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

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

1 Εισαγωγή

Οι σωροί επιτρέπουν την οργάνωση των δεδομένων με τέτοιο τρόπο έτσι ώστε το μεγαλύτερο στοιχείο να είναι συνεχώς προσπελάσιμο σε σταθερό χρόνο. Η δε λειτουργίες της εισαγωγής νέων τιμών στη δομή και της διαγραφή της μεγαλύτερης τιμής πραγματοποιούνται ταχύτατα. Σε αυτό το εργαστήριο θα παρουσιαστεί η υλοποίηση ενός σωρού μεγίστων και ο σχετικός με τη δομή αυτή αλγόριθμος ταξινόμησης, heapsort. Επιπλέον, θα παρουσιαστεί η δομή std::priority_queue που υλοποιεί στην STL της C++ τους σωρούς μεγίστων και ελαχίστων. 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 βασικά είδη σωρών: ο σωρός μεγίστων (MAXHEAP) και ο σωρός ελαχίστων (MINHEAP). 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:

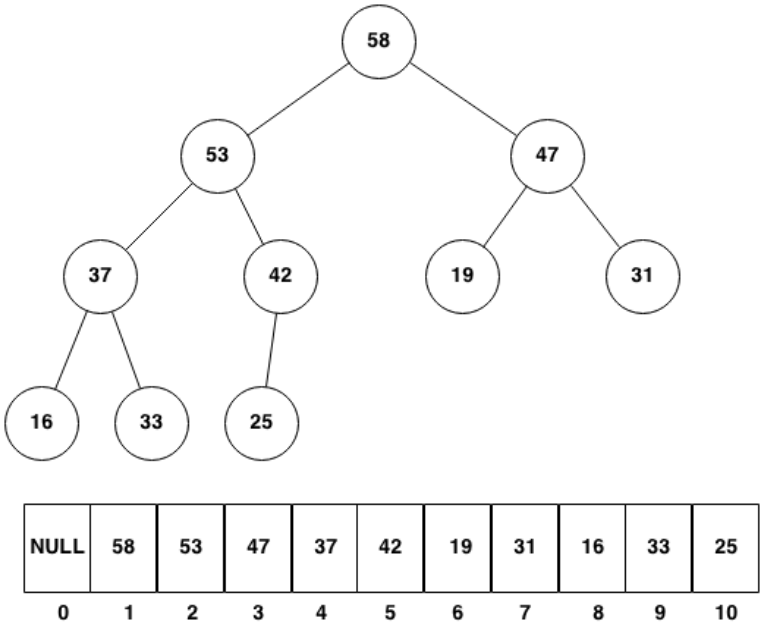
- Εύρεση του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Διαγραφή του στοιχείου με τη μεγαλύτερη τιμή κλειδιού.
- Εισαγωγή νέου κλειδιού στη δομή.

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 (χαμηλότερο) 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 MAXHEAP).

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 1). 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: Αναπαράσταση ενός σωρού μεγίστων ως πίνακα

Για το παράδειγμα του σχήματος ισχύουν τα ακόλουθα:

- Οι κόμβοι που είναι γονείς (έχουν τουλάχιστον ένα παιδί) βρίσκονται στις θέσεις από 1 μέχρι και 5.
- Οι κόμβοι που είναι φύλλα βρίσκονται στις θέσεις από 6 μέχρι και 10.
- Ο γονέας στη θέση 1 (η τιμή 58) έχει παιδιά στις θέσεις $2 * 1 = 2$ (τιμή 53) και $2 * 1 + 1 = 3$ (τιμή 57).
- Ο γονέας στη θέση 2 (η τιμή 53) έχει παιδιά στις θέσεις $2 * 2 = 4$ (τιμή 37) και $2 * 2 + 1 = 5$ (τιμή 42).
- Ο γονέας στη θέση 3 (η τιμή 47) έχει παιδιά στις θέσεις $2 * 3 = 6$ (τιμή 19) και $2 * 3 + 1 = 7$ (τιμή 31).
- Ο γονέας στη θέση 4 (η τιμή 37) έχει παιδιά στις θέσεις $2 * 4 = 8$ (τιμή 16) και $2 * 4 + 1 = 9$ (τιμή 33).
- Ο γονέας στη θέση 5 (η τιμή 42) έχει παιδιά στις θέσεις $2 * 5 = 10$ (τιμή 25).
- Ο κόμβος παιδί στη θέση 2 (η τιμή 53) έχει γονέα στη θέση $\lfloor \frac{2}{2} \rfloor = 1$ (τιμή 58).
- Ο κόμβος παιδί στη θέση 3 (η τιμή 47) έχει γονέα στη θέση $\lfloor \frac{3}{2} \rfloor = 1$ (τιμή 58).
- Ο κόμβος παιδί στη θέση 4 (η τιμή 37) έχει γονέα στη θέση $\lfloor \frac{4}{2} \rfloor = 2$ (τιμή 53).
- Ο κόμβος παιδί στη θέση 5 (η τιμή 42) έχει γονέα στη θέση $\lfloor \frac{5}{2} \rfloor = 2$ (τιμή 53).
- Ο κόμβος παιδί στη θέση 6 (η τιμή 19) έχει γονέα στη θέση $\lfloor \frac{6}{2} \rfloor = 3$ (τιμή 47).
- Ο κόμβος παιδί στη θέση 7 (η τιμή 31) έχει γονέα στη θέση $\lfloor \frac{7}{2} \rfloor = 3$ (τιμή 47).
- Ο κόμβος παιδί στη θέση 8 (η τιμή 16) έχει γονέα στη θέση $\lfloor \frac{8}{2} \rfloor = 4$ (τιμή 37).
- Ο κόμβος παιδί στη θέση 9 (η τιμή 33) έχει γονέα στη θέση $\lfloor \frac{9}{2} \rfloor = 4$ (τιμή 37).
- Ο κόμβος παιδί στη θέση 10 (η τιμή 25) έχει γονέα στη θέση $\lfloor \frac{10}{2} \rfloor = 5$ (τιμή 42).

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;
```

```
8
9  void clear_heap() {
10   for (int i = 0; i < HEAP_SIZE_LIMIT + 1; i++)
11     heap[i] = 0;
12   heap_size = 0;
13 }
14
15 void print_heap(bool newline = true) {
16   cout << "HEAP(" << heap_size << ") [";
17   for (int i = 1; i <= heap_size; i++)
18     if (i == heap_size)
19       cout << heap[i];
20     else
21       cout << heap[i] << " ";
22   cout << "]";
23   if (newline)
24     cout << endl;
25 }
26
27 void heapify(int k) {
28   int v = heap[k];
29   bool flag = false;
30   while (!flag && 2 * k <= heap_size) {
31     int j = 2 * k;
32     if (j < heap_size)
33       if (heap[j] < heap[j + 1])
34         j++;
35     if (v >= heap[j])
36       flag = true;
37     else {
38       heap[k] = heap[j];
39       k = j;
40     }
41   }
42   heap[k] = v;
43 }
44
45 void heap_bottom_up(int *a, int N, bool verbose = false) {
46   heap_size = N;
47   for (int i = 0; i < N; i++)
48     heap[i + 1] = a[i];
49   for (int i = heap_size / 2; i >= 1; i--) {
50     if (verbose)
51       cout << "heapify " << heap[i] << " ";
52     heapify(i);
53     if (verbose)
54       print_heap();
55   }
56 }
57
58 int top() { return heap[1]; }
59
60 void push(int key) {
61   heap_size++;
62   heap[heap_size] = key;
63   int pos = heap_size;
64   while (pos != 1 && heap[pos / 2] < heap[pos]) {
65     swap(heap[pos / 2], heap[pos]);
66     pos = pos / 2;
67   }
68 }
```

```
69
70 void pop() {
71   swap(heap[1], heap[heap_size]);
72   heap_size−−;
73   heapify(1);
74 }
```

Κώδικας 1: Σωρός μεγίστων με κλειδιά ακέραιες τιμές (max_heap.cpp)

Οι συναρτήσεις δημιουργίας σωρού από πίνακα (heap_bottom_up και heapify)

Ένας πίνακας μπορεί να μετασχηματιστεί ταχύτατα σε σωρό. Η διαδικασία ξεκινά από τον τελευταίο κόμβο γονέα του δένδρου (που βρίσκεται στη θέση $\lfloor \frac{n}{2} \rfloor$) και σταδιακά εφαρμόζεται μέχρι να φτάσει στον κόμβο στη θέση 1. Για καθένα από αυτούς τους κόμβους εξετάζεται από πάνω προς τα κάτω αν ισχύει η κυριαρχία γονέα και αν δεν ισχύει τότε γίνεται αντιμετάθεση με το μεγαλύτερο από τα παιδιά του επαναληπτικά.

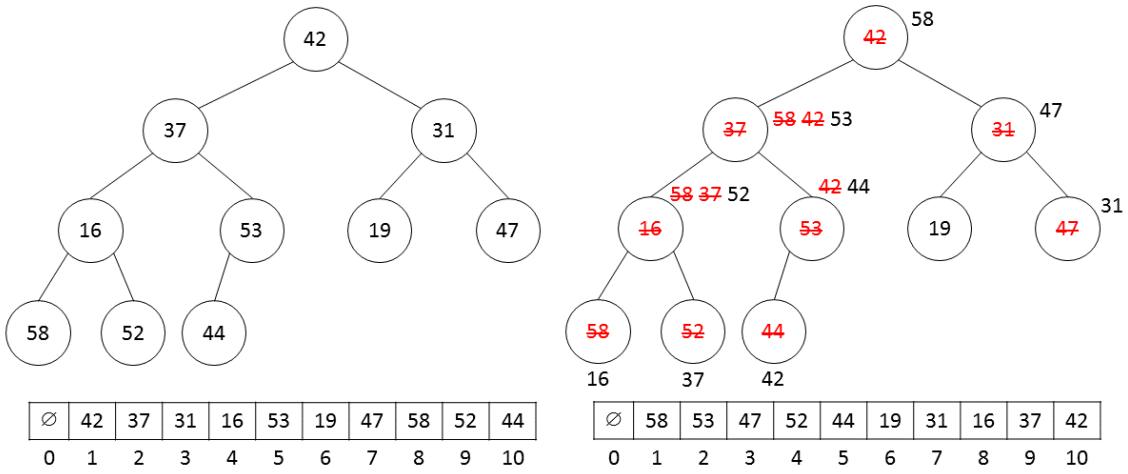
Ο ακόλουθος κώδικας χρησιμοποιεί τη συνάρτηση heap_bottom_up και μέσω αυτής τη συνάρτηση heapify προκειμένου να μετασχηματίσει έναν πίνακα ακεραίων σε σωρό μεγίστων.

```
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, 52, 44};
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 52 44]
3 heapify 16 HEAP(10) [42 37 31 58 53 19 47 16 52 44]
4 heapify 31 HEAP(10) [42 37 47 58 53 19 31 16 52 44]
5 heapify 37 HEAP(10) [42 58 47 52 53 19 31 16 37 44]
6 heapify 42 HEAP(10) [58 53 47 52 44 19 31 16 37 42]
7 HEAP(10) [58 53 47 52 44 19 31 16 37 42]
```

Στο σχήμα 2 παρουσιάζονται οι τιμές που έλαβε κάθε κόμβος του δένδρου προκειμένου να μετασχηματιστεί τελικά σε σωρό μεγίστων.



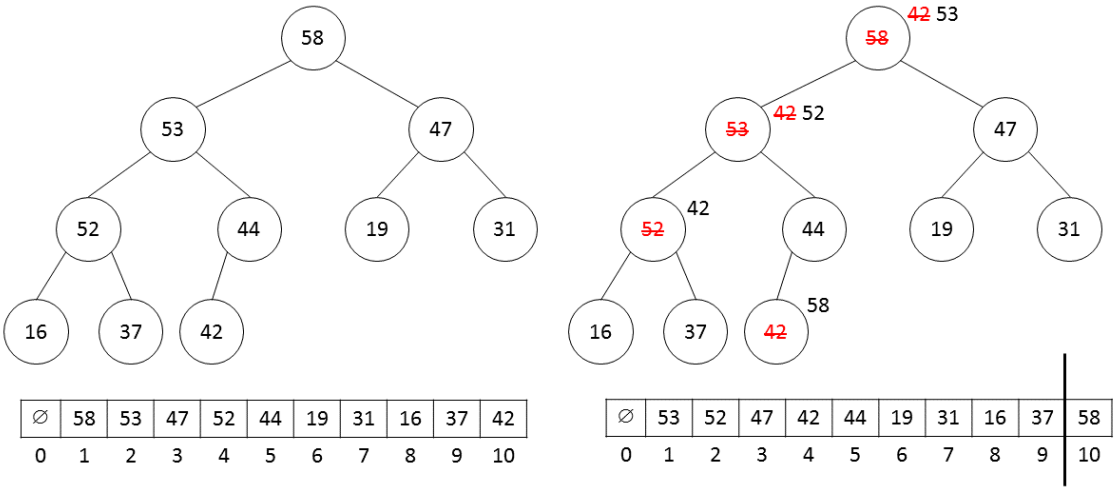
Σχήμα 2: Δημιουργία σωρού από πίνακα (heapify)

Η συνάρτηση λήψης της μεγαλύτερης τιμής από το σωρό (top)

Καθώς η μεγαλύτερη τιμή βρίσκεται πάντα στη θέση 1 του πίνακα που διατηρεί τα δεδομένα του σωρού η συνάρτηση top απλά επιστρέφει την τιμή αυτή.

Η συνάρτηση εξαγωγής της μεγαλύτερης τιμής από το σωρό (pop)

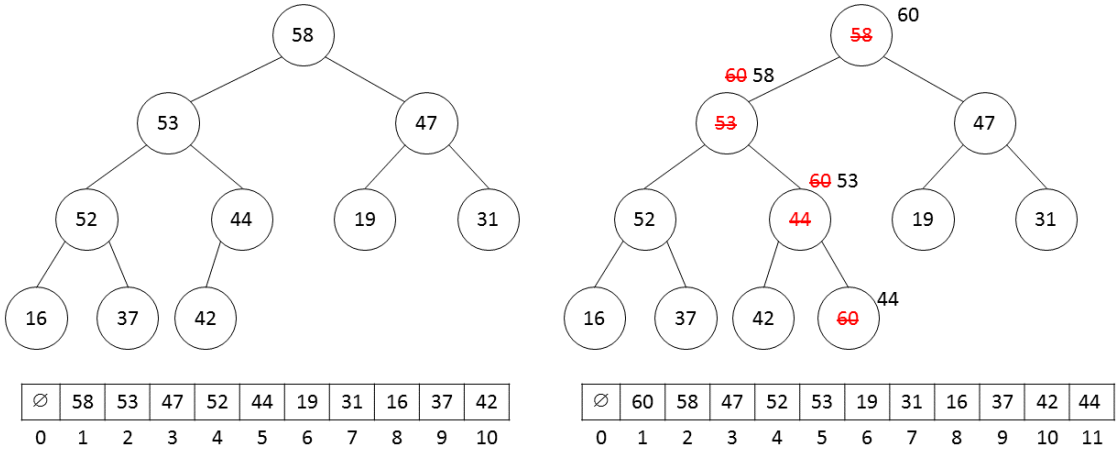
Η εξαγωγή της μεγαλύτερη τιμής γίνεται ως εξής. Το στοιχείο που βρίσκεται στην κορυφή του σωρού αντιμετατίθεται με το τελευταίο στοιχείο του σωρού. Στη συνέχεια το στοιχείο που έχει βρεθεί στην κορυφή του σωρού κατεβαίνει προς τα κάτω αν έχει παιδί που είναι μεγαλύτερό του πραγματοποιώντας αντιμετάθεση με το μεγαλύτερο στοιχείο από τα παιδιά του. Η διαδικασία επαναλαμβάνεται για τη νέα θέση του στοιχείου που αρχικά είχε μεταφερθεί στη κορυφή και μέχρι να ισχύσει ότι είναι μεγαλύτερο και από τα δύο παιδιά του. Στο σχήμα 3 παρουσιάζεται η εξαγωγή της κορυφαίας τιμής του σωρού.



Σχήμα 3: Εξαγωγή της μεγαλύτερης τιμής του σωρού (pop)

Η συνάρτηση εισαγωγής νέας τιμής στο σωρό (push)

Η εισαγωγή ενός στοιχείου γίνεται ως φύλλο στη πρώτη διαθέσιμη θέση από πάνω προς τα κάτω και από δεξιά προς τα αριστερά. Το στοιχείο αυτό συγκρίνεται με το γονέα του και αν είναι μεγαλύτερο αντιμετατίθεται με αυτόν. Η διαδικασία συνεχίζεται μέχρι είτε να βρεθεί το νέο στοιχείο στην κορυφή είτε να ισχύει η κυριαρχία γονέα. Στο σχήμα 4 παρουσιάζεται η εισαγωγή της τιμής 60 σε έναν σωρό μεγίστων.



Σχήμα 4: Εισαγωγή της τιμής 60 στο σωρό (push)

Παράδειγμα χρήσης των συναρτήσεων push και pop)

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

```
#include ”max_heap.cpp”

int main(void) {
   int a[10] = {42, 37, 31, 16, 53, 19, 47, 58, 33, 25};
   for (int i = 0; i < 10; i++) {
      print_heap(false);
      cout << ” ==> push key ” << a[i] << ” ==> ”;
      push(a[i]);
      print_heap();
   }
   while (heap_size > 0) {
      print_heap(false);
      cout << ” ==> pop ==> key=” << heap[1] << ”, ”;
      pop();
      print_heap();
   }
}
```

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

```
HEAP(0) [] ==> push key 42 ==> HEAP(1) [42]
HEAP(1) [42] ==> push key 37 ==> HEAP(2) [42 37]
HEAP(2) [42 37] ==> push key 31 ==> HEAP(3) [42 37 31]
HEAP(3) [42 37 31] ==> push key 16 ==> HEAP(4) [42 37 31 16]
HEAP(4) [42 37 31 16] ==> push key 53 ==> HEAP(5) [53 42 31 16 37]
HEAP(5) [53 42 31 16 37] ==> push key 19 ==> HEAP(6) [53 42 31 16 37 19]
HEAP(6) [53 42 31 16 37 19] ==> push key 47 ==> HEAP(7) [53 42 47 16 37 19 31]
HEAP(7) [53 42 47 16 37 19 31] ==> push key 58 ==> HEAP(8) [58 53 47 42 37 19 31 16]
HEAP(8) [58 53 47 42 37 19 31 16] ==> push key 33 ==> HEAP(9) [58 53 47 42 37 19 31 16 33]
HEAP(9) [58 53 47 42 37 19 31 16 33] ==> push key 25 ==> HEAP(10) [58 53 47 42 37 19 31 16 33 25]
HEAP(10) [58 53 47 42 37 19 31 16 33 25] ==> pop ==> key=58, HEAP(9) [53 42 47 33 37 19 31 16 25]
HEAP(9) [53 42 47 33 37 19 31 16 25] ==> pop ==> key=53, HEAP(8) [47 42 31 33 37 19 25 16]
HEAP(8) [47 42 31 33 37 19 25 16] ==> pop ==> key=47, HEAP(7) [42 37 31 33 16 19 25]
HEAP(7) [42 37 31 33 16 19 25] ==> pop ==> key=42, HEAP(6) [37 33 31 25 16 19]
HEAP(6) [37 33 31 25 16 19] ==> pop ==> key=37, HEAP(5) [33 25 31 19 16]
HEAP(5) [33 25 31 19 16] ==> pop ==> key=33, HEAP(4) [31 25 16 19]
HEAP(4) [31 25 16 19] ==> pop ==> key=31, HEAP(3) [25 19 16]
```

```
HEAP(3) [25 19 16] ==> pop ==> key=25, HEAP(2) [19 16]
HEAP(2) [19 16] ==> pop ==> key=19, HEAP(1) [16]
HEAP(1) [16] ==> pop ==> key=16, HEAP(0) []
```

4 Ταξινόμηση Heapsort

Ο αλγόριθμος Heapsort προτάθηκε από τον J.W.J.Williams το 1964 [1] και αποτελείται από 2 στάδια:

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

Το αποτέλεσμα είναι ότι τα στοιχεία αφαιρούνται από το σωρό σε φθίνουσα σειρά (για έναν σωρό μεγίστων). 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
```

Περισσότερες πληροφορίες για την ταξινόμηση heapsort μπορούν να βρεθούν στην αναφορά [2].

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
```

Περισσότερες πληροφορίες για τη δομή std::priority_queue μπορούν να βρεθούν στις αναφορές [3] και [4].

6 Παραδείγματα

6.1 Παράδειγμα 1

Χρησιμοποιώντας τον κώδικα 1, να γραφεί πρόγραμμα που να εισάγει 100.000 τυχαίες ακέραιες τιμές (στο διάστημα [-1.000.000,1.000.000]) σε έναν σωρό μεγίστων με τη συνάρτηση heap_bottom_up καθώς και με διαδοχικές κλήσεις της συνάρτησης push. Χρονομετρείστε τον κώδικα και στις δύο περιπτώσεις δημιουργίας του σωρού και εμφανίστε το κορυφαίο στοιχείο του σωρού. Επαναλάβετε τη διαδικασία χρησιμοποιώντας την std::priority_queue.

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


using namespace std::chrono;

int main(void) {
   constexpr int N = 100000;
   mt19937 mt(1821);
   int a[N];
   uniform_int_distribution<int> dist(-1000000, 1000000);
   for (int i = 0; i < N; i++)
      a[i] = dist(mt);

   auto t1 = high_resolution_clock::now();
   heap_bottom_up(a, N, false);
   auto t2 = high_resolution_clock::now();
   std::chrono::duration<double, std::micro> duration_micro_sec = t2 - t1;
   cout << "A. Top item: " << top() << endl;
   cout << "Time elapsed (heap_bottom_up): " << duration_micro_sec.count()
         << " microseconds " << endl;

   clear_heap();

   t1 = high_resolution_clock::now();
   for (int i = 0; i < N; i++)
      push(a[i]);
   t2 = high_resolution_clock::now();
   duration_micro_sec = t2 - t1;
   cout << "B. Top item: " << top() << endl;
   cout << "Time elapsed (push): " << duration_micro_sec.count()
         << " microseconds " << endl;

   t1 = high_resolution_clock::now();
   priority_queue<int> pq(a, a + N);
   t2 = high_resolution_clock::now();
   duration_micro_sec = t2 - t1;
   cout << "C. Top item: " << pq.top() << endl;
   cout << "Time elapsed (push): " << duration_micro_sec.count()
         << " microseconds " << endl;
}
```

Κώδικας 6: Χρόνος δημιουργίας MAXHEAP: A) με την heap_bottom_up B) με σταδιακές εισαγωγές (push) τιμών στο σωρό και C) με την std::priority_queue (lab06_ex1.cpp)

```
A. Top item: 999994
Time elapsed (heap_bottom_up): 3000.2 microseconds
B. Top item: 999994
Time elapsed (push): 6000.4 microseconds
```

```
C. Top item: 999994
Time elapsed (push): 11000.6 microseconds
```

6.2 Παράδειγμα 2

Έστω ένα παιχνίδι στο οποίο οι παίκτες έχουν όνομα (name) και επίδοση (score). Να γράψετε πρόγραμμα στο οποίο να εισέρχονται στο παιχνίδι 10 παίκτες στη σειρά (player1, player2, ...), πετυχαίνοντας κάποια επίδοση ο καθένας (τυχαίος ακέραιος από το 0 μέχρι το 50.000). Να εμφανίζεται μετά την εισαγωγή του κάθε παίκτη ο παίκτης που προηγείται και η επίδοση του. Τέλος, να εμφανίζονται τα ονόματα των παικτών με τις 3 υψηλότερες επιδόσεις.

```cpp
#include <iostream>
#include <queue>
#include <random>
#include <string>
#define N 10
#define TOP 3

using namespace std;
struct player {
  string name;
  int score;
  bool operator<(const player &other) const { return score < other.score; }
};

int main() {
  mt19937 mt(1821);
  uniform_int_distribution<int> dist(0, 50000);
  priority_queue<player> pq;
  int best_score = -1;
  for (int i = 0; i < N; i++) {
    player p;
    p.name = "player" + to_string(i + 1);
    p.score = dist(mt);
    pq.push(p);
    player top_player = pq.top();
    if (top_player.score != best_score)
      best_score = top_player.score;
    cout << "New player: " << p.name << " with score " << p.score << " best["
         << top_player.name << " score " << top_player.score << "]" << endl;
  }
  cout << "Top " << TOP << " players:" << endl;
  for (int i = 0; i < TOP; i++) {
    player p = pq.top();
    cout << i + 1 << " " << p.name << " " << p.score << endl;
    pq.pop();
  }
}
```

Κώδικας 7: Διατήρηση επιδόσεων σε σωρό (lab06_ex2.cpp)

```
New player: player1 with score 36323 best[player1 score 36323]
New player: player2 with score 21613 best[player1 score 36323]
New player: player3 with score 33218 best[player1 score 36323]
New player: player4 with score 32634 best[player1 score 36323]
New player: player5 with score 454 best[player1 score 36323]
New player: player6 with score 48987 best[player6 score 48987]
New player: player7 with score 25627 best[player6 score 48987]
New player: player8 with score 42239 best[player6 score 48987]
New player: player9 with score 9284 best[player6 score 48987]
```

```
New player: player10 with score 11639 best[player6 score 48987]
Top 3 players:
1 player6 48987
2 player8 42239
3 player1 36323
```

6.3 Παράδειγμα 3

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;
   }
```

```
40    return sum;
41  }
42
43  int main(int argc, char **argv) {
44    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 8: Υπολογισμός διαμέσου σε μια ροή τιμών (lab06_ex3.cpp)

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

7 Ασκήσεις

1. Να υλοποιηθεί ο σωρός μεγίστων που παρουσιάστηκε στον κώδικα 1 ως κλάση. Προσθέστε εξαιρέσεις έτσι ώστε να χειρίζονται περιπτώσεις όπως όταν ο σωρός είναι άδειος και ζητείται εξαγωγή της μεγαλύτερης τιμής ή όταν ο σωρός είναι γεμάτος και επιχειρείται εισαγωγή νέας τιμής.

2. Να γραφεί συνάρτηση που να δέχεται ως παράμετρο έναν πίνακα ακεραίων και έναν ακέραιο αριθμό κ και να επιστρέφει το κ-οστό μεγαλύτερο στοιχείο του πίνακα.

Αναφορές

[1] NIST, heapsort, https://xlinux.nist.gov/dads/HTML/heapSort.html

[2] PROGRAMIZ, Heap Sort Algorithm, https://www.programiz.com/dsa/heap-sort

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

[4] Cppreference.com, std::priority_queue, http://en.cppreference.com/w/cpp/container/priority_queue