HEAP(КУПА)

Підготувала: Повар Тетяна, 1 курс магістратури ФВЕ

Двійкова купа

- Двійкова куча являє собою повне бінарне дерево, для якого виконується основне властивість купи: пріоритет кожної вершини більше пріоритетів її нащадків.

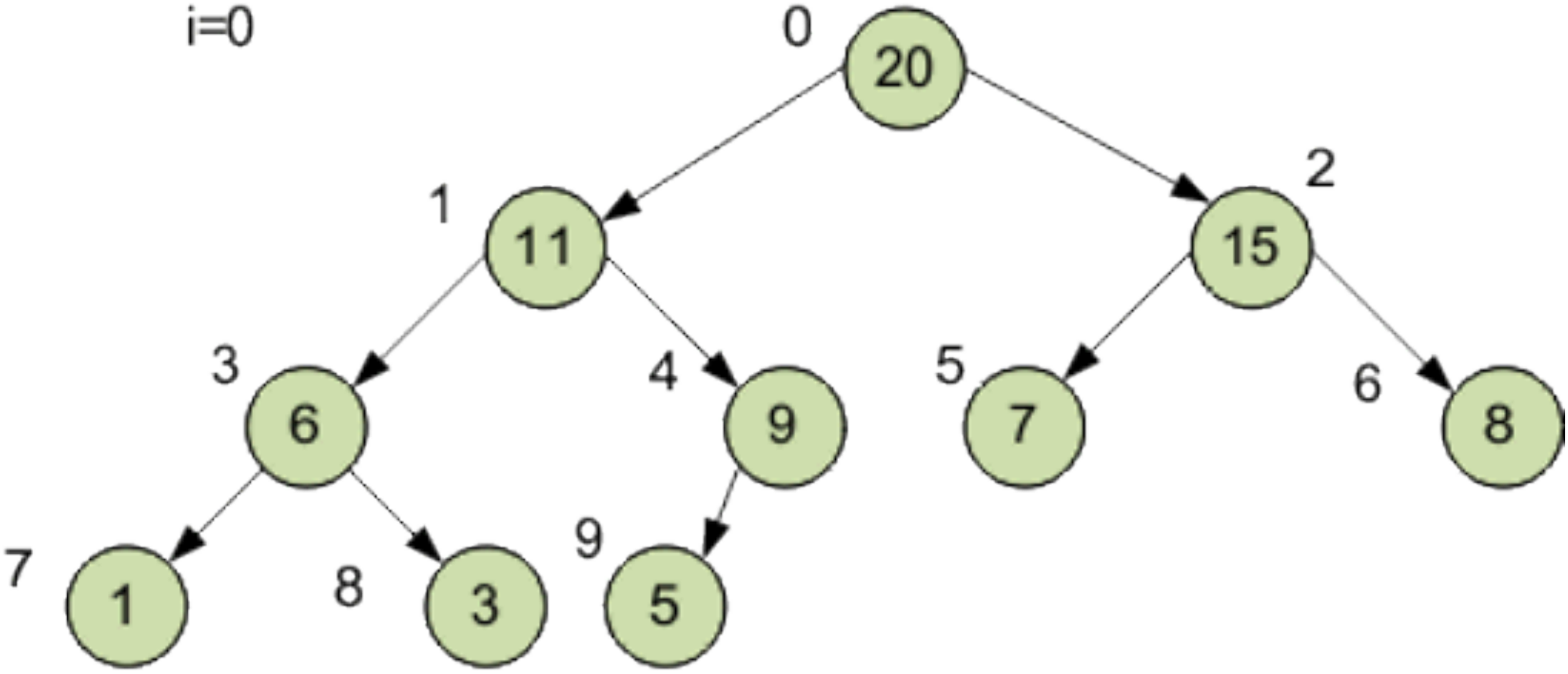
- У найпростішому випадку пріоритет кожної вершини можна вважати рівним її значенню. У такому випадку структура називається max-купа, оскільки корінь піддерева є максимумом зі значень елементів піддерева.

- В якості альтернативи, якщо порівняння перевернути, то найменший елемент буде завжди кореневим вузлом, такі купи називають min-купами.

- Двійкову куау зручно зберігати у вигляді одновимірного масиву, причому

  - лівий нащадок вершини з індексом i має індекс 2 * i + 1,

  - правий нащадок вершини з індексом i має індекс 2 * i + 2.



Двійкова купа

Висота двійкової купи

- Висота двійковій купи дорівнює висоті дерева, тобто

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

де N - кількість елементів масиву, ↑ - округлення в більшу сторону до найближчого цілого.

- Для представленої купи

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

- Спосіб побудувати купу з невпорядкованого масиву - це по черзі додати всі його елементи. Часова оцінка такого алгоритму оцінюється як

$$N \cdot \log_2 N.$$

- Можна побудувати купу за N кроків. Для цього спочатку слід побудувати дерево з усіх елементів масиву, не піклуючись про дотримання основного властивості купи, а потім викликати метод упорядкування для всіх вершин, у яких є хоча б один нащадок (так як піддерева, що складаються з однієї вершини без нащадків, вже впорядковані) .

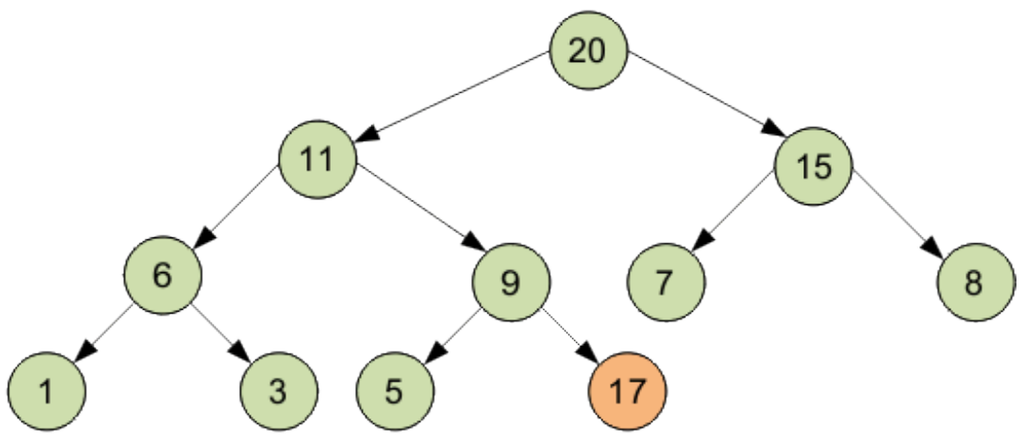
Конструктор купи

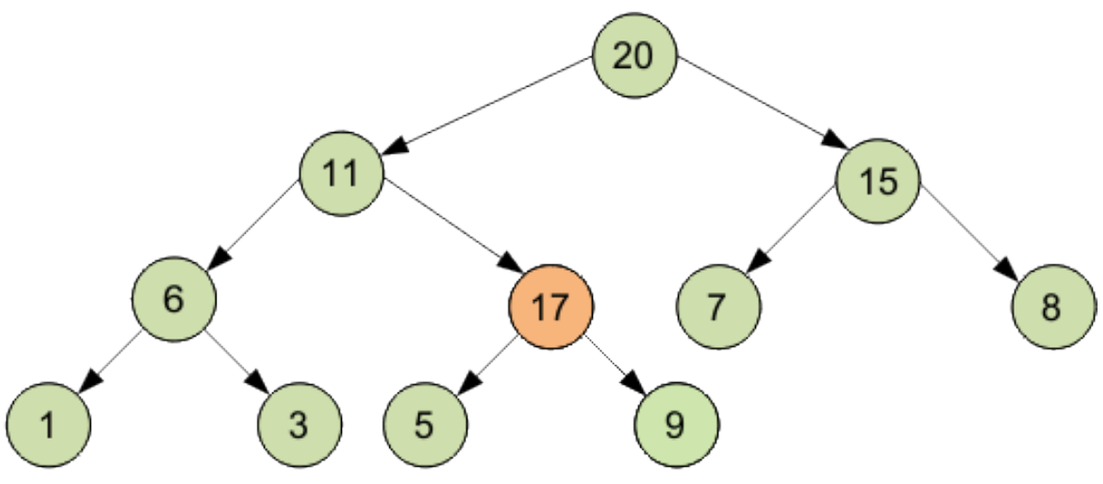
```
Heap :: Heap() {
  h = new int[SIZE];
  HeapSize = 0;
}
```

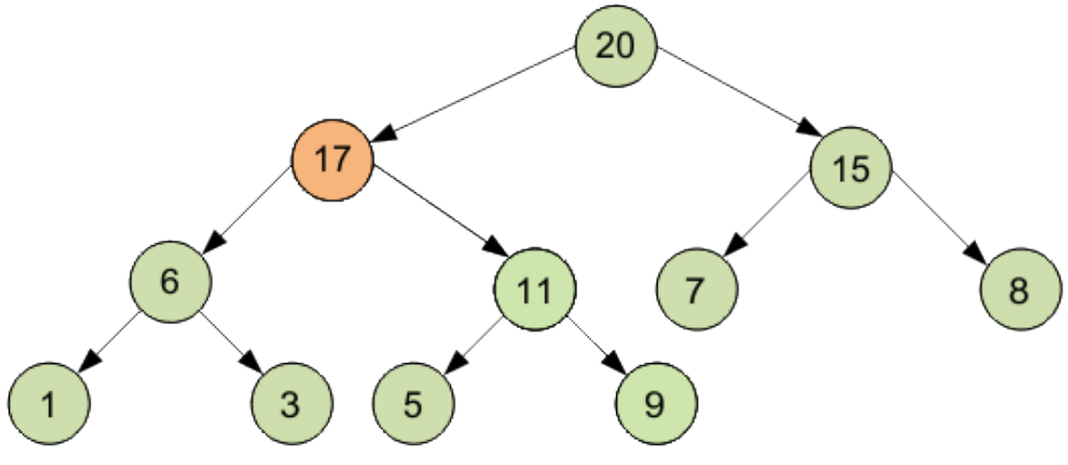
Реалізація класу купи

```
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
   int getmax();  // удаление вершины (максимального элемента)
  void heapify(int);  // упорядочение кучи
};
```

Додавання елемента купи







Складність алгоритму не перевищує висоти двійковій купи (так як кількість «підйомів" не більше висоти дерева), тобто дорівнює $\log_2 N$.

```
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++;
}
```

Видалення елементу з купи

- З купи можна видалити найбільший елемент, тобто той, який зберігається в вершині купі. На його місце потрібно поставити якийсь елемент купи. Поставимо останній елемент купи, видаливши його з кінця. Тепер в вершині купи може порушитися властивість купи, значить, верхній елемент потрібно змістити вниз, обмінявши його з одним зі своїх нащадків. При цьому з двох нащадків потрібно вибрати найбільший і якщо цей найбільший нащадок більше стоїть в вершині купи, обміняємо їх місцями.

- Тим самим елемент, який був узятий знизу купи, спуститься на один рівень вниз. Будемо далі опускати цей елемент до тих пір, поки обидва його нащадка не стануть менше його (або у нього не буде нащадків, також необхідно акуратно обробити випадок одного нащадка.

```
int Heap:: getmax(void) {
  int x;
  x = h[0];
  h[0] = h[HeapSize-1];
  HeapSize—;
  heapify(0);
  return(x);
}
```

Упорядкування купи

```
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);
    }
  }
}
```

Застосування купи

- Одне з найбільш відомих застосувань купи - сортування за допомогою купи або пірамідальна сортування (англ. Heapsort). У даному сортуванні з елементів списку спочатку будується купа, потім елементи по одному видаляються з купи - спочатку найбільший елемент, потім - найбільший з решти і т. Д. При цьому купу можна зберігати там же, де зберігаються елементи самого списку, тим самим пірамідальна сортування має складність O(n log n), але при цьому не вимагає додаткової пам'яті (як сортування злиттям) і не є ймовірнісним (як швидке сортування Хоара).

- Також за допомогою купи можна організувати структуру даних «чергу з пріоритетами». У черзі кожному елементу зіставляється пріоритет - деяке ціле число. При видаленні елемента з черги видаляється не той елемент, який був доданий раніше (як у звичайній черзі), а елемент з найбільшим пріоритетом. Тобто елементи в черзі з пріоритетами можна зберігати в купі, порівнюючи їх при цьому за пріоритетом.

- У черзі з пріоритетами також є операція зміни пріоритету елемента. Для цього реалізовані дві функції - підвищення і зниження пріоритету. При підвищенні пріоритету елемент піднімається вгору, тому ця функція реалізована аналогічно операції додавання елемента. При зниженні пріоритету елемент спускається вниз, як в операції видалення елемента.

Дякую за увагу