Лабораторная работа № 1

Тема: *Структура данных «Куча». Реализация базовых операций и их трудоёмкость.*

Цель: *ознакомиться с основами структуры данных «Куча», изучить основные алгоритмы обработки «Кучи» и реализации базовых операций и их трудоёмкость, научиться применять полученные знания на практике.*

1 Краткая теория

1.1 КУЧА

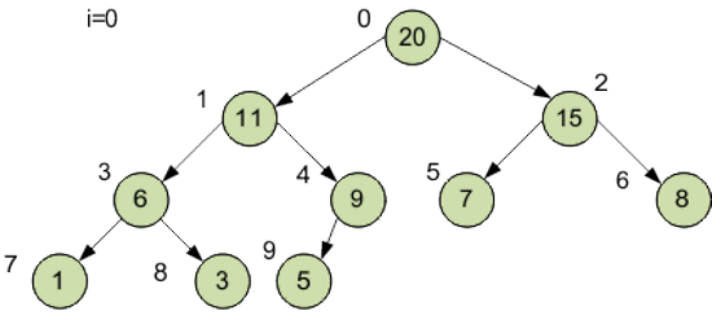
Двоичная куча представляет собой полное бинарное дерево, для которого выполняется основное свойство кучи: **приоритет каждой вершины больше приоритетов её потомков**.

В простейшем случае приоритет каждой вершины можно считать равным её значению. В таком случае структура называется ***max-куча***, поскольку корень поддерева является максимумом из значений элементов поддерева.

В качестве альтернативы, если сравнение перевернуть, то наименьший элемент будет всегда корневым узлом, такие кучи называют ***min-кучами***.

Двоичную кучу удобно хранить в виде одномерного массива, причем

- левый потомок вершины с индексом i имеет индекс 2*i+1,
- правый потомок вершины с индексом i имеет индекс 2*i+2.



Корень дерева (кучи) – элемент с индексом 0.

Высота двоичной кучи равна высоте дерева, то есть

$$\log_2 (N+1)\uparrow,$$

где **N** – количество элементов массива,

↑ – округление в большую сторону до ближайшего целого.

Для представленной кучи

$$\log_2 (10+1)\uparrow = 3{,}46\uparrow = 4$$

Способ построить кучу из неупорядоченного массива – это по очереди добавить все его элементы. Временная оценка такого алгоритма оценивается как

$$N \cdot \log_2 N.$$

Можно построить кучу за N шагов. Для этого сначала следует построить дерево из всех элементов массива, не заботясь о соблюдении основного свойства кучи, а потом вызвать метод упорядочения для всех вершин, у которых есть хотя

бы один потомок (так как поддеревья, состоящие из одной вершины без потомков, уже упорядочены).

Потомки гарантированно есть у первых **heapSize/2** вершин, где heapSize – размер кучи.

Реализация класса кучи

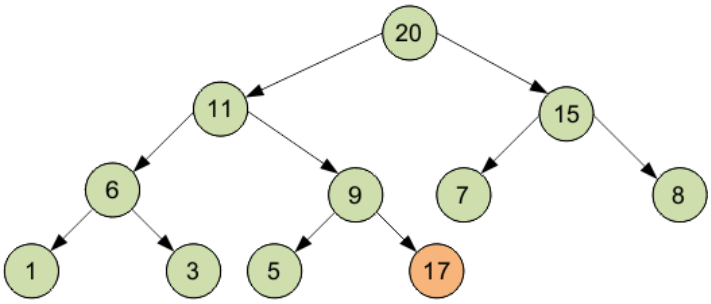
```
class Heap {
  static const int SIZE = 100; // максимальный размер кучи
  int *h;          // указатель на массив кучи
  int HeapSize; // размер кучи
public:
  Heap();  // конструктор кучи
  void addelem(int);  // добавление элемента кучи
  void outHeap();  // вывод элементов кучи в форме кучи
  void out();  // вывод элементов кучи в форме массива
  int getmax();  // удаление вершины (максимального элемента)
  void heapify(int);  // упорядочение кучи
};
```

Конструктор кучи

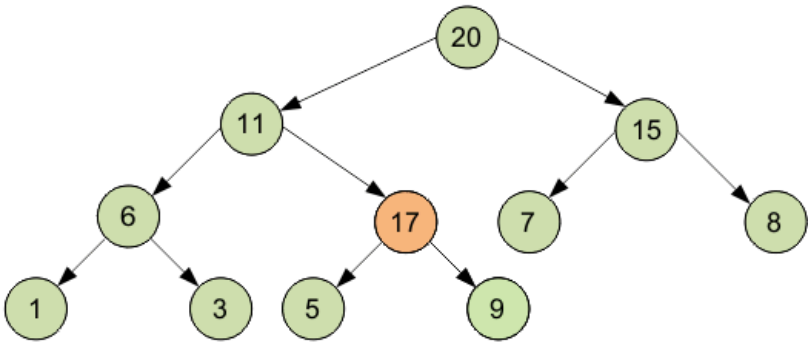
```
Heap :: Heap() {
  h = new int[SIZE];
  HeapSize = 0;
}
```

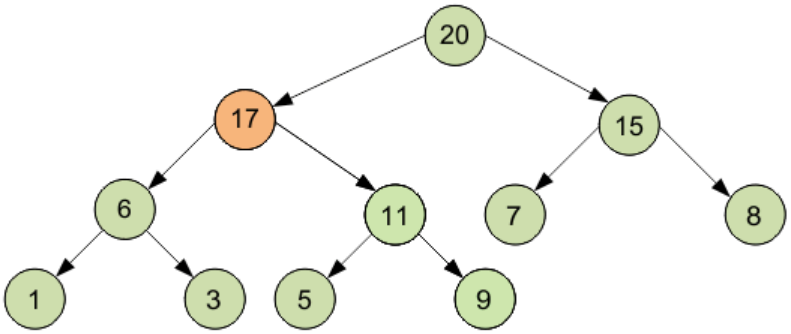
Добавление элемента кучи

Новый элемент добавляется на последнее место в массиве, то есть позицию с максимальным индексом.



Возможно, что при этом будет нарушено основное свойство кучи, так как новый элемент может быть больше родителя. В таком случае новый элемент «поднимается» на один уровень (менять с вершиной-родителем) до тех пор, пока не будет соблюдено основное свойство кучи.





Сложность алгоритма не превышает высоты двоичной кучи (так как количество «подъемов» не больше высоты дерева), то есть равна $\log_2 N$.

```
1   void Heap :: addelem(int n) {
2     int i, parent;
3     i = HeapSize;
4     h[i] = n;
5     parent = (i-1)/2;
6     while(parent >= 0 && i > 0)  {
7       if(h[i] > h[parent]) {
8         int temp = h[i];
9         h[i] = h[parent];
10        h[parent] = temp;
11      }
12      i = parent;
13      parent = (i-1)/2;
14    }
15    HeapSize++;
16  }
```

Вывод элементов кучи

Вывод элементов в форме кучи:

```
void Heap:: outHeap(void) {
  int i = 0;
  int k = 1;
  while(i < HeapSize) {
    while((i < k) && (i < HeapSize)) {
      cout << h[i] << " ";
      i++;
    }
    cout << endl;
    k = k * 2 + 1;
  }
}
```

Вывод элементов кучи в форме массива:

```
void Heap:: out(void) {
  for(int i=0; i< HeapSize; i++) {
    cout << h[i] << " "; }
  cout << endl;
}

```

Упорядочение кучи

```
void Heap:: heapify(int i) {
  int left, right;
  int temp;
  left = 2*i+1;
  right = 2*i+2;
  if(left < HeapSize) {
    if(h[i] < h[left]) {
      temp = h[i];
      h[i] = h[left];
      h[left] = temp;
      heapify(left);
    }
  }
  if(right < HeapSize) {
    if(h[i] < h[right]) {
      temp = h[i];
      h[i] = h[right];
      h[right] = temp;
      heapify(right);
    }
  }
}
```

В упорядоченном max-heap максимальный элемент всегда хранится в корне. Восстановить упорядоченность двоичной кучи после удаления максимального элемента можно, поставив на его место последний элемент и вызвав метод упорядочения для корня, то есть упорядочив все дерево.

Удаление вершины кучи (максимального элемента)

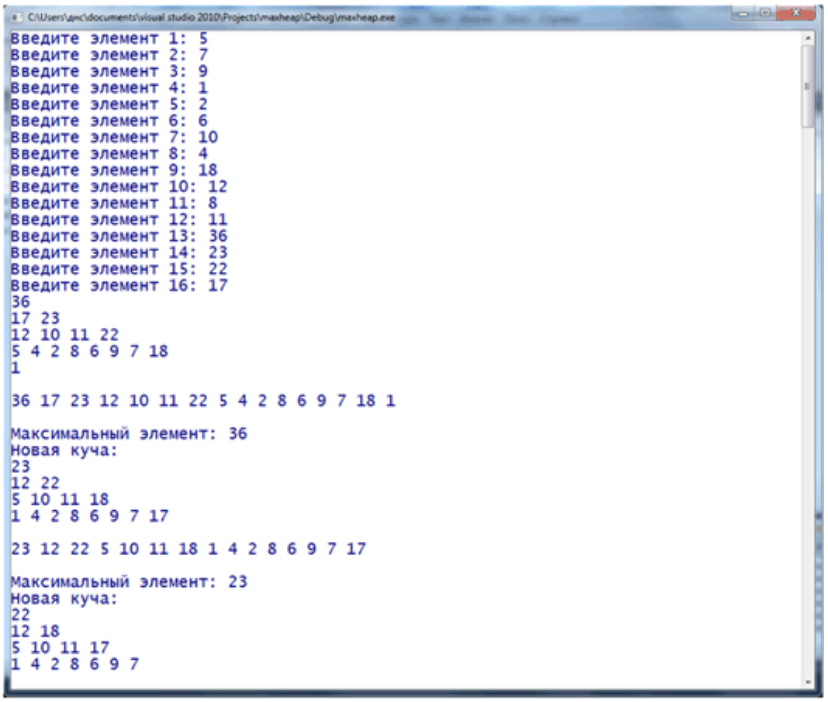
```
1   int Heap:: getmax(void) {
2     int x;
3     x = h[0];
4     h[0] = h[HeapSize-1];
5     HeapSize—;
6     heapify(0);
7     return(x);
8   }
```

Пример

Создать бинарную кучу из 16 элементов. Определить максимальный элемент.

```
1   int main() {
2     Heap heap;
3     int k;
4     system("chcp 1251");
5     system("cls");
6     for(int i=0; i<16; i++) {
7       cout << "Введите элемент " << i+1 << ": ";
8       cin >> k;
9       heap.addelem(k);
10    }
11    heap.outHeap();
12    cout  << endl;
13    heap.out();
14    cout  << endl << "Максимальный элемент: " << heap.getmax();
15    cout  << endl << "Новая куча: " << endl;
16    heap.outHeap();
17    cout  << endl;
18    heap.out();
19    cout  << endl << "Максимальный элемент: " << heap.getmax();
20    cout  << endl << "Новая куча: " << endl;
21    heap.outHeap();
22    cout  << endl;
23    heap.out();
24    cin.get();cin.get();
25    return 0;
26  }
```

Результат выполнения:



! Дополнительный материал !

1. Структура данных: двоичная куча (binary heap): *https://habr.com/ru/post/112222/*
2. Структура данных: пирамида (двоичная куча) в Java: *https://javarush.com/groups/posts/3083-strukturih-dannihkh-piramida-dvoichnaja-kucha-v-java*
3. Пирамидальная сортировка (HeapSort): *https://habr.com/ru/company/otus/blog/460087/*

2 Задания

Обязательное: *Создать кучу из n элементов. Найти минимальный элемент в куче. Удалить корневой узел в мин-куче.*

! Контрольные вопросы !

1. Определение понятия куча.
2. Опишите принцип работы кучи.
3. Виды кучи.
4. Операции для работы с кучей.
5. Перечислите способы реализации кучи.

Содержание отчёта

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.

5. Ответы на контрольные вопросы.
6. Выводы.