Алгоритмы и структуры данных

Лекция 4. Сортировки I.

Артур Кулапин

ВШПИ МФТИ

3 октября 2023 г.

О чем разговор?

1. Сортировки, основанные на сравнениях
2. Сортировка слиянием
   - Сортировка слиянием
   - Подсчет числа инверсий
3. Пирамидальная сортировка
   - Бинарная куча
   - Представление в виде массива

Сортировки, основанные на сравнениях

Def. Сортировка набора элементов $a_1, \ldots, a_n$ — задача поиска такой перестановки индексов σ, что $a_{\sigma(1)} \leq a_{\sigma(2)} \leq \ldots \leq a_{\sigma(n)}$.

Def. Сортировка основана на сравнениях, если про любые два элемента можно спросить «правда ли, что $x < y$?» и ничего более.

Теорема о сортировках

Теорема (б/д). Сортировка N элементов, основанная на сравнениях, требует $\Omega(N \log N)$ сравнений.

Процедура Merge

Задача. Даны два отсортированных массива $a_1 \leq \ldots \leq a_n$ и $b_1 \leq \ldots \leq b_m$. Надо слить их в один отсортированный за $O(n+m)$ времени и доппамяти.

Процедура Merge

Задача. Даны два отсортированных массива $a_1 \leq \ldots \leq a_n$ и $b_1 \leq \ldots \leq b_m$. Надо слить их в один отсортированный за $O(n+m)$ времени и доппамяти.

Решение.

1. Заведем указатели на первые элементы массивов i и j.

Процедура Merge

Задача. Даны два отсортированных массива $a_1 \leq \ldots \leq a_n$ и $b_1 \leq \ldots \leq b_m$. Надо слить их в один отсортированный за $O(n+m)$ времени и доппамяти.

Решение.

1. Заведем указатели на первые элементы массивов i и j.
2. По индексу $i+j$ выпишем меньший из a_i и b_j и сдвинем соответствующий указатель на один вперед.

Процедура Merge

Задача. Даны два отсортированных массива $a_1 \leq \ldots \leq a_n$ и $b_1 \leq \ldots \leq b_m$. Надо слить их в один отсортированный за $O(n+m)$ времени и доппамяти.

Решение.

1. Заведем указатели на первые элементы массивов i и j.
2. По индексу $i+j$ выпишем меньший из a_i и b_j и сдвинем соответствующий указатель на один вперед.
3. Так делаем, пока $i < n$ && $j < m$.

Процедура Merge

Задача. Даны два отсортированных массива $a_1 \leq \ldots \leq a_n$ и $b_1 \leq \ldots \leq b_m$. Надо слить их в один отсортированный за $O(n+m)$ времени и доппамяти.

Решение.

1. Заведем указатели на первые элементы массивов i и j.
2. По индексу $i+j$ выпишем меньший из a_i и b_j и сдвинем соответствующий указатель на один вперед.
3. Так делаем, пока $i < n$ && $j < m$.
4. Оставшийся хвост одного из массивов дописываем в конец.

MergeSort

Задача. Дан массив $a_1, \ldots, a_n$. Необходимо отсортировать его за $O(n \log n)$ времени и $O(n)$ доппамяти.

MergeSort

Задача. Дан массив $a_1, \ldots, a_n$. Необходимо отсортировать его за $O(n \log n)$ времени и $O(n)$ доппамяти.

Решение.

1. Разделим массив на две равные части.

MergeSort

Задача. Дан массив $a_1, \ldots, a_n$. Необходимо отсортировать его за $O(n \log n)$ времени и $O(n)$ доппамяти.

Решение.

1. Разделим массив на две равные части.
2. Каждую из них рекурсивно отсортируем.

MergeSort

Задача. Дан массив $a_1, \ldots, a_n$. Необходимо отсортировать его за $O(n \log n)$ времени и $O(n)$ доппамяти.

Решение.

1. Разделим массив на две равные части.
2. Каждую из них рекурсивно отсортируем.
3. Результаты сольем в один отсортированный массив.

Инверсии

Def. Пара $i < j$ называется инверсией, если $a_i > a_j$.

Задача. Дан массив $a_1, \ldots, a_n$. Необходимо найти число инверсий за $O(n \log n)$ времени и $O(n)$ доппамяти.

Подсчет числа инверсий

Решение.

1. Если при слиянии элемент левой части больше элемента правой части, то значит это и есть инверсия.

Подсчет числа инверсий

Решение.

1. Если при слиянии элемент левой части больше элемента правой части, то значит это и есть инверсия.
2. Пусть мы сравниваем в сортировке слиянием $l[i]$ и $r[j]$, тогда если $r[j] < l[i]$, то $r[j] < l[i] < l[i+1] < \ldots < l[N]$, то есть число инверсий это $N - i$ для $r[j]$.

Подсчет числа инверсий

Решение.

1. Если при слиянии элемент левой части больше элемента правой части, то значит это и есть инверсия.
2. Пусть мы сравниваем в сортировке слиянием $l[i]$ и $r[j]$, тогда если $r[j] < l[i]$, то $r[j] < l[i] < l[i+1] < \ldots < l[N]$, то есть число инверсий это $N - i$ для $r[j]$.
3. Тогда для одного слияния итоговое число инверсий равно сумме по j таких значений.

Подсчет числа инверсий

Решение.

1. Если при слиянии элемент левой части больше элемента правой части, то значит это и есть инверсия.
2. Пусть мы сравниваем в сортировке слиянием $l[i]$ и $r[j]$, тогда если $r[j] < l[i]$, то $r[j] < l[i] < l[i+1] < \ldots < l[N]$, то есть число инверсий это $N - i$ для $r[j]$.
3. Тогда для одного слияния итоговое число инверсий равно сумме по j таких значений.
4. Итоговый ответ равен сумме по всем слияниям.

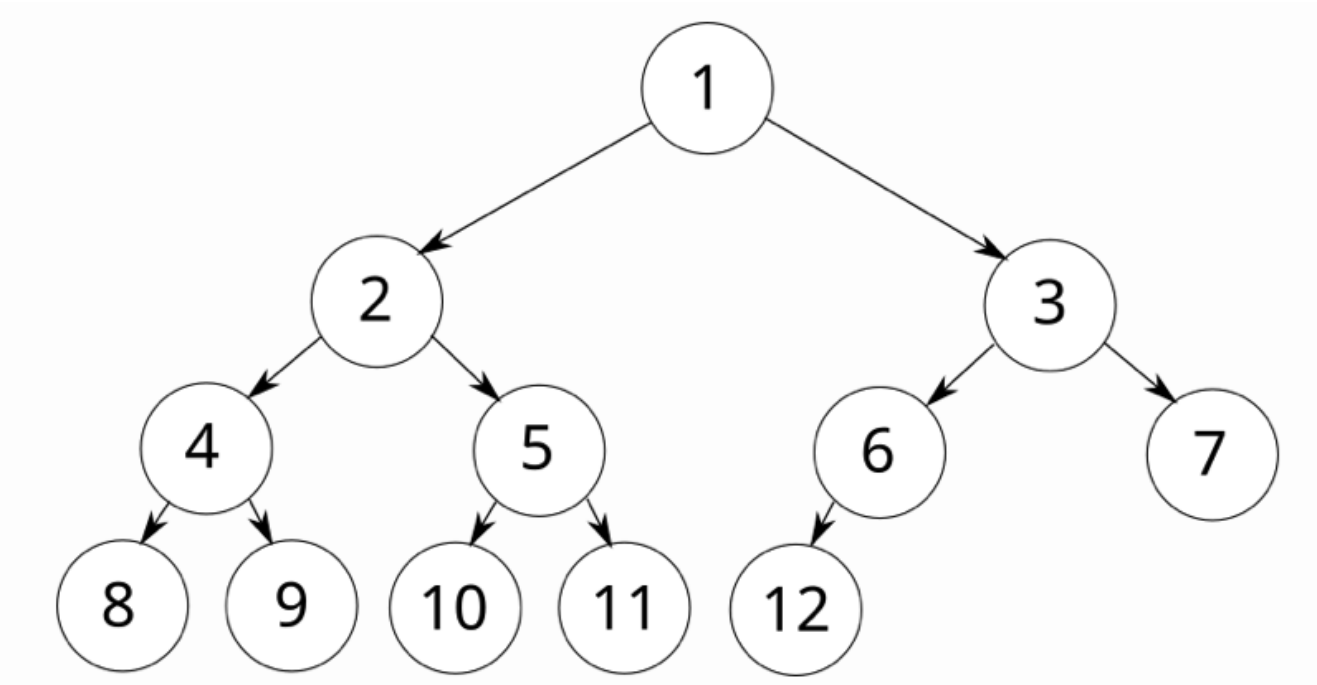
Куча

Def. Куча — структура данных, которая позволяет добавлять в себя элементы и получать/извлекать минимум за логарифмическое время.

Куча

Def. Куча — структура данных, которая позволяет добавлять в себя элементы и получать/извлекать минимум за логарифмическое время.

Def. Бинарная куча — структура данных в виде полного бинарного дерева, для которого верно свойство кучи: все дети больше родителя.



Просеивания

Def. Просеивание элемента вверх — процедура, поднимающая элемент как можно выше, пока не выполняется свойство кучи.

Def. Просеивание элемента вниз — процедура, погружающая элемент как можно ниже, пока не выполняется свойство кучи.

Основные операции

- `GetMin` — вернуть элемент, находящийся в корне.

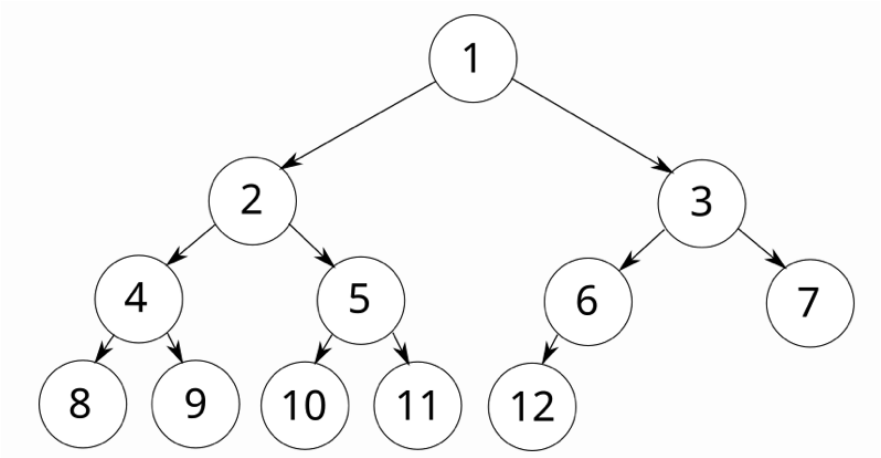
Основные операции

- `GetMin` — вернуть элемент, находящийся в корне.
- `Insert` — подвесить элемент в самое левое свободное место на нижнем уровне и просеять вверх.

Основные операции

- `GetMin` — вернуть элемент, находящийся в корне.
- `Insert` — подвесить элемент в самое левое свободное место на нижнем уровне и просеять вверх.
- `ExtractMin` — поменять местами корневой элемент и самый правый лист, лист отрезать, корень просеять вниз.

Куча как массив



Рассмотрим элементы по уровням

Уровень								
0	1							
1	2	3						
2	4	5	6	7				
3	8	9	10	11	12			

Куча как массив

То есть можно представлять полное бинарное дерево как массив. Для этого надо лишь определиться с индексацией.

Куча как массив

То есть можно представлять полное бинарное дерево как массив. Для этого надо лишь определиться с индексацией.

- Для узла с индексом i родителем является $\lfloor \frac{i-1}{2} \rfloor$.

Куча как массив

То есть можно представлять полное бинарное дерево как массив. Для этого надо лишь определиться с индексацией.

- Для узла с индексом i родителем является $\lfloor \frac{i-1}{2} \rfloor$.
- Для узла с индексом i левым сыном является $2i + 1$.

Куча как массив

То есть можно представлять полное бинарное дерево как массив. Для этого надо лишь определиться с индексацией.

- Для узла с индексом i родителем является $\lfloor \frac{i-1}{2} \rfloor$.
- Для узла с индексом i левым сыном является $2i + 1$.
- Для узла с индексом i правым сыном является $2i + 2$.

Heapify

Def. Heapify — процесс построения кучи на заданном массиве. То есть надо переупорядочить элементы в массиве так, чтобы по индексам на всех парах родитель-сын было соблюдено свойство кучи.

Heapify

Def. Heapify — процесс построения кучи на заданном массиве. То есть надо переупорядочить элементы в массиве так, чтобы по индексам на всех парах родитель-сын было соблюдено свойство кучи.

Решение. Будем просеивать вниз элементы, начиная с $\frac{N}{2}$-го и до нулевого.

Proof. Алгоритм корректен, так как процедура `SiftDown` гарантирует, что после ее выполнения при корректных поддеревьях результат окажется корректным.

Теорема (б/д). Время работы данного алгоритма: $O(n)$.

HeapSort

Задача. Отсортировать массив $a_1, \ldots, a_n$ за $O(n \log n)$ времени и $O(1)$ доппамяти.

Решение.

1. Выполнить Heapify.
2. Пусть `x = GetMin()`, далее надо сделать `ExtractMin()` и вместо последнего элемента выписать x.
3. Повторить это действие $n - 1$ раз.
4. Развернуть массив.