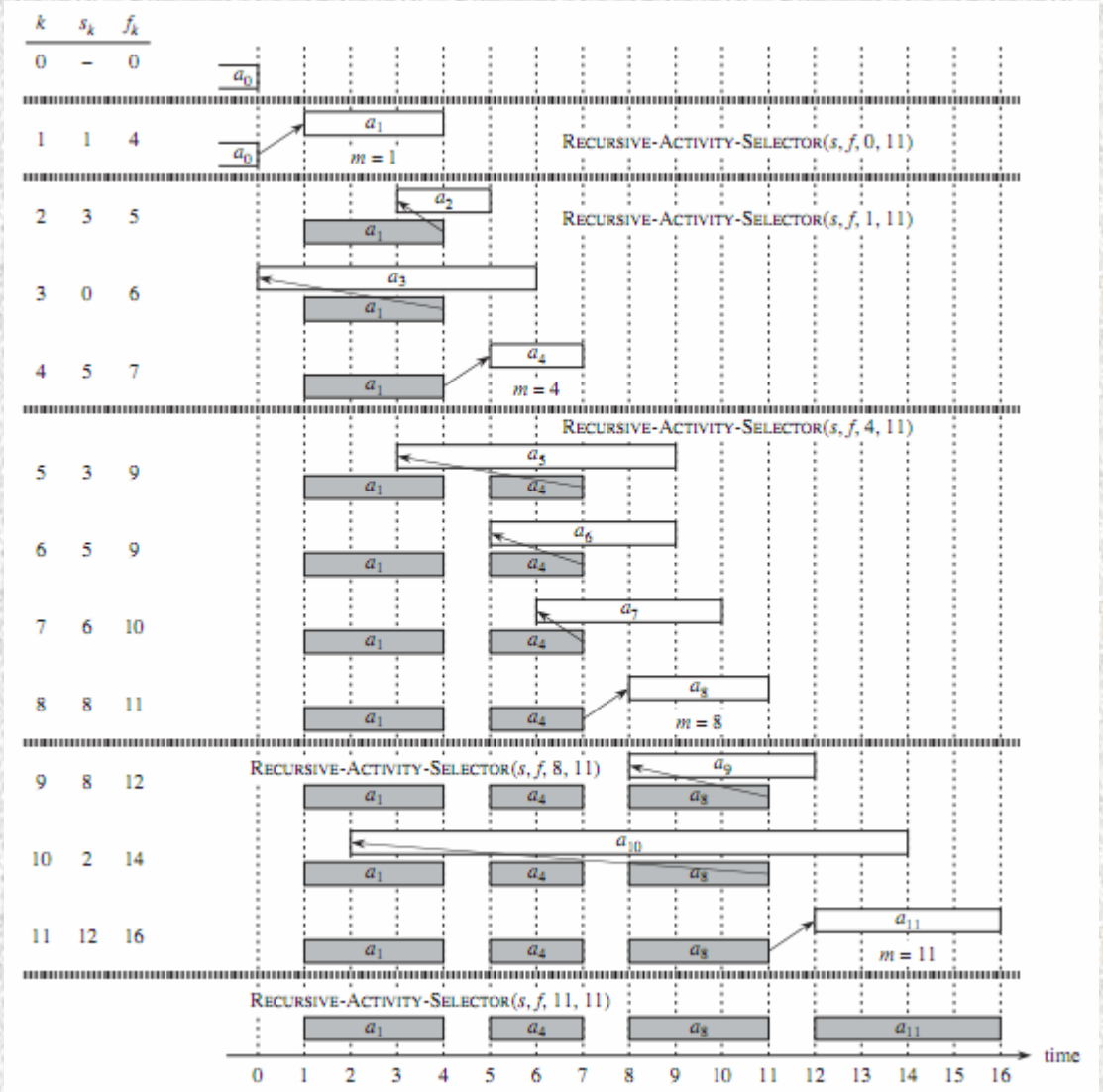
Лекция 3. Жадные алгоритмы

Задача о выборе заявок. Рекурсивный жадный алгоритм

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3        m = m + 1
4  if m ≤ n
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

Лекция 3. Жадные алгоритмы

Задача о выборе заявок. Рекурсивный жадный алгоритм



Лекция 3. Жадные алгоритмы

Задача о выборе заявок

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n ← length[s]
2  A ← {1}
3  j ← 1
4  for i ← 2 to n
5       do if s_i ⩾ f_j
6             then A ← A ∪ {i}
7                  j ← i
8  return A
```

Лекция 3. Жадные алгоритмы

Задача о выборе заявок

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

Рис. 1 Работа алгоритма GREEDY-ACTIVITY-SELECTOR для 11 заявок (таблица слева).

Лекция 3. Жадные алгоритмы

Когда применим жадный алгоритм?

Принцип жадного выбора

Говорят, что к оптимизационной задаче применим **принцип жадного выбора** (greedy-choice property), если последовательность локально оптимальных (жадных) выборов дает глобально оптимальное решение. Различие между жадными алгоритмами и динамическим программированием можно пояснить так: на каждом шаге жадный алгоритм берёт «самый жирный кусок», а потом уже пытается сделать наилучший выбор среди оставшихся, каковы бы они ни были; алгоритм динамического программирования принимает решение, просчитав заранее последствия для всех вариантов.

Как доказать, что жадный алгоритм даёт оптимальное решение? Это не всегда тривиально, но в типичном случае такое доказательство следует схеме, использованной в доказательстве теоремы 1. Сначала мы доказываем, что жадный выбор на первом шаге не закрывает пути к оптимальному решению: для всякого решения есть другое, согласованное с жадным выбором и не худшее первого. Затем показывается, что подзадача, возникающая после жадного выбора на первом шаге, аналогична исходной, и рассуждение завершается по индукции.

Лекция 3. Жадные алгоритмы

Когда применим жадный алгоритм?

Дискретная задача о рюкзаке (0-1 knapsack problem) состоит в следующем. Пусть вор пробрался на склад, на котором хранится n вещей. Вещь номер i стоит v_i долларов и весит w_i килограммов (v_i и w_i — целые числа). Вор хочет украсть товара на максимальную сумму, причём максимальный вес, который он может унести в рюкзаке, равен W (число W тоже целое). Что он должен положить в рюкзак?

Непрерывная задача о рюкзаке (fractional knapsack problem) отличается от дискретной тем, что вор может дробить краденые товары на части и укладывать в рюкзак эти части, а не обязательно вещи целиком (если в дискретной задаче вор имеет дело с золотыми слитками, то в непрерывной — с золотым песком).

Лекция 3. Жадные алгоритмы

Коды Хаффмена

Коды Хаффмена широко используются при сжатия информации (в типичной ситуации выигрыш может составить от 20% до 90% в зависимости от типа файла). Алгоритм Хаффмена находит оптимальные коды символов (исходя из частоты использования этих символов в сжимаемом тексте) с помощью жадного выбора.

Пусть в файле длины 100 000 известны частоты символов (рис. 3). Мы хотим построить **двоичный код** (binary character code), в котором каждый символ представляется в виде конечной последовательности битов, называемой **кодовым словом** (codeword). При использовании **равномерного кода** (fixed-length code), в котором все кодовые слова имеют одинаковую длину, на каждый символ (из шести имеющихся) надо потратить как минимум три бита, например, $\mathtt{a} = 000, \mathtt{b} = 001, \ldots, \mathtt{f} = 101$. На весь файл уйдет 300 000 битов — нельзя ли уменьшить это число?

Неравномерный код (variable-length code) будет экономнее, если часто встречающиеся символы закодировать короткими последо-

Лекция 3. Жадные алгоритмы

Коды Хаффмена

	a	b	c	d	e	f
Количество (в тысячах)	45	13	12	16	9	5
Равномерный код	000	001	010	011	100	101
Неравномерный код	0	101	100	111	1101	1100

Рис. 3 Задача о выборе кода. В файле длиной 100 000 символов встречаются только латинские буквы от a до f (в таблице указано, по скольку раз каждая). Если каждую букву закодировать тремя битами, то всего будет 300 000 битов. Если использовать неравномерный код (нижняя строка), то достаточно 224 000 битов.

вательностями битов, а редко встречающиеся — длинными. Один такой код показан на рис. 17.3: буква a изображается однобитовой последовательностью 0, а буква f — четырёхбитовой последовательностью 1100. При такой кодировке на весь файл уйдёт

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224\,000$$

битов, что даёт около 25% экономии.

Лекция 3. Жадные алгоритмы

Коды Хаффмена

Хаффмен построил жадный алгоритм, который строит оптимальный префиксный код. Этот код называется **кодом Хаффмена** (Huffman code). Алгоритм строит дерево T, соответствующее оптимальному коду, снизу вверх, начиная с множества из $|C|$ листьев и делая $|C|-1$ «слияний». Мы предполагаем, что для каждого символа $c \in C$ задана его частота $f[c]$. Для нахождения двух объектов, подлежащих слиянию, используется очередь с приоритетами Q, использующая частоты f в качестве рангов — сливаются два объекта с наименьшими частотами. В результате слияния получается новый объект (внутренняя вершина), частота которого считается равной сумме частот двух сливаемых объектов.

```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do z ← ALLOCATE-NODE()
5          x ← left[z] ← EXTRACT-MIN(Q)
6          y ← right[z] ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

Лекция 3. Жадные алгоритмы

Теоретические основы жадных алгоритмов

Матроидом (matroid) называется пара $M = (S, \mathcal{I})$, удовлетворяющая следующим условиям.

1. S — конечное непустое множество.
2. $\mathcal{I}$ — непустое семейство подмножеств S; входящие в $\mathcal{I}$ подмножества называют **независимыми** (independent). При этом должно выполняться такое свойство: из $B \in \mathcal{I}$ и $A \subseteq B$ следует $A \in \mathcal{I}$ (в частности, всегда $\emptyset \in \mathcal{I}$). Семейство $\mathcal{I}$, удовлетворяющее этому условию, называется **наследственным** (hereditary).
3. Если $A \in \mathcal{I}$, $B \in \mathcal{I}$ и $|A| < |B|$, то существует такой элемент $x \in B \setminus A$, что $A \cup \{x\} \in \mathcal{I}$. Это свойство семейства $\mathcal{I}$ называют **свойством замены** (exchange property).

Термин «матроид» принадлежит Хасслеру Уитни (Hassler Whitney). Он занимался **матричными матроидами** (matric matroids), у которых S — множество всех строк некоторой матрицы, и множество строк считается независимым, если эти строки линейно независимы в обычном смысле.

Лекция 3. Жадные алгоритмы

Теоретические основы жадных алгоритмов

```
GREEDY(M, w)
1  A ← ∅
2  отсортировать S[M] в порядке невозрастания весов
3  for x ∈ S[M] (перебираем все x в указанном порядке)
4       do if A ∪ {x} ∈ I[M]
5             then A ← A ∪ {x}
6  return A
```

Лекция 3. Жадные алгоритмы

Задача о расписании

В задаче о расписании для заказов равной длительности с единственным исполнителем, сроками и штрафами (scheduling unit-time tasks with deadlines and penalties for a single processor) исходными данными являются:

- множество $S = \{1, 2, \ldots, n\}$, элементы которого мы называем заказами;
- последовательность из n целых чисел $d_1, d_2, \ldots, d_n$, называемых **сроками** (deadlines) ($1 \leqslant d_i \leqslant n$ для всех i, срок d_i относится к заказу номер i);
- последовательность из n неотрицательных чисел $w_1, w_2, \ldots, w_n$, называемых **штрафами** (penalties) (если заказ номер i не выполнен ко времени d_i, взимается штраф w_i).

Требуется найти расписание для S, при котором сумма штрафов будет наименьшей.

Заказ номер i **просрочен** (late) для данного расписания, если его выполнение завершается позже момента d_i, в противном случае заказ считается **выполненным в срок** (early). Всякое расписание можно, не меняя суммы штрафов, модифицировать таким образом, чтобы все просроченные заказы стояли в нём после выполненных в срок. В самом деле, если просроченный заказ y идёт раньше выполненного в срок заказа x, то можно поменять их местами в расписании, и статус обоих заказов при этом не изменится.