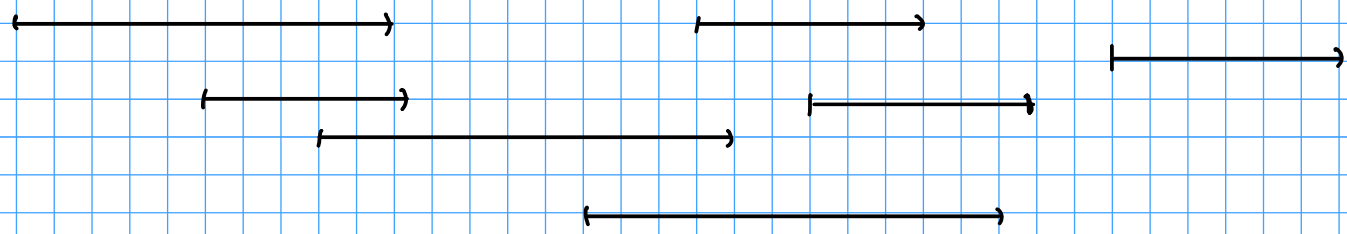
Жадные алгоритмы

Задача о выборе заявок

Вход: Набор заявок $\{[s_i, f_i)\}_{i=1}^{n}$

Задача: Удовлетворить max число заявок.



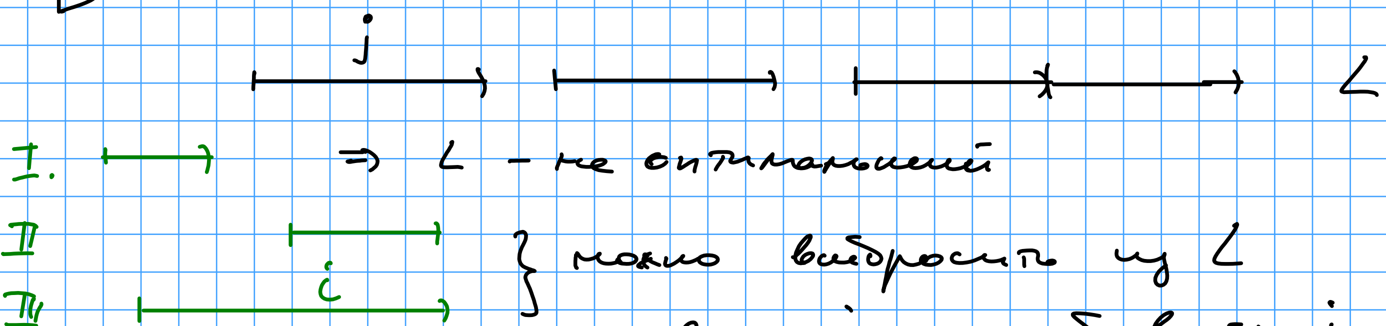
Алгоритм:

Отсортируем по f_i

Будем жадно выбирать мн-во заявок

Утв.: $\exists L$ — оптимальный набор размера $k \Rightarrow \exists L'$ — оптимальный набор, в который входит заявка с min f_i

$\triangleright$



I. $\Rightarrow L$ — не оптимальный

II, III } можно выбросить из L заявку j и добавить i

$$L' = (L \setminus \{j\}) \cup \{i\}$$

$\triangleleft$

Выполнимость хорновских формул

= Булевые формулы вида

1. $(\bar{x} \vee \bar{y} \vee \bar{z})$ дизъюнкция с отриц. лит.
2. $(x \wedge y) \Rightarrow z$ импликация, где посылка — конъюнкция положит. лит., а следствие — положит. литерал

$A \Rightarrow B \iff \neg A \vee B$

$2'.$ $(\overline{x} \vee \overline{y} \vee z)$

в т.г. $\Rightarrow z$

Вход: набор $\{\varphi_i\}_{i=1}^{n}$ – хорновские формулы

Выход: выполняющий набор α:

$\forall i: \varphi(\alpha) = 1$

Алгоритм:

Все переменные установить в false

Пока есть невыполненные импликации

взять первую и выполнить

(т.е. установить переменную из правой части в true)

Если получившийся набор – выполняющий

$\Rightarrow$ вернуть его

Если нет $\Rightarrow$ $\{\varphi_i\}_{i=1}^{n}$ – невыполнимо

Утв: Если алгоритм устанавливает

$x \leftarrow true \Rightarrow x = true$ в $\forall$ выполн. наборе

$\triangleright$ по индукции

База: $\Rightarrow z$

Переход: $\underbrace{(x \wedge \ldots \wedge y)}_{} \Rightarrow z$

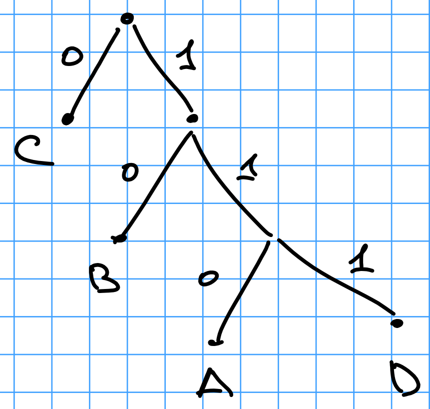
$= true$ в $\forall$ выпол. наборе $\Rightarrow z = true$ в $\forall$. в.н.

$\triangleleft$

Кодирование Хаффмана

prefix-free кодирование

Σ	f_i
A	20
B	30
C	40
D	5



Код A = 110

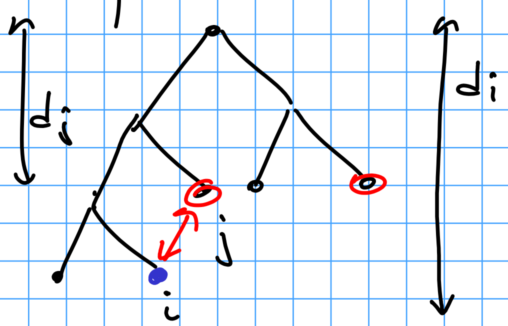
Задача: $\sum\limits_{a_i \in \Sigma} f_i \cdot |\text{код } a_i| \to \min$

$$\sum f_i \cdot d_i \to \min$$

Утв: В оптимальном дереве на нижнем уровне ≥ 2 вершины.

Утв: Вершины с min частотами висят на нижнем уровне

Д от обратного



$f_j < f_i$

$d_j < d_i$

Целевая функция уменьшится:

$$\ldots\ d_j \cdot f_j + d_i \cdot f_i \ldots \longrightarrow$$

$$\ldots\ d_j \cdot f_i + d_i \cdot f_j \ldots$$

Δ целевой функции: $\underline{d_j}\underbrace{(f_i - f_j)}_{>0} - \underline{d_i}\underbrace{(f_i - f_j)}_{>0} < 0$ ◁

Похожая задача: $a_1 \le a_2 \le a_3 \dots \le a_n$

$b_1 \le b_2 \le b_3 \dots \le b_n$

Разбить на пары:

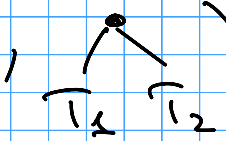
$$\sum a_i \cdot b_j \to \min$$

Алгоритм построения дерева Хаффмена:

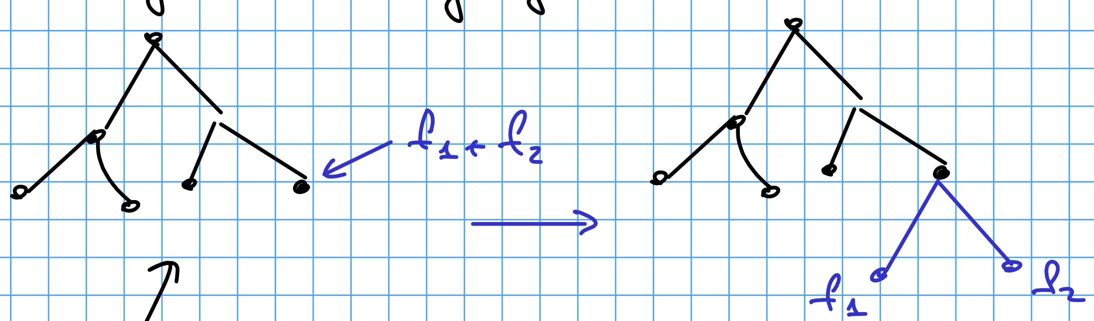
```
P ← make_priority_queue()
for i = 1 to n:
    P.insert((f_i, i))

while P.size() > 1:
    (a, T_1) = P.extract_min()
    (b, T_2) = P.extract_min()
    P.insert((a + b, <tree>))

return P.extract_min()
```



Утв. Если дерево для: $(f_1+f_2), f_3, f_4 \dots f_n$ — оптимально $\Rightarrow$ из него можно получить дерево оптимальное для исходной задачи



$$C' = \sum_{i=1}^{n-1} f'_i \cdot d'_i \to \min \qquad C = \sum_{i=1}^{n} f_i \cdot d_i \to \min$$

$$f'_1 = f_1 + f_2 \qquad C' - C = f_1 + f_2$$

$$\sum f_i \cdot d_i = \sum f(n)$$

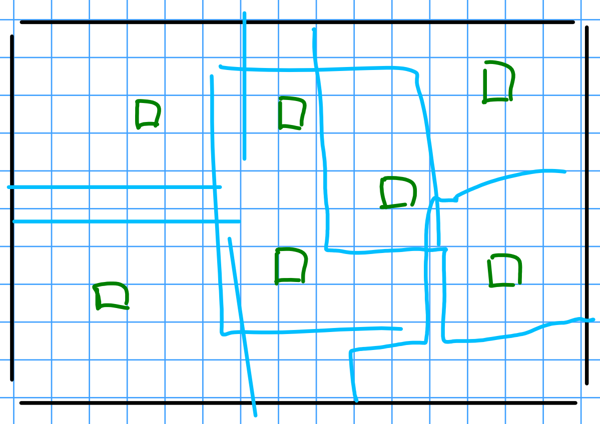
n - вершина дерева

см. Дасгупту

Задача о покрытии множествами

Вход: $U, I = \{S_i\}_{i=1}^{n} : S_i \subseteq U$

Выход: $\min \; J \subseteq I : \bigcup_{S_i \in J} S_i = U$



Алгоритм: на $\forall$ шаге выбираем подмножество, которое покрывает max число эл-ов, которые ещё не покрыли.

α - приближенный алгоритм A

$\forall x \; A(x) \leq A_{opt}(x) \cdot \alpha$

Утв: полученный алгоритм $\log n$ - приближ.

$\triangleright$ $\exists$ в оптимальном решении K множеств.

$\Rightarrow \exists$ мн-во, покрыв. $1/k$ всех эл-ов.

На $\forall$ шаге мы берём мн-во, кот. покрывает $1/k$ от оставшихся эл-ов.

$n_0 = n$

$n_1 \leq n_0 (1 - \frac{1}{k})$

$n_2 \leq n_1 \cdot (1 - 1/k)$

$\vdots$

n_i - эл-ов на i шаге

$$n_i = n_{i-1} \cdot (1 - 1/k) = n \cdot (1 - 1/k)^i$$

] $n_t < 1$, t – ?

$$n_t = n\cdot\left(1-\frac{1}{k}\right)^{t} = n\cdot\left(1-\frac{1}{k}\right)^{k\cdot\frac{t}{k}} \le n\cdot e^{-t/k} < 1$$

$$n < e^{t/k}$$

$\log n < t/k$ при $t = k\cdot\log n$ алгоритм останавливается

$\Rightarrow$ В полученном решении $\le t$ множеств $\le k\cdot\log$, где k – оптимум. $\triangleleft$