά και αποδοτικότεροι τρόποι για να επιτευχθεί αυτό. Η συνάρτηση copy (P) αντιγράφει τα πεδία του κελιού στο οποίο δείχνει η P. Θα πρέπει να μεριμνήσουμε όταν αντιγράφονται δομές δεδομένων να διασφαλίσουμε ότι διατηρείται η τοπολογία των δομών που μοιράζονται. Αν αποτύχουμε σ' αυτό θα δημιουργηθούν πολλαπλά αντίγραφα των μοιραζόμενων αντικειμένων, το οποίο στην καλύτερη περίπτωση θα αυξήσει απλώς την πληρότητα του σωρού, στη χειρότερη θα αλλοιώσει τη σημασιολογία του προγράμματος. Δηλαδή, μπορεί να αλλάξει η τιμή σε ένα αντίγραφο και μετά να διαβαστεί η παλιά τιμή από άλλο αντίγραφο. Επίσης, η αντιγραφή κυκλικών δομών χωρίς προσοχή σε αυτή τη λεπτομέρεια θα απαιτούσε πολύ χώρο! Οι συνεχείς αναδρομές μέσα στη δομή δεν θα τερματίζονταν ποτέ.

Οι συλλέκτες αντιγραφής διατηρούν το μοίρασμα δομών αφήνοντας μία *διεύθυνση αποστολής* στο αντικείμενο του αρχικού συνόλου όταν αντιγράφεται. Αυτή η διεύθυνση είναι η νέα του θέση στο τελικό σύνολο. Όποτε συναντάται ένα αντικείμενο στο αρχικό σύνολο, η συνάρτηση copy ελέγχει αν έχει ήδη αντιγραφεί. Αν συμβαίνει αυτό τότε απλώς επιστρέφει τη διεύθυνση που έχει οριστεί από την πρώτη αντιγραφή, διαφορετικά δεσμεύεται μνήμη για τη μετακίνηση στο τελικό σύνολο. Σε αυτόν τον αναδρομικό αλγόριθμο αντιγραφής, πρώτα ανατίθεται η νέα διεύθυνση στη δεσμευμένη μνήμη και μετά μετακινούνται τα πεδία του αντικειμένου στη νέα τους θέση. Με αυτό τον τρόπο διασφαλίζεται και το ότι ο αλγόριθμος θα τερματιστεί, και ότι το μοίρασμα αντικειμένων θα διατηρηθεί.

Η διεύθυνση αποστολής μπορεί να αποθηκεύεται σε δικό της πεδίο μέσα στο κελί. Συνήθως όμως γράφεται στην πρώτη λέξη του κελιού καθώς η τιμή της λέξης έχει εκ των προτέρων αντιγραφεί στη νέα θέση. Στον αλγόριθμο που παρουσιάσαμε παραπάνω θεωρήσαμε ότι η διεύθυνση αποστολής αποθηκεύεται στο πεδίο P[0] του κελιού P και χρησιμοποιήσαμε ισοδύναμα τις εκφράσεις forwarding_address (P) και P[0].

5.2. Πλεονεκτήματα και μειονεκτήματα της συλλογής με αντιγραφή

Τα πλεονεκτήματα της συλλογής σκουπιδιών με αντιγραφή σε σχέση με τις δύο προαναφερθείσες τεχνικές, της καταγραφής αναφορών και της εκκαθάρισης με σήμανση, οδήγησε στην ευρεία χρήση της μεθόδου αυτής. Το κόστος της ανάθεσης νέου χώρου στη μνήμη είναι ελάχιστο, αρκεί μία σύγκριση δεικτών για τον έλεγχο για επάρκεια μνήμης, η δέσμευση νέου χώρου γίνεται απλώς με την αντίστοιχη αύξηση στον δείκτη ελεύθερου χώρου και ο κατακερματισμός εκμηδενίζεται συγκεντρώνοντας τα ενεργά δεδομένα στη βάση του τελικού συνόλου. Το αντίστοιχο κόστος ανάθεσης είναι πολύ μεγαλύτερο για τους άλλους αλγόριθμους που δεν πραγματοποιούν συμπίεση των δεδομένων, κυρίως αν τα κελιά έχουν μεταβλητό μέγεθος. Δύσκολα μπορεί κάποιος να φανταστεί φτηνότερη ανάθεση μνήμης από αυτή που προσφέρει η συλλογή σκουπιδιών με αντιγραφή.

Στα μειονεκτήματα της μεθόδου, εμφανέστερο είναι η χρήση των δύο συνόλων με αποτέλεσμα να διπλασιάζονται οι ανάγκες σε μνήμη σε σχέση με τα άλλα είδη συλλεκτών. Συχνά χρησιμοποιείται το επιχείρημα ότι αυτό δεν είναι πρόβλημα σε συστήματα με εικονική μνήμη καθώς οι σελίδες του μη χρησιμοποιούμενου υποσυνόλου θα αποσυρθούν στη

δευτερεύουσα μνήμη, αλλά σε αυτή την άποψη παραλείπεται το κόστος της σελιδοποίησης. Ας συγκρίνουμε τη συμπεριφορά της εκκαθάρισης με σήμανση και της αντιγραφής σε έναν κύκλο συλλογής σκουπιδιών πάνω σε ένα σωρό δεδομένου μεγέθους. Κατά τη διάρκεια του κύκλου αυτού, ο συλλέκτης με αντιγραφή θα χρησιμοποιήσει κάθε σελίδα του σωρού ασχέτως της τοπικότητας του προγράμματος. Εκτός αν η φυσική μνήμη είναι αρκετά μεγάλη να χωρέσει συγχρόνως και το αρχικό και το τελικό σύνολο, ο συλλέκτης αντιγραφής θα προκαλέσει περισσότερα σφάλματα σελίδας από τον συλλέκτη εκκαθάρισης με σήμανση, καθώς χρησιμοποιεί διπλάσιο αριθμό σελίδων.

Από την άλλη, αυτό μπορεί να αντισταθμίζεται με το πλεονέκτημα της συμπίεσης. Τα δεδομένα στον απλό αλγόριθμο εκκαθάρισης με σήμανση είναι πιθανότατα κατακερματισμένα στο σωρό οδηγώντας σε μία αύξηση του αριθμού των σελίδων που χρησιμοποιεί το πρόγραμμα. Αυτό θα έχει την τάση να αυξήσει τον ρυθμό με τον οποίο εμφανίζονται σφάλματα σελίδας αν το σύνολο των χρησιμοποιούμενων σελίδων δεν χωράει στην κύρια μνήμη. Η έλλειψη τοπικότητας αναφορών επηρεάζει αρνητικά και την απόδοση της λανθάνουσας μνήμης, αλλά αυτό έχει πολύ μικρότερο αντίκτυπο στο συνολικό χρόνο εκτέλεσης του προγράμματος σε σχέση με την αναποτελεσματική σελιδοποίηση.

5.3. Σύγκριση μεταξύ της εκκαθάρισης με σήμανση και της αντιγραφής:

Το κύριο μειονέκτημα των συλλεκτών αντιγραφής είναι η ανάγκη να διχοτομήσουν τη διαθέσιμη μνήμη σε δύο υποσύνολα. Καθώς η πληρότητα του προγράμματος αυξάνει, η απόδοση του συλλέκτη μειώνεται αφού σε κάθε κύκλο συλλογής ανακυκλώνεται λιγότερος ελεύθερος χώρος και η συχνότητα των κύκλων αυξάνεται. Τα προγράμματα με ανάγκες σε μνήμη που είναι μεγαλύτερες από το μέγεθος ενός από τα δύο υποσύνολα θα οδηγηθούν σε αποτυχία. Η χρήση εικονικής μνήμης μπορεί να αντιμετωπίσει αυτά τα συμπτώματα. Με αυτήν το μέγεθος του υποσυνόλου μπορεί να είναι τόσο μεγάλο ή και ακόμα μεγαλύτερο από ότι είναι η φυσική μνήμη, και αν είναι απαραίτητο ο σωρός μπορεί να επεκταθεί ακόμα περισσότερο. Στις μεθόδους εκκαθάρισης με σήμανση όμως, η απόδοση του συλλέκτη μειώνεται σε σχέση με την πληρότητα του σωρού στη μισή ταχύτητα.

Η πολυπλοκότητα της συλλογής με αντιγραφή είναι μικρότερη από αυτήν της απλής εκκαθάρισης με σήμανση. Είναι ανάλογη του μεγέθους των ενεργών δομών δεδομένων και όχι του σωρού (ή του υποσυνόλου). Επιπλέον, αν η πλειοψηφία των κελιών δεν παραμένουν ζωντανά μέχρι τον επόμενο κύκλο συλλογής, όπως συμβαίνει συνήθως στις συναρτησιακές και αντικειμενοστραφείς γλώσσες προγραμματισμού, αρκεί να αντιγραφεί μόνο ένα μικρό μέρος του σωρού. Ας εξετάσουμε περισσότερο την πολυπλοκότητα των δύο μεθόδων. Αρχικά, ας αγνοήσουμε το κόστος ανάθεσης μνήμης. Επίσης, δε θα μας απασχολήσει η επίδραση της τοπικότητας αναφορών στην εικονική και την λανθάνουσα μνήμη. Έστω ότι M είναι το μέγεθος του σωρού και R το μέγεθος των ζωντανών δεδομένων.

Ο συλλέκτης αντιγραφής που περιγράφηκε παραπάνω, θα πρέπει να ακολουθήσει και ανανεώσει κάθε δείκτη που είναι αποθηκευμένος στις ρίζες και τα ενεργά δεδομένα, και να μεταφέρει αυτά τα στοιχεία στο τελικό σύνολο. Η πολυπλοκότητα σε χρόνο λοιπόν του συλλέκτη αντιγραφής στη φάση της συλλογής σκουπιδιών είναι συνάρτηση του R:

$$t_{Copy} = aR$$

Ο συλλέκτης εκκαθάρισης με σήμανση ακολουθεί όλους τους δείκτες προς ζωντανά δεδομένα στη φάση της σήμανσης και διασχίζει γραμμικά όλο το σωρό στη φάση της εκκαθάρισης. Η συνολική χρονική πολυπλοκότητα είναι:

$$t_{MS} = bR + cM$$

Ο όγκος της μνήμης που αποδεσμεύεται από κάθε συλλέκτη είναι:

$$m_{Copy} = \frac{M}{2} - R$$

για τον συλλέκτη αντιγραφής και:

$$m_{MS} = M - R$$

για τον συλλέκτη εκκαθάρισης με σήμανση.

Αν ορίσουμε την απόδοση, e, ενός αλγορίθμου ως την ποσότητα της μνήμης που ανακυκλώνεται στη μονάδα του χρόνου:

$$e = \frac{m}{t}$$

τότε έχουμε απόδοση:

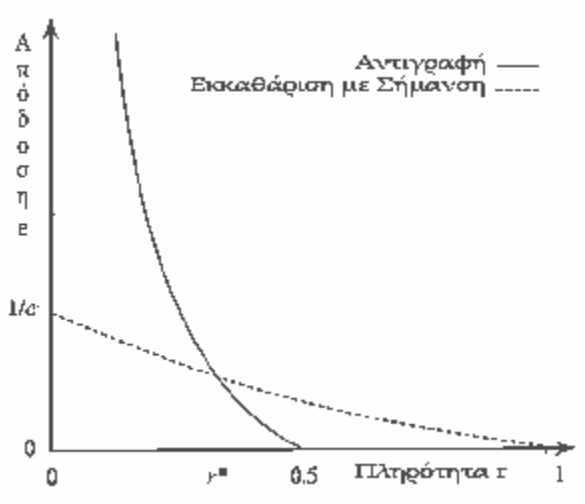
$$e_{Copy} = \frac{1}{2ar} - \frac{1}{a}$$

για τον συλλέκτη αντιγραφής και:

$$e_{MS} = \frac{1-r}{br + c}$$

για τον συλλέκτη εκκαθάρισης με σήμανση, όπου $r = \frac{R}{M}$ είναι η πληρότητα του σωρού του προγράμματος.

Παρατηρώντας το διάγραμμα απόδοσης που ακολουθεί βλέπουμε ότι γενικά ο συλλέκτης αντιγραφής είναι πιο αποδοτικός από την εκκαθάριση με σήμανση, με την προϋπόθεση ότι ο σωρός είναι αρκετά μεγάλος. Όμως, κάτω από μία ορισμένη πληρότητα r*, η δεύτερη είναι αποδοτικότερη:



Σίγουρα, πάντως, το θέμα αυτό είναι βαθύτερο. Οι συλλέκτες που περιγράφονται στην παρούσα έρευνα είναι πολύ απλοϊκοί και αναποτελεσματικοί. Η συμπεριφορά τους δεν μπορεί να χαρακτηρίσει και τις πιο καλογραμμένες εκδοχές τους. Το κόστος της αντιγραφής ενός ολόκληρου αντικειμένου πιθανότατα θα είναι μεγαλύτερο από έναν απλό έλεγχο και μεταβολή του bit σήμανσης, κυρίως αν το αντικείμενο είναι ογκώδες. Παρόλο που η εκκαθάριση με σήμανση πρέπει να σαρώσει ολόκληρο το σωρό, στην πράξη το πραγματικό κόστος είναι κατά κύριο λόγο η φάση σήμανσης, επειδή η γραμμική σάρωση του σωρού θα είναι λιγότερο δαπανηρή από την αναζήτηση δομών δεδομένων ακόμα και αν χρησιμοποιηθούν οι απλές τεχνικές που παρουσιάσαμε. Επομένως, στους παραπάνω τύπους ισχύει: $a > b > c$. Επιπλέον, οι βελτιωμένες μέθοδοι μπορούν να μειώσουν δραστικά το κόστος της εκκαθάρισης. Παρόλο που στο παρελθόν κυριάρχησαν οι συλλέκτες αντιγραφής, πρόσφατες μελέτες αποδεικνύουν ότι η τελική επιλογή μεταξύ αντιγραφής και εκκαθάρισης με σήμανση θα πρέπει να εξαρτάται τόσο από την συμπεριφορά του προγράμματος του χρήστη, όσο και από εγγενείς ιδιότητες των αλγορίθμων συλλογής σκουπιδιών.

6. Συγκριτική Μελέτη

Συνεχίζουμε αυτή τη σύντομη έρευνα με μία ανακεφαλαίωση όλων εκείνων των παραγόντων που θα πρέπει να συνυπολογίσουμε όταν καλούμαστε να αποφασίσουμε ποια τεχνική συλλογής σκουπιδιών είναι καταλληλότερη για να αντιμετωπίσει τις ανάγκες μας. Ο αναγνώστης, όμως, θα πρέπει να διαβάσει τα παρακάτω με κάποια επιφύλαξη. Ενώ θα ήταν πολύ καλοδεχούμενο να μπορούσαμε να κάνουμε συγκρίσεις σε έναν τομέα, θεωρώντας όλα τα υπόλοιπα «σταθερά», στην πράξη σπάνια κάτι τέτοιο είναι εφικτό. Στόχος αυτής της ενότητας είναι, λοιπόν, να δώσει έμφαση στην ανάλυση της συμπεριφοράς των συλλεκτών σε διαφορετικά περιβάλλοντα και όχι να αποτελέσει έναν τσελεμεντέ, ο οποίος θα ήταν απαραίτητα εξαπλουστευμένος. Αρχικά, εξετάζουμε πόσο καλά ο κάθε συλλέκτης ικανοποιεί τις απαιτήσεις που το κάθε πρόγραμμα μπορεί να θέσει στον διαχειριστή μνήμης του, στη συνέχεια μελετάμε και την απόδοση του καθενός.

Θα δώσουμε κυρίως έμφαση στις τρεις κλασικές μεθόδους που παρουσιάσαμε στην αρχή: την καταμέτρηση αναφορών, την εκκαθάριση με σήμανση και τη συλλογή αντιγραφής. Θα συνοψίσουμε τα σχετικά τους πλεονεκτήματα κάτω από διαφορετικές συνθήκες. Τονίζουμε και πάλι ότι οι υλοποιήσεις που παρουσιάσαμε για τις παραπάνω τεχνικές είναι απλοϊκές και η απόδοση και η συμπεριφορά της καθεμιάς δεν μπορεί να θεωρηθεί ότι έχει αυτόματη εφαρμογή και στις πιο καλογραμμένες υλοποιήσεις που μπορούν να υπάρξουν για τις τεχνικές αυτές.

6.1. Οι απαιτήσεις

Το πρώτο θέμα που θα πρέπει να εξεταστεί είναι η ανοχή του προγράμματος στις διακοπές που μπορεί να προκαλέσει ο διαχειριστής μνήμης του. Τόσο ο συλλέκτης εκκαθάρισης με σήμανση, όσο και ο συλλέκτης αντιγραφής είναι αλγόριθμοι που διακόπτουν τη ροή του προγράμματος όση ώρα ο συλλέκτης αποδεσμεύει τα σκουπίδια από τη μνήμη. Η καθυστέρηση που μπορεί να προκληθεί από την ολοκλήρωση ενός κύκλου συλλογής σε έναν μεγάλο σωρό με αυτούς τους απλούς αλγορίθμους κυμαίνεται από κλάσματα του δευτερολέπτου μέχρι και μερικά δευτερόλεπτα. Αυτή η συμπεριφορά προφανώς δεν είναι αποδεκτή από μια ευρεία μερίδα εφαρμογών, από αλληλεπιδραστικά προγράμματα, μέχρι και εφαρμογές με αυστηρές απαιτήσεις πραγματικού χρόνου. Στον αντίποδα, η καταμέτρηση αναφορών παρεμβάλει τις εντολές τις ανάμεσα στις εντολές του προγράμματος και επιτρέπει την εν γένει ομαλότερη λειτουργία του. Υπάρχει, όμως, μία εξαίρεση: ο απλώς αλγόριθμος στον οποίο αναφερθήκαμε νωρίτερα, στο δεύτερο κεφάλαιο, για την αποδέσμευση των σκουπιδιών, απελευθερώνει τη μνήμη άπληστα, με αναδρομή. Και σε αυτό το σημείο το κυρίως πρόγραμμα θα πρέπει πάλι να διακοπεί μέχρι ολόκληρη η δομή των σκουπιδιών να επιστραφεί στο ελεύθερο σύνολο. Πάντως, με πιο προσεγμένες μεθόδους, αυτό το μειονέκτημα μπορεί να αντιμετωπιστεί.

6.2. Αμεσότητα

Μια συνέπεια της παρεμβολής των λειτουργιών του καταμετρητή αναφορών στις εντολές του προγράμματος είναι ότι κάθε (μη-κυκλικό) σκουπίδι εντοπίζεται από το συλλέκτη τη στιγμή που γίνεται απρόσιτο. Για ορισμένες εφαρμογές και τρόπους προγραμματισμού αυτό είναι είτε επιθυμητό, είτε απαραίτητη προϋπόθεση. Κατ' αρχάς, επιτρέπει την άμεση επαναχρησιμοποίηση του χώρου που αποδεσμεύεται, το οποίο μπορεί να βελτιώσει την απόδοση. Για τις συναρτησιακές γλώσσες συγκεκριμένα, αυτό σημαίνει ότι μπορεί να εφαρμοστεί καταστροφική ανάθεση (destructive assignment), για παράδειγμα να ανανεωθούν οι τιμές ενός πίνακα επί τόπου και όχι με αντιγραφή. Δεύτερον, οι αντικειμενοστραφείς γλώσσες προγραμματισμού συνήθως επιτρέπουν ολοκλήρωση (finalization), με την οποία μπορεί να παρεμβληθεί μία διαδικασία ορισμένη από το χρήστη κατά τον θάνατο ενός αντικειμένου. Ένα τυπικό παράδειγμα χρήσης τέτοιας ολοκλήρωσης είναι να κλείνει ένα αρχείο μετά την τελευταία φορά που χρησιμοποιείται. Αν η ολοκλήρωση χρησιμοποιείται για την αποδέσμευση σπάνιων πόρων που ελέγχει ένα αντικείμενο, τότε θα πρέπει να παρεμβληθεί όσο το δυνατόν συντομότερα, αμέσως όταν το αντικείμενο γίνει σκουπίδι, ώστε να επιτρέψει την ανακύκλωση των πολύτιμων αυτών πόρων. Η καταμέτρηση αναφορών μπορεί να εγγυηθεί ότι η διαδικασία της ολοκλήρωσης θα κληθεί αμέσως όταν οι αναφορές προς το αντικείμενο μηδενιστούν. Οι έμμεσοι συλλέκτες, όπως είναι η εκκαθάριση με σήμανση και η αντιγραφή, δεν μπορούν να προσφέρουν κάτι αντίστοιχο, καθώς ο θάνατος του αντικειμένου δεν μπορεί να γίνει αντιληπτός παρά μόνο όταν γίνει ο επόμενος κύκλος συλλογής σκουπιδιών. Εν τούτοις έχουν υπάρξει ιδιαίτερα επιτυχημένες προσπάθειες συνδυασμού ολοκλήρωσης σε συστήματα που χρησιμοποιούν έμμεση συλλογή σκουπιδιών. Η, μακράν, πλέον συνηθισμένη λειτουργία της ολοκλήρωσης είναι απλώς να επιστραφεί ο χώρος που καταλαμβάνει το αντικείμενο στο διαχειριστή μνήμης και τον ρόλο αυτό μπορεί να τον αναλάβει ο συλλέκτης σκουπιδιών με απόλυτη επιτυχία.

6.3. Κυκλικές δομές δεδομένων

Σε πολλά προγράμματα είναι απαραίτητη η χρησιμοποίηση κυκλικών δομών δεδομένων. Εκτός από τις δομές οι οποίες έχουν εμφανώς αμιγώς κυκλική αναπαράσταση, όπως μία κυκλική προσωρινή μνήμη (buffer), άλλες συνηθισμένες δομές δεδομένων, όπως οι διπλά συνδεδεμένες λίστες και τα «δέντρα» των οποίων τα φύλλα έχουν δείκτες που επιστρέφουν πάλι στη ρίζα, είναι επίσης κυκλικές. Όπως είδαμε στην αναφορά μας στην καταμέτρηση αναφορών, η συγκεκριμένη μέθοδος, στην απλή τουλάχιστον μορφή της, δεν μπορεί να ανακυκλώσει τις κυκλικές δομές. Υπάρχουν παραλλαγές του αλγορίθμου που σε συνδυασμό με περιορισμούς στον τρόπο προγραμματισμού επιτρέπουν την ανάκτηση του χώρου που καταλαμβάνουν κυκλικές δομές από διαχειριστές μνήμης βασισμένους στην καταμέτρηση αναφορών. Οι έμμεσοι συλλέκτες, από την άλλη, μπορούν να διαχειριστούν τους κύκλους σωστά χωρίς την ανάγκη για ιδιαίτερη μέριμνα ή περιορισμούς στον τρόπο γραφής του κώδικα του προγράμματος.

6.4. Η αναζήτηση των ριζών και των δεικτών:

Οι έμμεσες μέθοδοι συλλογής σκουπιδιών έχουν την ανάγκη να βρίσκουν όλες τις ρίζες που χρησιμοποιούνται από το πρόγραμμα, και πιθανόν και όλους τους δείκτες που βρίσκονται στα ενεργά δεδομένα. Οι συλλέκτες που κάνουν μετακινήσεις, όπως οι συλλέκτες αντιγραφής, θα πρέπει να είναι σε θέση να εντοπίσουν όλες τις ρίζες ώστε να καταγράψουν τα ενεργά δεδομένα, και όλους τους ενεργούς δείκτες ώστε να ανανεώσουν την τιμή τους με τη νέα θέση του αντικειμένου στο οποίο αναφέρονται. Το σύνολο των ριζών και των ενεργών δεικτών θα πρέπει να υπολογιστεί με κάθε ακρίβεια. Μια υποτίμηση θα προκαλούσε τη μη ανανέωση ενός δείκτη, μια υπερεκτίμηση θα διακινδύνευε την ανανέωση δεδομένων που δεν είναι δείκτες με μια λανθασμένη τιμή. Ένας συλλέκτης αντιγραφής δεν μπορεί να χρησιμοποιηθεί αν δεν πληρούνται αυτές οι προϋποθέσεις, αν και υπάρχουν συντηρητικοί αλγόριθμοι συλλογής με αντιγραφή, οι οποίοι ανέχονται μια υπερεκτίμηση του συνόλου των ριζών. Αυτή η απαίτηση των μη συντηρητικών συλλεκτών τους αναγκάζει να συνεργαστούν στενά με τον μεταγλωττιστή.

Οι απαιτήσεις αυτές μπορεί να είναι λίγο πιο χαλαρές στην περίπτωση των έμμεσων συλλεκτών χωρίς μετακινήσεις. Μια υπερεκτίμηση δεν αποτελεί κίνδυνο, καθώς τα αντικείμενα δεν μετακινούνται και γι' αυτό οι τιμές των δεικτών δεν χρειάζεται να μεταβληθούν. Τα δεδομένα του σωρού δεν χρειάζεται να αλλάξουν καθόλου αν τα bit σήμανσης βρίσκονται σε έναν πίνακα εκτός του σωρού. Επιπλέον, δεν είναι απαραίτητο να αναζητηθούν όλοι οι ζωντανοί δείκτες. Αρκεί να διασφαλιστεί ότι έχει εντοπιστεί τουλάχιστον ένας δείκτης που να αντιστοιχεί σε κάθε ζωντανό αντικείμενο. Οπότε οι συλλέκτες εκκαθάρισης με σήμανση μπορούν να υλοποιηθούν με μικρότερη ανάγκη για συνεργασία από τον μεταγλωττιστή από ότι χρειάζονται οι συλλέκτες αντιγραφής. Ένας συντηρητικός συλλέκτης εκκαθάρισης με σήμανση μπορεί να υλοποιηθεί με ελάχιστη υποστήριξη από τον μεταγλωττιστή. Κυρίως χρειάζεται να γνωρίζει την έκταση του σωρού κατά τη διάρκεια εκτέλεσης του προγράμματος και τη θέση των ολικών (global) δεδομένων.

Οι καταμετρητές αναφορών έχουν ακόμα μικρότερες απαιτήσεις στο συγκεκριμένο τομέα. Το μόνο που είναι απαραίτητο είναι να εντοπίζονται όλοι οι δείκτες όταν διαγράφεται ένα αντικείμενο ως σκουπίδι. Γι' αυτό το λόγο οι καταμετρητές αναφορών μπορούν να υλοποιηθούν ως βιβλιοθήκη, χωρίς καμία υποστήριξη από τον μεταγλωττιστή και όντως υπάρχουν πολυάριθμα παραδείγματα συλλεκτών που είναι σχεδιασμένοι να συνεργάζονται με διάφορες γλώσσες προγραμματισμού.

6.5. Η απόδοση

Εκτός από τη συμμόρφωση με τις ανάγκες του προγράμματος και του περιβάλλοντος, η απόδοση είναι ένας ακόμα σημαντικός παράγοντας για την επιλογή μεταξύ των διαφόρων αλγορίθμων συλλογής σκουπιδιών. Η απόδοση μπορεί να υπολογιστεί βάση της χρονικής επιβάρυνσης στις λειτουργίες του προγράμματος, το κόστος σε χρόνο τόσο της εκχώρησης όσο και της αποδέσμευσης μνήμης, και η επιβάρυνση σε χώρο που οφείλεται είτε άμεσα στον συλλέκτη, είτε στις επιπλέον πληροφορίες που προστίθενται στα δεδομένα του χρήστη.

6.6. Κόστος επεξεργασίας

Η καταμέτρηση αναφορών είναι στενά συνδεδεμένη με το κυρίως πρόγραμμα, γεγονός το οποίο έχει δύο συνέπειες. Πρώτον, η συλλογή σκουπιδιών επιβαρύνει κάθε λειτουργία του κυρίως προγράμματος που σχετίζεται με τη μεταβολή δεικτών. Πολλές παραλλαγές της καταμέτρησης αναφορών μπορούν να χρησιμοποιηθούν ώστε να μειωθεί αυτή η επιβάρυνση, αλλά έτσι διακινδυνεύεται η ακρίβεια στη διαχείριση της μνήμης, την οποία η μέθοδος είναι σχεδιασμένη να υπηρετεί. Δεύτερη συνέπεια της στενής σύνδεσης της καταμέτρησης αναφορών με το κυρίως πρόγραμμα είναι ότι η συντήρηση και η ανάπτυξη του προγράμματος γίνονται πολύ πιο δύσκολα, καθώς θα πρέπει να διατηρείται η ακεραιότητα τον καταμετρητών αναφορών σε οποιαδήποτε αλλαγή χρειαστεί να γίνει στο πρόγραμμα. Οι απλοί, μη-γενεαλογικοί, έμμεσοι συλλέκτες από την άλλη, δεν προκαλούν επεξεργαστική επιβάρυνση στις εντολές του κυρίως προγράμματος. Γι' αυτό οι καλοσχεδιασμένοι έμμεσοι συλλέκτες έχουν χαμηλότερο συνολικό κόστος στο χρόνο εκτέλεσης του προγράμματος σε σχέση με την καταμέτρηση αναφορών.

Τόσο η καταμέτρηση αναφορών όσο και η εκκαθάριση με σήμανση τυπικά χρησιμοποιούν κάποια παραλλαγή της ελεύθερης λίστας για να διαχειριστούν το σύνολο των ελεύθερων αντικειμένων. Συνεπώς ο κατακερματισμός του σωρού γίνεται σημαντικό θέμα για αυτούς τους συλλέκτες. Ο κατακερματισμός όχι μόνο καταστρέφει την τοπικότητα των ενεργών δεδομένων αλλά κάνει πολύ πιο δύσκολη και την ανάθεση νέου χώρου σε αντικείμενα μεταβλητού μεγέθους, επομένως και πιο δαπανηρή σε χρόνο μηχανής. Στον αντίποδα, η συλλογή με αντιγραφή συγκεντρώνει τα ενεργά δεδομένα του σωρού. Η ανάθεση μπορεί να γίνει γραμμικά, κάνοντας το εξίσου εύκολο να διατεθεί χώρος για νέα αντικείμενα οποιουδήποτε μεγέθους.

6.7. Επιβάρυνση σε χώρο

Οι διαχειριστές μνήμης έκτος από χρονική προκαλούν και χωρική επιβάρυνση, και η συλλογή σκουπιδιών δεν αποτελεί εξαίρεση σ΄ αυτό τον κανόνα. Στην περίπτωση των συλλεκτών σκουπιδιών που εξετάζουμε, μπορεί να απαιτείται χώρος για να καθοδηγηθεί η διάσχιση του γράφου ή για να διαχειριστούν τα δεδομένα που βρίσκονται στο σωρό. Και οι τρεις κλασικές μέθοδοι χρησιμοποιούν αναδρομή - η εκκαθάριση με σήμανση και η αντιγραφή για να εντοπίσουν τα ενεργά δεδομένα, η καταγραφή αναφορών για να διαγράψει τα σκουπίδια - η οποία χρειάζεται χώρο για να υλοποιήσει τη στοίβα της. Πάντως, σε όλες τις μεθόδους είναι δυνατόν να αποφευχθεί αυτή η αναδρομή.

Η καταγραφή αναφορών απαιτεί χώρο για να αποθηκεύονται οι μετρητές αναφορών κάθε αντικειμένου. Καθώς, στη χειρότερη περίπτωση, σε ένα αντικείμενο μπορεί να

καταλήγουν δείκτες από κάθε άλλο αντικείμενο του σωρού και από όλες τις ρίζες, το μέγεθος του πεδίου του μετρητή θα πρέπει να ισούται με το μέγεθος ενός δείκτη. Πάντως στην πράξη, ένα μικρότερο πεδίο μπορεί να χρησιμοποιηθεί και υπάρχουν ασφαλείς τεχνικές με τις οποίες μπορεί να ελαττωθεί το μέγεθος του πεδίου του μετρητή χωρίς συνέπειες στην ακρίβεια της συλλογής. Αν τα αντικείμενα είναι πολλά και μικρά, η χωρική επιβάρυνση λόγω των μετρητών μπορεί να φτάσει και στο πενήντα τοις εκατό. Ο συλλέκτης εκκαθάρισης με σήμανση επίσης απαιτεί επιπλέον χώρο για κάθε αντικείμενο του σωρού για ένα bit σήμανσης. Το μέγεθος του χώρου που απαιτείται για το bit σήμανσης εξαρτάται από την αρχιτεκτονική του συστήματος και την υλοποίηση του προγραμματιστή. Μπορεί να καθορίζεται από το μέγεθος του ελάχιστου αντικειμένου που είναι σε θέση να διευθυνσηδοτήσει το σύστημα, αλλά αν ο προγραμματιστής το επιθυμεί μπορεί να το ενσωματώσει σε κάποια άλλη λέξη που χρησιμοποιείται για άλλο σκοπό. Για παράδειγμα, ένα αχρησιμοποίητο bit στη διεύθυνση ενός δείκτη μπορεί να έχει το ρόλο του bit σήμανσης. Οι συλλέκτες αντιγραφής, τέλος, απαιτούν χώρο διευθύνσεων διπλάσιου μεγέθους της μέγιστης πληρότητας του σωρού του προγράμματος, ώστε να μπορούν να στεγάσουν και τα δύο υποσύνολα που χρησιμοποιούν.

6.8. Η πληρότητα του σωρού

Θεωρητικά, οι συλλέκτες αντιγραφής είναι πολύ ευκολότερο να υλοποιηθούν από τους συλλέκτες εκκαθάρισης με σήμανση. Η γραμμική ανάθεση που επιτρέπουν οι πρώτοι είναι ταχύτατη και η πολυπλοκότητα της συλλογής σκουπιδιών είναι ανάλογη του αριθμού των ζωντανών δεικτών στο σωρό, άρα σχεδόν ανάλογη των ζωντανών δεδομένων. Αντίθετα, η πολυπλοκότητα της εκκαθάρισης με σήμανση είναι ανάλογη του μεγέθους του σωρού. Η απόδοση και των δύο έμμεσων συλλεκτών μειώνεται καθώς αυξάνει η πληρότητα του σωρού. Από την άλλη, στην καταμέτρηση αναφορών η πληρότητα του σωρού δεν προκαλεί προβλήματα άμεσα, αν και εμμέσως ο κατακερματισμός που συνεπάγεται δυσχεραίνει την ανάθεση μνήμης για νέα αντικείμενα.

Πάντως, υπάρχουν κάποιες παγίδες όταν προσπαθούμε να συγκρίνουμε την πολυπλοκότητα της εκκαθάρισης με σήμανση και της αντιγραφής. Πρώτον, η αντιγραφή έχει ένα ξεκάθαρο θεωρητικό πλεονέκτημα έναντι των απλών υλοποιήσεων της εκκαθάρισης με σήμανση. Ακόμα κι έτσι όμως, το πλεονέκτημα αυτό εξαλείφεται όταν αυξάνει η πληρότητα του σωρού του κυρίως προγράμματος. Όπως εξηγήσαμε και νωρίτερα, στο τέλος της ενότητας για τους αλγόριθμους αντιγραφής, καθώς αυξάνεται η πληρότητα πέρα από ένα οριακό σημείο (συνήθως περίπου στο ένα τρίτο του σωρού), οι αλγόριθμοι αντιγραφής αρχίζουν να καταρρέουν και το πλεονέκτημα περνάει στους αλγόριθμους εκκαθάρισης με σήμανση. Δεύτερον, οι σταθερές που εμπλέκονται στις συναρτήσεις πολυπλοκότητας είναι πολύ σημαντικές. Το κόστος της αντιγραφής ενός αντικειμένου εξαρτάται από το μέγεθός του, σε κάθε περίπτωση όμως θα είναι αρκετά μεγαλύτερο από το κόστος της μεταβολής ενός bit σήμανσης. Επιπλέον, ένας συλλέκτης αντιγραφής θα μεταφέρει συνεχώς τα μακρόβια αντικείμενα από το ένα υποσύνολο στο άλλο. Η επιλογή λοιπόν μεταξύ συλλογής με αντιγραφή και εκκαθάρισης με σήμανση θα επηρεάζεται από το μέγεθος και τη διάρκεια ζωής των αντικειμένων του σωρού, και από το αν είναι σημαντικότερη η συλλογή σκουπιδιών ή η ανάθεση νέου χώρου κατά τη διάρκεια της διαχείρισης της μνήμης.

7. Βελτιωτικοί Αλγόριθμοι και Τεχνικές

7.1. Συλλογή συγκέντρωσης με σήμανση (Mark-Compact):

Η συλλογή σκουπιδιών εκκαθάρισης με σήμανση μπορεί να γίνει ιδιαίτερα ανταγωνιστική με την συλλογή με αντιγραφή σε ορισμένες περιπτώσεις. Συγκεκριμένα, η εκκαθάριση με σήμανση έχει καλύτερη συμπεριφορά με την χρήση εικονικής μνήμης, αλλά το κύριο μειονέκτημά της είναι η τάση να κατακερματίζει το σωρό αν χρειαστεί να διαχειριστεί μια ποικιλία αντικειμένων διαφορετικού μεγέθους. Μετά από κάθε κύκλο συλλογής ο σωρός είναι πιθανόν να περιέχει αρκετές μικρές «οπές». Για αυτό το λόγο έχει προταθεί ένας παρόμοιος αλγόριθμος, ο οποίος όμως μετακινεί τα δεδομένα, η συλλογή συγκέντρωσης με σήμανση. Ας δούμε όμως πρώτα αναλυτικότερα το πρόβλημα του κατακερματισμού.

7.1.1. Το πρόβλημα του κατακερματισμού:

Ο κατακερματισμός μπορεί να προκαλέσει την εξής περίεργη κατάσταση: μπορεί να γίνει ανέφικτη η τοποθέτηση ενός καινούργιου ογκώδους αντικειμένου χωρίς την επέκταση του σωρού στην περίπτωση που καμιά από τις οπές δεν είναι αρκετά μεγάλη για να το φιλοξενήσει, ακόμα και αν ο συνολικός ελεύθερος χώρος είναι επαρκής. Έχει ζωτική σημασία το δίλημμα που αντιμετωπίζουμε όταν εκχωρούμε μνήμη σε μικρά αντικείμενα. Ποια αρχή θα πρέπει να ακολουθήσουμε; Θα πρέπει να τοποθετήσουμε το αντικείμενο στην πρώτη οπή που θα συναντήσουμε με αρκετό χώρο (first-fit), διακινδυνεύοντας την

επιδείνωση του κατακερματισμού ή θα πρέπει να αφιερώσουμε περισσότερο χρόνο αναζητώντας την οπή με το πλησιέστερο μέγεθος που μπορεί να φιλοξενήσει το αντικείμενο (best-fit); Ο προβληματισμός αυτός δεν αφορά μόνο τους συλλέκτες εκκαθάρισης με σήμανση. Αντιμετωπίζεται και σε κάθε σύστημα που εκχωρεί μνήμη σε αντικείμενα μεταβλητού μεγέθους χωρίς να τα μετακινεί. Οι καταμετρητές αναφορών αλλά και τα συστήματα όπου ο προγραμματιστής διαχειρίζεται χειροκίνητα τη μνήμη μοιράζονται τέτοια προβλήματα.

Αντίθετα, οι συλλέκτες που συγκεντρώνουν τα ενεργά δεδομένα στη βάση του σωρού, μεταξύ των οποίων συμπεριλαμβάνονται και οι συλλέκτες αντιγραφής, είναι ιδιαίτερα αποδοτικοί στην εκχώρηση μνήμης. Η στρατηγική εκχώρησης του σωρού σε αυτά τα συστήματα είναι εξαιρετικά απλή και πειθαρχημένη: Η περιοχή του σωρού που χρησιμοποιείται (δηλαδή τα ενεργά δεδομένα και τα σκουπίδια που δεν έχουν εντοπιστεί ακόμα) αυξάνει συνεχώς μέχρι να συμβεί ένας κύκλος συλλογής σκουπιδιών, οπότε, αν όλα πάνε καλά, θα μειωθεί σημαντικά ο όγκος της. Ως εκ τούτου, απλή είναι και η εκχώρηση νέου χώρου στη μνήμη. Δεδομένου ότι υπάρχει αρκετός χώρος στο σωρό, ένα νέο αντικείμενο μπορεί να εισαχθεί αν απλώς αυξήσουμε κατά το μέγεθος του αντικειμένου την τιμή της αναφοράς που δείχνει στο τέλος του χρησιμοποιούμενου μέρους του σωρού.

Ένας ελκυστικός τρόπος οργάνωσης για τους συλλέκτες που δεν κάνουν μετακινήσεις είναι να διατηρούν ξεχωριστές λίστες ελεύθερων διαφορετικού μεγέθους αντικειμένων. Σε αυτή την περίπτωση το κόστος εκχώρησης μνήμης δεν είναι πολύ μεγαλύτερο από το συλλέκτη αντιγραφής. Πάντως, αν και αυτή η τεχνική διευκολύνει την ανάθεση και αποδέσμευση αντικειμένων δεδομένου μεγέθους, δεν αντιμετωπίζει το πρόβλημα του κατακερματισμού στη ρίζα του. Είναι ακόμα δυνατό μία περιοχή που ελέγχεται από μία λίστα να είναι πλήρης, ενώ μία άλλη να είναι σχετικά άδεια.

7.1.2. Εκχώρηση δύο επιπέδων (two level allocation):

Μια πιο καλοδουλεμένη τεχνική, που βελτιώνει αρκετά την κατάσταση, είναι να γίνεται η εκχώρηση σε δύο επίπεδα. Υπάρχουν δύο εκχωρητές, ένας σε χαμηλό επίπεδο και ένας σε υψηλό. Ο εκχωρητής χαμηλού επιπέδου παίρνει πακέτα μνήμης (τυπικά έχουν μέγεθος 4 Kbytes, αλλά αυτό δεν είναι υποχρεωτικό) από το λειτουργικό σύστημα χρησιμοποιώντας μια απλή συνάρτηση εκχώρησης (όπως τη malloc στη C). Ο εκχωρητής υψηλού επιπέδου διαχειρίζεται μια σειρά από ελεύθερες λίστες, καθεμία απ' τις οποίες αντιστοιχεί σε αντικείμενα διαφορετικού μεγέθους (όπως και στην περίπτωση που περιγράψαμε στην προηγούμενη παράγραφο). Αν μία ελεύθερη λίστα αδειάσει δεν υπάρχει πρόβλημα (όπως προηγουμένως), καθώς μπορεί να πάρει άλλο ένα ελεύθερο πακέτο μνήμης από τον χαμηλό εκχωρητή. Αν και η λίστα του χαμηλού εκχωρητή αδειάσει, τότε γίνεται νέα αίτηση για κομμάτια μνήμης από το λειτουργικό σύστημα. Όσα πακέτα μνήμης βρεθούν τελείως άδεια κατά τη διάρκεια της φάσης εκκαθάρισης μπορούν να επιστραφούν πάλι στον χαμηλό εκχωρητή ώστε να ανακυκλωθούν μεταξύ των διάφορων ελεύθερων λιστών. Με την εκχώρηση σε δύο επίπεδα η ανάθεση μικρών αντικειμένων μπορεί να γίνει αρκετά γρήγορα, αν η αντίστοιχη ελεύθερη λίστα δεν είναι άδεια. Επίσης, ένα ακόμα πλεονέκτημα αυτής της μεθόδου είναι ότι ο σωρός δεν χρειάζεται να είναι ενιαίος.

Η εκχώρηση δύο επιπέδων δεν θεραπεύει πλήρως τον κατακερματισμό. Η εκχώρηση αντικειμένων μεγαλύτερων σε μέγεθος από ένα πακέτο μπορεί να αποτελέσει πρόβλημα, καθώς θα χρειαστεί να βρεθούν αρκετά γειτονικά πακέτα για να φιλοξενήσουν το αντικείμενο. Μία πιθανή λύση σε αυτό το πρόβλημα είναι να διαχειριζόμαστε τα μεγάλα αντικείμενα ξεχωριστά, διαχωρίζοντάς τα σε δύο κομμάτια, μια επικεφαλίδα δεδομένου μεγέθους και το κυρίως σώμα. Η επικεφαλίδα θα ελέγχεται από τον συλλέκτη εκκαθάρισης με σήμανση

χρησιμοποιώντας την ελεύθερη λίστα που αντιστοιχεί στο μέγεθός της, όπως γίνεται και για τα υπόλοιπα δεδομένα στο σωρό. Το κυρίως σώμα θα καταλαμβάνει μια ξεχωριστή θέση στο σωρό, την *περιοχή των μεγάλων αντικειμένων*, στην οποία θα εφαρμόζεται διαφορετική στρατηγική, ίσως με τη χρήση ενός συλλέκτη αντιγραφής.

Και στην εκχώρηση δύο επιπέδων εμφανίζεται κατακερματισμός μέσα στα κομμάτια που διαχειρίζεται μία ελεύθερη λίστα. Αν και σε αυτή την περίπτωση δεν επιβραδύνει την εκχώρηση μνήμης, ο κατακερματισμός αυτός μπορεί να επηρεάσει την χωρική τοπικότητα του κυρίως προγράμματος. Μετά τη συλλογή σκουπιδιών περιοχές κενού χώρου θα βρίσκονται διάσπαρτες μεταξύ των ζωντανών αντικειμένων. Αυτές οι κενές περιοχές στη συνέχεια θα γεμίσουν με νεότερα αντικείμενα, κάνοντας τις σελίδες της εικονικής μνήμης να περιέχουν αντικείμενα διαφορετικών ηλικιών, τα οποία ανήκουν σε διαφορετικά σημεία του προγράμματος. Το τελικό αποτέλεσμα είναι ότι το ενεργό σύνολο των δεδομένων του προγράμματος θα εξαπλωθεί σε περισσότερες σελίδες από όσο χρειάζεται, με αποτέλεσμα να αυξάνεται η διακίνηση σελίδων εικονικής μνήμης. Για αυτό το λόγο οι απλοί συλλέκτες εκκαθάρισης με σήμανση θεωρούνται ακατάλληλοι για περιβάλλοντα που χρησιμοποιούν εικονική μνήμη. Το επιχείρημα αυτό ισχύει και για άλλα είδη συλλεκτών που δεν μετακινούν τα δεδομένα στο σωρό, όπως οι καταμετρητές αναφορών, ή για συστήματα που μετακινούν τα δεδομένα αλλά χωρίς να λαμβάνουν υπόψη το θέμα της τοπικότητας.

Πάντως το θέμα της τοπικότητας ίσως να μην είναι τόσο σοβαρό όσο φαίνεται στην παραπάνω απλή ανάλυση. Τα αντικείμενα που είναι ενεργά συγχρόνως, συνήθως δημιουργούνται την ίδια στιγμή και έχουν παρεμφερή διάρκεια ζωής. Αν υπάρχουν όντως τέτοια σύνολα αντικειμένων που δημιουργούνται και πεθαίνουν ταυτόχρονα, τότε τα αντικείμενα αυτά θα εκχωρούνται κοντά, τόσο χωρικά όσο και χρονικά, και πιθανόν θα ανακαλούνται την ίδια περίπου στιγμή.

7.1.3. Μέθοδοι συλλογής συγκέντρωσης με σήμανση:

Υπάρχουν διάφορες μέθοδοι με τις οποίες μπορεί να γίνει η συγκέντρωση των ζωντανών δεδομένων στο σωρό. Με τον όρο *συγκέντρωση (compaction)* εννοούμε ότι όταν περατωθεί η αντίστοιχη φάση (δηλαδή η φάση της συγκέντρωσης) από τον συλλέκτη, ο σωρός (ή το κομμάτι του όπου εφαρμόστηκε η συγκέντρωση) θα είναι χωρισμένος σε δύο συνεχείς περιοχές. Στη μία περιοχή θα είναι αποθηκευμένα όλα τα ζωντανά δεδομένα ενώ όλος ο ελεύθερος χώρος θα βρίσκεται στην άλλη. Διευκρινίζεται ότι οι τεχνικές αυτές δεν έχουν καμία σχέση με συμπίεση (compression) των δεδομένων στη μνήμη ώστε να μειωθεί ο συνολικός τους όγκος. Στην πράξη μπορεί να είναι επιθυμητός ο συνδυασμός των δύο αυτών τεχνικών: τα δεδομένα να συμπιέζονται κατά τη μεταφορά τους. Τέτοιες τεχνικές όμως έχουν αρχίσει να εγκαταλείπονται, καθώς η ανάγνωση συμπιεσμένων δεδομένων προκαλεί καθυστέρηση η οποία δεν έχει νόημα εξαιτίας της μείωσης του κόστους των μνημών. Πάντως, ορισμένοι συγγραφείς προσφάτως επέστρεψαν στην τεχνική της συμπίεσης θεωρώντας την χρήσιμη στην ελάττωση των απαιτήσεων σε μνήμη και σε αναγνώσεις στον δίσκο (επειδή οι επιδόσεις των επεξεργαστών συνεχίζουν να αυξάνονται πολύ ταχύτερα από την ταχύτητα των δίσκων).

Οι αλγόριθμοι συγκέντρωσης κάνουν αρκετά περάσματα στη δομή των ενεργών δεδομένων ή το σωρό. Ο αριθμός των περασμάτων αυτών ποικίλει εξαρτώμενος από τον αλγόριθμο που χρησιμοποιείται και από το αν είναι εφικτές βελτιώσεις που θα συνδυάσουν τα περάσματα. Γενικά, οι συλλέκτες συγκέντρωσης έχουν τρεις φάσεις, παρόλο που αυτό μπορεί να αλλάζει αν η ανάθεση των κελιών γίνεται πριν ή μετά την ανανέωση των δεικτών:

α) Σήμανση των ενεργών δεδομένων
β) Συγκέντρωση των δεδομένων με μετακίνηση των κελιών τους και
γ) Ανανέωση των τιμών των δεικτών που αναφέρονται σε κελιά που μετακινήθηκαν.

Ιδιαίτερη μέριμνα θα πρέπει να υπάρξει κατά τη μετακίνηση των κελιών. Ιδανικά η διάταξη των κελιών στο σωρό θα καθρεφτίζει τον τρόπο με τον οποίο έχει πρόσβαση σε αυτά το κυρίως πρόγραμμα. Αν τα αντικείμενα δεν τοποθετηθούν σωστά θα μειωθεί η απόδοση της εικονικής μνήμης και οι ευστοχίες στην λανθάνουσα μνήμη. Οι αλγόριθμοι μπορούν να διαχωριστούν σε τρεις κατηγορίες ανάλογα με τον τρόπο που διατάσσουν τα δεδομένα μετά τη συγκέντρωση:

Τυχαίοι: τα κελιά μετακινούνται χωρίς να λαμβάνεται υπόψη η αρχική τους σειρά ή αν αναφέρονται το ένα προς το άλλο. Τέτοιες μέθοδοι συνήθως είναι εύκολες στην υλοποίηση και γρήγορες στην εκτέλεση, ιδίως αν οι κόμβοι έχουν κοινό δεδομένο μέγεθος, αλλά τα αποτελέσματα στη χωρική τοπικότητα είναι τις περισσότερες φορές ανεπαρκή.

Γραμμικοποιητές: τα κελιά που αρχικά αναφέρονταν το ένα προς το άλλο μετακινούνται σε γειτονικές θέσεις, στο μέτρο που αυτό είναι δυνατόν. Οι δομές δεδομένων στη συνέχεια μπορούν να συμπιεστούν αν αυτό κριθεί επιθυμητό.

Μετατόπισης: τα κελιά μετατοπίζονται προς το ένα άκρο του σωρού ενώ τα κενά ανάμεσά τους εκτοπίζονται. Σε αυτούς τους αλγόριθμους διατηρείται η αρχική διάταξη των δεδομένων.

Οι δύο τελευταίες κατηγορίες συγκέντρωσης προσφέρουν αρκετά πλεονεκτήματα. Σε ορισμένα συστήματα είναι ιδιαίτερα σημαντικό η χωρική διάταξη των δεδομένων στο σωρό να αντικατοπτρίζει τον τρόπο ανάθεσής τους. Για παράδειγμα, οι υλοποιήσεις της Prolog μπορεί να αξιοποιήσουν αυτή την ιδιότητα όταν οπισθοδρομούν για να αποδεσμεύσουν απεριόριστη ποσότητα μνήμης σε σταθερό χρόνο, αντιμετωπίζοντας το σωρό σαν στοίβα. Αρκετοί επιχειρηματολογούν ότι μια απλή μετατόπιση τείνει να δώσει την καλύτερη τοπικότητα αναφορών και δεν αξίζει τον κόπο να καταπιανόμαστε με πιο πολύπλοκες μεθόδους. Πολλές μελέτες συγκλίνουν στην άποψη ότι πολλά αντικείμενα δημιουργούνται και πεθαίνουν σε ομάδες. Τα αντικείμενα της ίδιας ομάδας πιθανότατα θα τοποθετούνται σχετικά γειτονικά στη μνήμη κατά την ανάθεσή τους και η συγκέντρωση με μετατόπιση θα τα διατηρήσει κοντά, ή τουλάχιστον δεν θα τα απομακρύνει μεταξύ τους.

Άλλα θέματα που θα πρέπει να σκεφτούμε όταν συγκρίνουμε αλγόριθμους συγκέντρωσης είναι αν ο αλγόριθμος μπορεί να χειριστεί δεδομένα διαφορετικών μεγεθών, πόσα περάσματα θα πρέπει να γίνουν στο σωρό για να μετακινηθούν τα δεδομένα και να ανανεωθούν οι δείκτες, πόσος επιπλέον χώρος χρειάζεται, αν χρειάζεται, από τον αλγόριθμο και αν τοποθετούνται περιορισμοί στη χρήση των δεικτών. Για παράδειγμα, κάποιοι αλγόριθμοι δεν επιτρέπουν τους εσωτερικούς δείκτες, δηλαδή αυτούς που αναφέρονται στο εσωτερικό του αντικειμένου και όχι στο κεφάλι του.

Πολλοί αλγόριθμοι συγκέντρωσης αναφέρονται στη σχετική βιβλιογραφία. Παρακάτω, αναφέρουμε απλώς τέσσερις κατηγορίες ενδεικτικά με μια πολύ σύντομη περιγραφή για την καθεμία:

α) Αλγόριθμοι των δύο δεικτών: Χρησιμοποιούνται δύο δείκτες, ο ένας δείχνει στην επόμενη ελεύθερη θέση και ο άλλος στο επόμενο στοιχείο που πρέπει να μετακινηθεί. Καθώς μετακινείται ένα κελί, η διεύθυνση προορισμού του αποθηκεύεται στην παλιά του θέση για να γίνει στη συνέχεια η ανανέωση των αναφορών που έδειχναν σε αυτό.

Πρόκειται για απλούς αλγορίθμους τυχαίας διάταξης των αντικειμένων στο σωρό. Είναι γρήγοροι και απαιτούν μόνο δύο περάσματα, αλλά δεν βελτιώνουν την τοπικότητα αναφορών

και μάλιστα είναι πολύ πιθανό να τη χειροτερέψουν. Έχουν εφαρμογή μόνο σε δεδομένα σταθερού μεγέθους ή, το πολύ, μιας δεδομένης γκάμας μεγεθών.

β) Αλγόριθμοι με διεύθυνση προορισμού: Σε αυτούς τους αλγόριθμους η διεύθυνση προορισμού αποθηκεύεται σε ένα επιπλέον πεδίο μέσα στο κάθε κελί προτού αυτό μετακινηθεί. Τέτοιες μέθοδοι είναι κατάλληλες για συλλογή κόμβων διαφορετικών μεγεθών.

γ) Μέθοδοι με χρήση πίνακα: Ένας πίνακας μετακινήσεων δημιουργείται στο σωρό είτε πριν, είτα κατά τη διάρκεια την ανάθεσης των κελιών. Ο πίνακας αυτός χρησιμοποιείται αργότερα για να υπολογιστούν οι νέες τιμές των δεικτών.

δ) Μέθοδοι με νήματα: Κάθε κελί συνδέεται με μια λίστα των κελιών που αναφέρονται σε αυτό. Όταν το κελί μετακινείται, διατρέχεται η λίστα για να μεταβληθούν οι τιμές των δεικτών προς αυτό.

Πλεονεκτήματα των συλλεκτών συγκέντρωσης με σήμανση:

Η φάση της συγκέντρωσης είναι σίγουρα δαπανηρή σε σχέση με την απλή εκκαθάριση, αλλά υπάρχουν αρκετοί λόγοι για τους οποίους κάποιος θα προτιμούσε έναν τέτοιο αλγόριθμο σε σχέση με έναν συλλέκτη αντιγραφής ή εκκαθάρισης με σήμανση, καθώς φαίνεται να αντισταθμίζει τα μειονεκτήματα των δύο αυτών μεθόδων:

α) Μειωμένο κόστους εκχώρησης μνήμης: Καταρχάς, το κόστος εκχώρησης μνήμης είναι ελάχιστο, ίδιο με του συλλέκτη αντιγραφής. Αν η ελεύθερη περιοχή του σωρού είναι συνεχής, νέα αντικείμενα οποιουδήποτε μεγέθους μπορούν να δημιουργηθούν απλώς με αντίστοιχη αύξηση ενός δείκτη. Πάντως, όπως είδαμε νωρίτερα όταν μελετούσαμε το πρόβλημα του κατακερματισμού, παρεμφερές πλεονέκτημα μπορούμε να κερδίσουμε και στην εκκαθάριση με σήμανση εφαρμόζοντας εκχώρηση δύο επιπέδων με ξεχωριστές ελεύθερες λίστες για αντικείμενα διαφορετικού μεγέθους.

β) Μικρότερο εύρος διευθύνσεων: Ένας αλγόριθμος αντιγραφής μπορεί να είναι ανεπιθύμητος εξαιτίας της μεγάλης ποσότητας διευθύνσεων που απαιτεί στη μνήμη για ένα πρόγραμμα δεδομένης πληρότητας (αν και το ένα σύνολο είναι πάντα ανενεργό, παραμένει δεσμευμένο για τις ανάγκες του συλλέκτη και δεν μπορεί να χρησιμοποιηθεί). Αν ο συνολικός απαιτούμενος χώρος είναι μεγαλύτερος της φυσικής μνήμης, αυτό θα δυσκολέψει την σελιδοποίηση της, σε αντίθεση με έναν αλγόριθμο εκκαθάρισης ή συγκέντρωσης με σήμανση όπου ολόκληρος ο σωρός θα χωρούσε στην φυσική μνήμη. Το μικρό εύρος διευθύνσεων είναι ιδιαίτερα χρήσιμο στην περίπτωση μικρών μηχανημάτων, όπως για παράδειγμα είναι οι προσωπικοί υπολογιστές, στους οποίους μπορεί να μην εφαρμόζεται και σελιδοποίηση.

γ) Αποφυγή επανάληψης αντιγραφών: Οι απλοί, μη γενεαλογικοί συλλέκτες αντιγραφής μεταφέρουν συνεχώς τα μακρόβια δεδομένα από το ένα σύνολο στο άλλο. Στους αλγόριθμους συγκέντρωσης αυτά τα δεδομένα είναι απίθανο να χρειαστεί να μετακινηθούν ξανά. Για καλύτερα αποτελέσματα, βέβαια, θα πρέπει να χρησιμοποιηθούν γενεαλογικοί αλγόριθμοι για τους οποίους θα μιλήσουμε παρακάτω.

δ) Διαχείριση ακανόνιστης πληρότητας: Ο Sansom πρότεινε μια ενδιαφέρουσα λύση στο δίλημμα του περιορισμένου εύρους διευθύνσεων. Είδαμε στο τέταρτο κεφάλαιο, στην αναφορά μας στους αλγόριθμους αντιγραφής, ότι η απόδοση των συλλεκτών αντιγραφής μειώνεται ραγδαία όταν η πληρότητα του προγράμματος υπερβεί το μισό του συνολικού μεγέθους του σωρού. Όμως, το να αυξήσουμε το μέγεθος του σωρού σε μια τέτοια περίπτωση δεν είναι συνετή λύση. Ο Sansom πρότεινε ότι το οριακό σημείο μεταξύ της χρήσης συλλέκτη αντιγραφής και συγκέντρωσης είναι όταν ο σωρός είναι γεμάτος κατά περίπου τριάντα τοις εκατό (αν και αυτό εξαρτάται από την υλοποίηση). Χρησιμοποίησε λοιπόν έναν

υβριδικό συλλέκτη σκουπιδιών για την υλοποίηση της καθαρά συναρτησιακής γλώσσας Haskell. Ο συλλέκτης του χρησιμοποιούσε την πληρότητα του σωρού ως κριτήριο για την δυναμική εναλλαγή μεταξύ ενός μη γενεαλογικού συλλέκτη αντιγραφής (όσο η πληρότητα ήταν μικρότερη του τριάντα τοις εκατό) και ενός συλλέκτη συγκέντρωσης με σήμανση (αν η πληρότητα αυξανόταν). Αν και ο συλλέκτης συγκέντρωσης που χρησιμοποίησε ήταν πολύ πιο αργός από τον συλλέκτη αντιγραφής, πίστευε ότι θα ήταν χρήσιμος σε προγράμματα που τυπικά η πληρότητα ήταν αρκετά χαμηλότερη από την οριακή τιμή, αλλά παρουσίαζαν περιστασιακές αυξήσεις.

ε) Τοπικότητα: Ο κυριότερος λόγος για την επιλογή χρήσης ενός συλλέκτη συγκέντρωσης με σήμανση είναι για να βελτιωθεί η χωρική τοπικότητα των αντικειμένων που βρίσκονται στο σωρό και έτσι να μειωθεί ο αριθμός των σφαλμάτων σελίδας που θα προκαλέσει το κυρίως πρόγραμμα. Για το σκοπό αυτό, ένας συλλέκτης γραμμικοποίησης ή μετατόπισης είναι απαραίτητος. Μπορεί να μην είναι απαραίτητο να συγκεντρώσουμε τα δεδομένα στο σωρό σε κάθε συλλογή, αλλά να το κάνουμε αυτό περιστασιακά όταν τα κριτήρια που έχουμε θέσει υποδεικνύουν ότι η βελτίωση στη σελιδοποίηση θα αξίζει το κόστος της φάσης συγκέντρωσης.

Ο τρόπος με τον οποίο τα συστήματα διαχείρισης μνήμης διανέμουν τα δεδομένα στο σωρό μπορεί να είναι πολύ σημαντικός για τη συνολική επίδοση του προγράμματος. Όπως είπαμε και νωρίτερα, αρκετές μελέτες έχουν δείξει ότι τα αντικείμενα θα πρέπει να είναι διατεταγμένα στο σωρό με τέτοιο τρόπο ώστε τα δεδομένα που αναφέρονται το ένα στο άλλο, ή συνδέονται με κάποιον άλλο τρόπο, να τοποθετούνται σε κοντινή απόσταση, για να μειώνεται το λειτουργικό σύνολο του προγράμματος. Υπάρχουν αρκετές αποδείξεις ότι η σειρά εκχώρησης είναι καλή ένδειξη για την ύπαρξη σχέσεων μεταξύ αντικειμένων, και ένας συγκεντρωτής μετατόπισης διατηρεί αυτή τη σειρά. Μερικά συστήματα, όπως η Prolog, πρέπει να διατηρούν προσωρινές πληροφορίες ώστε να λειτουργούν αποδοτικά. Η αντιστοίχιση της σειράς στη μνήμη με τη σειρά δημιουργίας είναι ένας πολύ αποτελεσματικός τρόπος να γίνει αυτό.

7.1.4. Επιλογή μεταξύ των αλγορίθμων συγκέντρωσης:

Υπάρχουν αρκετά θέματα προς μελέτη εκτός από την επίδραση της συγκέντρωσης στην διάταξη των δεδομένων στο σωρό. Αρχικά, θα πρέπει να εξετάσουμε αν ο αλγόριθμος επιβάλει κάποια ανεπιθύμητη απαγόρευση στα δεδομένα του χρήστη. Για παράδειγμα, οι περισσότεροι αλγόριθμοι τυχαίας συγκέντρωσης δεν μπορούν να διαχειριστούν δεδομένα διαφόρων μεγεθών, εκτός αν ο σωρός είναι διαχωρισμένος σε περιοχές, κάθε μία από τις οποίες περιέχει αντικείμενα ενός μόνο μεγέθους. Οι αλγόριθμοι με νήματα (για τους οποίους μιλήσαμε επιγραμματικά προηγουμένως) μπορεί είτε να απαιτούν την ξεκάθαρη διάκριση μεταξύ των δεικτών και των υπόλοιπων δεδομένων, είτε να προβάλλουν περιορισμούς στην κατεύθυνση των δεικτών, είτε να απαιτούν τη σάρωση του σωρού για ζωντανά αντικείμενα και στις δύο κατευθύνσεις. Οι αλγόριθμοι επίσης μπορεί να απαγορεύουν τη χρήση εσωτερικών δεικτών.

Η χωρική και χρονική επίδοση του συλλέκτη συγκέντρωσης έχει επίσης σημασία. Ο αλγόριθμος που χρησιμοποιούσε η Lisp 2 απαιτούσε ένα ξεχωριστό πεδίο, μεγέθους δείκτη, να αποθηκεύει σε κάθε αντικείμενο τη διεύθυνση προορισμού του. Άλλοι αλγόριθμοι είτε δεν χρησιμοποιούν επιπλέον χώρο, είτε αξιοποιούν άλλα πεδία του αντικειμένου ή κενά στο σωρό για να καταγράψουν αυτές τις πληροφορίες. Ο χρόνος εκτέλεσης του κάθε συλλέκτη είναι μια πιο ύπουλη ερώτηση. Ο αριθμός των περασμάτων που κάνει ο αλγόριθμος στο σωρό είναι δύο ή τρία, αλλά το πρώτο πέρασμα ενός αλγορίθμου μπορεί να συνδυαστεί με τη διαδικασία σήμανσης. Επίσης, ο επεξεργαστικός φόρτος σε κάθε βήμα είναι σημαντικός. Οι

απλοί αλγόριθμοι τυχαίας συγκέντρωσης κάνουν ελάχιστα σε κάθε βήμα, ενώ άλλες, πιο καλοδουλεμένες μέθοδοι χρειάζεται να καθυστερήσουν περισσότερο. Για παράδειγμα, οι μέθοδοι με νήματα μπορεί για κάθε κελί να ζητήσουν πρόσβαση σε άλλα αντικείμενα του σωρού, διακινδυνεύοντας ακόμα περισσότερες αστοχίες λανθάνουσας μνήμης και σφάλματα σελίδας.

7.2. Γενεαλογικοί αλγόριθμοι συλλογής σκουπιδιών (Generational Garbage Collection):

7.2.1. Το πρόβλημα:

Οι απλοί έμμεσοι συλλέκτες, όπως η εκκαθάριση με σήμανση και η αντιγραφή, υποφέρουν από μια σειρά μειονεκτήματα: Επειδή όλα τα ενεργά δεδομένα πρέπει να σημανθούν ή να αντιγραφούν, οι καθυστερήσεις που προκαλούνται από τη συλλογή σκουπιδιών μπορεί να είναι ενοχλητικές. Έρευνες στις δεκαετίες του 1970 και του 1980 έδειχναν ότι τα ογκώδη προγράμματα σε Lisp τυπικά δαπανούσαν μεταξύ 25 και 40 τοις εκατό του χρόνου εκτέλεσής τους στη συλλογή σκουπιδιών. Για αυτό το λόγο αρκετά συστήματα που προορίζονταν κυρίως για αλληλεπιδραστικές εφαρμογές, χρησιμοποιούσαν αναγκαστικά την καταμέτρηση αναφορών για να κατανείμουν το κόστος της συλλογής σκουπιδιών ισόποσα μέσα στο πρόγραμμα, παρόλη την αυξημένη συνολική επιβάρυνση του επεξεργαστή και την αδυναμία της μεθόδου να συλλέξει κύκλους. Προσθετικοί αλγόριθμοι συλλογής σκουπιδιών έχουν επίσης χρησιμοποιηθεί στην προσπάθεια να διαμοιραστεί το κόστος ανάκλησης μνήμης πιο ομαλά. Πάντως το επιπλέον υπολογιστικό κόστος αυτών των συστημάτων είναι αυξημένο, εκτός αν υπάρχει ειδική υποστήριξη από την εικονική μνήμη ή από εξειδικευμένο υλικό (hardware).

Πολλοί συγγραφείς υποστηρίζουν ότι ο ρόλος του συλλέκτη σκουπιδιών δεν είναι απλώς η ανάκτηση μνήμης, αλλά θα πρέπει να βελτιώνει την συνολική τοπικότητα του συστήματος. Η ελλιπής σχεδίαση του συλλέκτη θα προκαλέσει προβλήματα στην σωστή λειτουργία τόσο της εικονικής μνήμης, όσο και της λανθάνουσας. Η αναζήτηση των ενεργών αντικειμένων απαιτεί την επίσκεψη σε κάθε ζωντανό κελί. Στην περίπτωση της συλλογής με αντιγραφή, θα πρέπει να επισκεφτούμε κάθε σελίδα του σωρού κάθε δύο κύκλους συλλογής, αν κα μόνο το μισό του σωρού χρησιμοποιείται από το κυρίως πρόγραμμα σε κάθε δεδομένη στιγμή. Μια τόσο ελλιπής τοπικότητα αναφορών οδηγεί σε σωρεία αστοχιών της λανθάνουσας μνήμης και σφάλματα σελίδας της εικονικής, εκτός αν η κύρια μνήμη του συστήματος είναι αρκετά μεγάλη για να χωρέσει και τα δύο σύνολα, αν και αυτό βελτιώνεται κάπως από την ισχυρά διαδοχική συμπεριφορά της συλλογής με αντιγραφή. Επίσης, στον συλλέκτη εκκαθάρισης με σήμανση μπορούν να αποφευχθούν κάποια σφάλματα σελίδας αν χρησιμοποιηθεί ένας καλύτερος αλγόριθμος στη σήμανση, για παράδειγμα με τη χρήση ενός πίνακα για την αποθήκευση των bit σήμανσης, ώστε να μη χρειάζεται να επισκεφτούμε ένα αντικείμενο για να δούμε αν είναι ήδη σημαδεμένο.

Οι έμμεσοι αλγόριθμοι επίσης σπαταλούν αρκετό χρόνο για να ασχοληθούν ανεπιτυχώς με σχετικά μακρόβια αντικείμενα (ανεπιτυχώς με την έννοια ότι σκοπός της συλλογής σκουπιδιών είναι η αποδέσμευση μνήμης). Οι συλλέκτες είτε συνεχώς θα επισκέπτονται και θα σημαδεύουν αυτά τα αντικείμενα, είτε θα τα μετακινούν από το ένα υποσύνολο στο άλλο. Ο χρόνος που δαπανάται από τους συλλέκτες αντιγραφής για τον εντοπισμό και την μετακίνηση μακρόβιων αντικειμένων μπορεί να μειωθεί αν διαχωρίσουμε το σωρό σε τρία τμήματα: το ψευδο-στατικό, το μόνο για ανάγνωση και το δυναμικό. Τα αντικείμενα του σωρού που βρίσκονται στο στατικό μέρος θα πρέπει να ελέγχονται για τους δείκτες τους,

αλλά δεν χρειάζεται να μετακινούνται. Τα αντικείμενα που προορίζονται μόνο για ανάγνωση είναι εγγυημένο ότι περιέχουν δείκτες μόνο προς τη στατική και την για ανάγνωση περιοχή, άρα δε χρειάζονται ούτε καν έλεγχο. Δυστυχώς όμως, η διάρκεια ζωής των αντικειμένων δεν μπορεί, εν γένει, να εκτιμηθεί στατικά, ώστε να μετακινηθούν τα αντικείμενα στην κατάλληλη περιοχή.

Στον αντίποδα, η διάρκεια ζωής πολλών αντικειμένων είναι μικρή. Από το 1976 ένας ερευνητής παρατηρούσε ότι, βάσει στατιστικών, τα αντικείμενα που δημιουργήθηκαν πρόσφατα είτε θα παραμείνουν ενεργά για πολύ, είτε θα γίνουν πολύ σύντομα σκουπίδια. Το 98 τοις εκατό του χώρου που αποδεσμεύεται σε έναν κύκλο συλλογής, έχει εκχωρηθεί μετά τον προηγούμενο κύκλο. Πολύ άλλοι ερευνητές έχουν συγκεντρώσει αρκετά στοιχεία ώστε να στηρίξουν την *ασθενή γενεαλογική υπόθεση* ότι «τα περισσότερα αντικείμενα πεθαίνουν νέα». Η ιδέα πίσω από την γενεαλογική συλλογή σκουπιδιών είναι ότι η ανάκτηση χώρου μπορεί να γίνει πιο αποτελεσματικά και με λιγότερη χρονοτριβή αν επικεντρώσουμε την προσπάθειά μας στην ανακύκλωση αυτών των αντικειμένων που είναι πιο πιθανό να γίνουν σκουπίδια, δηλαδή στα νέα αντικείμενα.

Ένας αριθμός πλεονεκτημάτων παρουσιάζονται αν αυτό μπορεί να γίνει αποδοτικά. Συλλέγοντας μόνο ένα μέρος του σωρού η διάρκεια των παύσεων θα εκμηδενιστεί. Αν αυτές οι καθυστερήσεις μειωθούν σημαντικά, τότε η συλλογή σκουπιδιών γίνεται ρεαλιστική λύση και για αλληλεπιδραστικά συστήματα. Ένα συνηθισμένο κριτήριο για αυτές τις περιπτώσεις είναι «Μπορώ να κάνω συλλογή σκουπιδιών ενώ ο χρήστης χρησιμοποιεί ομαλά το ποντίκι;». Επιπλέον, αποφεύγουμε την επαναλαμβανόμενη επεξεργασία των μακρόβιων αντικειμένων, μειώνοντας το συνολικό κόστος της συλλογής σκουπιδιών στο πρόγραμμα. Η τοπικότητα του συλλέκτη μπορεί επίσης να βελτιωθεί, αφού συγκεντρωνόμαστε σε ένα μικρό μέρος του σωρού. Πάντως, υπάρχει ένα τίμημα. Το σύστημα θα πρέπει να διαχωρίζει τα νεότερα από τα παλαιότερα αντικείμενα. Συγκεκριμένα, το κόστος αποθήκευσης ενός δείκτη από ένα παλαιό σε ένα νέο αντικείμενο αυξάνεται αρκετά.

Η γενεαλογική στρατηγική είναι να διασπάσουμε τα αντικείμενα σε δύο ή περισσότερες περιοχές του σωρού που ονομάζονται γενιές. Οι διάφορες γενιές θα μπορούν μετά να συλλέγονται με διαφορετική συχνότητα, με την νεότερη γενιά να ελέγχεται συχνότερα και τις παλαιότερες πολύ πιο σπάνια, ή ακόμα, στην περίπτωση της παλαιότερης γενιάς, να μην ελέγχεται καθόλου. Κατά μία έννοια, οι γενιές αποτελούν τον δυναμικό (κατά τη διάρκεια της εκτέλεσης) διαχωρισμό του σωρού στις τρεις περιοχές που αναφέραμε παραπάνω: την μόνο για ανάγνωση, τη στατική και τη δυναμική. Ο αριθμός των γενεών διαφέρει από υλοποίηση σε υλοποίηση. Σε κάποιες περιπτώσεις χρησιμοποιούνται μόνο δύο, άλλοι φτάνουν μέχρι και επτά! Δεν λείπουν και οι υλοποιήσεις στις οποίες ο αριθμός καθορίζεται δυναμικά, ανάλογα με τις ανάγκες του κυρίως προγράμματος. Η γενεαλογική συλλογή σκουπιδιών συχνά χρησιμοποιείται σε συνδυασμό με προσθετικές μεθόδους, αν και υπάρχουν αρκετές διαφορές μεταξύ τους, και η γενεαλογική συλλογή δεν εξαρτάται από αυτές. Πράγματι, η γενεαλογική συλλογή μπορεί να χρησιμοποιηθεί επαρκώς, αν η μερική συλλογή των νεότερων γενεών μπορεί να είναι αρκετά σύντομη και η ολική συλλογή όλων των γενεών μπορεί να γίνει διακριτικά προς τον χρήστη.

Οι γενεαλογικές τεχνικές θεωρούνται πολύ επιτυχείς και τυγχάνουν ευρείας διάδοσης. Για πολλές σύγχρονες εφαρμογές (αλλά όχι όλες βέβαια), η γενεαλογική συλλογή είναι η πρώτη επιλογή, αλλά το αν αυτή η στρατηγική είναι αποτελεσματική εξαρτάται από την ίδια την εφαρμογή. Τα ερωτήματα που θα πρέπει να σκεφτούμε είναι: Τα περισσότερα αντικείμενα πεθαίνουν νέα; Αν τα νέα αντικείμενα δεν έχουν υψηλή θνησιμότητα, τότε η συλλογή σκουπιδιών δε θα ανακτά αρκετό χώρο. Πόσο συχνά θα χρησιμοποιούνται δείκτες

μεταξύ αντικειμένων διαφορετικών γενεών; Ποια θα είναι η επιβάρυνση για αυτούς τους δείκτες;

7.2.2. Διάρκεια ζωής των αντικειμένων:

Για να μπορέσουμε να μετρήσουμε την ηλικία ενός αντικειμένου, είναι απαραίτητο να διαλέξουμε μια μονάδα μέτρησης για τον χρόνο. Το πιο προφανές είναι να χρησιμοποιήσουμε μονάδες πραγματικού χρόνου. Η κατανομή της διάρκειας ζωής των αντικειμένων σε μονάδες πραγματικού χρόνου μας δίνει μια εικόνα των δημογραφικών στοιχείων της συγκεκριμένης υλοποίησης, αλλά εξαρτάται από το μηχάνημα στο οποίο γίνεται η μέτρηση. Συγκεκριμένα εξαρτάται και από το μηχάνημα και από την υλοποίηση. Μια καλύτερη μονάδα είναι να μετράμε τον αριθμό των bytes που δεσμεύονται στο σωρό. Εκτός του ότι είναι ανεξάρτητη του μηχανήματος, αυτή η μονάδα αντανακλά καλύτερα τις ανάγκες που δημιουργούνται από το υποσύστημα διαχείρισης μνήμης. Συγκεκριμένα, είναι στενά συνδεδεμένο με τη συχνότητα της συλλογής σκουπιδιών αφού αυτή εξαρτάται από το ποσό του διαθέσιμου σωρού.

Πάντως, η ποσότητα δέσμευσης του σωρού δεν είναι η τέλεια μονάδα. Οι αλγόριθμοι εικονικής μνήμης μπορεί να μετρούν το χρόνο με βάση την προσπέλαση σελίδων. Τα αντικείμενα που υποστηρίζουν ανθρώπινη αλληλεπίδραση έχουν διάρκεια ζωής που εξαρτάται από τη δραστηριότητα του χρήστη. Και τα δύο αυτά θέματα επηρεάζουν τον συλλέκτη και μας οδηγούν στη μέτρηση του πραγματικού χρόνου. Επιπλέον, κάποιες γλώσσες έχουν μεγαλύτερη κατανάλωση σε μνήμη. Οι υλοποιήσεις της Smalltalk και των συναρτησιακών γλωσσών τυπικά τοποθετούν αντικείμενα στο σωρό τα οποία άλλες πιο συμβατικές, προστακτικές γλώσσες αποθηκεύουν στη στοίβα ή στους καταχωρητές του επεξεργαστή. Αυτές οι υλοποιήσεις όχι μόνο εκχωρούν μνήμη ταχύτερα, άλλα επίσης αποδεσμεύουν και δεδομένα με ταχύτερο ρυθμό.

Πολλά σημερινά συστήματα, ιδιαίτερα αυτά που είναι γραμμένα σε σύγχρονες συναρτησιακές, αντικειμενοστραφείς ή λογικές γλώσσες, έχουν παράλογες απαιτήσεις σε μνήμη. Ρυθμοί ανάθεσης ενός megabyte ανά δευτερόλεπτο είναι συνηθισμένοι. Υπάρχουν προγράμματα που αποθηκεύουν μια νέα λέξη για κάθε τριάντα εντολές που εκτελούνται. Τα προγράμματα που είναι γραμμένα σε αντικειμενοστραφείς γλώσσες τείνουν να έχουν πολύ μεγαλύτερο αριθμό δομών δεδομένων που αποθηκεύονται στο σωρό από τους προγόνους τους. Πάντως, και σήμερα τα στοιχεία συγκλίνουν στο ότι η συντριπτική πλειοψηφία των αντικειμένων πεθαίνει πολύ νέα, αν και μια μικρή μερίδα μπορεί να ζήσει για αρκετό καιρό. Σύμφωνα με μια πρόσφατη έρευνα, ο Wilson βρήκε ότι 80 με 98 τοις εκατό των αντικειμένων έχουν πεθάνει προτού εκχωρηθεί άλλο ένα megabyte στο σωρό. Έχουν γίνει και πολλές στατιστικές για συγκεκριμένες γλώσσες, όλα όμως τα στοιχεία υποστηρίζουν την ασθενή γενεαλογική υπόθεση.

Αντίθετα, η *ισχυρή γενεαλογική υπόθεση*, ότι όσο παλαιότερο είναι ένα αντικείμενο τόσο λιγότερο πιθανό είναι να πεθάνει, δεν φαίνεται να ισχύει γενικά. Η διάρκεια ζωής των αντικειμένων, σε συνάρτηση με το χρόνο, δεν μειώνεται ομαλά. Αν και τα περισσότερα αντικείμενα πεθαίνουν νέα, μερικά μπορεί να αντέξουν πολύ περισσότερο. Σίγουρα η συμπεριφορά των αντικειμένων δεν εκφράζεται με εκθετική φθίνουσα συνάρτηση, στην οποία η μείωση είναι διαρκής. Στον αντίποδα, η πιθανότητα ένα αντικείμενο να πεθάνει είναι συχνά αντιστρόφως ανάλογη της ηλικίας του. Μετρήσεις σε γενεαλογικούς συλλέκτες πολλών γενεών δείχνουν μεγάλη μείωση στο ποσοστό ανάκτησης χώρου από γενιά σε γενιά. Η κατανομή έχει ανώμαλη μορφή. Ο Hayes βρήκε ότι περισσότερο από το 80 τοις εκατό των αντικειμένων που αποδεσμεύονται διαδοχικά διαφέρουν σε ηλικία λιγότερο από 1 kilobyte,

και ότι αυτό το ποσοστό αυξάνεται περισσότερο αν χαλαρώσουμε το κριτήριο ελέγχοντας τα «σχεδόν διαδοχικά» αντικείμενα που αποδεσμεύονται.

Λιγότερη συμφωνία υπάρχει για το αν η μακροζωία ενός αντικειμένου σχετίζεται με το μέγεθός του. Παρόλο που κάποιοι ερευνητές βρήκαν ότι τα ογκώδη αντικείμενα παρουσιάζουν την τάση να ζουν περισσότερο, άλλοι δεν έχουν βρει μια τέτοια συσχέτιση. Κάποιοι γενεαλογικοί αλγόριθμοι πάντως, χρησιμοποιούν ως κριτήριο για την προαγωγή των αντικειμένων σε μεγαλύτερη γενιά, εκτός από την ηλικία τους, και το μέγεθος τους.

7.2.3. Ο αλγόριθμος

Οι γενεαλογικές τεχνικές συλλογής σκουπιδιών χωρίζουν το σωρό σε δύο ή περισσότερες γενιές, διαχωρίζοντας τα αντικείμενα κατά ηλικία. Τα αντικείμενα αρχικά τοποθετούνται στη νεότερη γενιά, αλλά *προάγονται* σε μεγαλύτερες γενιές αν ζήσουν αρκετά. Αποδεχόμενοι την ασθενή υπόθεση ότι τα περισσότερα αντικείμενα πεθαίνουν νέα, οι γενεαλογικοί αλγόριθμοι συγκεντρώνουν τις προσπάθειές τους στην ανάκτηση χώρου στη νεότερη γενιά, όπου και αναμένεται να εντοπιστεί ο περισσότερος ανακυκλώσιμος χώρος. Αντί για περιστασιακές αλλά χρονοβόρες παύσεις για συλλογή σε όλο το σωρό, ελέγχεται μόνο η νεότερη γενιά σε πιο συχνά διαστήματα. Καθώς η νεότερη γενιά είναι μικρή, οι παύσεις θα είναι συγκριτικά σύντομες. Επιπλέον, καθώς τα παλαιότερα αντικείμενα προάγονται σε μεγαλύτερες γενιές, δεν χρειάζεται να δαπανηθεί χρόνος μηχανής για την επεξεργασία των μακρόβιων αντικειμένων, αν και χρειάζεται να ελεγχθούν μερικά από τα παλαιότερα αντικείμενα για δείκτες προς τις νεότερες γενιές.

7.2.4. Συμπεράσματα:

Η γενεαλογική συλλογή σκουπιδιών έχει αποδειχθεί ιδιαίτερα επιτυχής σε ένα μεγάλο εύρος εφαρμογών. Μπορεί να μειώσει τη διάρκεια των παύσεων σε ένα επίπεδο που να γίνεται ανταγωνιστική με τις προσθετικές μεθόδους για χρήση σε ορισμένα προγράμματα. Συγκεντρώνοντας τις προσπάθειες για εκχώρηση και συλλογή μνήμης σε ένα μικρότερο κομμάτι του σωρού, η σελιδοποίηση και η αξιοποίηση της λανθάνουσας μνήμης βελτιώνονται και για το κυρίως πρόγραμμα, και για το συλλέκτη. Τέλος, καθυστερώντας τη συλλογή των μακρόβιων αντικειμένων, οι γενεαλογικοί αλγόριθμοι βελτιώνουν και το συνολικό κόστος της συλλογής σκουπιδιών. Οι προγραμματιστές που εκχωρούν μεγάλους αριθμούς βραχύβιων αντικειμένων και που δεν μεταβάλουν συχνά τις τιμές των δεικτών μετά την αρχικοποίησή τους, επωφελούνται τα μέγιστα από αυτή την προσέγγιση. Πάντως, η γενεαλογική συλλογή σκουπιδιών δεν είναι πανάκεια και σε συγκεκριμένες περιπτώσεις μπορεί να μην ικανοποιείται η ασθενής γενεαλογική υπόθεση.

Ο στόχος των μικρών παύσεων δεν επιτυγχάνεται αν το σύνολο των ριζών του προγράμματος είναι πολύ μεγάλο. Αυτό μπορεί να συμβεί με οποιονδήποτε συνδυασμό από τα παρακάτω: το πρόγραμμα είναι πάρα πολύ μεγάλο, υπάρχει ένας ασυνήθιστα μεγάλος αριθμός ολικών (global) μεταβλητών που δείχνουν στο σωρό ή γίνονται πολλές αναδρομικές κλήσεις και δημιουργείται μια πολύ μεγάλη στοίβα.

8. Επίλογος

Η συλλογή σκουπιδιών είναι ένα θέμα που απασχολεί τους προγραμματιστές για σχεδόν πέντε δεκαετίες. Πολλές ιδέες, βελτιώσεις και λύσεις σε προβλήματα που ανέκυψαν έχουν παρουσιαστεί και το θέμα θα πρέπει να θεωρείται ότι έχει ωριμάσει, τουλάχιστον για τις τετριμμένες περιπτώσεις. Οι αισιόδοξοι θεωρούν ότι το πρόβλημα της συλλογής σκουπιδιών έχει λυθεί, οι απαισιόδοξοι πιστεύουν ότι ο πολυσύνθετος χαρακτήρας του το καθιστά μη επιλύσιμο.

Γεγονός είναι ότι όποιος προγραμματιστής ενδιαφέρεται για το ζήτημα πλέον, αρκεί να μελετήσει τη σχετική βιβλιογραφία, να εξετάσει διεξοδικά το πρόβλημα που καλείται να αντιμετωπίσει με το πρόγραμμά του και να κάνει τις επιλογές του, σχετικά με το αν θα χρησιμοποιήσει συλλέκτη σκουπιδιών και ποια τεχνική θα τον εξυπηρετήσει καλύτερα.

Για το μέλλον το ζήτημα παραμένει ανοικτό σε εξειδικευμένες μόνο μορφές του όπως είναι τα κατανεμημένα συστήματα ή οι υπολογιστές με πολλαπλούς επεξεργαστές. Το τελευταίο μάλιστα θέμα δείχνει να έχει πολύ ενδιαφέρον καθώς η τάση της αγοράς δείχνει ότι τα επόμενα χρόνια τα συστήματα με περισσότερους του ενός επεξεργαστές θα κατακλύσουν την αγορά ακόμα και τον προσωπικών υπολογιστών.

Βιβλιογραφία

[1] Νικόλαος Παπασπύρου και Εμμανουήλ Σκορδαλάκης, *Μεταγλωττιστές*, Εκδόσεις Συμμετρία, 2002

[2] *Subroutines Overview, General Programming Concepts*, AIX version 3.2 edition.

[3] David L. Andre. *Paging in Lisp programs*. Master's thesis, University of Maryland, College Park, Maryland, 1986.

[4] Andrew W. Appel and Kai Li. "Virtual Memory Primitives for User Programs". *ACM SIGPLAN Notices*, 26(4):96-107, 1991.

[5] Andrew W. Appel. "Garbage Collection can be Faster than Stack Allocation". *Information Processing Letters*, 25(4):275-279, 1987.

[6] Tomas H. Axford. "Reference Counting of Cyclic Graphs for Functional Programs". *Computer Journal*, 33(5):466-470, 1990.

[7] H. D. Baecker. "Garbage Collection for Virtual Memory Computer Systems". *Communications of the ACM*, 15(11):981-986, November 1972.

[8] E. C. Berkeley and Daniel G. Bobrow, editors. *The Programming Language LISP: Its Operation and Applications*, Information International, Inc., Cambridge, MA, fourth edition, 1974.

[9] Hans-Juergen Boehm. "Hardware and Operating System Support for Conservative Garbage Collection", in L.-F. Cabrera et al. editors, *Proceedings of the International Workshop on Object Orientation in Operating Systems* (IWOOS'91), pp. 61-67. 1991.

[10] Hans-Juergen Boehm. "Space Efficient Conservative Garbage Collection", in *Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and Implementation*, pp.197-206, 1993.

[11] Edsgar W. Dijkstra and C. S. Scholten. "Termination Detection for Diffusing Computations". *Information Processing Letters*, 11, August 1989.

[12] Richard Jones and Rafael Lins, *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*, John Wiley & Sons Ltd, 1996.