Δομές Δεδομένων

Υλοποίηση Δυαδικού Σωρού Ελαχίστου σε γλώσσα C

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Δυαδικός Σωρός

ως ΑΤΔ

Ένας δυαδικός σωρός ελαχίστου είναι ένας ΑΤΔ που υποστηρίζει τις εξής λειτουργίες:

- Εισαγωγή ενός νέου αντικειμένου x με κλειδί k, INSERT(H,x,k).
- Εύρεση του στοιχείου με το ελάχιστο κλειδί (υψηλότερη προτεραιότητα), FINDMIN(H).
- Διαγραφή του στοιχείου με το ελάχιστο κλειδί (υψηλότερη προτεραιότητα), DELMIN(H).
- Επιστροφή του αριθμού των στοιχείων που είναι στον σωρό, SIZE(H)
- Έλεγχος αν ο σωρός είναι άδειος, ISEMPTY(H).

Δυαδικός Σωρός

Ένα δέντρο είναι **διατεταγμένο σε σωρό** (heap-ordered) αν το κλειδί κάθε κόμβου είναι μικρότερο ή ίσο από τα κλειδιά όλων των παιδιών του.

Δυαδικός Σωρός

Ένα δέντρο είναι **διατεταγμένο σε σωρό** (heap-ordered) αν το κλειδί κάθε κόμβου είναι μικρότερο ή ίσο από τα κλειδιά όλων των παιδιών του.

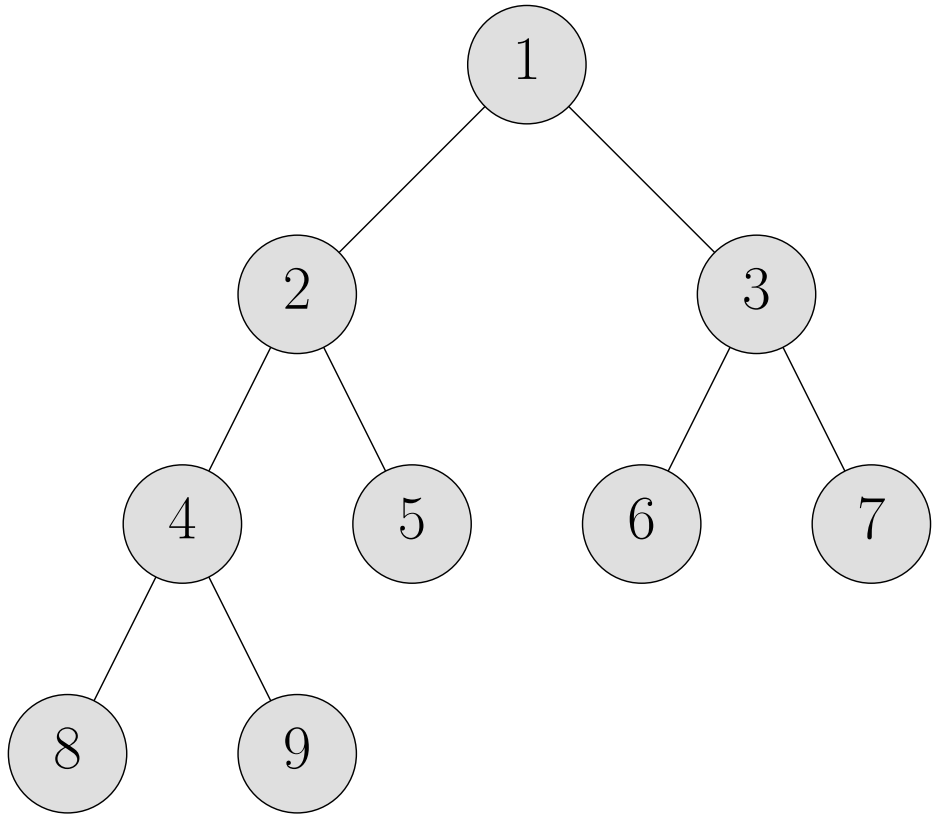
Δυαδικός Σωρός Ελαχίστου (binary min heap)

Ο **σωρός** είναι ένα σύνολο κόμβων με κλειδιά τοποθετημένα σε ένα πλήρες δυαδικό δέντρο το οποίο είναι διατεταγμένο σε σωρό και αναπαριστάται ως πίνακας.

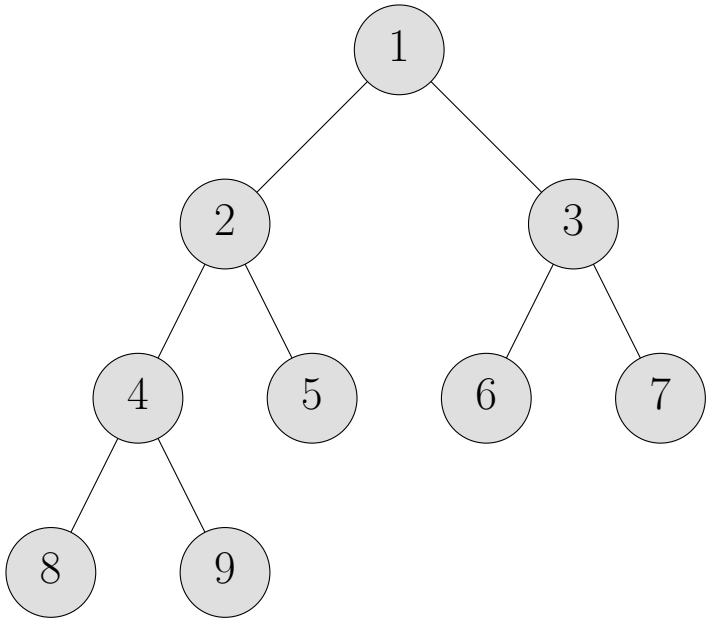
Πλήρες Δυαδικό Δέντρο

Ορισμός

Ένα δυαδικό δέντρο όπου όλα τα επίπεδα, εκτός ίσως του τελευταίου, είναι συμπληρωμένα, το οποίο συμπληρώνεται από τα αριστερά προς τα δεξιά.



Πλήρες Δυαδικό Δέντρο σε Πίνακα



1	2	3	4	5	6	7	8	9

- $\mathit{parent}(i) = \lfloor i/2 \rfloor$
- $\mathit{left} - \mathit{child}(i) = 2i$
- $\mathit{right} - \mathit{child}(i) = 2i + 1$

Δυαδικός Σωρός

διασύνδεση σε C (minheap.h)

```c
#ifndef _MINHEAP_H
#define _MINHEAP_H

typedef int key_type;
typedef struct _minheap* minheap;

minheap minheap_create();
minheap minheap_heapify(const key_type* array, int n);
void    minheap_destroy(minheap);

int     minheap_findmin(minheap);
void    minheap_insert(minheap, key_type);
void    minheap_deletemin(minheap);

int     minheap_is_empty(minheap);
int     minheap_size(minheap);
void    minheap_clear(minheap);

#endif
```

Υλοποίηση Δυαδικού Σωρού

Αναπαράσταση (minheap.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "minheap.h"

struct _minheap {
    key_type* array;
    int max_size;
    int cur_size;
};
```

1. `array` είναι ο πίνακας με τα κλειδιά
2. `max_size+1` είναι το μέγεθος του πίνακα
3. `cur_size` είναι η τελευταία θέση του πίνακα που χρησιμοποιείται

Υλοποίηση Δυαδικού Σωρού

Δημιουργία (minheap.c)

```
minheap minheap_create() {
    minheap h = (minheap) malloc(sizeof(struct _minheap));
    if (h == NULL) {
        fprintf(stderr, "Not enough memory!\n");
        abort();
    }

    h->max_size = 64;
    h->cur_size = 0;
    h->array = (key_type*) malloc( \
        sizeof(key_type)*(h->max_size+1));
    if (h->array == NULL) {
        fprintf(stderr, "Not enough memory!\n");
        abort();
    }

    return h;
}
```

Υλοποίηση Δυαδικού Σωρού

Καταστροφή (minheap.c)

```
void minheap_destroy(minheap h) {
    assert(h);
    free(h->array);
    free(h);
}
```

Υλοποίηση Δυαδικού Σωρού

Διπλασιασμός Χωρητικότητας (minheap.c)

```
static
void minheap_double_capacity(minheap h) {
    // create double the array
    int new_max_size = 2 * h->max_size;
    key_type* new_array = (key_type*) malloc( \
        sizeof(key_type)*(new_max_size+1));
    if (new_array == NULL) {
        fprintf(stderr, "Not enough memory!\n");
        abort();
    }

    /* copy old elements to new array */
    for(int i = 1; i <= h->cur_size; i++) {
        new_array[i] = h->array[i];
    }

    /* free old array and place new in position */
    free(h->array);
    h->array = new_array;
    h->max_size = new_max_size;
}
```

Υλοποίηση Δυαδικού Σωρού

Αντιμετάθεση (minheap.c)

```
static
void minheap_swap(minheap h, int i, int j) {
    assert (h && i >=1 && i <= h->cur_size &&
            j >= 1 && j <= h->cur_size);
    key_type tmp = h->array[i];
    h->array[i] = h->array[j];
    h->array[j] = tmp;
}
```

Υλοποίηση Δυαδικού Σωρού

Κάτω-Πάνω Τακτοποίηση (minheap.c)

```
static
void minheap_fixup(minheap h, int k) {
    assert(h && k >= 1 && k <= h->cur_size);

    while (k>1 && h->array[k] < h->array[k/2]) {
        minheap_swap(h, k/2, k);
        k /= 2;
    }
}
```

Υλοποίηση Δυαδικού Σωρού

Πάνω-Κάτω Τακτοποίηση (minheap.c)

```
static
void minheap_fixdown(minheap h, int k) {
    assert(h);

    while (2*k <= h->cur_size) {
        int j = 2*k;
        if (j < h->cur_size && h->array[j+1] < h->array[j])
            j++;
        if (h->array[k] <= h->array[j])
            break;

        minheap_swap(h, k, j);
        k = j;
    }
}
```

Υλοποίηση Δυαδικού Σωρού

Εισαγωγή (minheap.c)

```
void minheap_insert(minheap h, key_type key) {
    assert(h);

    // make sure there is space
    if (h->cur_size == h->max_size)
        minheap_double_capacity(h);

    // add at the bottom, as a leaf
    h->array[++h->cur_size] = key;

    // fix its position
    minheap_fixup(h, h->cur_size);
}
```

Υλοποίηση Δυαδικού Σωρού

Έυρεση Ελαχίστου (minheap.c)

```
int minheap_findmin(minheap h) {
    if (minheap_is_empty(h)) {
        fprintf(stderr, "Heap is empty!\n");
        abort();
    }

    // min is always in first position
    return h->array[1];
}
```

Υλοποίηση Δυαδικού Σωρού

Διαγραφή Ελαχίστου (minheap.c)

```
void minheap_deletemin(minheap h) {
    if (minheap_is_empty(h)) {
        fprintf(stderr, "Heap is empty!\n");
        abort();
    }

    // swap first with last
    minheap_swap(h, 1, h->cur_size);

    // delete last
    h->cur_size--;

    // fixdown first
    minheap_fixdown(h, 1);
}
```

Υλοποίηση Δυαδικού Σωρού

Μέγεθος και Έλεγχος Κενού (minheap.c)

```c
int minheap_size(minheap h) {
    assert(h);
    return h->cur_size;
}

int minheap_is_empty(minheap h) {
    assert(h);
    return h->cur_size <= 0;
}
```

Υλοποίηση Δυαδικού Σωρού

Διαγραφή Όλων (minheap.c)

```c
void minheap_clear(minheap h) {
    assert(h);
    h->cur_size = 0;
}
```

Υλοποίηση Δυαδικού Σωρού

Heapify (minheap.c)

```
minheap minheap_heapify(const key_type* array, int n) {
    assert(array && n > 0);

    minheap h = (minheap) malloc(sizeof(struct _minheap));
    if (h == NULL) {
        fprintf(stderr, "Not enough memory!\n");
        abort();
    }
    h->max_size = n;
    h->cur_size = 0;
    h->array = (key_type*) malloc(sizeof(key_type)*(h->max_size+1));
    if (h->array == NULL) {
        fprintf(stderr, "Not enough memory!\n");
        abort();
    }

    h->cur_size = n;
    for(int k = 0; k < n; k++)
        h->array[k+1] = array[k];

    for(int k = (h->max_size+1)/2; k > 0; k--)
        minheap_fixdown(h, k);

    return h;
}
```

Χρησιμοποιώντας τον Δυαδικό Σωρό

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "minheap.h"

int main() {
    int i;
    srand(time(NULL));

    minheap h = minheap_create();

    for(i = 0; i < 100; i++)
        minheap_insert(h, rand() % 1000);

    while(!minheap_is_empty(h)) {
        printf("%4d", minheap_findmin(h));
        minheap_deletemin(h);
    }

    minheap_destroy(h);

    return 0;
}
```

HeapSort

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "minheap.h"

void heapsort(int *array, int n) {
    minheap h = minheap_heapify(array, n);

    int i = 0;
    while(!minheap_is_empty(h)) {
        array[i++] = minheap_findmin(h);
        minheap_deletemin(h);
    }

    minheap_destroy(h);
}
```

HeapSort (συνέχεια)

```
int main() {
    srand(time(NULL));

    int array[SIZE];
    for(int i = 0; i < SIZE; i++) {
        array[i] = rand() % MAX_NUMBER;
    }

    heapsort(array, SIZE);

    for(int i = 1; i < SIZE; i++) {
        assert(array[i-1] <= array[i]);
    }
    return 0;
}
```