Алгоритмы и структуры данных

Экзамен, 2-й семестр

1 Медиана и порядковые статистики. Алгоритм с линейным временем работы для медианы

2 Сортировка массива: пузырьковая, mergesort, quicksort. Алгоритмы и оценки сложности.

3 Списки: односвзяный и двусвязный

4 Бинарные деревья поиска. Вставка, удаление, оценки сложности.

5 Графы. Способы их представления в памяти компьютера. Матрицы смежности, матрицы инцедентности, списки связности, представление на двух массивах (CSR).

6 Обходы графов. Обход в ширину, обход в глубину.

7 Алгоритмы поиска кратчайших путей. Алгоритм Дейкстры, алгоритм Беллмана-Форда.

8 Остовные деревья. Алгоритмы Прима, Краскала, Борувки.

9 Эвристики для поиска кратчайших путей, алгоритм A*.

10 Потоки в сетях. Максимальный поток и минимальный разрез.]

11 Хэш-функции. Коллизии. Хеш-таблицы. Хэширование. Фильтр Блюма.

12 Предикат поворота. Задача пересечения двух отрезков

13 Выпуклые оболочки, алгоритма Джарвиса, Грэхема и QuickHull.

14 kD- деревья. Окто- и квадро-деревья.

15 Многочлены. Метод Горнера. Умножение Карацубы.

1 Основная теорема для рекуррентных соотношений. Схема доказательства.

Теорема

Пусть $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d)$. Тогда

$$T(n) = \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases},$$

где $a \geq 1$ — количество частей, на которые мы дробим задачу, $b > 1$ — во сколько раз легче становится решить задачу, d — степень сложности входных данных.

Схема доказательства:

Рассмотрим дерево рекурсии данного соотношения. Всего в нем будет $\log_b n$ уровней. На каждом таком уровне, количество детей в дереве будет умножаться на a на уровне i будет a^i потомков. Также известно, что каждый ребенок на уровне i размера $\dfrac{n}{b^i}$. Ребенок размера $\dfrac{n}{b^i}$ требует $O\left(\left(\dfrac{n}{b^i}\right)^d\right)$ дополнительных затрат, поэтому общее количество совершенных действий на уровне i:

$$O\left(a^i \cdot \left(\frac{n}{b^i}\right)^d\right) = O\left(n^d \cdot \left(\frac{a^i}{b^{id}}\right)\right) = O\left(n^d \cdot \left(\frac{a}{b^d}\right)^i\right)$$

Решение разбивается на три случая: когда $\dfrac{a}{b^d}$ больше 1, равна 1 или меньше 1. Переход между этими случаями осуществляется при

$$\frac{a}{b^d} = 1 \Leftrightarrow a = b^d \Leftrightarrow \log_b a = d \cdot \log_b b \Leftrightarrow \log_b a = d$$

Распишем всю работу в течение рекурсивного спуска:

$$T(n) = \sum_{i=0}^{\log_b n} O\left(n^d \left(\frac{a}{b^d}\right)^i\right) + O(1) = O\left(n^d \cdot \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i\right)$$

Отсюда получаем:

1. $d > \log_b a \Rightarrow T(n) = O(n^d)$ (так как $\left(\dfrac{a}{b^d}\right)^i$ — бесконечно убывающая геометрическая прогрессия)

2. $d = \log_b a \Rightarrow T(n) = O\left(n^d \cdot \sum\limits_{i=0}^{\log_b n} \left(\dfrac{a}{b^d}\right)^i\right) =$

   $= O\left(n^d \cdot \sum\limits_{i=0}^{\log_b n} (1)^i\right) = O(n^d + n^d \cdot \log_b n) = O(n^d \cdot \log_b n)$

3. $d < \log_b a \Rightarrow T(n) = O\left(n^d \cdot \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i\right) = O\left(n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n}\right)$, но

$$n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \cdot \left(\frac{a^{\log_b n}}{b^{d \log_b n}}\right) = n^d \cdot \left(\frac{n^{\log_b a}}{n^d}\right) = n^{\log_b a} \Rightarrow T(n) = O(n^{\log_b a})$$

2 АВЛ-деревья. Повороты, балансировка.

АВЛ-дерево - сбалансированное двоичное дерево поиска со следующим свойством: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. Названо в честь изобретателей Г. М. Адельсона-Вельского и Е. М. Ландиса.

Теорема: АВЛ-дерево имеет высоту $h = O(\log n)$.

Доказательство: Высоту поддерева с корнем x будем обозначать как $h(x)$. Пусть m_h - минимальное число вершин в АВЛ-дереве высоты h. Тогда легко видеть, что $m_{h+2} = m_{h+1} + m_h + 1$ по индукции. Равенством $m_h = F_{h+2} - 1$ докажем.

База: $m_1 = F_3 - 1$ - верно, $m_1 = 1$, $F_3 = 2$.

Шаг: Допустим $m_h = F_{h+2} - 1$ - верно. Тогда $m_{h+1} = m_h + m_{h-1} + 1 = F_{h+2} - 1 + F_{h+1} - 1 + 1 = F_{h+3} - 1$.

$F_h = \Omega(\varphi^h)$ где $\varphi = \frac{\sqrt{5}+1}{2}$. То есть $n \geq \varphi^h \Rightarrow \log_\varphi n \geq h$. Высота АВЛ-дерева из n вершин - $O(\log n)$.

Балансировка: Балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев $|h(L) - h(R)| = 2$ изменяет связи предок - потомок в поддереве данной вершины так, чтобы восстановилось свойство дерева $|h(L) - h(R)| = 1$, иначе ничего не меняет. $(diff[i] = h(L) - h(R))$.

Малое вращение: $(O(1))$

- Левое используется когда $h(b) - h(L) = 2$ и $h(c) \leq h(R)$.
- Правое используется, когда $h(a) - h(R) = 2$ и $h(c) \leq h(L)$.

Большое вращение: $(O(1))$ Левое используется когда $h(b) - h(L) = 2$ и $h(c) > h(R)$. Правое используется когда $h(b) - h(R) = 2$ и $h(c) > h(L)$.

Большое вращение состоит из двух малых.

Вставка (insert): Спускаемся по дереву, как при поиске. Если мы стоим в вершине a и там надо идти в поддерево b, то делаем b листом, а вершину a корнем. Поднимаемся вверх по пути поиска и пересчитываем баланс у вершин. Если мы поднялись в вершину i из левого поддерева, то $diff[i] = h(L) - h(R)$ увеличилось на 1. Если из правого - уменьшается на 1. Если пришли в вершину и баланс стал равен 0, то высота не изменилась и подъём останавливается. Если пришли в вершину и её баланс стал равен 1 или -1, то высота поддерева изменилась и подъём продолжается. Если пришли в вершину и её баланс стал равен 2 или -2, то делаем одно из 4 вращений и, если после вращения баланс стал 0, то останавливаемся, иначе продолжаем подъём. Сложность: $O(\log n)$, т.к. в процессе добавления вершины мы рассматриваем не более $O(\log n)$ вершин, и для каждой запускаем балансировку не более одного раза.

Удаление (delete): Если вершина лист, удаляем её. Иначе найдём самую близкую по значению вершину и, переместим её на место удаляемой вершины, а затем удалим эту вершину. От удалённой вершины будем подниматься к корню и пересчитывать баланс у вершин. $diff[]$ уменьшается. Если поднялись из левого поддерева, то $diff[]$ уменьшается на 1. Если из правого - увеличивается на 1. Если пришли в вершину и её баланс стал равен 1 или -1, то высота поддерева не изменилась и подъём можно остановить. Если пришли в вершину и баланс стал равен 0, то высота поддерева уменьшилась и подъём нужно продолжить. Если пришли в вершину и её баланс стал равен 2 или -2, то делаем одно из 4 вращений и, если после вращения баланс стал 0, то подъём продолжается, иначе - останавливается. Аналогично с вставкой: $O(\log n)$.

Поиск (find): Как в обычном бинарном дереве поиска. $O(\log n)$.

3 Красно-черные деревья. Балансировка (схема).

4 Бинарные кучи. Реализация с указателями и на массиве. Добавление и удаление элемента в бинарную кучу.

5 Очереди с приоритетами. Наивная реализация, реализация на бинарной куче.

6 Потоки в сетях. Задача о максимальном потоке. Алгоритм Форда – Фалкерсона.

7 Амортизационный анализ: групповой анализ, банковский метод. Амортизационный анализ для бинарного счётчика

Амортизационный анализ: Метод анализа производительности алгоритмов, учитывающий общую стоимость последовательности операций, а не только стоимость одной. Идея:

- Пересчитать на более скрытые или редкие операции.
- Стоимость операции как бы «размазывается» на все операции, входящие в последовательность.
- Отражает наихудший случай.
- Не учитывает вероятность.

Метод группировки: В случае, когда в наихудшем случае суммарное время выполнения k операций $T_k(N)$ не зависит от N, то амортизированная стоимость каждой операции $A(N) = T_k(N)/k$.

Пример: Бинарный счетчик

Счетчик имеет n бит (например, N значений). Поддерживает операции $Increment$ и $Reset$. Пусть $N = 5$. Изначально счетчик равен 00000. Пример:
0. 00000
1. 00001
2. 00010
3. 00011
4. 00100

За N элементов каждый раз k операций. k - количество операций. Например, для $N = 4$: $000 \to 001$ (1 раз) $001 \to 010$ (2 раза) $010 \to 011$ (1 раз) $011 \to 100$ (3 раза) $\Rightarrow N$ бит менялись k раз.

Тогда амортизированная стоимость $O(k)$, где $k = 2N$. $A(N) = N + \frac{N}{2} + \frac{N}{4} + \cdots + 1 = 2N$. Сложность $O(1)$.

Бухгалтерский метод (взвешивания)

1. На операции тратится столько условных единиц, сколько больше (меньше) реально.

2. Если уж стоит и реально получится, то разность сил и цен на k-ю операцию.

3. Условные деньги вводится так, чтобы их хватало на любые операции. Это для того, чтобы кредит не были отрицательными деньги.

8 Амортизационный анализ: метод потенциалов для динамического массива.

9 Алгоритм Рабина-Карпа, алгоритм Кнута-Морриса-Пратта . Структура данных «бор», алгоритм Ахо-Корасик.

10 Алгоритм Бойера-Мура. Эвристики стоп-символа и хорошего суффикса.

Цель: найти все подстроки str2 в str1.
Суть: базовый алгоритм для поиска:

- Проходимся по всем str1 от i-го до $i + \text{len(str2)} - 1$.
- Посимвольно сравниваем.

Модернизированный алгоритм: обходит некоторое количество символов.

Эвристика стоп-символа:

- Мы спускаемся в рамках i окна слева-направо.
- И находим несовпадение в str2.
- Из того, что мы знаем про str2 (какой символ там должен быть), смотрим на $str1[i+k]$ и $str2[k]$.
- И если $str1[i+k]$ совпадает с символом в $str2$, то сдвигаем окно так, чтобы $str1[i+k]$ совпал с этим символом. Иначе сдвигаем окно на длину str2.

Пример: str1: abcabcab str2: abab
В этом случае сравнения $str1[3]$ и $str2[3]$ дадут "c" и "b" соответственно.
str1 ...c... str2 ...b...
Как это влияет на наши дальнейшие действия? Мы в $str1[3]$ видим "c". Если в $str2$ нет "c", то мы сдвигаем окно на $length(str2)$, начиная с $i + k$.

Эвристика хорошего суффикса:

Мы находим подстроку в str2, которая совпадает с подстрокой в str1, то есть $str1[i + k \dots i + \text{len(str2)} - 1]$ совпадает с $str2[k \dots \text{len(str2)} - 1]$. Это называется "хороший суффикс". Дальше мы ищем повторение "хорошего суффикса" в $str2$.
Пример: str1 абабаб ищем абаб.

```
str1: абабаб
str2: абаб
```

Шаги:

```
str1: абабаб
str2:  абаб
```

1. Во-вторых, мы можем попытаться присоединить str2 к str1 так, чтобы у нас совпали несколько символов.

2. Из этого вытекает: $str1[i \dots i + \text{len(str2)} - 1]$ должен совпадать $str2[0 \dots \text{len(str2)} - 1]$.

Пример: str1: aabbabcaba str2: abab

11 Расстояние Левенштейна, алгоритм Вагнера-Фишера.

12 Сканирующая прямая. Алгоритм Бентли-Оттоманна для поиска пересечения отрезков

13 Диаграммы Вороного. Алгоритм Форчуна.

14 Триангуляция Делоне, связь с диаграммами Вороного. Алгоритм построения

15 Сумма Минковского. Задача планирования движения робота в среде с препятствиями. Граф видимости.

16 Отсечение невидимых поверхностей. Z-буфер и алгоритм художника.