Γλώσσες Προγραμματισμού ΙΙ
Άσκηση 8 – Διαχείριση μνήμης

Σταύρος Μπιρμπίλης
ΑΜ: 03112116

1.

Αρχικά, είναι γεγονός ότι με τη μέθοδο reference counting δεν είναι δυνατόν να εντοπιστούν οι μη προσβάσιμες κυκλικές δομές δεδομένων, αφού οι αναφορές τους δεν μηδενίζονται ποτέ. Τώρα αν θέλουμε να επεκτείνουμε τη μέθοδο, ώστε να μην υπάρχουν memory leaks, είτε θα πρέπει να καλούμε παράλληλα, αλλά με μικρότερη συχνότητα, κάποια άλλη μέθοδο συλλογής σκουπιδιών (πχ. mark and sweep), είτε να βασιστούμε σε άλλους, όπως ο σχεδιαστής του λειτουργικού που θα μπορούσε να μην επιτρέπει τις αυτο-αναφορές ή ο προγραμματιστής που θα μπορούσε να χρησιμοποιεί implicitly weak references (πχ. όπως στη Java). Σε κάθε περίπτωση όμως, η μέθοδος RefCount++ δεν μας επωφελεί ιδιαίτερα. Ωστόσο, ο συμφοιτητής μας θα μπορούσε να έχει δίκιο και να αποφεύγει τα memory leaks.

2.

Η παραπάνω στρατηγική έχει ένα μεγάλο λάθος. Το πρόβλημα είναι ότι βασίζεται στο liveness analysis των μεταβλητών και όχι των πραγματικών αναφορών. Δηλαδή τον παρακάτω κώδικα…

```
int *x, *y;
x = malloc(sizeof(int));
y = x;

foo(y);
```

… θα τον μετασχημάτιζε ως εξής…

```
int *x, *y;
x = malloc(sizeof(int));
y = x;
free(x);
foo(y);
```

… όπου το πρόβλημα είναι προφανές.

3.

Αυτός ο μετασχηματισμός είναι λάθος. Το πρόβλημα είναι κατά τη δέσμευση μνήμης μέσα σε συναρτήσεις, οι οποίες αφού επιστρέψουν, γίνεται pop το πλαίσιο τους από τη στοίβα και έτσι θα χάνονται τα αντικείμενα. Στον παρακάτω κώδικα το πρόβλημα φαίνεται καθαρά.

```
char* ref_char(char letter) {
      char *c;
      c = malloc(sizeof(char));
      *c = letter;
      return c;
}
```

```
int main() {

      char *alpha;
      alpha = ref_char(‘a’);

      // it will print trash here
      printf(“letter %c\n”, *alpha);

      return 0;
}
```

4.

→ Για τη δέσμευση μιας νέας εγγραφής στο σωρό το κόστος θα εξαρτάται σημαντικά από το πόσο έχει γίνει fragmented ο σωρός από προγενέστερες συλλογές σκουπιδιών. Αντίθετα στην υλοποίηση στοίβας κόστος δέσμευσης δεν υπάρχει.

→ Για την ενέργεια push το κόστος είναι δύο αναθέσεις δεικτών, του prev και του r, ενω στη στοίβα είναι μία, του stack pointer.

→ Για το pop το κόστος είναι ίδιο, δηλάδη μία ανάθεση δείκτη.

→ Για την αποδεύσμευση δεν υπάρχει κόστος την στιγμή που γίνεται και για τις δύο περιπτώσεις. Το μόνο θέμα είναι ότι όταν ο σωρός φιλοξενεί και τις εγγραφές δραστηριοποίησης, τότε θα χρειάζεται παραπάνω χρόνο εκτέλεσης ο αλγόριθμος mark and sweep.

5.

→ Σε αυτήν την περίπτωση η δέσμευση στον σωρό δεν έχει κόστος όπως και στη στοίβα, αφού ο σωρός είναι συμπαγής και η εγγραφή εισάγεται στο τέλος του.

→ Για το κόστος του push δεν αλλάζει κάτι.

→ Για το κόστος του pop δεν αλλάζει κάτι.

→ Ξανά σε αυτήν την περίπτωση δεν υπάρχει κόστος αποδέσμευσης την στιγμή που γίνεται. Επίσης, έχουμε stop and copy συλλογή σκουπιδιών και οπότε η μέθοδος mark and sweep δεν θα έχει επιπλέον κόστος εκτέλεσης, αφού αυτό εξαρτάται απο το μέγεθος των reachable δεδομένων. Το μόνο που θα μπορούσαμε να πούμε είναι ότι μπορεί ο συλλέκτης σκουπιδιών να καλείται πιο συχνά σε αυτήν την περίπτωση.