Вопросы к зачоту по структурам

1 Асимптотические оценки времени работы алгоритмов и их смысл

Пусть f, g функции принимающие только положитльные значения

Определение 1 (Тета) $f = \Theta(g) : \exists c_1, c_2 > 0 n_0 : \forall n > n_0 \; c_1 g(n) \leq f(n) \leq c_2 g(n)$

Пример 1.1

$$\frac{n^2}{2} - 3n = \Theta(n^2).$$

Чтоб доказать нужно найти нужные константы

$$c_1 n^2 \leq \frac{n^2}{2} - 3n \leq c_2 n^2.$$

$$c_1 < \frac{1}{2} - \frac{3}{n} \leq c_2.$$

$$c_1 = 1/4.$$

$$c_2 = 1/2.$$

при $n \geq 12$

Пример 1.2

$$c_1 2^n \leq 2^{n+1} \leq c_2 2^n.$$

$$c_1 \leq 2 \leq c_2.$$

Определение 2 (O)

$$f = O(g) : \exists c\, n_0 \forall n > n_0 : f(n) \le cg(n).$$

Определение 3 (Омега)

$$f = \Omega(n) : \exists c, n_0 \forall n > n_0 f(n) \ge cg(n).$$

Определение 4 (о-малое)

$$f = o(g) : \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

2 Теорема о рекурсии

Есть рекурсивный алгоритм. Его время работы зависит от параметра n (размера входных данных)

1. Если $n < k, k = const$ решаем задачу нерекурсивно относительно n
2. иначе разбиваем задачу на a подзадач размера $\frac{n}{b}$ каждую решаем рекрсувно

Время работы такого алгоритма описываем такой формулой

$$T(n) = \begin{cases} g(n), n < k \\ a * T(\frac{n}{b}) + f(n), n \ge k \end{cases}.$$

$g(n)$ время для нерекурсивного решения , $f(n)$ время для определения подзадач и собирания результатов рекурсивных вызовов

Теорема 1 *Пусть время работы рекурсивного алгоритма выражается формулой*

$$T(n) = aT(\frac{n}{b}) + f(n).$$

1. *Если* $f(n) = O(n^c), c < \log_b a$ *то* $T(n) = \Theta(n^{\log_b a})$
2. *Если* $f(n) = \Theta(n^c(\log_b a)^k$ *для* $k \ge 0, c = \log_b a$*, то* $T(n) = \Theta(n^c(\log n)^{k+1})$

3. $f(n) = \Omega(n^c) c > \log_b a$, *то* $T(n) = \Theta(f(n))$ *для это еще должно быть*

$$af(\frac{n}{b}) < kf(n) \ k < 1.$$

Частный случай при $a = b$

$$T(n) = aT(\frac{n}{a}) + f(n).$$

1. *Если* $f(n) = O(n^c), c < 1$ *то* $T(n) = \Theta(n)$
2. *Если* $f(n) = \Theta(n)$ *то* $T(n) = \Theta(n \log n)$
3. *Если* $f(n) = \Omega(n^c), c > 1$, *То* $T(n) = \Theta(f(n))$

Пример 1.1

$$T(n) = 9T(\frac{n}{3}) + n.$$

$$\log_b a = 2.$$

$$n = O(n) \iff T(n) = \Theta(n^2).$$

Пример 1.2

$$T(n) = T(\frac{2n}{3}) + 1.$$

$$f(n) = 1.$$

$$a = 1.$$

$$b = \frac{3}{2}.$$

$$\log_{3/2} 1 = 0.$$

3 Сортировка слиянием

Оценка времени работы $T(n) = 2T(\frac{n}{2}) + n$

$$n = \Theta(n).$$

По теореме $T = \Theta(\log n)$

4 Длинная арифметика

Число представляем как список int, каждое число меньше некого числа, которое помешается в int. $a = \{a_0, a_1, \ldots, a_{n-1}\}, \forall i < n-1\ a_i < b$

$$a = \sum_{i=0}^{n-1} a_i b^i.$$

За b удобно брать большую степень 10 , например миллион. Еще удобнее брать большую степень двух как пример 2^{31}

4.1 Сложение

Сложение двух длинных чисел происходит точно так как в столибик. Для сложения нужно сделать n элементарных операций, где n количесвто цифр самого длинного числа.

4.2 Деление с остатотком

Опять алгоритм деления в столбик

4.3 Умножение

Если умножать в столбик, придется сделать n^2 элементарных операций

4.3.1 Алгоритм карацубы

Пусть есть числа a, b длины n, c основание системы счисления

$$x = c^{n/2}.$$

$$a = \alpha_1 x + \alpha_2.$$

$$b = \beta_1 x + \beta_2.$$

$$ab = \alpha_1 \beta x^2 + \alpha_1 \beta_2 x + \alpha_2 \beta_1 x + \alpha_2 \beta_2 = \alpha_1 \beta_1 x^2 + (\alpha_1 \beta_2 + \alpha_2 \beta_1) x + \alpha_2 \beta_2.$$

$$T(n) = 4T(n/2) + O(n).$$

$$\log_2 4 = 2.$$

$$T(n) = \Theta(n^2).$$

$$\alpha_1 \beta_1 x^2 + (\alpha_1 \beta_2 + \alpha_2 \beta_1) x + \alpha_2 \beta_2 = \alpha_1 \beta_1 x^2 + ((\alpha_1 + \beta_1)(\alpha_2 + \beta_2) - \alpha_1 \beta_1 - \alpha_2 \beta_2) + \alpha_2 \beta_2.$$

$$T(n) = 3T(\frac{n}{2}) + O(n).$$

5 Алгоритм рабина карпа

Есть строка, есть подстрока, есть хешфункция, все хеши подстрок нужной длины сранвивем с хешем искомой, если равны, сравниваем посимвольно. Для опитимизации, надо сделать хорошую хеш функцию, чтоб было малок коллизий, чтобы она например зависела от позиций.

6 Грамматики,Регулярные выражения

Можно определить язык через регулярные выражения. Рассмотрим регулярки в джаве. В джаве есть класс Pattern и класс Mathcher. Pattern содержит компилированны выражения, Mathcer ищут в тексте штуки по регулярке.

Рассмотрим задачу, есть строка s, нужно проверить подходит ли под регулярку

```
s.mathcher("kjdskjdsk");
```

такая фигня возвращает boolean