3η μέρα

Καλοκαιρινό καμπ για μικρούς 2023

Discord: MavrosOplarxigos#0428

Πρόγραμμα

- STL map
- STL priority_queue

Black Box – Μαύρο Κουτί

- Για τις δομές που θα δούμε σήμερα δεν θα δείξουμε πως αυτές «λειτουργούν στα παρασκήνια». Θα κάνουμε απλή αναφορά στην επιστημονική ονομασία της δομής που χρησιμοποιείται και την ιδέα τους πολύ γενικά.
- Πρώτιστος στόχος είναι να ξέρουμε να εκμεταλλευτούμε τις δομές που μας δίνονται έτοιμες από την STL.
- Άρα θα τις χρησιμοποιούμε ως μαύρα κουτιά που η χρήση τους θα μας δίνει κάποιο επιθυμητό αποτέλεσμα.

STL map

- Πρόβλημα: Ένας γιατρός θέλει να αποθηκεύσει στον Η/Υ τα ύψη των πελατών του.
- *(# Ταυτότητας, Ύψος)*
- Θέλει να χρησιμοποιήσει μια δομή αποθήκευσης η οποία να:
  - Εισάγει γρήγορα την εγγραφή καινούργιου πελάτη
  - Διαγράφει γρήγορα την εγγραφή πελάτη που φεύγει
  - Να βρίσκει γρήγορα αν κάποιος πελάτης είναι γραμμένος ή όχι
  - Να μπορεί να αλλάζει το ύψος κάποιου πελάτη γρήγορα

Ποια δομή θα προτείνουμε που μας δίνεται έτοιμη από την STL της C++?

map

Γιατί;

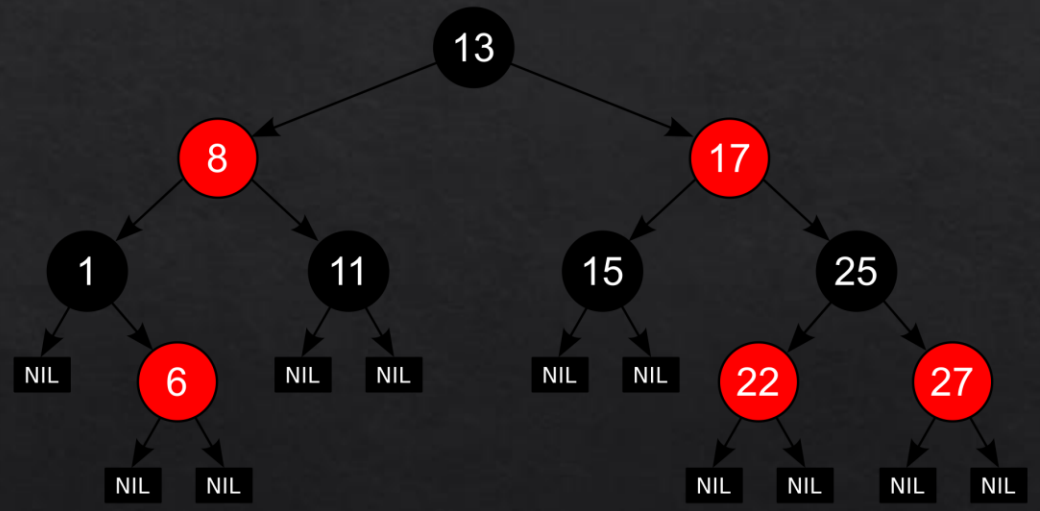
Η συγκεκριμένη δομή έχει λογαριθμική πολυπλοκότητα ως προς το μέγεθος της για όλες τις πράξεις της:

- Εισαγωγή (insert) – O(logN)
- Διαγραφή (erase) – O(logN)
- Αναζήτηση (find) – O(logN)
- Αλλαγή τιμής (value modification) – O(logN)

Διάσχιση (iteration): δηλαδή να περάσω από όλα τα στοιχεία; - O(N)

Πώς;

- Το μαύρο κουτί μας σε αυτήν την περίπτωση είναι τα Red-Black trees.
- Είναι ένα παράδειγμα αυτό-εξισορροπούμενου δυαδικού δέντρου αναζήτησης.



Source: Wikipedia

Κλειδί και Τιμή

- Έχουμε συσχέτιση ενός μοναδικού κλειδιού με μία τιμή. Γι'αυτό και το STL map ονομάζεται συσχετιστική δομή (associative container).
- **Κλειδί:** Είναι μοναδικό, δηλαδή μπορεί να εμφανίζεται μόνο μια φορά μέσα στην δομή μας.
- **Τιμή:** Κάθε κλειδί συσχετίζεται με κάποια τιμή. Πολλά κλειδιά μπορούν να συσχετίζονται με την ίδια τιμή.
- Από το παράδειγμα μας: # Ταυτότητας, Ύψος
- Κλειδί: # Ταυτότητας. Αφού μόνο μια φορά μπορεί να εμφανιστεί (δεν μπορούν >1 άτομα να έχουν τον ίδιο αριθμό ταυτότητας και ένας # ταυτότητας δεν μπορεί να συσχετίζεται με >1 ύψη)
- Τιμή: Ύψος. Πολλά άτομα μπορούν να έχουν το ίδιο ύψος.

Αρχικοποίηση

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5
6         // map < τύπος κλειδίου , τύπος τιμής > όνομα;
7         map < int , double > hm;
8
```

Εισαγωγή (insert)

```
 9          // Εισαγωγή στοιχείου (insertion)
10          // 1ος τρόπος: με χρήση της συνάρτησης insert
11          hm.insert( pair < int , double > (1337,1.70) );
12          hm.insert( make_pair(2020,1.80) );
13
14          // 2ος τρόπος: με χρήση του τελεστή []
15          hm[1001] = 2.01;
```

Διαγραφή (erase)

```
16
17          cout << hm.size() << endl; // 3
18          cout << hm[1001] << endl; // 2.01
19          // Διαγραφή στοιχείου (deletion)
20          hm.erase(1001);
21          hm.erase(15); // Τίποτε δεν θα γίνει
22          cout << hm.size() << endl; // 2
23          cout << hm[1001] << endl; // 0
24
```

Αλλαγή τιμής (modification)

```
25          cout << hm[2020] << endl; // 1.8
26          // Αλλαγή τιμής
27          // 1ος τρόπος: με χρήση του τελεστή []
28          hm[2020] = 1.95;
29          cout << hm[2020] << endl; // 1.95
30
31          // με χρήση της insert? ΟΧΙ!
32          hm.insert( make_pair(2020,2.10) );
33          cout << hm[2020] << endl; // 1.95
34
```

Αναζήτηση (find)

```
35          // Αναζήτηση
36          hm[1234] = 1.55;
37          hm[5678] = 1.56;
38          // 1ος τρόπος: με χρήση της count
39          cout << hm.count(1234) << endl; // 1
40          cout << hm.count(3000) << endl; // 0
41
42          // 2ος τρόπος: με χρήση της find
43          cout << ( hm.find(1234) == hm.end() ) << endl; // 0
44          cout << ( hm.find(3000) == hm.end() ) << endl; // 1
45
46          if( hm.find(1234) != hm.end() ){
47                  cout << "1234 exists!" << endl; // θα τυπωθεί
48          }
49
50          // 3ο τρόπος: με χρήση [] μόνο αν defautl value σημαίνει δεν υπάρχει
51          if( hm[5678] != 0 ){
52                  cout << "5678 exists!" << endl; // θα τυπωθεί
53          }
54
```

Διάσχιση (iteration)

```
54
55          // Διάσχιση (iteration)
56          map<int,double>::iterator it;
57          for(it=hm.begin(); it!=hm.end();it++){
58                  cout << it->first << " " << it->second << endl;
59          }
60
61          /*
62           * 1001 0
63           * 1234 1.55
64           * 1337 1.7
65           * 2020 1.95
66           * 5678 1.56
67           */
68
```

Κώδικας: https://shorturl.at/jlENT

STL priority queue

- Πρόβλημα: Βάζω διάφορα κουτιά με διάφορα βάρη σε ένα φορτηγό και θέλω ανά πάσα στιγμή να ξέρω ποιο είναι το πιο μεγάλο βάρος κουτιού στο φορτηγό μου για να το βγάλω έξω.
- Άρα θέλουμε μία δομή που να:
  - Εισάγει (push) γρήγορα ένα καινούργιο βάρος.
  - Βρίσκει γρήγορα το μεγαλύτερο βάρος (top).
  - Βγάζει γρήγορα (pop) το μεγαλύτερο βάρος από την δομή.

Ποια δομή θα προτείνουμε που μας δίνεται έτοιμη από την STL της C++?

priority_queue

Γιατί;

Η συγκεκριμένη δομή για το μεγαλύτερο (ή μικρότερο που θα δούμε μετά) χρειάζεται πάντα σταθερό χρόνο για να το:

- Αναφέρει (top) να μας πει ποιο είναι – O(1)
- Να το διαγράψει (pop) – O(1)

Επίσης για να μπει (push) ένα καινούργιο αντικείμενο στην δομή χρειαζόμαστε – O(logN)

Αν θα βάλουμε N αντικείμενα στην δομή ποια θα είναι η χρονική πολυπλοκότητα;

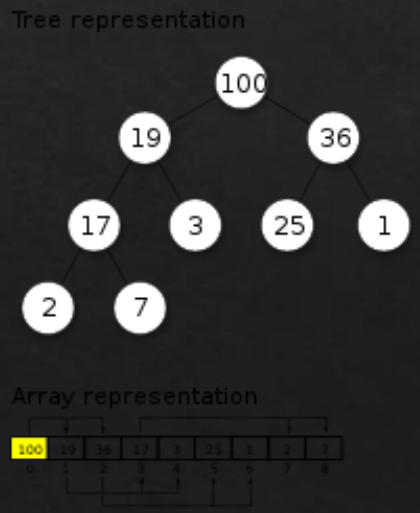
O(logN)

Αν έχουμε N αντικείμενα στην δομή ήδη και θέλουμε να τα βγάλουμε από το μεγαλύτερο στο μικρότερο;

O(N)

Πώς;

- Το μαύρο κουτί μας σε αυτήν την περίπτωση είναι ο Σωρός (heap). Max Heap στην default υλοποίηση της STL.
- Δεντρική δομή.
- Διάφορες υλοποιήσεις με την πιο κοινή ως δυαδικό δένδρο.



Source: Wikipedia

Αρχικοποίηση

Max Heap (το μεγαλύτερο στην κορυφή)

```
6           // MAX HEAP
7           // priority_queue < τύπος αντικειμένου > όνομα;
8           priority_queue < int > q;
```

Min Heap (το μικρότερο στην κορυφή)

```
9
10          // MIN HEAP
11          // priority_queue < τύπος , vector < τύπος > , greater < τύπος >  > όνομα;
12          priority_queue < int , vector < int > , greater < int >  > min_q_example;
13          priority_queue < int > min_q_minus;
14
```

Εισαγωγή (push)

```
15          // Εισαγωή στοιχείων
16          int a[5] = { 1, 2, 4, 2, 8};
17          for(int i=0;i<5;i++){
18                  q.push(a[i]);
19                  min_q_example.push(a[i]);
20                  // χρήση τελεστή - για την min_q_minus
21                  min_q_minus.push(-a[i]); // e.g. -8 < -2
22          }
23
```

Πρόσβαση/Διαγραφή κορυφής (top/pop)

```
24          while(!q.empty()){
25                  cout << q.top() << " ";
26                  q.pop();
27          }
28          cout << endl;
29          // 8 4 2 2 1
30
31          while(!min_q_example.empty()){
32                  cout << min_q_example.top() << " ";
33                  min_q_example.pop();
34          }
35          cout << endl;
36          // 1 2 2 4 8
37
38          while(!min_q_minus.empty()){
39                  cout << min_q_minus.top() << " ";
40                  min_q_minus.pop();
41          }
42          cout << endl;
43          // -1 -2 -2 -4 -8
44
```

Κώδικας: https://shorturl.at/cHZ78

Ανακεφαλαίωση – Επίλυση προβλημάτων

- Προβλήματα για map: (30 minutes solving session - 30 minutes analysis)

https://codeforces.com/contest/855/problem/A

https://codeforces.com/problemset/problem/1703/B

https://www.spoj.com/problems/RPLD/

- Προβλήματα για priority_queue: (30 minutes solving session - 30 minutes analysis)

https://leetcode.com/problems/kth-largest-element-in-an-array/description/

https://www.hackerrank.com/challenges/jesse-and-cookies/problem?isFullScreen=true

https://www.hackerrank.com/challenges/qheap1/problem?isFullScreen=true

Ανακεφαλαίωση – Επίλυση προβλημάτων

- Προβλήματα για maps:
- (30 minutes solving session - 30 minutes analysis)
- - https://codeforces.com/contest/855/problem/A (naive λύση VS λύση με maps: πολυπλοκότητα)
- - https://codeforces.com/problemset/problem/1703/B (naive λύση -> πίνακες, λύση με maps, συζήτηση γιατί προτιμούμε πίνακες αντί maps στην προκειμένη περίπτωση και ποια αλλαγή στην εκφώνηση θα μας οδηγούσε στο να χρησιμοποιήσουμε maps αντί πίνακες).
- - https://www.spoj.com/problems/RPLD/ (λύση με map, λύση με ταξινόμηση και πίνακες;)

- Προβλήματα για priority_queues:
- (30 minutes solving session - 30 minutes analysis)
- - https://leetcode.com/problems/kth-largest-element-in-an-array/description/ (solve this with priority_queues)
- - https://www.hackerrank.com/challenges/jesse-and-cookies/problem?isFullScreen=true (very simple priority queue implementation)
- - https://www.hackerrank.com/challenges/qheap1/problem?isFullScreen=true (upsolving, 2 priority queues: how to delelte the element linear vs logarithmic)