Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 6
Σωροί ελαχίστων (MINHEAP) και σωροί μεγίστων (MAXHEAP), ταξινόμηση heapsort, ουρά προτεραιότητας (priority_queue) της STL

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Εισαγωγή

Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa.

2 Σωροί

Ο σωρός είναι μια μερικά ταξινομημένη δομή δεδομένων. Υπάρχουν δύο ειδών σωροί, ο σωρός μεγίστων (Max-Heap) και ο σωρός ελαχίστων (Min-Heap). Οι ιδιότητες των σωρών που θα περιγραφούν στη συνέχεια αφορούν τους σωρούς μεγίστων αλλά αντίστοιχες ιδιότητες ισχύουν και για τους σωρούς ελαχίστων. Ειδικότερα, ένας σωρός μεγίστων υποστηρίζει ταχύτατα τις ακόλουθες λειτουργίες:

- Εύρεση του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Διαγραφή του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Εισαγωγή νέου κλειδιού στη δομή.

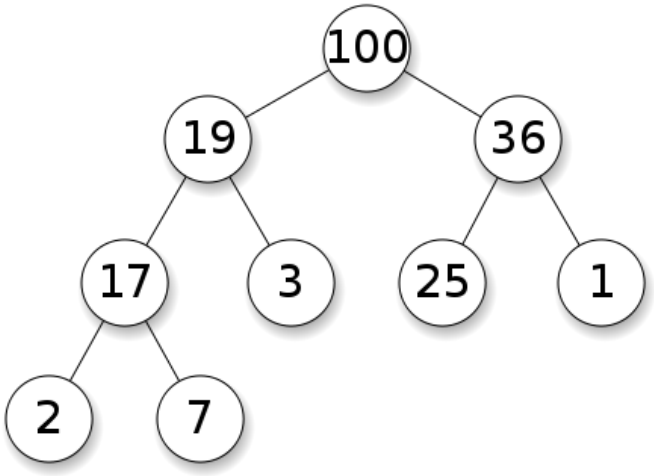
Ένας σωρός μπορεί να θεωρηθεί ως ένα δυαδικό δένδρο για το οποίο ισχύουν οι ακόλουθοι δύο περιορισμοί:

- Πληρότητα: το δυαδικό δένδρο είναι συμπληρωμένο, δηλαδή όλα τα επίπεδά του είναι πλήρως συμπληρωμένα εκτός πιθανά από το τελευταίο επίπεδο στο οποίο μπορούν να λείπουν μόνο κάποια από τα δεξιότερα φύλλα.
- Κυριαρχία γονέα: το κλειδί σε κάθε κορυφή είναι μεγαλύτερο ή ίσο από τα κλειδιά των παιδιών (σε Max-Heap).

Στο σχήμα 1 παρουσιάζεται ένα παράδειγμα σωρού μεγίστων στη δενδρική του μορφή.

Ένας σωρός μπορεί να υλοποιηθεί με ένα πίνακα καταγράφοντας στον πίνακα στη σειρά τα στοιχεία του δυαδικού δένδρου από αριστερά προς τα δεξιά και από πάνω προς τα κάτω (σχήμα 2). Μερικές σημαντικές ιδιότητες οι οποίες προκύπτουν εφόσον τηρηθεί ο παραπάνω τρόπος αντιστοίχισης των στοιχείων του δένδρου στα στοιχεία του πίνακα είναι οι ακόλουθες:

- Στον πίνακα τα κελιά γονείς βρίσκονται στις πρώτες $\lfloor \frac{n}{2} \rfloor$ θέσεις ενώ τα φύλλα καταλαμβάνουν τις υπόλοιπες θέσεις.
- Στον πίνακα τα παιδιά για κάθε κλειδί στις θέσεις i από 1 μέχρι και $\lfloor \frac{n}{2} \rfloor$ βρίσκονται στις θέσεις $2 * i$ και $2 * i + 1$.
- Στον πίνακα ο γονέας για κάθε κλειδί στις θέσεις i από 2 μέχρι και n βρίσκεται στη θέση $\lfloor \frac{i}{2} \rfloor$.



Σχήμα 1: Σωρός μεγίστων στη δενδρική του απεικόνιση

3 Υλοποίηση ενός σωρού

Στη συνέχεια παρουσιάζεται η υλοποίηση ενός σωρού μεγίστων που περιέχει ακέραιες τιμές-κλειδιά.

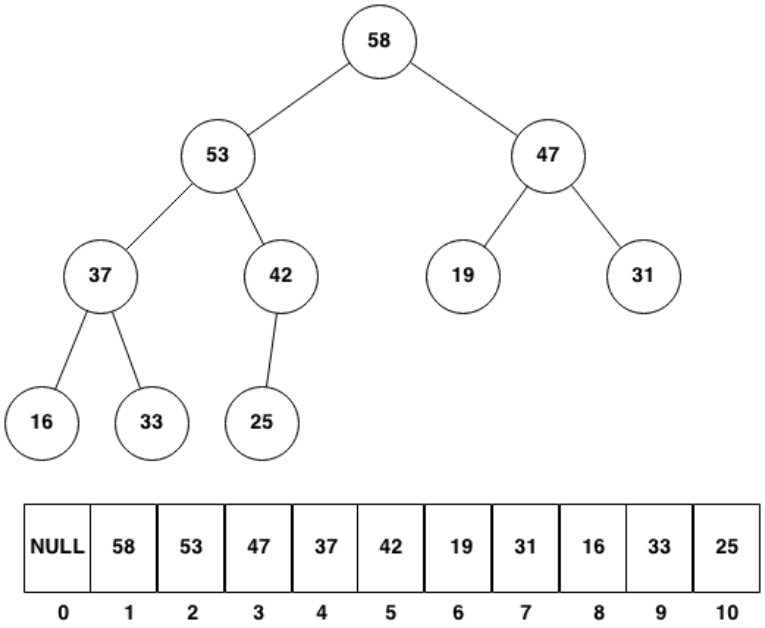
```
#include <iostream>
using namespace std;

// MAXHEAP
const int static HEAP_SIZE_LIMIT = 100000;
int heap[HEAP_SIZE_LIMIT + 1];
int heap_size = 0;

void clear_heap() {
  for (int i = 0; i < HEAP_SIZE_LIMIT + 1; i++)
    heap[i] = 0;
  heap_size = 0;
}

void print_heap() {
  cout << "HEAP(" << heap_size << ") [";
  for (int i = 1; i <= heap_size; i++) {
    if (i == heap_size)
      cout << heap[i];
    else
      cout << heap[i] << " ";
  }
  cout << "]" << endl;
}

void heapify(int k) {
  int v = heap[k];
  bool flag = false;
  while (!flag && 2 * k <= heap_size) {
    int j = 2 * k;
    if (j < heap_size)
      if (heap[j] < heap[j + 1])
        j++;
    if (v >= heap[j])
```



Σχήμα 2: Αναπαράσταση ενός σωρού μεγίστων ως πίνακα

```
35          flag = true;
36        else {
37          heap[k] = heap[j];
38          k = j;
39        }
40      }
41      heap[k] = v;
42    }
43
44    void heap_bottom_up(int *a, int N, bool verbose = false) {
45      heap_size = N;
46      for (int i = 0; i < N; i++)
47        heap[i + 1] = a[i];
48      for (int i = heap_size / 2; i >= 1; i--) {
49        if (verbose)
50          cout << "heapify " << heap[i] << " ";
51        heapify(i);
52        if (verbose)
53          print_heap();
54      }
55    }
56
57    void insert_key(int key) {
58      heap_size++;
59      heap[heap_size] = key;
60      int pos = heap_size;
61      while (pos != 1 && heap[pos / 2] < heap[pos]) {
62        swap(heap[pos / 2], heap[pos]);
63        pos = pos / 2;
64      }
65    }
66
67    void maximum_key_deletion() {
68      swap(heap[1], heap[heap_size]);
69      heap_size--;
```

```
70    heapify(1);
71  }
```

Κώδικας 1: Σωρός μεγίστων ακεραίων (max_heap.cpp)

Ο ακόλουθος κώδικας χρησιμοποιεί τη συνάρτηση heap_bottom_up και μέσω αυτής τη συνάρτηση heapify προκειμένου να μετασχηματίσει έναν πίνακα ακεραίων σε σωρό μεγίστων.

```
1  #include "max_heap.cpp"
2
3  int main(void) {
4    cout << "#### Test heap construction with heapify ####" << endl;
5    int a[10] = {42, 37, 31, 16, 53, 19, 47, 58, 33, 25};
6    heap_bottom_up(a, 10, true);
7    print_heap();
8  }
```

Κώδικας 2: Δημιουργία σωρού από πίνακα με heapify (heap1.cpp)

```
1  #### Test heap construction with heapify ####
2  heapify 53 HEAP(10) [42 37 31 16 53 19 47 58 33 25]
3  heapify 16 HEAP(10) [42 37 31 58 53 19 47 16 33 25]
4  heapify 31 HEAP(10) [42 37 47 58 53 19 31 16 33 25]
5  heapify 37 HEAP(10) [42 58 47 37 53 19 31 16 33 25]
6  heapify 42 HEAP(10) [58 53 47 37 42 19 31 16 33 25]
7  HEAP(10) [58 53 47 37 42 19 31 16 33 25]
```

Ο ακόλουθος κώδικας δημιουργεί σταδιακά έναν σωρό εισάγοντας δέκα τιμές με τη συνάρτηση insert_key. Στη συνέχεια πραγματοποιούνται εξαγωγές τιμών με τη συνάρτηση maximum_key_deletion μέχρι ο σωρός να αδειάσει.

```
1  #include "max_heap.cpp"
2
3  int main(void) {
4    int a[10] = {42, 37, 31, 16, 53, 19, 47, 58, 33, 25};
5    for (int i = 0; i < 10; i++) {
6      cout << "key " << a[i] << " inserted ==> ";
7      insert_key(a[i]);
8      print_heap();
9    }
10   while (heap_size > 0) {
11     cout << "key " << heap[1] << " deleted ==> " << heap[1] << " ";
12     maximum_key_deletion();
13     print_heap();
14   }
15 }
```

Κώδικας 3: Δημιουργία σωρού με εισαγωγές τιμών και εν συνεχεία άδειασμα του σωρού με διαδοχικές διαγραφές της μέγιστης τιμής (heap2.cpp)

```
1  key 42 inserted ==> HEAP(1) [42]
2  key 37 inserted ==> HEAP(2) [42 37]
3  key 31 inserted ==> HEAP(3) [42 37 31]
4  key 16 inserted ==> HEAP(4) [42 37 31 16]
5  key 53 inserted ==> HEAP(5) [53 42 31 16 37]
6  key 19 inserted ==> HEAP(6) [53 42 31 16 37 19]
7  key 47 inserted ==> HEAP(7) [53 42 47 16 37 19 31]
8  key 58 inserted ==> HEAP(8) [58 53 47 42 37 19 31 16]
9  key 33 inserted ==> HEAP(9) [58 53 47 42 37 19 31 16 33]
10 key 25 inserted ==> HEAP(10) [58 53 47 42 37 19 31 16 33 25]
11 key 58 deleted ==> 58 HEAP(9) [53 42 47 33 37 19 31 16 25]
12 key 53 deleted ==> 53 HEAP(8) [47 42 31 33 37 19 25 16]
13 key 47 deleted ==> 47 HEAP(7) [42 37 31 33 16 19 25]
14 key 42 deleted ==> 42 HEAP(6) [37 33 31 25 16 19]
```

```
key 37 deleted ==> 37 HEAP(5) [33 25 31 19 16]
key 33 deleted ==> 33 HEAP(4) [31 25 16 19]
key 31 deleted ==> 31 HEAP(3) [25 19 16]
key 25 deleted ==> 25 HEAP(2) [19 16]
key 19 deleted ==> 19 HEAP(1) [16]
key 16 deleted ==> 16 HEAP(0) []
```

4 Ταξινόμηση Heapsort

Ο αλγόριθμος Heapsort προτάθηκε από τον J.W.J.Williams το 1964 και αποτελείται από 2 στάδια:

- Δημιουργία σωρού με τα n στοιχεία ενός πίνακα τα στοιχεία του οποίου ζητείται να ταξινομηθούν.
- Εφαρμογή της διαγραφής της ρίζας n -1 φορές.

Το αποτέλεσμα είναι ότι τα στοιχεία αφαιρούνται από το σωρό σε φθίνουσα σειρά. Καθώς κατά την αφαίρεσή του κάθε στοιχείου, αυτό τοποθετείται στο τέλος του σωρού, τελικά ο σωρός περιέχει τα αρχικά δεδομένα σε αύξουσα σειρά. Στη συνέχεια παρουσιάζεται η υλοποίηση του αλγορίθμου HeapSort. Επιπλέον ο κώδικας ταξινομεί πίνακες μεγέθους 10.000, 20.000, 40.000 80.000 και 100.000 που περιέχουν τυχαίες ακέραιες τιμές και πραγματοποιείται σύγκριση με τους χρόνους εκτέλεσης που επιτυγχάνει η std::sort.

```
#include "max_heap.cpp"
#include <algorithm>
#include <chrono>
#include <random>

using namespace std::chrono;

void heapsort() {
  while (heap_size > 0)
    maximum_key_deletion();
}

int main(void) {
  high_resolution_clock::time_point t1, t2;
  mt19937 mt(1940);
  uniform_int_distribution<int> uni(0, 200000);
  int problem_sizes[] = {10000, 20000, 40000, 80000, 100000};
  for (int i = 0; i < 5; i++) {
    clear_heap();
    int N = problem_sizes[i];
    int *a = new int[N];
    for (int i = 0; i < N; i++)
      a[i] = uni(mt);
    heap_bottom_up(a, N);
    t1 = high_resolution_clock::now();
    heapsort();
    t2 = high_resolution_clock::now();
    duration<double, std::milli> duration1 = t2 - t1;
    for (int i = 0; i < N; i++)
      a[i] = uni(mt);
    t1 = high_resolution_clock::now();
    sort(a, a + N);
    t2 = high_resolution_clock::now();
    duration<double, std::milli> duration2 = t2 - t1;
    cout << "SIZE " << N << " heap sort " << duration1.count()
         << "ms std::sort " << duration2.count() << "ms" << endl;
    delete[] a;
  }
}
```

Κώδικας 4: Ο αλγόριθμος heapsort (heapsort.cpp)

```
SIZE 10000 heap sort 4.0003ms std::sort 4.0003ms
SIZE 20000 heap sort 5.0003ms std::sort 4.0002ms
SIZE 40000 heap sort 10.0006ms std::sort 10.0006ms
SIZE 80000 heap sort 19.0011ms std::sort 18.001ms
SIZE 100000 heap sort 24.0014ms std::sort 22.0013ms
```

5 Η δομή priority_queue της STL

Η STL της C++ περιέχει υλοποίηση της δομής std::priority_queue (ουρά προτεραιότητας) η οποία είναι ένας σωρός μεγίστων. Κάθε στοιχείο που εισέρχεται στην ουρά προτεραιότητας έχει μια προτεραιότητα που συνδέεται με αυτό και το στοιχείο με τη μεγαλύτερη προτεραιότητα βρίσκεται πάντα στην αρχή της ουράς. Οι κυριότερες λειτουργίες που υποστηρίζονται από την std::priority_queue είναι οι ακόλουθες:

- push: εισαγωγή ενός στοιχείου στη δομή
- top: επιστροφή χωρίς εξαγωγή του στοιχείου με τη μεγαλύτερη προτεραιότητα
- pop: απώθηση του στοιχείου με τη μεγαλύτερη προτεραιότητα
- size: πλήθος των στοιχείων που υπάρχουν στη δομή
- empty: επιστρέφει true αν η δομή είναι άδεια αλλιώς επιστρέφει false

Ένα παράδειγμα χρήσης της std::priority_queue ως σωρού μεγίστων αλλά και ως σωρού ελαχίστων παρουσιάζεται στη συνέχεια.

```
#include <algorithm>
#include <iostream>
#include <queue>

using namespace std;

int main(void) {
   int a[10] = {15, 16, 13, 23, 45, 67, 11, 22, 37, 10};
   cout << "priority queue (MAXHEAP): ";
   priority_queue<int> pq1(a, a + 10);
   while (!pq1.empty()) {
      int x = pq1.top();
      pq1.pop();
      cout << x << " ";
   }
   cout << endl;

   cout << "priority queue (MINHEAP): ";
   priority_queue<int, std::vector<int>, std::greater<int>> pq2(a, a + 10);
   while (!pq2.empty()) {
      int x = pq2.top();
      pq2.pop();
      cout << x << " ";
   }
   cout << endl;
}
```

Κώδικας 5: Παράδειγμα με priority_queue της STL (stl_priority_queue.cpp)

```
priority queue (MAXHEAP): 67 45 37 23 22 16 15 13 11 10
priority queue (MINHEAP): 10 11 13 15 16 22 23 37 45 67
```

6 Παραδείγματα

6.1 Παράδειγμα 1

Διάμεσος ενός δείγματος N παρατηρήσεων οι οποίες έχουν διαταχθεί σε αύξουσα σειρά ορίζεται ως η μεσαία παρατήρηση, όταν το N είναι περιττός αριθμός, ή ο μέσος όρος (ημιάθροισμα) των δύο μεσαίων παρατηρήσεων όταν το N είναι άρτιος αριθμός. Έστω ότι για διάφορες τιμές που παράγονται με κάποιον τρόπο ζητείται ο υπολογισμός της διάμεσης τιμής καθώς παράγεται κάθε νέα τιμή και για όλες τις τιμές που έχουν προηγηθεί μαζί με την τρέχουσα τιμή όπως φαίνεται στο επόμενο παράδειγμα:
5 ⇒ διάμεσος 5
5, 7 ⇒ διάμεσος 6
5, 7, 13 ⇒ διάμεσος 7
5, 7, 13, 12 ⇒ 5, 7, 12, 13 ⇒ διάμεσος 9.5
5, 7, 13, 12, 2 ⇒ 2, 5, 7, 12, 13 ⇒ διάμεσος 7

```cpp
#include <chrono>
#include <iomanip>
#include <iostream>
#include <queue>
#include <random>

using namespace std;
using namespace std::chrono;

double medians(int a[], int N) {
  priority_queue<int, std::vector<int>, std::less<int>> pq1;
  priority_queue<int, std::vector<int>, std::greater<int>> pq2;
  int first = a[0];
  int second = a[1];
  if (first < second) {
    pq1.push(first);
    pq2.push(second);
  } else {
    pq2.push(first);
    pq1.push(second);
  }
  double sum = first + (first + second) / 2.0;
  for (int i = 2; i < N; i++) {
    int x = a[i];
    if (x <= pq1.top())
      pq1.push(x);
    else
      pq2.push(x);
    if (pq1.size() < pq2.size()) {
      pq1.push(pq2.top());
      pq2.pop();
    }
    double median;
    if (pq1.size() == pq2.size())
      median = (pq1.top() + pq2.top()) / 2.0;
    else
      median = pq1.top();
    sum += median;
  }
  return sum;
}

int main(int argc, char **argv) {
  high_resolution_clock::time_point t1, t2;
```

```
45   t1 = high_resolution_clock::now();
46   mt19937 mt(1940);
47   uniform_int_distribution<int> uni(0, 200000);
48   int N = 500000;
49   int *a = new int[N];
50   for (int i = 0; i < N; i++)
51      a[i] = uni(mt);
52   double sum = medians(a, N);
53   delete[] a;
54   t2 = high_resolution_clock::now();
55   duration<double, std::milli> duration = t2 − t1;
56   cout.precision(2);
57   cout << "Moving medians sum = " << std::fixed << sum << " elapsed time "
58         << duration.count() << "ms" << endl;
59 }
```

Κώδικας 6: Υπολογισμός διαμέσου σε μια ροή τιμών (lab06_ex1.cpp)

```
1 Moving medians sum = 54441518145.50 elapsed time 132.52ms
```

6.2 Παράδειγμα 2

7 Ασκήσεις

1. a
2. b
3. Να υλοποιηθεί ο σωρός μεγίστων που παρουσιάστηκε στον κώδικα X ως κλάση.
4. d

Αναφορές

[1] Geeks for Geeks, Priority Queue in C++ Standard Template Library (STL), http://www.geeksforgeeks.org/priority-queue-in-cpp-stl/