

Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Παναγιώτης Φωτόπουλος
sdi1300195@di.uoa.gr

Μάνος Πιτσικάλης
sdi1300143@di.uoa.gr

1 Εισαγωγή - Γενικά

2 Λεπτομέρειες υλοποίησης Part 1

- **Δομές αποθήκευσης**

Για την αποθήκευση του γράφου χρησιμοποιούνται δύο ευρετήρια (index-buffer) ένα για τις εξερχόμενες ακμές και ένα για τις εισερχόμενες ακμές. Στις αναζητήσεις και όπου χρειάζονται λίστες, για την αποφυγή πολλών malloc, χρησιμοποιούνται λίστες υλοποιημένες με πίνακα, οι οποίες και επαναχρησιμοποιούνται όταν χρειάζεται. Δηλαδή γίνεται μια φορά malloc, αν χρειαστεί γίνεται realloc, και με την λήξη της εκτέλεσης γίνεται η αποδέσμευση της μνήμης. Επιπλέον, στις αναζητήσεις και όπου αλλού χρειάζεται δομή για να σημαδεύονται οι κόμβοι (π.χ. στην bidirectional-bfs δομή visited) χρησιμοποιείται πίνακας A ίσος με των αριθμό των στοιχείων με versioning, δηλαδή αν το στοιχείο x έχει επισκεφτεί όταν γίνεται αναζήτηση με version v τότε $A[x]=v$, αν καποίο στοιχείο y δεν έχει επισκεφτεί τότε θα ισχύει $A[y]<v$, έτσι η αρχικοποίηση γίνεται μόνο μία φορά και ο έλεγχος γίνεται σε $O(1)$.

- **Αναζήτηση**

Για την αναζήτηση χρησιμοποιήθηκε η bidirectional Breadth-First Search, όπως περιγράφεται στην εκφώνηση, με τις εξής λεπτομέρειες και βελτιώσεις:

- Στην αρχή της κάθε επανάληψης, θα επιλεχθεί για επέκταση η BFS η οποία έχει την χαμηλότερη τιμή στην εξής ευρετική: $\langle \text{αριθμός "παιδιών" προς επέκταση} \rangle + \langle \text{αριθμός "εγγονιών" προς επέκταση (σε επόμενη επανάληψη)} \rangle$. Με αυτόν τον τρόπο, αποφεύγεται η σπατάλη χρόνου σε καταστάσεις όπου από τη μία κατεύθυνση υπάρχουν πάρα πολλά πιθανά μονοπάτια, ενώ από την άλλη λίγα ή μόνο ένα.
- Η αναζήτηση γίνεται ανά "επίπεδα": Σε κάθε επανάληψη, η τρέχουσα BFS επισκέπτεται όλα τα παιδιά του τρέχοντος βάθους.

- **Unit testing**

Έχει υλοποιηθεί unit testing στις βασικές δομές.

- **Εισαγωγή ακμών**

Κατά την εισαγωγή ακμών, ο έλεγχος ύπαρξης ακμής γίνεται στον κόμβο με τις λιγότερες ακμές στην αντίστοιχη κατεύθυνση (εισερχόμενες-εξερχόμενες) για εξοικονόμηση χρόνου. Δοκιμάστηκε η χρήση hashtable για να γίνεται σε $O(1)$ ο έλεγχος αυτός, αλλά αυτή η προσέγγιση αποδείχθηκε μη αποδοτική καθώς επηρέαζε σημαντικά τη χρήση μνήμης, καθώς και τον συνολικό χρόνο (ενημέρωση του hashtable).

3 Λεπτομέρειες υλοποίησης Part 2

- **StronglyConnectedComponents**

Για την εύρεση των SCCs, χρησιμοποιήθηκε ο αλγόριθμος του Tarjan, αλλά τροποποιημένος ώστε αντί αναδρομικά να τρέχει επαναληπτικά. Αυτή η αλλαγή επέτρεψε την επεξεργασία μεγαλύτερων workloads, στα οποία στην αναδρομική προσέγγιση "έσκαγε" η στοίβα.

- **Grail Index**

Και εδώ τροποποιήθηκε η αρχική αναδρομική υλοποίηση σε επαναληπτική. Επίσης, ο αριθμός των labels που επιλέχθηκε (πειραματικά, μετά από διάφορες δοκιμές) είναι 5.

- **(Weakly) Connected Components Index**

Για την αρχική εύρεση των CC χρησιμοποιήθηκε Depth-First-Search. Η αποθήκευση τους γίνεται σε έναν πίνακα με μέγεθος ίσο με το πλήθος των κόμβων, και όταν γίνει ένωση μεταξύ δύο components χρησιμοποιείται η δομή Updated η οποία υλοποιήθηκε και δοκιμάστηκε με τους εξής τρόπους :

- Η πρώτη υλοποίηση έγινε με την δομή της εκφώνησης η οποία φάνηκε να είναι η χειρότερη σε σχέση με τις άλλες δύο που δοκιμάστηκαν.
- Η δεύτερη υλοποίηση χρησιμοποιούσε ένα πίνακα με δείκτες σε λίστες (τις οποίες παίρνει από μία δομή με επαναχρησιμοποιήσιμες λίστες (list pool)) με τα συνδεδεμένα components του κάθε cc. Ωστόσο αυτή η λύση απαιτούσε rebuild. Σε κάθε rebuild ενημερώνεται το ευρετήριο και "αδειάζονται" οι λίστες του updated. Κι αυτή η υλοποίηση όμως φάνηκε να μην είναι η καλύτερη από αυτές που δοκιμάστηκαν.
- Η τρίτη και αυτή που επιλέξαμε για τον πίνακα updated χρησιμοποιεί disjoint sets με χρήση rank και path compression. Περισσότερες πληροφορίες βρίσκονται εδώ: [Disjoint Sets \(Wikipedia\)](#)

4 Λεπτομέρειες υλοποίησης Part 3

- **JobScheduler**

Ο JobScheduler υλοποιήθηκε όπως ζητείται στην εκφώνηση. Ο τερματισμός των threads γίνεται με αποστολή queries με version 0.

- **Πολυνηματισμός**

Έγιναν αλλαγές στον κώδικα των προηγούμενων parts, ώστε να είναι συμβατός με τον πολυνηματισμό. Σε αυτές περιλαμβάνονται: error handling & printing, οι ουρές και οι δομές για μαρκάρισμα για τις αναζητήσεις (και γενικά όπου ήταν απαραίτητο) μεταφέρθηκαν στα thread αντί να είναι μέλος του γράφου, καθώς και το σύστημα versioning για την σωστή εκτέλεση των ερωτημάτων στους δυναμικούς γράφους.

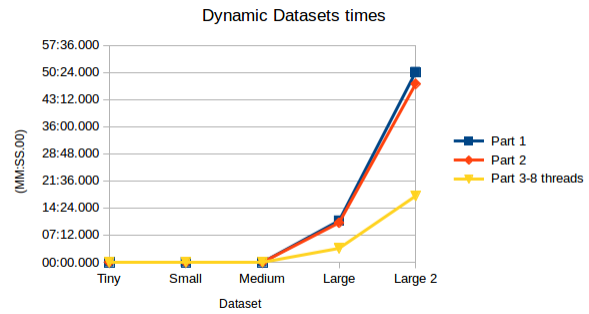
5 Δοκιμές στην υλοποίηση

6 Μετρήσεις

Σύγκριση Part1 Part2 Part3

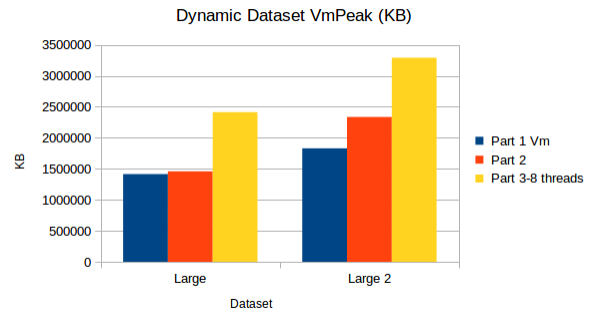
Πίνακας 1: Dynamic Datasets times

| Dynamic Datasets times (MM:SS.000) | | | |
|------------------------------------|-----------|-----------|------------------|
| | Part 1 | Part 2 | Part 3-8 threads |
| Tiny | 00:00.001 | 00:00.002 | 00:00.004 |
| Small | 00:00.453 | 00:00.519 | 00:00.596 |
| Medium | 00:01.539 | 00:01.760 | 00:02.530 |
| Large | 11:01.551 | 10:28.650 | 03:41.225 |
| Large 2 | 50:29.401 | 47:21.940 | 17:35.641 |



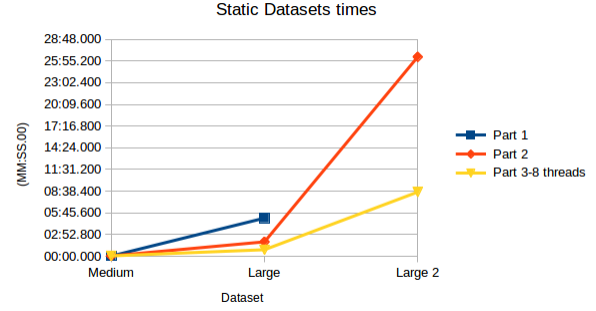
Πίνακας 2: Dynamic Datasets Memory Usage

| Dynamic Dataset VmPeak (KB) | | | |
|-----------------------------|-----------|---------|------------------|
| | Part 1 Vm | Part 2 | Part 3-8 threads |
| Large | 1414896 | 1458116 | 2414616 |
| Large 2 | 1829748 | 2335964 | 3291848 |



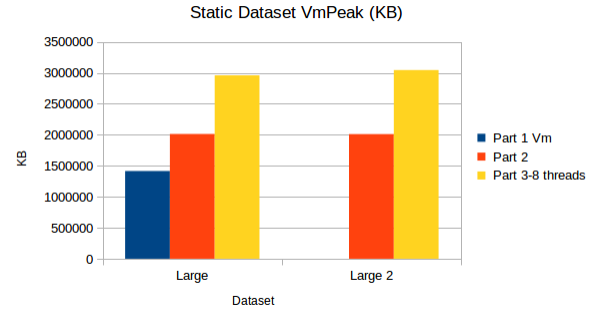
Πίνακας 3: Static Datasets times

| Static Datasets times (MM:SS.000) | | | |
|-----------------------------------|-----------|-----------|------------------|
| | Part 1 | Part 2 | Part 3-8 threads |
| Medium | 00:01.434 | 00:02.538 | 00:02.637 |
| Large | 05:03.604 | 01:54.640 | 00:51.800 |
| Large 2 | >3H | 26:28.035 | 08:31.369 |



Πίνακας 4: Static Datasets Memory Usage

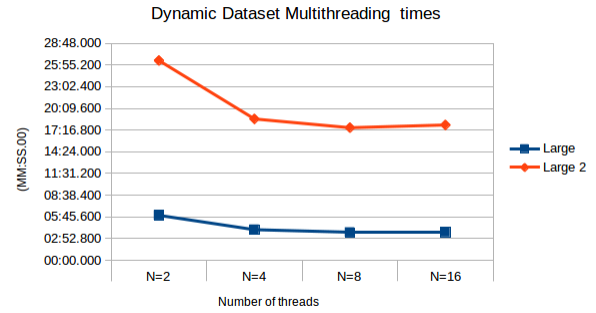
| Static Datasets times (MM:SS.000) | | | |
|-----------------------------------|-----------|-----------|------------------|
| | Part 1 | Part 2 | Part 3-8 threads |
| Medium | 00:01.434 | 00:02.538 | 00:02.637 |
| Large | 05:03.604 | 01:54.640 | 00:51.800 |
| Large 2 | >3H | 26:28.035 | 08:31.369 |



Σύγκριση Part3 αριθμός νημάτων

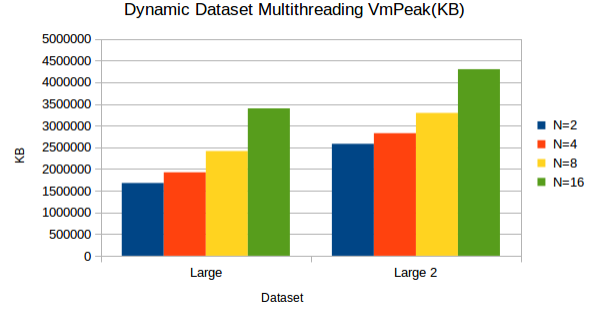
Πίνακας 5: Dynamic Dataset Multithreading times

| Dynamic Dataset Multithreading times (MM:SS.000) | | | | |
|--|-----------|-----------|-----------|-----------|
| Threads | N=2 | N=4 | N=8 | N=16 |
| Large | 05:57.266 | 04:03.300 | 03:41.225 | 03:42.664 |
| Large 2 | 26:30.534 | 18:45.376 | 17:35.641 | 17:57.566 |



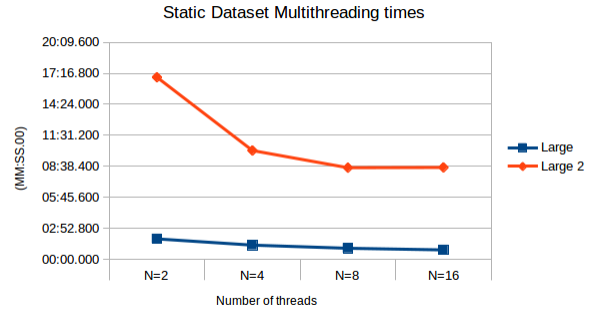
Πίνακας 6: Dynamic Dataset Multithreading Memory Usage

| Dynamic Dataset Multithreading VmPeak(KB) | | | | |
|---|---------|---------|---------|---------|
| Threads | N=2 | N=4 | N=8 | N=16 |
| Large | 1677284 | 1923060 | 2414616 | 3397252 |
| Large 2 | 2579536 | 2825312 | 3291848 | 4299972 |



Πίνακας 7: Static Dataset Multithreading times

| Static Dataset Multithreading times (MM:SS.000) | | | | |
|---|-----------|-----------|-----------|-----------|
| Threads | N=2 | N=4 | N=8 | N=16 |
| Large | 01:53.454 | 01:18.555 | 01:01.018 | 00:51.800 |
| Large 2 | 16:54.585 | 10:05.976 | 08:30.127 | 08:31.690 |



Πίνακας 8: Static Dataset Multithreading Memory Usage

| Static Dataset Multithreading VmPeak(KB) | | | | |
|--|---------|---------|---------|---------|
| Threads | N=2 | N=4 | N=8 | N=16 |
| Large | 2222564 | 2468340 | 2959892 | 3943000 |
| Large 2 | 2307580 | 2553356 | 3044908 | 3950336 |

