Адрианов Н.М.
Иванов А.Б.

Алгоритмы и структуры данных

Жадные алгоритмы (greedy)

Известные нам жадные алгоритмы

- Алгоритм Дейкстры.
- Алгоритм Прима.
- Алгоритм Крускала.

Задача о расписании

- Процессор (исполнитель) – может выполнять 1 задачу в каждый момент времени.
- Список задач P_1, P_2, ... P_n.
- Для каждой задачи заданы длительность l_i и приоритет w_i.

Пусть выбран порядок выполнения задач P_{m_1}, P_{m_2}, P_{m_3}, ...
Время окончания выполнения задачи = сумма длительностей и задачи и всех задач перед ней:

$$C_{m_k} = \sum_{i=1}^{k} l_{m_i}$$

Требуется: найти порядок выполнения задач, при котором величина

$$\sum_{i=1}^{n} w_i C_i$$

будет минимальна.

Задача о расписании: жадный алгоритм

Утв. Упорядочим задачи по *убыванию* величины w_i/l_i.
Этот порядок будет оптимальным.

Док-во. Предположим, что оптимальный порядок другой.
Тогда в нем найдется пара задач, идущих подряд

$$\ldots, P_i, P_j, \ldots$$

для которых

$$w_i/l_i < w_j/l_j.$$

Поменяем порядок выполнения задач P_i и P_j: ...

Задача о кэшировании (cache)

Память компьютера разбита на страницы. Чтобы процессор получил доступ к данным, надо зачитать их в кэш. Читается страница целиком. В кэш помещается m страниц.

Дано: последовательность запросов к страницам.

Требуется: определить оптимальный сценарий сброса страниц из кэша, при котором количество промахов (cache miss) будет минимально.

Пример: $m = 3$. Последовательность запросов

$$a,\ b,\ c,\ a,\ d,\ d,\ c,\ d,\ a,\ b$$

Задача о кэшировании: жадный алгоритм

Теорема (Bélády, 1966). Выкидываем из кэша ту страницу, которая понадобится в наиболее далеком будущем. Это оптимальная стратегия.

Доказательство использует похожую идею с перестановкой, как в задаче о расписании, но рассуждение более сложное.

Выбор непересекающихся интервалов

Дано: набор (возможно пересекающихся) временных интервалов (s_i, f_i).

Требуется: выбрать максимально возможное количество непересекащихся интервалов.

Выбор непересекающихся интервалов

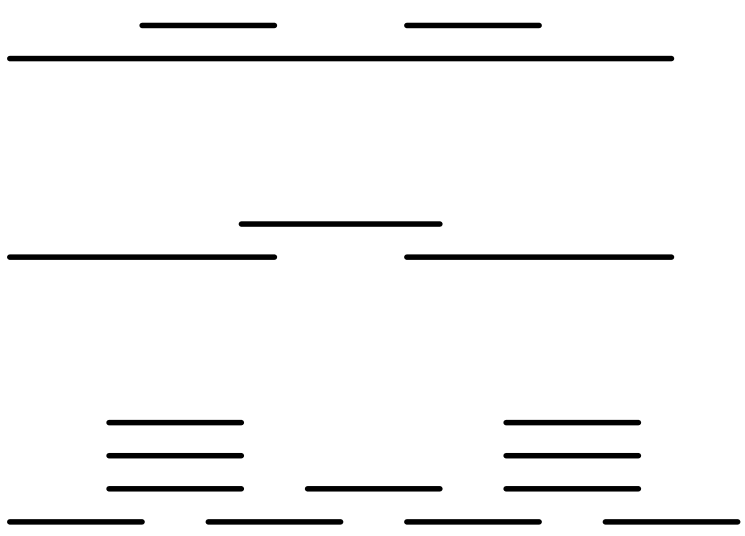
Идеи, как организовать жадный алгоритм: выбираем интервал

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 Σ

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.

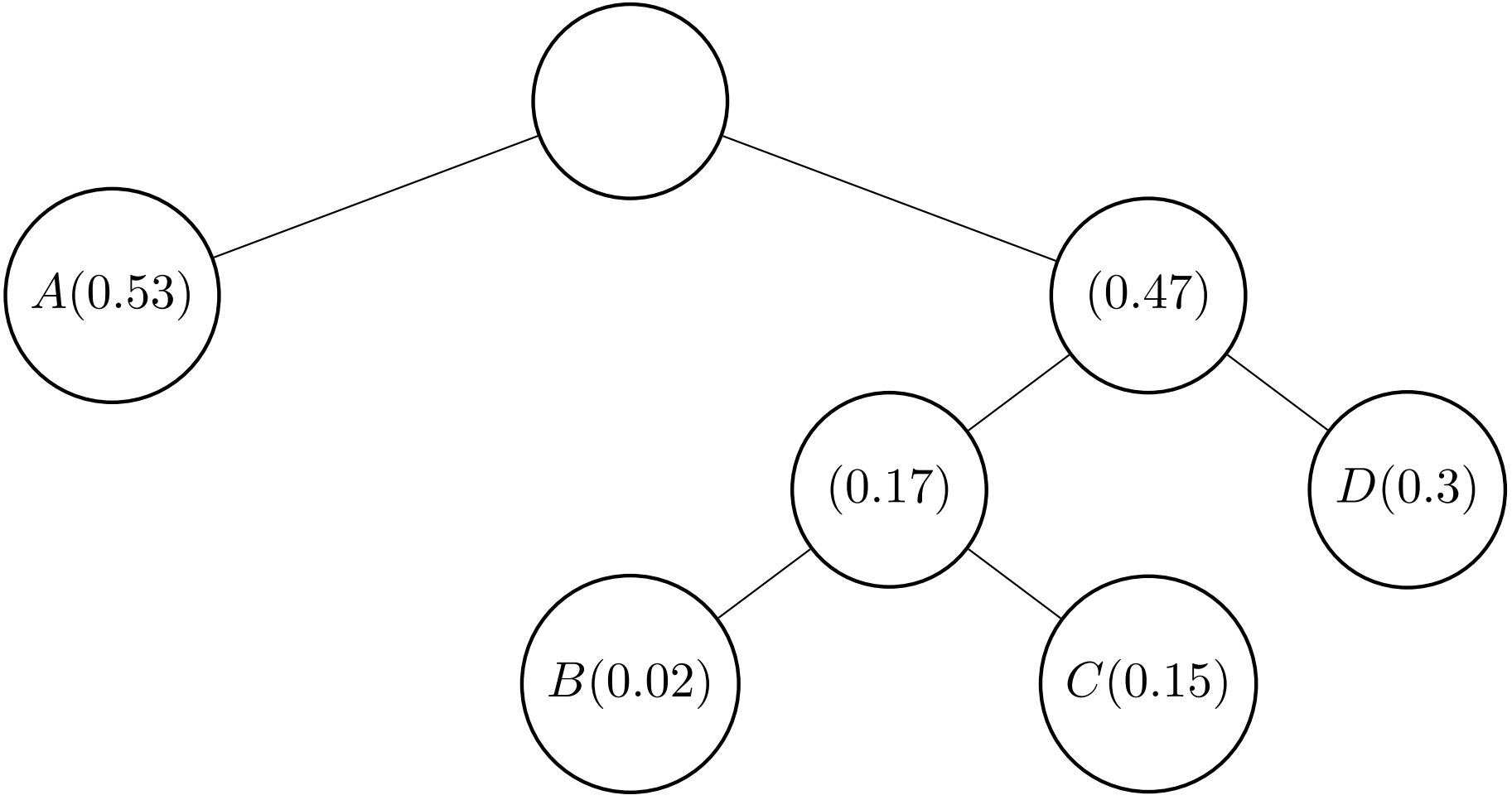
Требуется: найти беспрефиксный код, для которого сумма

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

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

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;

  ...
}
```