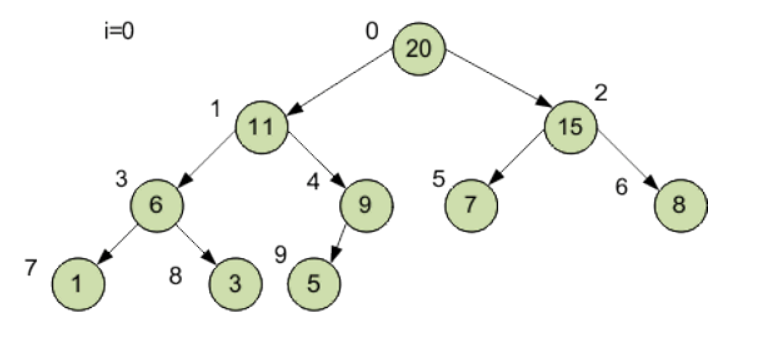
Структуры данных: двоичная куча (binary heap)

Двоичная куча (binary heap) – просто реализуемая структура данных, позволяющая быстро (за логарифмическое время) добавлять элементы и извлекать элемент с максимальным приоритетом (например, максимальный по значению).



Двоичную кучу удобно хранить в виде одномерного массива, причем

1). левый потомок вершины с индексом i имеет индекс 2\*i+1,

2). правый потомок вершины с индексом i имеет индекс 2\*i+2.

3). корень дерева (кучи) – элемент с индексом 0.

Высота двоичной кучи равна высоте дерева, то есть **log2 (N+1)↑,** где **N** – количество элементов массива, **↑** – округление в большую сторону до ближайшего целого.

Способ построить кучу из неупорядоченного массива – это по очереди добавить все его элементы. Временная оценка такого алгоритма оценивается как **N·log2N**.

Реализация класса бинарной кучи на С++ выглядит следующим образом:

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

int getmax(); // удаление вершины (максимального элемента)

void heapify(int); // упорядочение кучи

};

Конструктор выделяет память для нашей кучи количеством нашего максимального размера **SIZE** и размер кечи **HeapSize** приравнивает к 0.

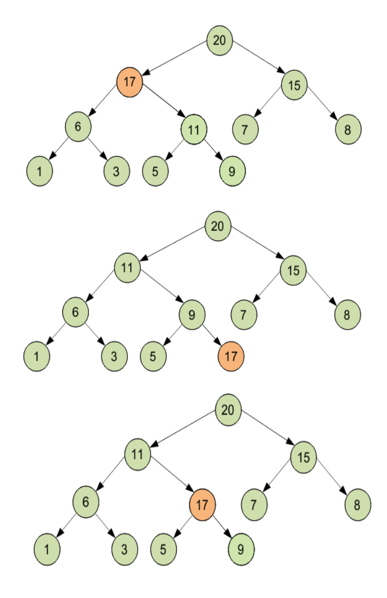
Heap::Heap() {

h = new int[SIZE];

HeapSize = 0;

}

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 log2 N.



void Heap::addelem(int n) {

int i, parent;

i = HeapSize;

h[i] = n;

parent = (i - 1) / 2;

while (parent >= 0 && i > 0) {

if (h[i] > h[parent]) {

int temp = h[i];

h[i] = h[parent];

h[parent] = temp;

}

i = parent;

parent = (i - 1) / 2;

}

HeapSize++;

}

Вывод элементов осуществляется следующим образом:

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;

}

}

Упорядочивание элементов:

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);

}

}

}

Удаление вершины кучи (максимального элемента)

int Heap::getmax(void) {

int x;

x = h[0];

h[0] = h[HeapSize - 1];

HeapSiz--;

heapify(0);

return(x);

}