

# Разработка и анализ алгоритмов

## Лекция 5

Поиск. Жадные алгоритмы.

Сергей Леонидович Бабичев

# Абстракция хранилище.

# Абстракция хранилище

- Хранилище содержит *ключи и значения*.
- Помимо операций создания и уничтожения хранилища реализуются операции:
  - ▶ create — создать новую пару;
  - ▶ read — найти пару по ключу;
  - ▶ update — изменить значение по ключу;
  - ▶ delete — удалить пару по ключу.

Структуры данных, реализующие эту абстракцию называются CRUD-структурами.

# Задача поиска. Абстракция поиска.

# Задача поиска. Абстракция поиска

Информация нужна для того, чтобы ей пользоваться.

Расширенная задача поиска:

- 1 Накопление информации (сбор)
- 2 Организация информации (переупорядочивание, сортировка)
- 3 Извлечение информации (собственно поиск)

# Расширенная задача поиска

- Задача: построение эффективного хранилища данных.
- Требования:
  - ▶ Поддержка больших объёмов информации.
  - ▶ Возможность быстро находить данные.
  - ▶ Возможность быстро модифицировать данные.
- Реализация абстракций:
  - ▶ *Create*
  - ▶ *Read*
  - ▶ *Update*
  - ▶ *Delete*

# Задача поиска. Абстракция поиска

- Имеется множество ключей

$$a_1, a_2, \dots, a_n$$

- Требуется определить индекс ключа, совпадающего с заданным значением  $key$ , `find` — основа для `Read/Update/Delete`.

```
bunch a;  
index = find(a, key);
```

*bunch* — абстрактное хранилище элементов, содержащих ключи (массив, множество, дерево, список...).

**Хорошая организация хранилища входит в расширенную задачу поиска.**

# Последовательный поиск.



# Последовательный поиск

Ситуация: к поиску не готовились, ключи не упорядочены.

Индекс	0	1	2	3	4	5	6	7	8	9
Ключ	132	612	232	890	161	222	123	861	120	330
Данные	АВ	СА	ЯФ	АВ	АА	НД	ОР	ОС	ЗЛ	УГ

$\text{find}(a, 222) = 5$

$\text{find}(a, 999) = 10$  (элемент за границей поиска).

## Последовательный поиск

```
int dummysearch(int *a, int N, int key) {
    for (int i = 0; i < N; i++) {
        if (a[i] == key) {
            return i;
        }
    }
    return N;
}
```

Вероятность найти ключ в  $i$ -м элементе  $P_i = \frac{1}{N}$

Матожидание числа поисков  $E = \frac{N}{2}$

Число операций сравнения в худшем случае  $2N$ .

$$T(N) = 2 \cdot N = O(N)$$

# Последовательный поиск

Небольшая подготовка:

Индекс	0	1	2	3	4	5	6	7	8	9	10
Ключ	132	612	232	890	161	222	123	861	120	330	999
Данные	AB	CA	ЯФ	AB	AA	НД	ОР	ОС	ЗЛ	УГ	??

Результаты не изменились.

$\text{find}(a, 222) = 5$

$\text{find}(a, 999) = 10$  (элемент за границей поиска).

# Последовательный поиск

```
int cleversearch(int *a, int N, int key) {  
    // Required: capacity of a >= N+1  
    a[n] = key;  
    int i;  
    for (i = 0; a[i] != key; i++)  
        ;  
    return i;  
}
```

Число операций сравнения  $N$  в худшем случае.

$$T(N) = N = O(N)$$

Поиск ускорен в два раза!

**Без подготовки лучших результатов не добиться.**

# Неупорядоченный массив

- Сложность операций:
  - ▶ *Create* —  $O(1)$
  - ▶ *Read* —  $O(N)$
  - ▶ *Update* —  $O(N)$
  - ▶ *Delete* —  $O(N)$

# Поиск с сужением зоны.

# Поиск с сужением зоны

Если в зоне поиска имеется упорядочивание — всё становится значительно лучше.  
Возможное действие: упорядочить по отношению.

- Имеется множество ключей

$$a_1 \leq a_2 \leq \dots \leq a_n$$

- Требуется определить индекс ключа, совпадающего с заданным значением *key*.

# Поиск с сужением зоны

Принцип «разделяй и властвуй».

- 1 Искомый элемент равен центральному? Да — нашли.
- 2 Искомый элемент меньше центрального? Да — рекурсивный поиск в левой половине.
- 3 Искомый элемент больше центрального? Да — рекурсивный поиск в правой половине.



## Поиск с сужением зоны

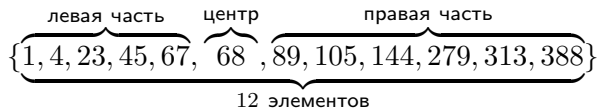
- Вход алгоритма: упорядоченный по возрастанию массив, левая граница поиска, правая граница поиска.
- Выход алгоритма: номер найденного элемента или -1.

```
int binarySearch(int val, int a[], int left, int right) {
    if (left >= right) return a[left] == val? left : -1;
    int mid = (left+right)/2;
    if (a[mid] == val) return mid;
    if (a[mid] > val) {
        return binarySearch(val, a, left, mid-1);
    } else {
        return binarySearch(val, a, mid+1, right);
    }
}
```

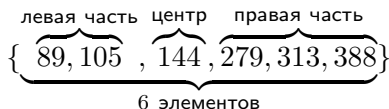
# Поиск с сужением зоны

Оценка глубины рекурсии.

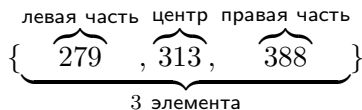
Поиск ключа 313:



$313 > 68 \rightarrow$  ключ справа



$313 > 144 \rightarrow$  ключ справа



$313 = 313 \rightarrow$  ключ найден

# Поиск с сужением зоны

Попрактикуемся в мастер-теореме для вывода и так очевидного результата.

- Количество подзадач  $a = 1$ .
- Каждая подзадача уменьшается в  $b = 2$  раза.
- Сложность консолидации  $O(1) = O(N^0) \rightarrow d = 0$

$$d = \log_b a \rightarrow T(N) = \log N$$

## Поиск с сужением зоны

Переход от рекурсии к итерации.

```
int binarySearch(int val, int a[], int left, int right) {
    while (left < right) {
        int mid = (left + right)/2;
        if (a[mid] == val) return mid;
        if (a[mid] > val) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return a[left] == val? left : -1;
}
```

Распределяющий поиск. Поиск с использованием свойств ключа.

# Распределяющий поиск

- Можно ли найти ключ в неотсортированном массиве быстрее, чем за  $O(N)$ ?

# Распределяющий поиск

- Можно ли найти ключ в неотсортированном массиве быстрее, чем за  $O(N)$ ?
- Без предварительной подготовки и вспомогательных данных — нет.

# Распределяющий поиск

- Можно ли найти ключ в неотсортированном массиве быстрее, чем за  $O(N)$ ?
- Без предварительной подготовки и вспомогательных данных — нет.
- Нужно найти  $M$  раз значение в неотсортированном массиве размером  $N$ .  
Какова сложность алгоритма?



# Распределяющий поиск

- Можно ли найти ключ в неотсортированном массиве быстрее, чем за  $O(N)$ ?
- Без предварительной подготовки и вспомогательных данных — нет.
- Нужно найти  $M$  раз значение в неотсортированном массиве размером  $N$ .  
Какова сложность алгоритма?
- Вариант ответа: если  $M > \log N$ , то предварительной сортировкой.  
Сложность составит  $O(N \log N) + M \cdot O(\log N) = O(N \log N)$ .
- Можно и быстрее.

# Распределяющий поиск

- Если  $|D(Key)|$  невелико, то имеется способ, похожий на сортировку подсчётом.
- Создаётся **инвертированный массив**.

$$a = \{2, 7, 5, 3, 8, 6, 3, 9, 12\}. |D(a)| = 12 - 2 + 1 = 11.$$

$$a_{inv}[2..12] = \{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1\}$$

$$a[0] = 2 \rightarrow a_{inv}[2] = 0$$

$$a[1] = 7 \rightarrow a_{inv}[7] = 1$$

$$a[2] = 5 \rightarrow a_{inv}[5] = 2$$

$$a_{inv}[2..12] = \{0, 6, -1, -1, 5, -1, 4, 7, -1, -1, 8\}$$

# Распределяющий поиск

index	0	1	2	3	4	5	6	7	8
key	2	7	5	3	8	6	3	9	12

key	2	7	5	3	8	6	3	9	12
index	0	1	2	3	4	5	6	7	8

# Распределяющий поиск

Два этапа. Первый этап — инвертирование.

```
int * prepare(int *a, int N, int *min, int *max) {
    *min = *max = a[0];
    for (int i = 1; i < N; i++) {
        if (a[i] > *max) *max = a[i];
        if (a[i] < *min) *min = a[i];
    }
    if (*max - *min > THRESHOLD) return NULL;
    int *ret = malloc(sizeof(int)*(*max - *min + 1));
    for (int i = *min; i <= *max; i++) {
        ret[i] = -1;
    }
    for (int i = 0; i < N; i++) {
        ret[a[i] - *min] = i;
    }
    return ret;
}
```

# Распределяющий поиск

Второй этап: поиск.

```
// Preparation
int min, max;
int *ainv = prepare(a, N, &min, &max);
// Where is the key?
result = -1; // Not found value
if (key >= min && key <= max) result = ainv[key - min];
...
free(ainv);
```

- $O(N)$  на подготовку.
- $O(M)$  на поиск  $M$  элементов.
- $T(N, M) = O(N) + O(M) = O(N)$

# Распределяющий поиск

- Сложность операций после подготовки:
  - ▶ *Create* —  $O(1)$
  - ▶ *Read* —  $O(1)$
  - ▶ *Update* —  $O(1)$
  - ▶ *Delete* —  $O(1)$
- Жёсткие ограничения на множество ключей.
- Сложность по памяти —  $O(N) + O(|E|)$
- Запрет на повторяющиеся ключи.
- При наличии  $f(key)$  сводится к хеш-поиску.

# Жадные алгоритмы

# Задача о резервных копиях

**Задача 1.** Имеется распределённая система, состоящая из  $N$  хранилищ различной ёмкости, причём в  $i$ -м хранилище можно разместить  $A_i$  блоков информации.

Хранение одного блока считается надёжным, если имеется две его копии в различных хранилищах. Требуется определить наибольшее количество надёжных блоков, которое можно разместить во всех хранилищах.



## Задача о резервных копиях: пример

4 хранилища размерами (8, 7, 4, 3).

8	7	4	3
1	1	2	2
3	3	4	4
5	5	6	6
7	7	-	-
8	8	-	-
9	9	-	-
10	10	-	-
11	-	11	-

## Задача о резервных копиях

Сведём задачу к эквивалентной: имеется  $N$  кучек камней, каждым ходом можно выбрать по одинаковому количеству камней из любой пары кучек. Найти такой порядок игры, при котором останется минимальное количество камней.

- Это же ним-задача!
- Но мы не знаем игру Шпрага-Гранди и попробуем решить жадным алгоритмом.
- Будем считать позицию  $P_1$  более простой, чем позиция  $P_2$ , если позиция  $P_2$  может возникнуть после конечного количества ходов из позиции  $P_1$ .
- Назовём *стратегией* детерминированное действие, зависящее от позиции, и приводящее к более простой позиции.
- Например, стратегией является «выбрать две наибольшие кучи и взять оттуда максимально возможное количество камней».
- Задачу можно решать многими стратегиями.
- Какая из стратегий оптимальная?

# Решение жадным алгоритмом

- Применение выбранной стратегии на каждом шаге создаёт более простую позицию.
- Стратегия в новой позиции не меняется.
- Такая стратегия есть *локально-оптимальный* алгоритм.
- Применение выбранного локально-оптимального алгоритма на протяжении решения всей задачи есть *жадный алгоритм*.

# Задача о резервных копиях: стратегия 1

Первая стратегия: выбрать две наименьших кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
8	7	1	0
8	6	0	0
2	0	0	0

Решение неверное.

## Задача о резервных копиях: стратегия 2

Вторая стратегия: выбрать наибольшую и наименьшую кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
5	7	4	0
5	3	0	0
2	0	0	0

Опять не повезло.

## Задача о резервных копиях: стратегия 3

Третья стратегия: выбрать две наибольших кучи и взять из них одинаковое наибольшее количество камней.

8	7	4	3
1	0	4	3
1	0	1	0
0	0	0	0

Повезло. Как обычно в задачах на жадность: нужен контрпример, *зелёная ворона*.

## Задача о резервных копиях: стратегия 3

Новая попытка:

8	7	7	6	5
1	0	7	6	5
1	0	1	0	5
0	0	1	0	4
0	0	0	0	3

И снова неудача. Есть ли вообще нужная стратегия?

## Задача о резервных копиях: стратегия 4

Четвёртая стратегия: выбрать наибольшую и наименьшую кучи, и взять из них по одному камню.

8	7	4	3
7	7	4	2
7	6	4	1
6	6	4	0
5	6	3	0
5	5	2	0
4	5	1	0
4	4	0	0
0	0	0	0



## Задача о резервных копиях: стратегия 4

8	7	7	6	5
7	7	7	6	4
7	7	6	6	3
7	6	6	6	2
6	6	6	6	1
6	6	6	5	0
6	6	5	4	0
6	5	5	3	0
5	5	5	2	0
5	5	4	1	0
5	4	4	0	0
4	4	3	0	0
4	3	2	0	0
3	3	1	0	0
3	2	0	0	0
1	0	0	0	0

# Задача о резервных копиях: корректность алгоритма

- Очевидно, что на каждой «большой» итерации наименьшая из куч опустошается, так как из неё по условиям алгоритма всегда производится взятие.
- Когда останется три кучи  $A \geq B \geq C$ , то возможны следующие ситуации:
  - ▶  $A > B + C$ . Тогда ответ:  $A - B - C$ . Так как на каждом ходе  $A$  оставалось наибольшим, на каждом их ходов, приведшем к позиции происходило вычитание из  $A$ , это значит, что  $A > \sum$  всех оставшихся, что даёт верное решение.
  - ▶  $A = B + C$ . Тогда результат равен нулю.
  - ▶  $A < B + C$ . Тогда, после некоторой, возможно нулевой последовательности ходов достигается ситуация, когда  $A' = B' > C'$ , которая сведётся к позиции  $A' - \left\lfloor \frac{C'}{2} \right\rfloor, A' - \left\lfloor \frac{C'}{2} \right\rfloor$ .

# Задача о распределении работ

**Задача 2.** Имеется список работ. Каждая работа выполняется ровно один день, должна быть выполнена строго до заданного времени и приносит заданное количество прибыли.

Номер	Дедлайн	Прибыль
1	5	19
2	2	16
3	3	33
4	1	20
5	3	45
6	2	70

Как решать такую задачу?

# Задача о распределении работ

- Предлагается следующий вариант:

- 1 Выбирается самый большой по прибыли элемент.
- 2 Он добавляется в очередь.
- 3 Если всё ещё имеются заказы — идём к шагу (1).
- 4 В полученной очереди берём первый элемент очереди.
- 5 Если на этот день или любой из ранних дней работа не запланирована — планируем на самый поздний день, иначе — пропускаем.
- 6 Если в очереди остались элементы, идём к 4.
- 7 Конец алгоритма.

# Задача о распределении работ

- Это — опять жадный алгоритм.
- На каждом шаге принимается локально-оптимальное решение.
- Мы должны иметь *безопасный шаг* выбора очередного элемента.
- После каждого шага *структура задачи* остаётся подобной предыдущей.
- Если очередной шаг остаётся безопасным, задача решается.
- Стратегии в задаче о резервных копиях есть попытки выбора безопасных шагов.

# Задача о банкомате

**Задача 3.** В банкомате имеется неограниченное количество купюр номиналами  $a_1, a_2, \dots$ . Нужно выдать точную сумму  $x$  наименьшим количеством банкнот.

- Имеется жадный алгоритм.

1 Очередной шаг: выдаём наибольшую из возможных купюр.

2 Если выдать не удалось, то конец алгоритма с неудачей.

3 Если полная сумма выдана, то конец алгоритма с успехом.

4 Идём к шагу 1.

- Пусть имеются купюры номиналом  $\{10, 6, 1\}$ .

● Оптимальное решение для 12:  $12 = 6 + 6$ . Алгоритм даст  $12 = 10 + 1 + 1$ .

● Для набора  $\{10, 5, 3, 1\}$  алгоритм всегда будет давать точное решение.

Почему какие-то задачи допускают точное жадное решение, а какие-то — нет?

- Не имеют жадного решения:  $\{10, 6, 1\}$ ,  $\{10, 7, 2, 1\}$ ,  $\{4, 3, 1\}$ .
- Имеют жадное решение:  $\{10, 5, 3, 1\}$ ,  $\{4, 2, 1\}$ .
- Гипотезы?

- Не имеют жадного решения:  $\{10, 6, 1\}$ ,  $\{10, 7, 2, 1\}$ ,  $\{4, 3, 1\}$ .
- Имеют жадное решение:  $\{10, 5, 3, 1\}$ ,  $\{4, 2, 1\}$ .
- Гипотезы?
- Не имеют жадного решения:  $\{9, 4, 1\}$ ,  $\{25, 15, 1\}$ .
- Имеют жадное решение:  $\{9, 5, 1\}$ ,  $\{25, 15, 10, 5, 1\}$ .
- Гипотезы?



# Алгоритм: как найти алгоритм?

- Начинаем с первого и, если он подходит, отлично.
  - 1 Наивное решение.
  - 2 Жадный алгоритм.
  - 3 Разделяй и властвуй. Рекурсия.
  - 4 Динамическое программирование.
  - 5 Что-то специфическое.

Жадные алгоритмы, которые нам предстоят: алгоритмы Хаффмена, Куна, Прима, Краскала, Дейкстры, Форда-Фалкерсона, Эдмондса-Карпа, Z- и префикс-функции, Ахо-Корасик, Укконена, ...

Для многих придётся использовать готовые или строить новые структуры данных. Но они остаются жадными.

Все жадные алгоритмы допускают математическое представление в виде *матроида*.

# Матроиды

# Матроиды

## Definition (Матроид)

Матроид  $\mathbb{I}$  есть система множеств, обладающая свойствами:

1. Пустое множество принадлежит  $I$ .
2. Если какое-то множество  $S$  принадлежит  $\mathbb{I}$ , то все его подмножества принадлежат  $\mathbb{I}$ .
3. Если для множеств  $S_1$  и  $S_2$   $|S_1| > |S_2|$ , то существует такой элемент  $e \in S_1$ , что  $S_2 \cup \{e\} \in I$ .
- 3'. Вариант формулировки: то существует такой элемент  $e \in S_1 - S_2$ , что  $S_2 \cup \{e\} \in I$ .

## Definition (Носитель матроида)

Все множества, входящие в матроид, состоят из элементов *носителя* матроида.

## Definition (База матроида)

База матроида  $\mathbb{I}$  — максимальное по включению множество, входящее в матроид  $\mathbb{I}$

# Матроиды и графы

- Матроид  $\mathbb{I}$  представляет граф  $G = (V, E)$ , если:
  1. множество носителей есть множество рёбер  $E$ .
  2. Если  $E_1 \subset E$ , то  $A \in \mathbb{I}$  тогда и только тогда, когда  $E_1$  — ациклическое.

# Алгоритм Радо-Эдмондса

- Пусть имеется матроид  $\mathbb{I}$  с носителем  $X$ .
- Пусть каждому элементу  $x_i \in X$  сопоставлен вес  $w_i$ .
- Пусть вес подмножества есть сумма весов элементов.
- Тогда алгоритм Радо-Эдмондса находит в  $\mathbb{I}$  *базу минимального веса*, выбирая на каждом шаге элемент с наименьшим весом.
- $A_0 = \emptyset$
- $A_i = A_{i-1} + \{x\}$ , где  $x = \min_{w_j} (y_j \in X - A_{i-1} : A_{i-1} + \{y_j\} \in I)$ .

# Решение задачи о распределении работ

- Назовём множество работ *выполнимым*, если все работы можно выполнить без просрочки.
- Система выполнимых множеств является матроидом так как выполняются пункты:
  1. Пустое выполнимое множество существует.
  2. Любое подмножество выполнимого подмножества принадлежит  $\mathbb{I}$ .
  3. Если существуют выполнимые множества  $S_1$  и  $S_2$  такие, что  $|S_1| > |S_2|$ , то, действительно, если элемент  $e$ , который принадлежит  $S_1$  добавить в  $S_2$ , то итоговое множество будет принадлежать  $\mathbb{I}$ .

# Алгоритм Хаффмена.

# Сжатие информации: алгоритм Хаффмана

**Задача 4.** Имеется текст, состоящий из символов. Закодировать его таким образом, чтобы:

- каждый из встречающихся символов получил свой двоичный код
- множество кодов было префиксным
- суммарная длина всех кодов для всех символов была бы минимальной.



# Алгоритм Хаффмана

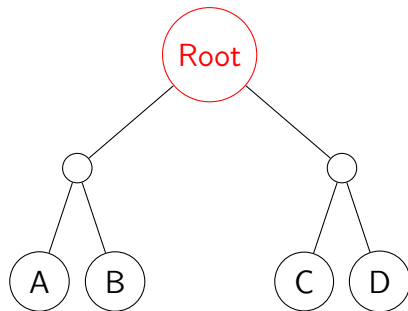
- Пусть имеется текст, состоящий из множества из четырёх символов:  
AAAABAABABABABCBSCAAAD
- Его длина — 21 символ.
- Один из возможных кодов:
  - ▶ A → 00
  - ▶ B → 01
  - ▶ C → 10
  - ▶ D → 11
- Всего 42 бита на кодирование текста.

# Префиксный код

- Неформальное определение: код, в котором не имеется кодовых слов, начинающихся с других кодовых слов.
- Не префиксный код:
  - ▶ A → 00
  - ▶ B → 10
  - ▶ C → 01
  - ▶ D → 101
- Ещё одно название — код, удовлетворяющий *условиям Фано*.
- Наша задача — найти *минимальный префиксный код* для множества.

# Кодирование с помощью дерева

- A → 00
- B → 01
- C → 10
- D → 11



# Алгоритм Хаффмана

- AAAABAABABABABCBСАААD
- Определим частоты символов:
  - ▶  $F_A = 12$
  - ▶  $F_B = 6$
  - ▶  $F_C = 2$
  - ▶  $F_D = 1$
- Нужно построить дерево, вес которого минимален.

$$\sum_{i=1}^n F_i \cdot L_i \rightarrow \min,$$

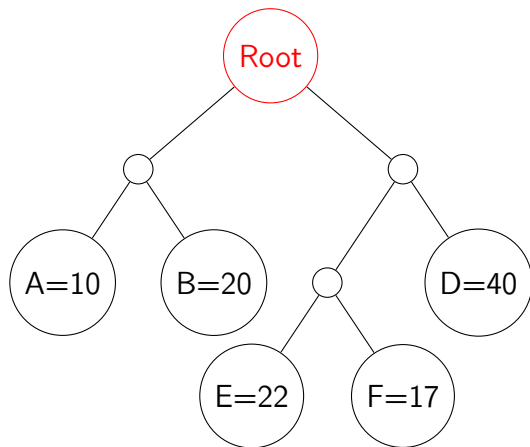
где  $L_i$  - глубина  $i$ -го символа.

# Алгоритм Хаффмана

Свойства оптимального дерева:

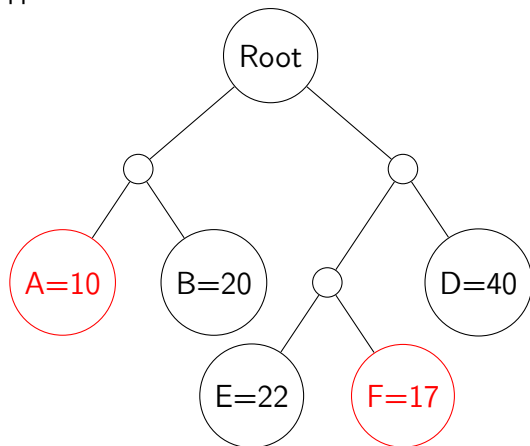
- Из каждого узла должно исходить ровно два пути.
- Не должно быть пустых вершин.
- Самое длинное кодовое слово должно быть парным.

# Алгоритм Хаффмана



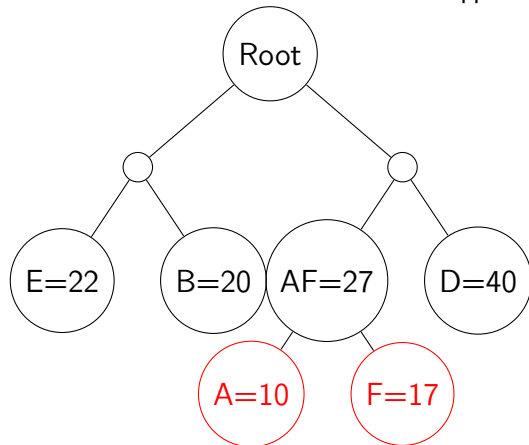
# Алгоритм Хаффмана

Находим два самых редких символа.



# Алгоритм Хаффмана

Два самых редких символа должны находиться на самой длинной ветке. Если нет, то можно их поменять местами с символами на самой длинной ветке.



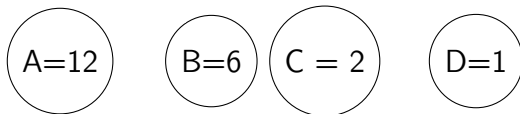


# Алгоритм Хаффмана

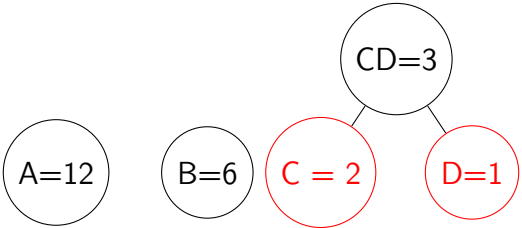
Жадный алгоритм:

1. Помещаем в каждый узел частоту символа
2. Располагаем узлы согласно убыванию частот.
3. Для двух узлов с наименьшей частотой добавляем узел, который их соединяет.
4. В узел помещаем сумму частот детей
5. Помечаем узлы или вершины, как уже обработанные (отправляем вниз)
6. Если необработанных не осталось, то конец алгоритма
7. Переходим к шагу 2.

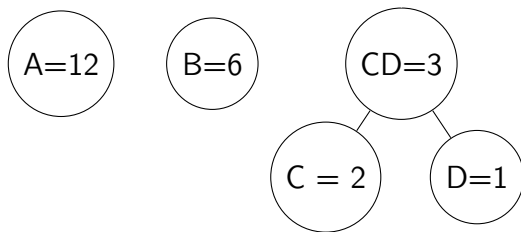
После шага 1



Шаги 2,3,4

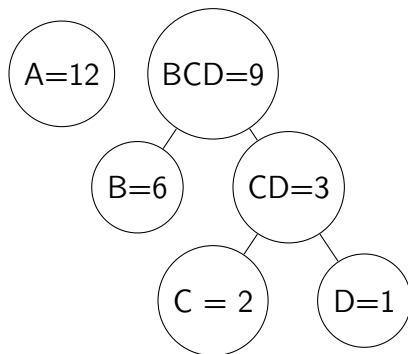


## Шаг 5



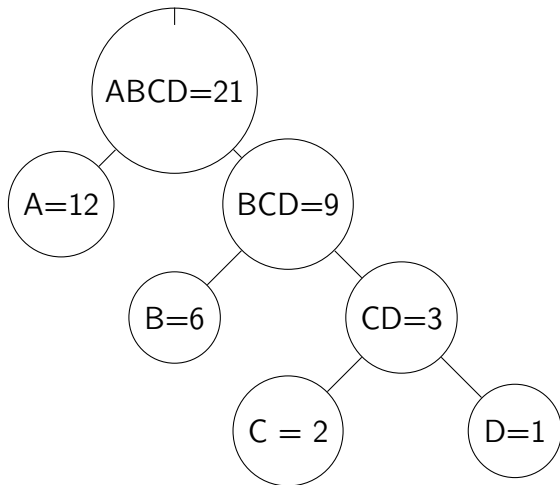
# Алгоритм Хаффмана

Следующая итерация:



# Алгоритм Хаффмана

Заключительное состояние:



# Алгоритм Хаффмана

Получившиеся коды:

- $A \rightarrow 0$
- $B \rightarrow 10$
- $C \rightarrow 110$
- $D \rightarrow 111$

Общая длина всех кодовых слов  $12 \cdot 1 + 6 \cdot 2 + 2 \cdot 3 + 1 \cdot 3 = 33 < 42$