Рекуррентные соотношения

* Метода декомпозиции «разделяй и властвуй»
* Парадигма, лежащая в основе метода декомпозиции
* Разделение
* Властвование
* Комбинирование

Рекуррентное соотношение

* Представляет собой уравнение или неравенство, которое описывает функцию через ее значения для меньших аргументов
* Пример – Ханойские башни: T(n-1) = 2T(n-1) + 1

Амортизационный анализ – время, необходимое для выполнения последовательности операций над структурой данных, которое усредняется по всем выполняемым операциям.

Групповой анализ заключается в том, что определяется верхняя граница полной стоимости всех операций и вычисляет среднее значение – отношение полной стоимости к количеству операций.

Бухгалтерский учет (кредитный метод) – каждая операция характеризуется своей амортизационной стоимостью. Вводится переоценка (кредит) для добавления стоимости операции.

Метод потенциалов – допустим перерасчет стоимости.

Push(s, x) O(1)

Pop(s) O(1)

MultiPop(s, k)

while StackIsEmpty == f and k > 0

Pop(s)

k = k – 1

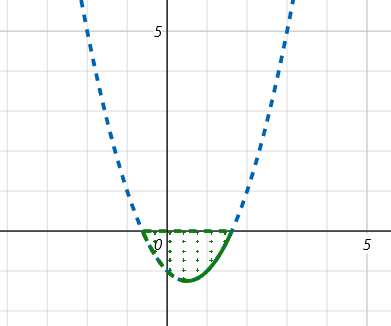
O(min(s, k))

O(n)

Add(x)

T(n) = n \* O(1) + n

|  |  |
| --- | --- |
|  |  |
| 3 | 1.3 |
| 4 | 1.4 |
| 5 | 1.53 |
| 6 | 1.57 |
| 7 | 1.59 |
| 22 | 1.61 |



get(i) – O(1)

set(i, x) – O(1)

add(x) –

del() (последний элемент) –

size() – O(1)

Бухгалтерский метод

add(x) – 3 O (3 монеты)

|O|\_|\_|\_||xO|

i – n/2

|O|O|O|O|O|xO|xO|xO|xO| <- x

|

\/

|\_\_\_\_\_\_\_|

O(1) – добавление

i mod n/4

O(1) – удаление

Метод потенциалов

Ф(m, n) =

m – размер массива

n – число элементов

add()

1. n = m (n/m = 1)

|\_|\_|\_|\_|\_|x|\_\_\_\_

|<-----m------>|

1. m/2 <= n < m, не расширяемся
2. n < m/2, (n+1) >= m/2

|\_|\*||x|\_|

del()

1. n/m = 1/4, сжатие
2. m/4 < n < m/2
3. n >= m/2, n-1 < m/2

Priority Queue

Неупорядоченный массив

peek() – O(n)

dequeue(x) – O(1)

enqueue(x) – O(1)

Упорядоченный массив

peek() – O(1)

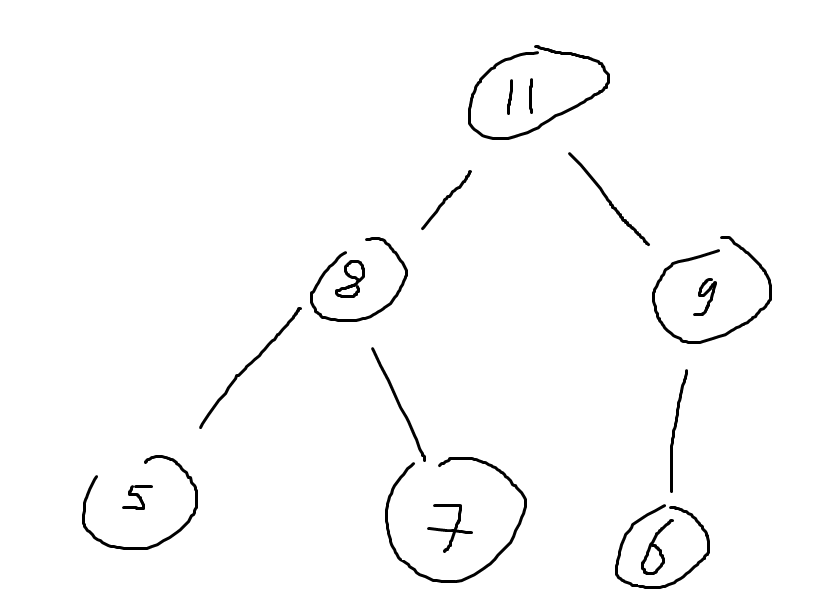
dequeue(x) – O(n)

enqueue(x) – O(n)

Heap (пирамида, куча) – определяется как бинарное дерево с ключами, для которого выполняется 2 следующих условия:

требование к форме дерева – бинарное дерево должно быть полным или практически полным, то есть все его уровни должны быть заполнены, кроме, может быть, последнего, в котором могут отсутствовать некоторые крайние листья;

требование доминирования – для родительских узлов ключ не меньше ключей его дочерних узлов



H = |\_ log n \_|

Свойства:

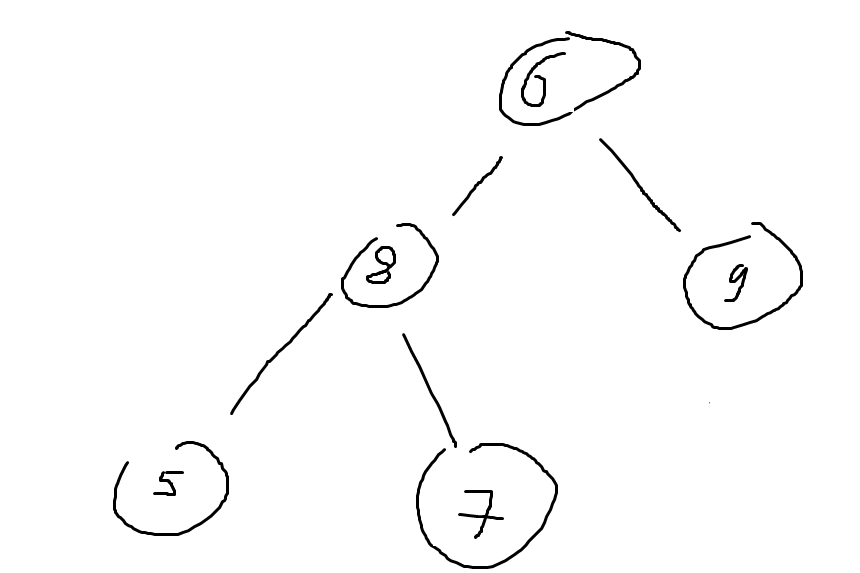
На пирамиде имеется ровно одно практически полное бинарное дерево с n узлами

В корне пирамиды всегда находится ее наибольший элемент

Любой узел пирамиды со всеми его потомками также является пирамидой

Пирамиду можно реализовать в виде массива (11 – 8 – 9 – 5 – 7 – 6) (2х и 2х+1 – потомки)

H[i] = max{ H[2i], H[2n] }, i =1, …, |\_ n/2 \_|



SiftDown(i)

While (i <= n)

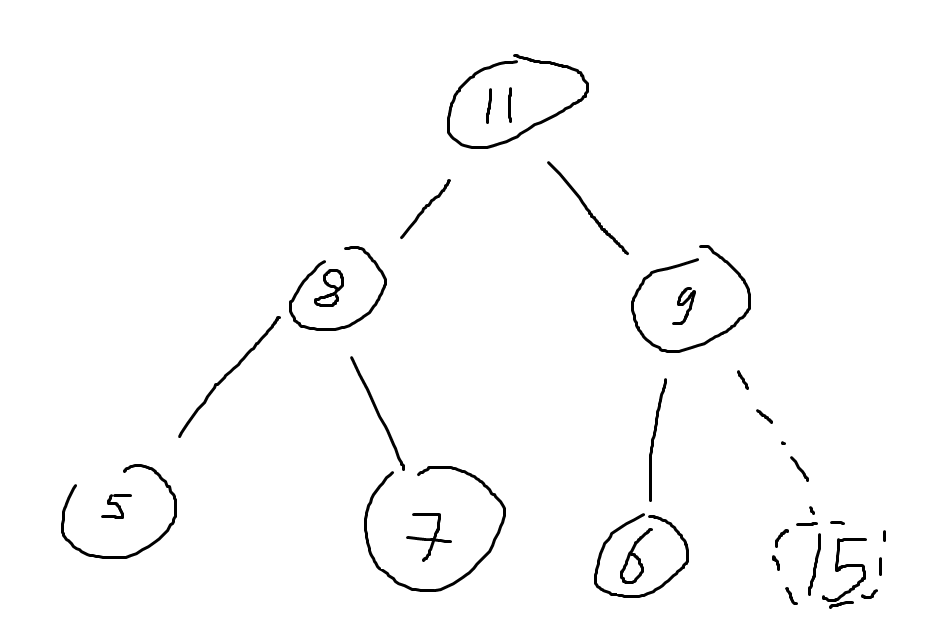
m = max(i, 2i, 2i + 1) <- индекс максимального из элементов

if i = m break

swap(H[m], H[i])

i = m

O(log n)



SiftUp(i)

While (i > 1) and (H[ i/2 ] > H[ i ])

Swap(H[ i ], H[ i/2 ])

i = |\_ i/2 \_|

O(log n)

BuildHeap(H)

for i = |\_ n/2 \_| down to 1

SiftDown(i)

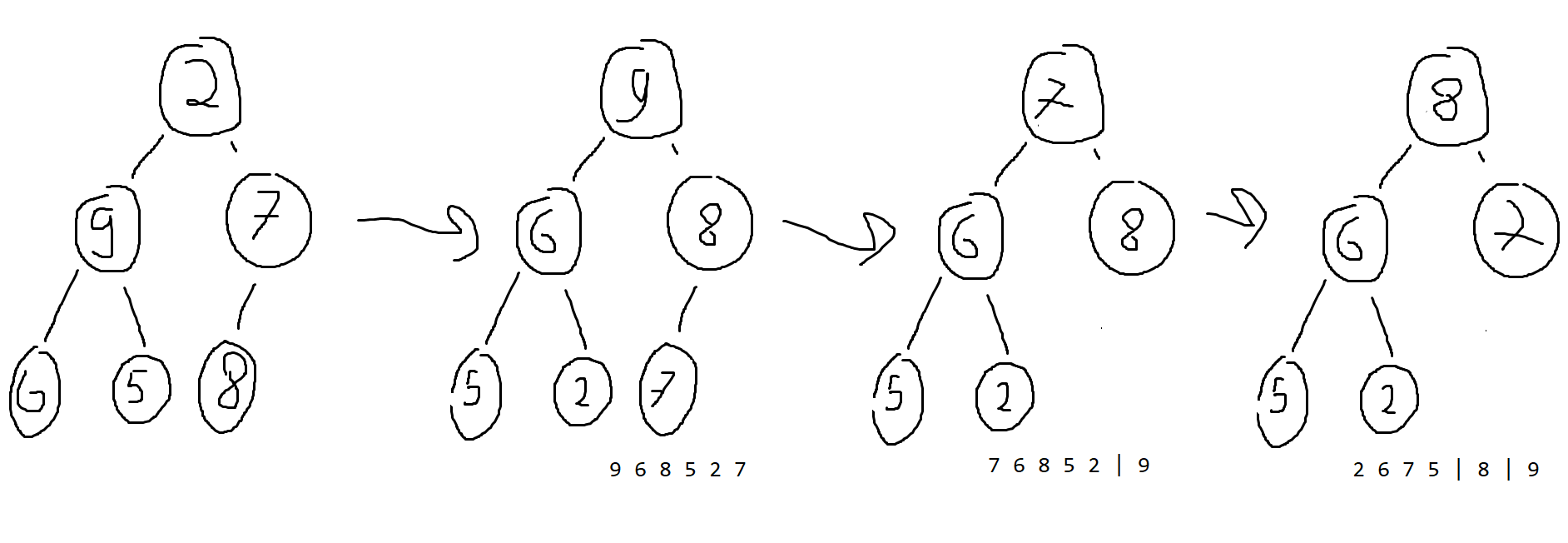
O(n)

Сортировки

HeapSort(H)

Алгоритм пирамидальной сортировки:

1. построение пирамиды
2. удаление наибольшего элемента n-1 раз



BuildHeap(H)

For i = 1 to H.size

Swap(H[i], H[n-i+1])

H.size--

SiftDown(A, 1)

O(n) <= 2 |\_ log(n-1) \_| + 2 |\_log(n-2) \_| + 2 |\_ log 1 \_| <= 2 sum(i=1, i<=n-1, log(n-i)) <= 2 sum(i=1, i<=n-1, log(n-1)) = 2(n-1)log(n-1) <= 2nlogn = O(n log n)

D-ичная пирамида – пирамида с D потомками

SiftUp = O(log3 n)

SiftDown != O(n)

Сортировка слиянием

2 9 7 6 5 8

2 9 7 6 5 8

2 9 7 6 5 8

2 9 7 6 5 8

MergeSort(A, p, r)

If p < r

q=|\_(p+r)/2\_| # D(n) = O(1)

MergeSort(A, p, q)

MergeSort(A, q, r) # C(n) = O(n)

Merge(A, p, q, r)

Merge(A, p, q, r)

n1 = q-p+1

n2 = r-q

Задаем массивы L[1..n1] и R[1..n2]

Копируем A[p..q] в L[1..n1]

Копируем A[q..r] в R[1..n2]

i = 1, j = 1

for k = p to r

if L[i] < R[j]

A[k] = L[i]

i = i + 1

else

A[k] = R[j]

j = j + 1

\*1 Инвариант цикла – логическое выражение, которое истинно после каждого прохода тела цикла и перед началом выполнения цикла, зависящее от переменных, изменяющихся в цикле

\*2 Пусть на вход алгоритма QuickSort поступает массив с n различными элементами. Тогда математическое ожидание времени работы алгоритма при случайном равномерном и независимом выборе разделителя составляет O(n log n). Мат. ожидание глубины рекурсии оценивается как D(n) = O(log n)

Taverage(n) – среднее время

\*3 Если рассматривать вариацию алгоритма быстрой сортировки, детерминировано выбирающего в качестве опорного элемента первый элемент текущего подмассива, а на вход алгоритму поступает случайная последовательность, в которой все элементы различны и все перестановки равновероятны, тогда среднее время работы алгоритма – O(n log n)

\*4 Любой алгоритм сортировки сравнения в наихудшем случае требует сравнений

\*5 Сортировки за линейное время:

Сортировка подсчетом – входные данные – целые числа из некоторого интервала [0, k]. k = O(n) => . Для каждого элемента х определяется количество элементов меньше х. Размещаем элемент х в той позиции выходного массива, где он должен находиться.

\*6 Поразрядная сортировка

\*7 Лемма: пусть имеется n d-значных чисел, в которых каждая цифра принимает одно из k возможных значений. Тогда алгоритм поразрядной сортировки выполняет сортировку за время T(n) = O(d(n+k)); если сортировка устойчива, то

Карманная сортировка – входные данные подчиняются равномерному закону распределения. T(n) = O(n)