Иванов А.Б.

# Алгоритмы и структуры данных

## Жадные алгоритмы (greedy)

# Выбор непересекающихся интервалов

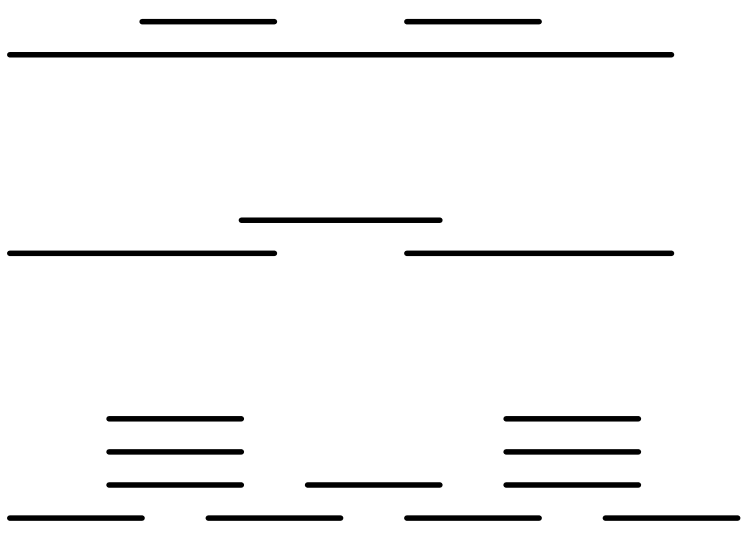
Идеи, как организовать жадный алгоритм: выбираем интервал

1. с минимальной начальной точкой $s_i$
2. минимальной длины $f_i - s_i$
3. с минимальным количеством пересечений

# Выбор непересекающихся интервалов

Идеи, как организовать жадный алгоритм: выбираем интервал

1. с минимальной начальной точкой $s_i$
2. минимальной длины $f_i - s_i$
3. с минимальным количеством пересечений



# Выбор непересекающихся интервалов

Выбираем интервал с самым ранним временем окончания $f_i$. Выкидываем все интервалы, пересекающиеся с ним. Повторяем.

Утв. Это оптимальная стратегия.

Док-во. Пусть $I_0$ – интервал с самым ранним временем окончания. Возьмем оптимальный набор интервалов. Если $I_0$ в него не входит – заменим самый левый интервал в наборе на $I_0$.

# Бинарное кодирование

Имеется алфавит $\Sigma$

Для каждой буквы $\alpha \in \Sigma$ определяем код –
последовательность бит (коды могут быть разной длины)

$$c : \Sigma \to \{0,1\}^*$$

Единственный ли способ раскодировать слово?

# Безпрефиксные коды (prefix-free)

Код называется безпрефиксным, если не существует двух букв таких, что код одной является началом кода другой.

$$\nexists\, \alpha,\, \beta \in \Sigma :\; c(\alpha) \text{ является началом } c(\beta)$$

Безпрефиксный код раскодируется однозначно.

Если коды букв алфавита изобразить в виде бинарного дерева – буквы будут стоять только в листьях дерева.

# Оптимальный беспрефиксный код

**Дано:** для каждого символа $\alpha \in \Sigma$ известна частота $p(\alpha)$ появления его в тексте.

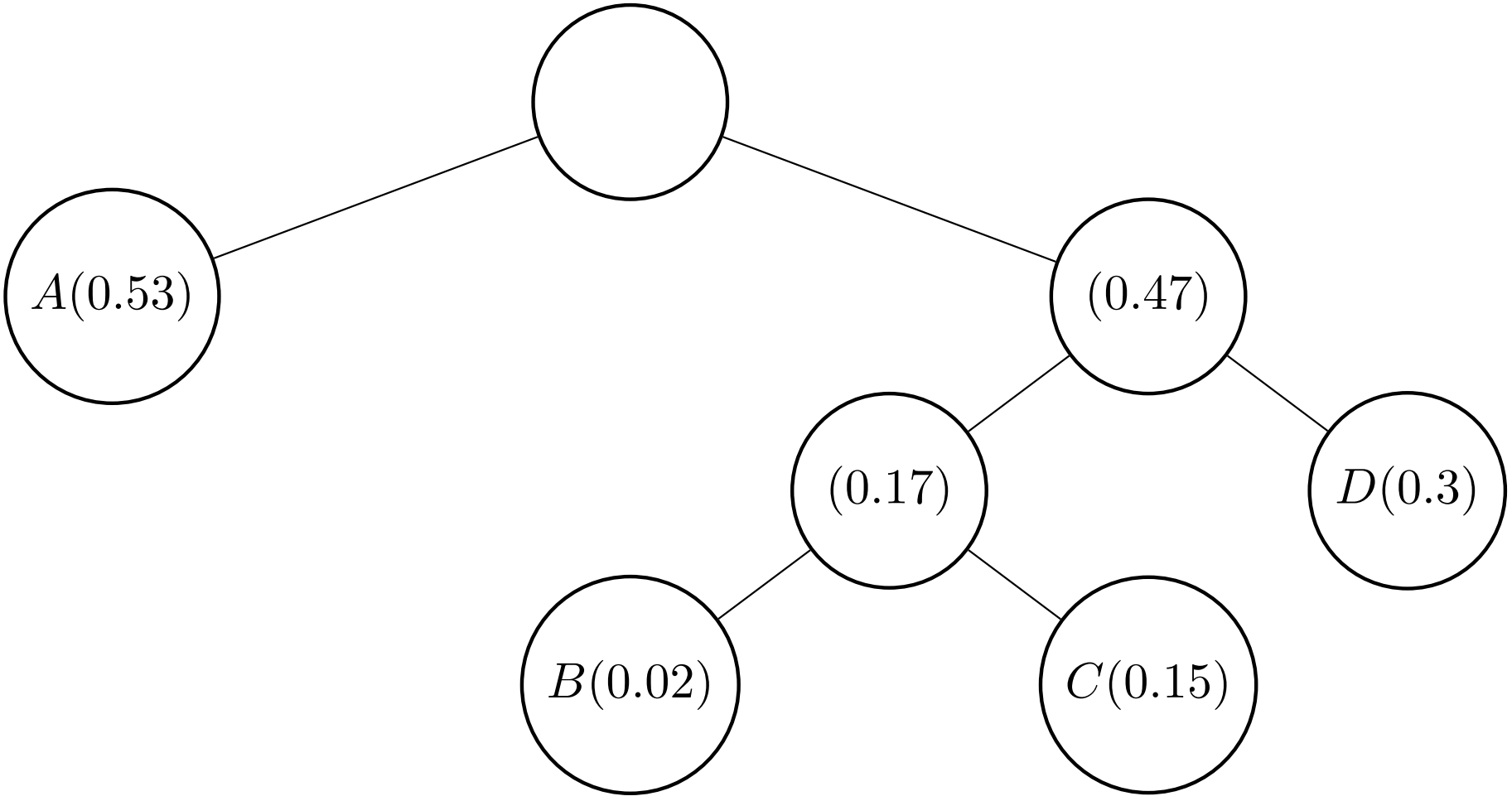
**Требуется:** найти беспрефиксный код, для которого сумма

$$\sum_{\alpha \in \Sigma} p(\alpha)|c(\alpha)|$$

минимальна ($|c(a)|$ – длина кодового слова для символа $\alpha$).

# Алгоритм Хаффмана

p(A)=0.53, p(B)=0.02, p(C)=0.15, p(D)=0.3



# Алгоритм Хаффмана

```
procedure Huffman( chars, p ):
  // chars - массив символов (алфавит)
  // p - массив частот символов
  // (кол-во символов = кол-во частот = n)
  Q = priority queue
  // создадим n листьев
  for i = 0 to n-1:
      node = вершина(
          ch = chars[i],
          freq = p[i])
      Q.add(node)

  // создадим еще n-1 внутренних вершин
  for k = n to 2n-2:
      node1 = Q.deleteMin()
      node2 = Q.deleteMin()
      node = вершина(
          freq  = node1.freq + node.freq,
          left  = node1,
          right = node2)
      Q.add(node)
  end
end
```

```
class Node {
  char ch;
  int  freq;
  Node left;
  Node right;

  ...
}
```

# Задача о рюкзаке

**Дано:**

- Рюкзак максимальной вместимости $W$ (по весу)
- $n$ предметов, для которых задан вес $w_i$ и ценность $v_i$

**Требуется:** Выбрать предметы общего веса $\leq W$ и максимальной суммарной ценности.

# Задача о рюкзаке: жадный алгоритм

Предположим, что можно брать дробные части предметов. (например, есть песок золотой, серебрянный и медный и можно насыпать любое количество из имеющегося в наличии)

Очевидно: надо насыпать в рюкзак самый ценный (золотой). Если в рюкзаке осталось место – следующий по ценности (серебрянный) и т.д.

Жадный алгоритм дает решение с ценностью

$$v_{greedy-fractional} = v_{optimal-fractional}$$

Причем

$$v_{optimal-fractional} \geq v_{optimal}$$

# Задача о рюкзаке: жадный алгоритм

**Алгоритм (не очень хороший).**

1. Упорядочим предметы
   по убыванию относительной ценности $v_i/w_i$.

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \frac{v_3}{w_3} \geq \dots$$

2. Кладем в рюкзак предметы $1, 2, \dots k$, пока они помещаются.

**Пример:** $W = 1000$

$$\begin{array}{ll} v_1 = 2 & w_1 = 1 \\ v_2 = 1000 & w_2 = 1000 \end{array}$$

$$v_{optimal} = 1000$$

$$v_{greedy} = 2$$

# Задача о рюкзаке: жадный алгоритм

**Алгоритм (хороший).**

1. Упорядочим предметы
   по убыванию относительной ценности $v_i/w_i$.

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \frac{v_3}{w_3} \geq \dots$$

2. Кладем в рюкзак предметы $1, 2, \dots k$, пока они помещаются.
   Получили некоторое решение с ценностью
   $v_{greedy} = v_1 + \dots + v_k$.
3. Если есть предмет $(w_{max}, v_{max})$: $w_{max} \leq W$, $v_{greedy} < v_{max}$, то вместо решения, полученного на шаге 2 берем решение, состоящее из одного этого предмета.

# Анализ эффективности жадного алгоритма

**Утв.** Ценность решения по «хорошему» жадному алгоритму не меньше $\frac{1}{2}$ ценности оптимального решения ($\frac{1}{2}$-аппроксимация).

**Пример:** $W = 1000$

$$\begin{array}{ll} v_1 = 502 & w_1 = 501 \\ v_2 = 500 & w_2 = 500 \\ v_3 = 500 & w_3 = 500 \end{array}$$

$$v_{optimal} = 1000$$

$$v_{greedy-good} = 502$$

# Анализ эффективности жадного алгоритма

**Утв.** Ценность решения по «хорошему» жадному алгоритму не меньше $\frac{1}{2}$ ценности оптимального решения ($\frac{1}{2}$-аппроксимация).

**Доказательство:**

$$v_{greedy} = \sum_{i=1}^{k} v_i$$

$$v_{greedy-good} = \max\left(\sum_{i=1}^{k} v_i,\ v_{max}\right)$$

$$v_{greedy-good} \geq \sum_{i=1}^{k} v_i$$

$$v_{greedy-good} \geq v_{max} \geq v_{k+1}$$

$$2 \cdot v_{greedy-good} \geq \sum_{i=1}^{k} v_i + v_{k+1} \geq v_{greedy-fractional} \geq v_{optimal}$$

# Покрытие множествами

**Дано:** множество $B$ и его подмножества $S_1, \dots S_m \subseteq B$.

**Требуется:** найти минимальный набор множеств $S_{i_1}, \dots S_{i_k}$, которые покрывают все $B$.

# Покрытие множествами: жадный алгоритм

Критерий: выбираем $S_i$ с максимальным числом непокрытых элементов.

**Утв.** Пусть $|B| = n$ и оптимальное покрытие состоит из $k$ элементов. Тогда жадный алгоритм даст решение из не более, чем $k \ln n$ множеств.

Пусть $n_t$ – количество непокрытых элементов после шага $t$ в жадном алгоритме. Они покрываются $k$ подмножествами (оптимальным покрытием). Значит, существует подмножество, покрывающее не менее $n_t/k$ элементов, следовательно, на следующем шаге жадного алгоритма останутся непокрытыми

$$n_{t+1} \leq n_t - \frac{n_t}{k} = \left(1 - \frac{1}{k}\right) n_t$$

Поскольку $n_0 = n$ и $1 - x \leq e^{-x}$, то

$$n_t \leq \left(1 - \frac{1}{k}\right)^t n_0 < e^{-t/k} n$$

При $t = k \ln n$ получаем

$$n_t < e^{-k \ln n/k} n = 1 \quad \Rightarrow \quad n_t = 0$$

# Задача $k$-центра

**Дано:** полный взвешенный граф $(G, V)$, $k \in \mathbb{N}$.

**Требуется:** найти $C \subseteq V$, $|C| = k$, минимизирующее

$$\max_{v \in V} d(v, C)$$

($d(v, C)$ - расстояние до ближайшей вершины из $C$).

**Идея:** Первую вершину выберем произвольно. Если у нас уже есть $i$ вершин, следующую выберем наиболее далекую от них.

**Утв.:** Такой алгоритм ***для метрического графа*** дает решение не хуже, чем 2 оптимальных.

# Задача об упаковке в контейнеры

**Дано:** конечное множество $U$ и "размер"каждого элемента $s(u) \in [0, 1]$.

**Требуется:** найти минимальное разбиение на $k$ подмножеств $U_i \subseteq U : \forall i = 1 \dots k \sum\limits_{u \in U_i} s(u) \leq 1$.

**Идея:** (FF - First Fit) кладем следующий элемент в первое подмножество, куда он помещается.

**Утв.:** Такой алгоритм дает решение не хуже, чем 2 оптимальных.

Есть улучшенный жадный алгоритм FFD (First Fit Decreasing), который дает результат $\frac{11}{9}OPT + 4$.

# Динамическое программирование

Термин: Ричард Беллман, 1950-е годы

The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making,... But planning, is not a good word for various reasons. I decided therefore to use the word, "programming"... [Dynamic] has a very interesting property as an adjective, and that is its impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. Its impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

# Максимальное независимое множество

Maximal Weighted Independent Set (MAX-WIS)

Множество вершин графа называется *независимым*, если в нем нет соседних вершин.

**Дано:** граф, в каждой вершине задан *неотрицательный* вес.

**Требуется:** найти независимое множество вершин с максимальной суммой весов.

Задача сложная (NP-полная). Однако если граф – дерево, то существует алгоритм, работающий за линейное время.

# MAX-WIS в дереве

Пусть граф $G$ = дерево-цепочка ○–○–○–○ с $n$ вершинами.
В вершинах заданы веса $w_i \geq 0$, $i = 1, ...n$.
Как решить задачу о максимальном независимом множестве?

Пусть $G'$ – дерево с удаленной вершиной $n$,
$G''$ – дерево с удаленными вершинами $n$ и $n-1$.

- Если вершина $n$ не входит в оптимальное решение, то оптимальное решение является оптимальным решением для $G'$.
- Если вершина $n$ входит в оптимальное решение, то оптимальное решение получается из оптимального решения для $G''$ добавлением вершины $n$.

# MAX-WIS в дереве: динамика

Обозначим сумму весов в оптимальном решении MAX-WIS($G$). Тогда

$$\text{MAX-WIS}(G) = \max\Big(\text{MAX-WIS}(G'),\ \text{MAX-WIS}(G'') + w_n\Big)$$

- Итак, задача легко решается, если известны двух задач меньшего размера.
- Нельзя использовать рекурсию!
- Решаем последовательно задачи для $G_1$, $G_2$, …, $G_n$.
- Формула дает максимальный вес, а как восстановить максимальное множество?
  - можно хранить для каждой решенной задачи множество (память $= O(n^2)$)
  - но лучше хранить признаки – включается ли вершина $k$ в решение задачи $G_k$, тогда максимальное множество восстанавливается за линейное время.

# Наибольшая возраст. последовательность

**Дано:** последовательность чисел $a_1, a_2, \ldots a_n$

**Найти:** самую длинную возрастающую подпоследовательность $a_{i_1}, a_{i_2}, \ldots a_{i_k}$:

$$i_1 < i_2 < \ldots < i_k$$

$$a_{i_1} < a_{i_2} < \ldots < a_{i_k}$$

5 2 8 6 3 6 9 7 3

# Наибольшая возраст. последовательность

5 2 8 6 3 6 9 7 3

$A[j]$ = максимальная длина возрастающей подпоследовательности, заканчивающейся в $j$

$$A[j] = 1 + \max\{A[i] : i < j,\ a_i < a_j\}$$

Решение начальной задачи: $\max(A[j])$

# Идея динамического программирования

1. исходная задача включается в семейство задач;
2. решение каждой задачи выражается через решения задач «меньшего размера»;
3. решаем задачи последовательно в правильном порядке.

1-мерная динамика

- Максимальное независимое множество в дереве
- Наибольшая возрастающая последовательность

Далее: 2-мерная динамика

- Расстояние редактирования
- Задача о рюкзаке
- Задача коммивояжёра (динамика по подмножествам)

# Выравнивание (расстояние редактирования)

(Sequence alignment, minimum edit distance)

**Дано:**

- слова $X = x_1 x_2 \dots x_m$ и $Y = y_1 y_2 \dots y_n$
- стоимость замены $\alpha_{ab}$ для каждой пары букв $a, b$
- стоимость вставки буквы $\alpha_+$, стоимость удаления $\alpha_-$

Пример: $X =$ `AGGGCT`, $Y =$ `AGGCAC`

```
AGGGCT-
AGG-CAC
```

стоимость $= \alpha_- + \alpha_{TA} + \alpha_+$

**Требуется:** найти расстояние редактирования = выравнивание последовательностей минимальной стоимости.

(расстояние Левенштейна: $\alpha_- = \alpha_+ = \alpha_{xy} = 1$ при $x \neq y$)

# Выравнивание: динамика

$X_i :=$ первые $i$ букв слова $X$
$Y_j :=$ первые $j$ букв слова $Y$
$A[i, j] :=$ стоимость оптимального выравнивания $X_i$ и $Y_j$.

$$A[i,j] = \min\Big(\alpha_{x_i y_j} + A[i-1, j-1],\ \alpha_- + A[i-1, j],\ \alpha_+ + A[i, j-1]\Big)$$

# Выравнивание: пример

| | | S | I | T | T | I | N | G |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| K | | | | | | | | |
| I | | | | | | | | |
| T | | | | | | | | |
| T | | | | | | | | |
| E | | | | | | | | |
| N | | | | | | | | |

# Выравнивание: динамика

```
procedure SequenceAlignment( X, Y, α )
  A = 2-мерный массив размера)
  for i = 0,1,2,...m:
    A[i,0] = i*α_
  for i = 0,1,2,...n:
    A[0,i] = i*α+

  for i = 1,2,...m:
    for j = 1,2,...n:
       A[i,j] = min( A[i-1,j-1] + α_{x_i y_j},
                     A[i-1,j] + α_,
                     A[i,j-1] + α+)
  return A[m,n]
end
```

Сложность: $O(mn)$

При заполненном массиве $A$ восстанавливаем оптимальное выравнивание, двигаясь в обратном направлении, за $O(m+n)$.