ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Приоритетна редица со heap

Алгоритми и податочни структури

Аудиториска вежба 9 – помошни материјали

Приоритетна редица

- Приоритетна редица е податочна структура во која што за секој елемент се поврзува приоритет (на пример приоритет според големина, или според лексикографски редослед, или според година на раѓање)
- Секоја приоритетна редица ги подржува следните основни операции:
  - Внесување на нов елемент со одреден приоритет
  - Отстранување на елементот со најголем приоритет

Приоритетна редица - C

```
// vraka 1 ako a ima pogolem prioritet
// vraka -1 ako b ima pogolem prioritet
// vraka 0 ako a i b imaat ist prioritet
int compare(int a, int b) {        // vie go definirate prioritetot
                                   // na elementite
    if (a > b)
        return 1;
    if (a < b)
        return -1;
    return 0;
}

typedef int info_t;                // vie go definirate tipot na elementite
```

Приоритетна редица - C

```
void initialize(heap *h) {
    h->elements = (info_t *)calloc(NUM_OF_ELEMENTS, sizeof(info_t));
    h->size = 0;
}

int getParent(int i) {
    return (i+1)/2-1;
}

info_t getAt(heap *h, int i) {
    return h->elements[i];
}

int getLeft(int i) {
    return (i+1)*2-1;
}

int getRight(int i) {
    return (i+1)*2;
}

void setElement(heap *h, int index, info_t elem) {
    h->elements[index] = elem;
}
```

Приоритетна редица - C

```
void swap(heap *h, int i, int j) {
    info_t tmp = h->elements[i];
    h->elements[i] = h->elements[j];
    h->elements[j] = tmp;
}

void adjust(heap *h, int i, int n) {
    int left, right, largest;

    while (i < n) {
        left = getLeft(i);
        right = getRight(i);
        largest = i;

        if ((left < n)&&(compare(h->elements[left], h->elements[largest]) > 0))
            largest = left;
        if ((right < n)&&(compare(h->elements[right], h->elements[largest]) > 0))
            largest = right;

        if (largest == i)
            break;

        swap(h, i, largest);
        i = largest;
    }

}
```

Приоритетна редица - C

```
void buildHeap(heap *h) {
    int i;
    for (i=h->size/2-1;i>=0;i--)
        adjust(h, i, h->size);
}

void adjustUp(heap *h, int i, int n){
    // vasiot kod ovde
    int parent;

    while (i > 0) {
        parent = getParent(i);
        if (compare(h->elements[i], h->elements[parent]) <= 0) {
            break;
        } else {
            swap(h, i, parent);
            i = parent;
        }
    }
}
```

Приоритетна редица - C

```
// returns 1 if the element is inserted into the heap
// returns 0 if the element is not inserted into the heap
int insert(heap *h, info_t elem) {
    if (h->size == NUM_OF_ELEMENTS)
        return 0;   // there is not enough space in the array
    h->elements[h->size] = elem;
    h->size++;
    adjustUp(h, h->size-1, h->size);
    return 1;
}

info_t removeMax(heap *h) {
    if (h->size == 0)
        return -1;
    info_t tmp = h->elements[0];
    h->elements[0] = h->elements[h->size-1];
    h->size--;
    adjust(h, 0, h->size);
    return tmp;
}
```

Приоритетна редица - C

```
info_t getMax(heap *h) {
    if (h->size == 0)
        return NULL;
    return h->elements[0];
}

int isEmpty(heap *h) {
    if (h->size == 0)
        return 1;
    else
        return 0;
}

int getSize(heap *h) {
    return h->size;
}
```

Приоритетна редица - Java

```
import java.util.Comparator;

class Heap<E extends Comparable<E>> {

    private int N;
    private E elements[];

    private Comparator<? super E> comparator;

    private int compare(E k1, E k2) {
   return (comparator==null ? k1.compareTo(k2) : comparator.compare(k1,
   k2));
    }

    int getParent(int i) {
        return (i+1)/2-1;
    }

    public E getAt(int i) {
        return elements[i];
    }
```

Приоритетна редица - Java

```
int getLeft(int i) {
    return (i+1)*2-1;
}

int getRight(int i) {
    return (i+1)*2;
}

void setElement(int index, E elem) {
    elements[index] = elem;
}

void swap(int i, int j) {
    E tmp = elements[i];
    elements[i] = elements[j];
    elements[j] = tmp;
}
```

Приоритетна редица - Java

```
void adjust(int i, int n){

    while (i < n) {

        int left = getLeft(i);
        int right = getRight(i);
        int largest = i;

        if ((left < n)&&(elements[left].compareTo(elements[largest]) > 0))
            largest = left;
        if ((right < n)&&(elements[right].compareTo(elements[largest]) > 0))
            largest = right;

        if (largest == i)
            break;

        swap(i, largest);
        i = largest;

    }

}
```

Приоритетна редица - Java

```
void buildHeap() {
    int i;
    for (i=elements.length/2-1;i>=0;i--)
        adjust(i, elements.length);
}

@SuppressWarnings("unchecked")
public Heap(int size) {
    elements = (E[])new Comparable[size];
    N = 0;
}
```

Приоритетна редица - Java

```
public boolean insert(E elem) {
    if (N == elements.length)
        return false;   // there is not enough space in the array
    elements[N] = elem;
    N++;
    adjustUp(N-1, N);
    return true;
}

public E getMax() {
    if (N == 0)
        return null;
    return elements[0];
}

 public E removeMax() {
    if (N == 0)
        return null;
    E tmp = elements[0];
    elements[0] = elements[N-1];
    N--;
    adjust(0, N);
    return tmp;
}
```

Приоритетна редица - Java

```
 void adjustUp(int i, int n){
        while (i > 0) {
            int parent = getParent(i);
            if (elements[i].compareTo(elements[parent]) <= 0) {
                break;
            } else {
                swap(i, parent);
                i = parent;
            }
        }
    }

    public boolean isEmpty() {
        if (N == 0)
            return true;
        return false;
    }

    public int getSize() {
        return N;
    }

}
```