Вводные слайды, 1D, 2D

Иван Сергеевич Казменко

Санкт-Петербургский Государственный Университет

Четверг, 17 февраля 2022 года

Содержание

1. Вводная информация
   - Ссылка
   - Структура курса
   - Структура занятия
   - Как получить зачёт
   - Проверка решений
   - Чем пользоваться
2. Одномерные задачи
   - Числа Фибоначчи
   - Лестница
   - Динамика и индукция
   - Анти-лестница
   - Кузнечик
   - Восстановление ответа
3. Двумерные задачи
   - Число сочетаний
   - Наибольшая общая подпоследовательность
   - Редакционное расстояние
   - Строка и шаблон

Содержание

1. Вводная информация
   - Ссылка
   - Структура курса
   - Структура занятия
   - Как получить зачёт
   - Проверка решений
   - Чем пользоваться
2. Одномерные задачи
   - Числа Фибоначчи
   - Лестница
   - Динамика и индукция
   - Анти-лестница
   - Кузнечик
   - Восстановление ответа
3. Двумерные задачи
   - Число сочетаний
   - Наибольшая общая подпоследовательность
   - Редакционное расстояние
   - Строка и шаблон

Ссылка

Ссылка на материалы занятий:
`http://acm.math.spbu.ru/~gassa/dp-2022`

Предупреждение: всё далее — предварительная информация, она может поменяться!

Структура курса

Что мы будем делать?

- Слушать теорию
- Решать задачи за компьютером
- Рассказывать решения на разборе
- Делать свою задачу

Структура занятия

Примерная структура занятия:

- Разбираем друг другу задачи с предыдущих занятий
- Обсуждаем новую тему, теорию и реализацию
- Решаем задачи на новую тему между занятиями

Как получить зачёт

Как получить зачёт?

- Набрать баллы за решения (40% или 50% или 60%)
  - Обычные задачи: дедлайн в 23:59 перед следующим занятием
  - Задачи со звёздочкой: сложнее, но без дедлайна
  - После дедлайна: решение даёт половину баллов
- Сделать свою задачу

Проверка решений

Проверка решений

- Автоматическая проверяющая система: `http://acm.math.spbu.ru/tsweb`
- Логины и пароли будут разосланы на почту
- Submit: послать решение на проверку
- Monitor: кто что решил
- Clarifications: задать вопрос по формальному условию задачи
- Submissions: предыдущие попытки и их результаты (`NO` $\rightarrow$ `OK`, `WA`, `TL`, ...)
- Statements: условия задач
- List of all contests: предыдущие занятия

Чем пользоваться

Чем можно пользоваться?

- Документация по языку, конспекты, Wikipedia, Google, OEIS, ...
- Задавать вопросы преподавателю
- Вместе придумывать решения

Что нужно делать самим

- Писать код решения
- Делать свою задачу

Содержание

1 Вводная информация
- Ссылка
- Структура курса
- Структура занятия
- Как получить зачёт
- Проверка решений
- Чем пользоваться

2 Одномерные задачи
- Числа Фибоначчи
- Лестница
- Динамика и индукция
- Анти-лестница
- Кузнечик
- Восстановление ответа

3 Двумерные задачи
- Число сочетаний
- Наибольшая общая подпоследовательность
- Редакционное расстояние
- Строка и шаблон

Числа Фибоначчи

Определение: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ при $n \geqslant 2$.
Задача: дано n, вычислите F_n.

Числа Фибоначчи

C++, рекурсия:

```cpp
#include <iostream>
using namespace std;




int fib (int n) {
    if (n < 2) return n;
    return fib (n - 1) + fib (n - 2);
}



int main () {
    int n;
    while (cin >> n)
        cout << fib (n) << endl;
    return 0;
}
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

C++, рекурсия:

```
 1  #include <iostream>
 2  using namespace std;
 3
 4
 5
 6
 7  int fib (int n) {
 8      if (n < 2) return n;
 9      return fib (n - 1) + fib (n - 2);
10  }
11
12
13
14  int main () {
15      int n;
16      while (cin >> n)
17          cout << fib (n) << endl;
18      return 0;
19  }
```

Пример $n = 5$:

- $5 \to 4, 3$
- . $4 \to 3, 2$
- . . $3 \to 2, 1$
- . . . $2 \to 1, 0$
- 1
- 0
- . . . 1
- . . $2 \to 1, 0$
- . . . 1
- . . . 0
- . $3 \to 2, 1$
- . . $2 \to 1, 0$
- . . . 1
- . . . 0
- . . 1

Числа Фибоначчи

C++, рекурсия с запоминанием (мемоизация):

```cpp
#include <iostream>
using namespace std;

int f [99];

int fib (int n) {
    if (f[n] == 0) {
        if (n < 2) f[n] = n;
        else f[n] = fib (n - 1) + fib (n - 2);
    }
    return f[n];
}

int main () {
    int n;
    while (cin >> n)
        cout << fib (n) << endl;
    return 0;
}
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

C++, рекурсия с запоминанием (мемоизация):

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  int f [99];
 5
 6  int fib (int n) {
 7      if (f[n] == 0) {
 8          if (n < 2) f[n] = n;
 9          else f[n] = fib (n - 1) + fib (n - 2);
10      }
11      return f[n];
12  }
13
14  int main () {
15      int n;
16      while (cin >> n)
17          cout << fib (n) << endl;
18      return 0;
19  }
```

Пример $n = 5$:

- $5 \to 4, 3$
- . $4 \to 3, 2$
- . . $3 \to 2, 1$
- . . . $2 \to 1, 0$
- 1
- 0
- . . . 1 запомнено
- . . 2 запомнено
- . 3 запомнено

Числа Фибоначчи

C++, итеративное решение:

```cpp
#include <iostream>
using namespace std;

int main () {
    int f [99];
    f[0] = 0;
    f[1] = 1;
    for (int i = 2; i < 99 /*???*/; i++)
        f[i] = f[i - 1] + f[i - 2];
    int n;
    while (cin >> n)
        cout << f[n] << endl;
    return 0;
}
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

Python, рекурсия:

```
 1  import sys
 2
 3
 4  def fib (n):
 5      if n < 2:
 6          return n
 7      return fib (n - 1) + fib (n - 2)
 8
 9  for line in sys.stdin:
10      n = int (line)
11      print (fib (n))
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

Python, рекурсия с запоминанием (мемоизация):

```
 1  import functools, sys
 2
 3  @functools.lru_cache (maxsize = None)
 4  def fib (n):
 5      if n < 2:
 6          return n
 7      return fib (n - 1) + fib (n - 2)
 8
 9  for line in sys.stdin:
10      n = int (line)
11      print (fib (n))
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

D, рекурсия:

```d
import std;


int fib (int n) {
    if (n < 2) return n;
    return fib (n - 1) + fib (n - 2);
}

void main () {
    foreach (line; stdin.byLine) {
        auto n = line.to !(int);
        writeln (fib (n));
    }
}
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

Числа Фибоначчи

D, рекурсия с запоминанием (мемоизация):

```d
import std;

alias fib = memoize !(fibImpl);
int fibImpl (int n) {
    if (n < 2) return n;
    return fib (n - 1) + fib (n - 2);
}

void main () {
    foreach (line; stdin.byLine) {
        auto n = line.to !(int);
        writeln (fib (n));
    }
}
```

Ввод:

```
5
30
40
45
```

Вывод:

```
5
832040
102334155
1134903170
```

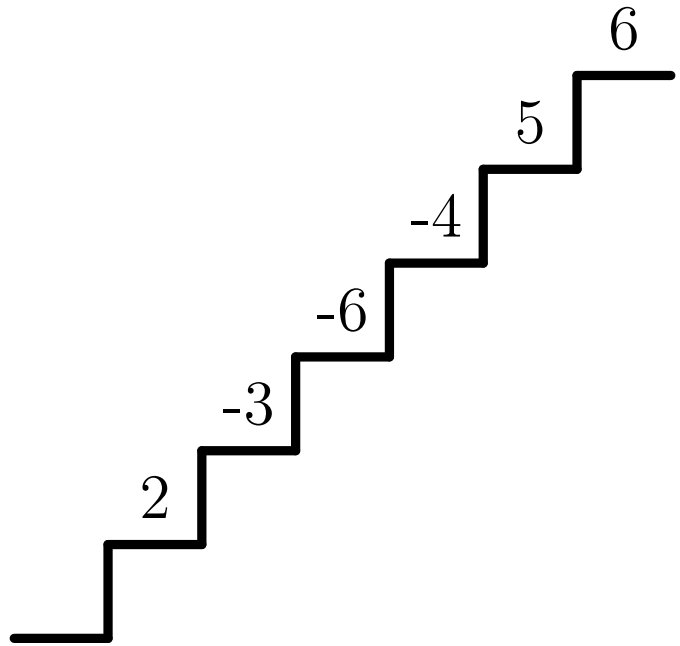
Лестница

- Есть лестница из n ступенек.
- Мы начинаем у подножия лестницы, на нулевой ступеньке.
- На ступеньках написаны числа $a_1, a_2, \ldots, a_n$.
- Мы должны оказаться на n-й ступеньке.
- За один шаг мы поднимаемся либо на одну, либо на две ступеньки.
- Наши баллы — сумма чисел, на которые мы шагнули.
- Как подняться по лестнице, чтобы максимизировать баллы?

Лестница

- Есть лестница из n ступенек.
- Мы начинаем у подножия лестницы, на нулевой ступеньке.
- На ступеньках написаны числа $a_1, a_2, \ldots, a_n$.
- Мы должны оказаться на n-й ступеньке.
- За один шаг мы поднимаемся либо на одну, либо на две ступеньки.
- Наши баллы — сумма чисел, на которые мы шагнули.
- Как подняться по лестнице, чтобы максимизировать баллы?

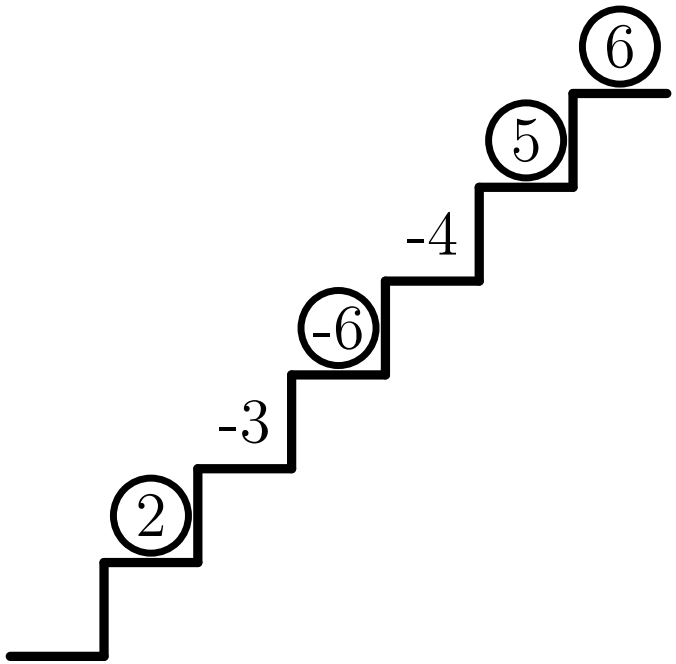
Пример 1:



Лестница

- Есть лестница из n ступенек.
- Мы начинаем у подножия лестницы, на нулевой ступеньке.
- На ступеньках написаны числа $a_1, a_2, \ldots, a_n$.
- Мы должны оказаться на n-й ступеньке.
- За один шаг мы поднимаемся либо на одну, либо на две ступеньки.
- Наши баллы — сумма чисел, на которые мы шагнули.
- Как подняться по лестнице, чтобы максимизировать баллы?

Пример 1, решение:

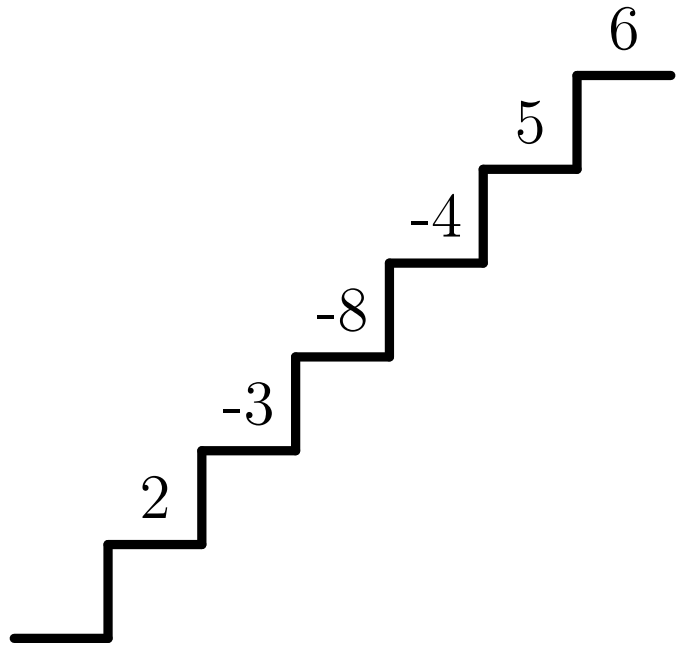


Ответ:
$2 + (-6) + 5 + 6 = 7$.

Лестница

- Есть лестница из n ступенек.
- Мы начинаем у подножия лестницы, на нулевой ступеньке.
- На ступеньках написаны числа $a_1, a_2, \ldots, a_n$.
- Мы должны оказаться на n-й ступеньке.
- За один шаг мы поднимаемся либо на одну, либо на две ступеньки.
- Наши баллы — сумма чисел, на которые мы шагнули.
- Как подняться по лестнице, чтобы максимизировать баллы?

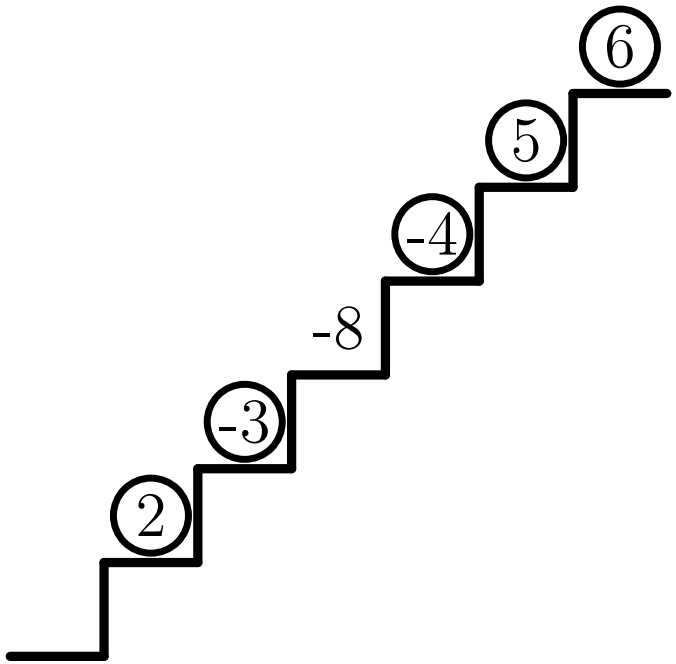
Пример 2:



Лестница

- Есть лестница из n ступенек.
- Мы начинаем у подножия лестницы, на нулевой ступеньке.
- На ступеньках написаны числа $a_1, a_2, \ldots, a_n$.
- Мы должны оказаться на n-й ступеньке.
- За один шаг мы поднимаемся либо на одну, либо на две ступеньки.
- Наши баллы — сумма чисел, на которые мы шагнули.
- Как подняться по лестнице, чтобы максимизировать баллы?

Пример 2, решение:



Ответ:
$2 + (-3) + (-4) + 5 + 6 = 6.$

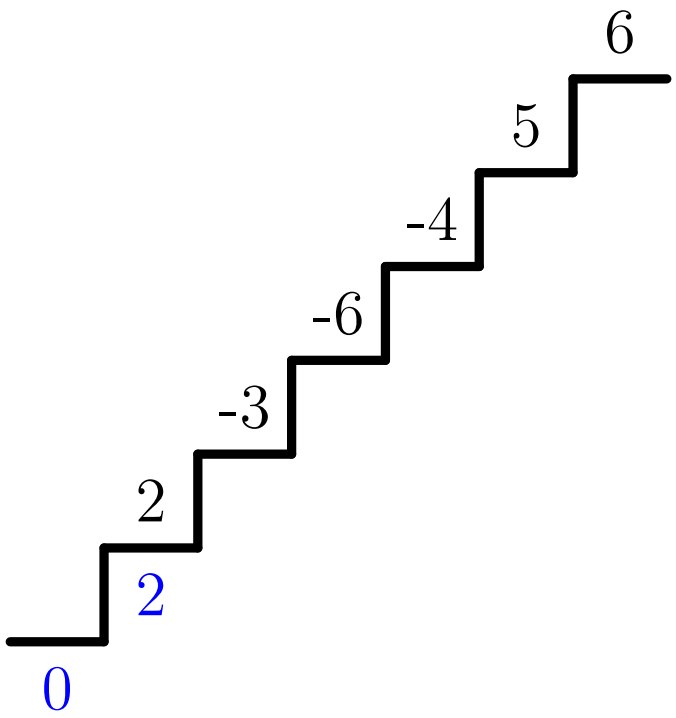
Лестница, решение

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Как подняться на k-ю ступеньку, чтобы максимизировать баллы?

Лестница, решение

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Как подняться на k-ю ступеньку, чтобы максимизировать баллы?
- Назовём это значение $f(k)$.
- База: $f(0) = 0$.
- База: $f(1) = a_1$.

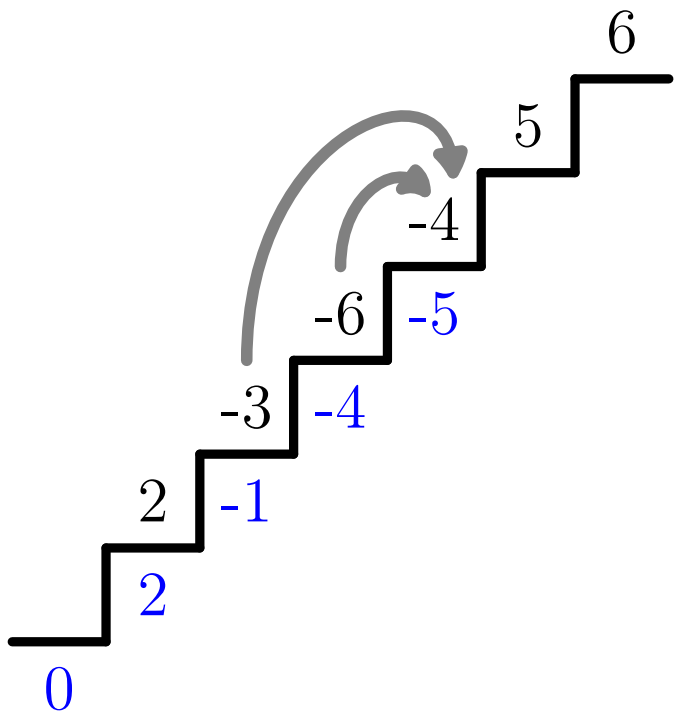
Пример 1:



Лестница, решение

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Как подняться на k-ю ступеньку, чтобы максимизировать баллы?
- Назовём это значение $f(k)$.
- База: $f(0) = 0$.
- База: $f(1) = a_1$.
- Пусть мы уже нашли $f(k')$ для всех $k' < k$.
- Как подняться на k-ю ступеньку?
- Мы пришли либо с $k-1$-й ступеньки, либо с $k-2$-й ступеньки.

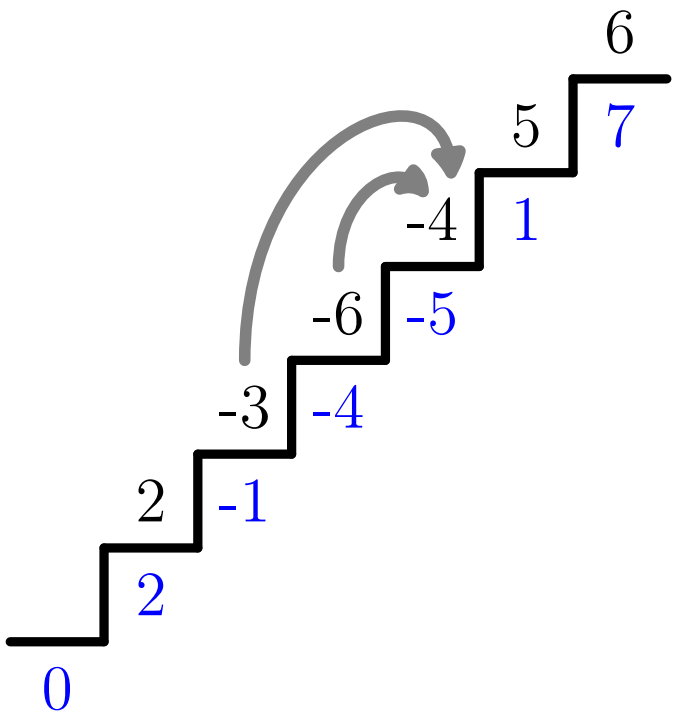
Пример 1, $k = 4$:



Лестница, решение

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Как подняться на k-ю ступеньку, чтобы максимизировать баллы?
- Назовём это значение $f(k)$.
- База: $f(0) = 0$.
- База: $f(1) = a_1$.
- Пусть мы уже нашли $f(k')$ для всех $k' < k$.
- Как подняться на k-ю ступеньку?
- Мы пришли либо с $k-1$-й ступеньки, либо с $k-2$-й ступеньки.

Пример 1, $k = 4$:



$$f(k) = a_k + \max(f(k-1), f(k-2))$$

Лестница, код

Рекурсия:

```
 1  #include <algorithm>
 2  #include <iostream>
 3  using namespace std;
 4
 5  int a [10001];
 6
 7
 8  int f (int n) {
 9      if (n == 0) return 0;
10      if (n == 1) return a[1];
11
12      return a[n] + max (f (n - 1), f (n - 2));
13  }
14
15
16  int main () {
17      int n;  cin >> n;
18      for (int i = 1; i <= n; i++)  cin >> a[i];
19
20      cout << f (n) << endl;
21      return 0;
22  }
```

Лестница, код

Рекурсия с запоминанием (мемоизация):

```
 1  #include <algorithm>
 2  #include <iostream>
 3  using namespace std;
 4
 5  int a [10001];
 6  int mf [10001];
 7
 8  int f (int n) {
 9      if (n == 0) return 0;
10      if (n == 1) return a[1];
11      if (mf[n] == INT_MAX)
12          mf[n] = a[n] + max (f (n - 1), f (n - 2));
13      return mf[n];
14  }
15
16  int main () {
17      int n;  cin >> n;
18      for (int i = 1; i <= n; i++)  cin >> a[i];
19      for (int i = 0; i <= n; i++)  mf[i] = INT_MAX;
20      cout << f (n) << endl;
21      return 0;
22  }
```

Лестница, код

Итеративное решение:

```
#include <algorithm>
#include <iostream>
using namespace std;

int a [10001];
int f [10001];

int main () {
    int n;  cin >> n;
    for (int i = 1; i <= n; i++)  cin >> a[i];
    f[0] = 0;
    f[1] = a[1];
    for (int i = 2; i <= n; i++)
        f[i] = a[i] + max (f[i - 1], f[i - 2]);
    cout << f[n] << endl;
    return 0;
}
```

Динамика и индукция

Динамическое программирование:

- Найдём $f(n)$.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже найдены для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$.
- Ответ: $f(n)$.

Математическая индукция:

- Докажем, что $f(n)$ посчитано верно.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже посчитаны верно для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$ тоже посчитано верно.
- Значит, $f(n)$ посчитано верно.

Динамика и индукция

Динамическое программирование:

- Найдём $f(n)$.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже найдены для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$.
- Ответ: $f(n)$.

Математическая индукция:

- Докажем, что $f(n)$ посчитано верно.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже посчитаны верно для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$ тоже посчитано верно.
- Значит, $f(n)$ посчитано верно.

Динамика и индукция

Динамическое программирование:

- Найдём $f(n)$.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже найдены для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$.
- Ответ: $f(n)$.

Математическая индукция:

- Докажем, что $f(n)$ посчитано верно.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже посчитаны верно для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$ тоже посчитано верно.
- Значит, $f(n)$ посчитано верно.

Динамика и индукция

Динамическое программирование:

- Найдём $f(n)$.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже найдены для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$.
- Ответ: $f(n)$.

Математическая индукция:

- Докажем, что $f(n)$ посчитано верно.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже посчитаны верно для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$ тоже посчитано верно.
- Значит, $f(n)$ посчитано верно.

Динамика и индукция

Динамическое программирование:

- Найдём $f(n)$.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже найдены для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$.
- Ответ: $f(n)$.

Математическая индукция:

- Докажем, что $f(n)$ посчитано верно.
- База: $f(0) = 0$, $f(1) = a_1$.
- Предположение: пусть $f(k')$ уже посчитаны верно для всех $k' < k$.
- Переход: тогда $f(k) = a_k + \max(f(k-1), f(k-2))$ тоже посчитано верно.
- Значит, $f(n)$ посчитано верно.

Анти-лестница

- Есть последовательность из n чисел: $a_1, a_2, \ldots, a_n$.
- Мы хотим выбрать подпоследовательность.
- При этом нельзя выбирать два соседних элемента.
- Как максимизировать сумму выбранных чисел?

Анти-лестница

- Есть последовательность из n чисел: $a_1, a_2, \ldots, a_n$.
- Мы хотим выбрать подпоследовательность.
- При этом нельзя выбирать два соседних элемента.
- Как максимизировать сумму выбранных чисел?

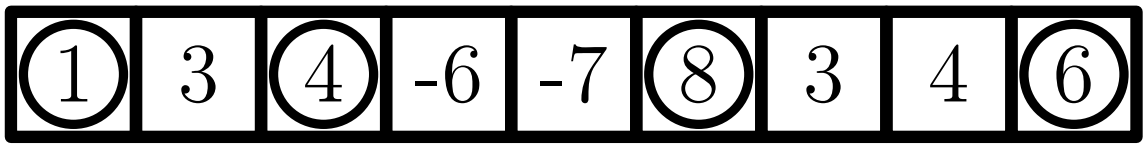
Пример 1:

1	3	4	-6	-7	8	3	4	6

Анти-лестница

- Есть последовательность из n чисел: $a_1, a_2, \ldots, a_n$.
- Мы хотим выбрать подпоследовательность.
- При этом нельзя выбирать два соседних элемента.
- Как максимизировать сумму выбранных чисел?

Пример 1, решение:



Ответ: $1 + 4 + 8 + 6 = 19$.

Анти-лестница

- Есть последовательность из n чисел: $a_1, a_2, \ldots, a_n$.
- Мы хотим выбрать подпоследовательность.
- При этом нельзя выбирать два соседних элемента.
- Как максимизировать сумму выбранных чисел?

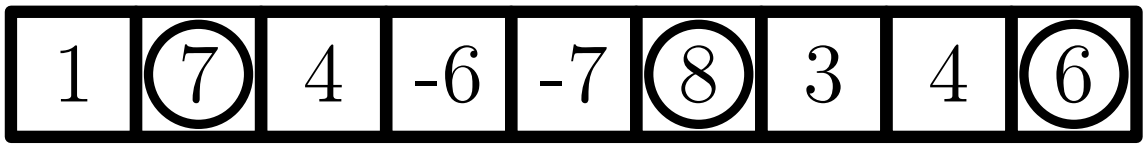
Пример 2:

1	7	4	-6	-7	8	3	4	6

Анти-лестница

- Есть последовательность из n чисел: $a_1, a_2, \ldots, a_n$.
- Мы хотим выбрать подпоследовательность.
- При этом нельзя выбирать два соседних элемента.
- Как максимизировать сумму выбранных чисел?

Пример 2, решение:

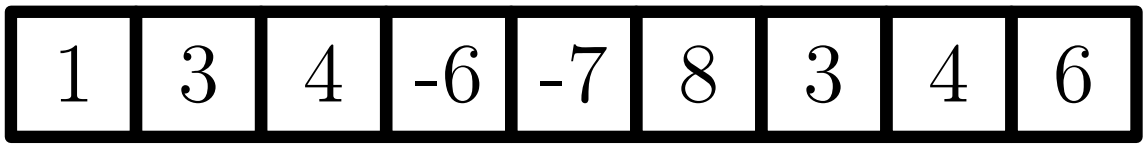


Ответ: $7 + 8 + 6 = 21$.

Анти-лестница, решение 1

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Пусть мы взяли k-е число последним.
- Как при этом максимизировать сумму?

Пример 1:

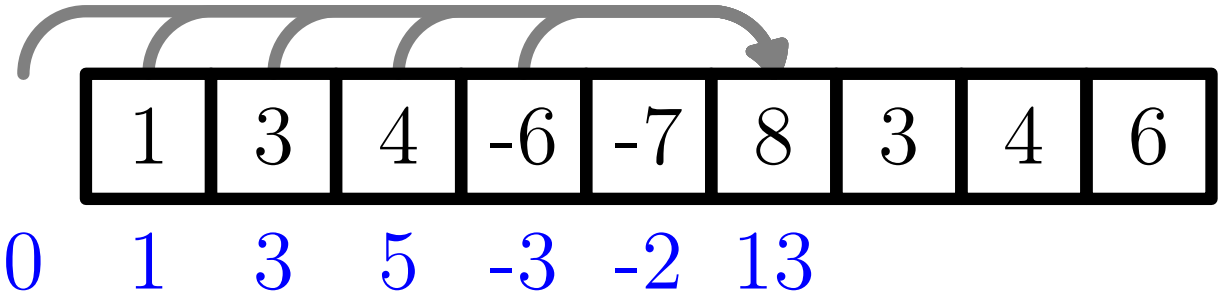


- Назовём это значение $f(k)$.
- База: пусть $f(0) = 0$.
- База: $f(1) = a_1$.

Анти-лестница, решение 1

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Пусть мы взяли k-е число последним.
- Как при этом максимизировать сумму?

Пример 1:

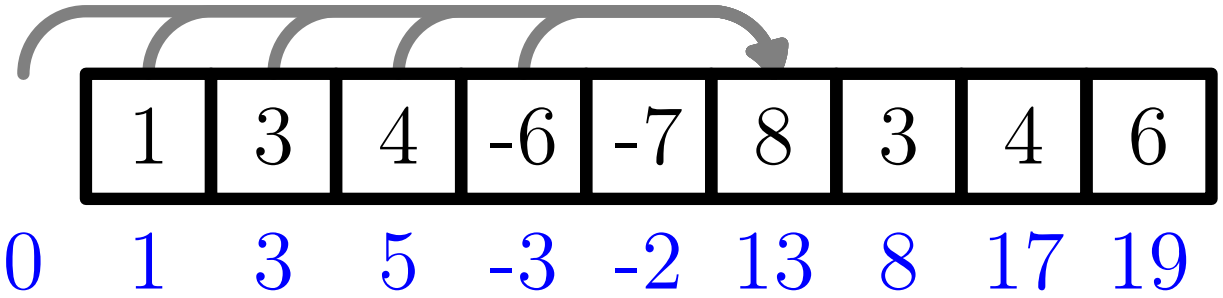


- Пусть мы уже нашли $f(k')$ для всех $k' < k$.
- Как посчитать $f(k)$?
- Предыдущим мы могли взять $k-2$-е число, или $k-3$-е, ..., или первое, или нулевое (то есть ничего не взять).

Анти-лестница, решение 1

- Давайте решать более общую задачу.
- Для каждого k от 0 до n:
- Пусть мы взяли k-е число последним.
- Как при этом максимизировать сумму?

Пример 1:

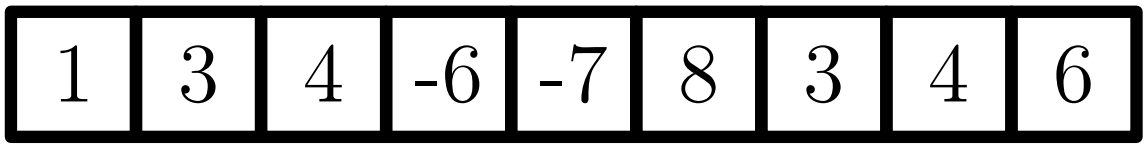


- Получается $f(k) = a_k + \max(f(k-2), f(k-3), \ldots, f(1), f(0))$.
- Ответ — это $\max(f(0), f(1), \ldots, f(n))$.
- Решение работает за $O(n^2)$.

Анти-лестница, решение 2

- Давайте решать другую более общую задачу.
- Для каждого k от 0 до n:
- Рассмотрим префикс последовательности: $a_1, a_2, \ldots, a_k$.
- Как при этом максимизировать сумму?

Пример 1:



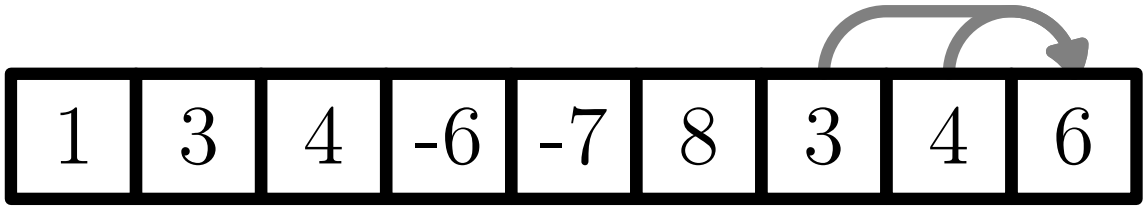
0 1

- Назовём это значение $g(k)$.
- База: $g(0) = 0$.
- База: $g(1) = \max(0, a_1)$.

Анти-лестница, решение 2

- Давайте решать другую более общую задачу.
- Для каждого k от 0 до n:
- Рассмотрим префикс последовательности: $a_1, a_2, \ldots, a_k$.
- Как при этом максимизировать сумму?

Пример 1:



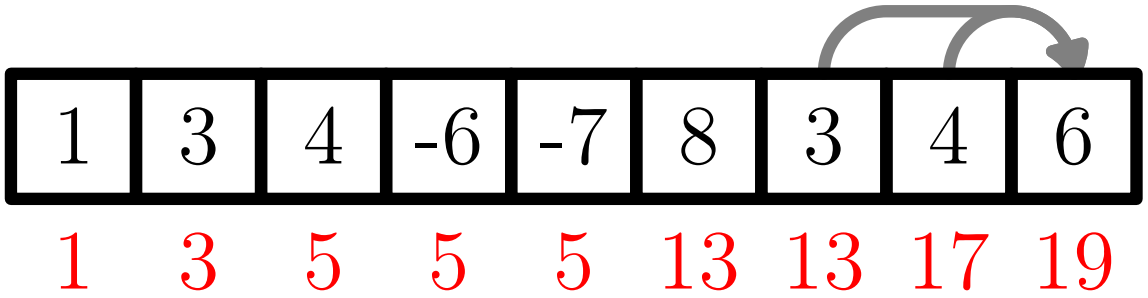
Как найти $g(k)$, если мы уже знаем $g(k')$ для всех $k' < k$?

- Мы можем не брать a_k, тогда возьмём ответ из $g(k-1)$.
- Мы можем взять a_k, тогда добавим ответ из $g(k-2)$.

Анти-лестница, решение 2

- Давайте решать другую более общую задачу.
- Для каждого k от 0 до n:
- Рассмотрим префикс последовательности: $a_1, a_2, \ldots, a_k$.
- Как при этом максимизировать сумму?

Пример 1:



- Получается $g(k) = \max(g(k-1), a_k + g(k-2))$.
- Ответ — это $g(n)$.
- Решение работает за $O(n)$.

Кузнечик — может или нет

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- Пример: $a = 2$, $b = 3$, $c = 5$.
- Может ли кузнечик добраться до клетки n?

Кузнечик — может или нет

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- Пример: $a = 2$, $b = 3$, $c = 5$.
- Может ли кузнечик добраться до клетки n?

Обобщим задачу: для каждой клетки k выясним, может ли кузнечик до неё добраться.

- База: $f(1) = \texttt{true}$.
- Пересчёт: $f(k) = f(k - a) \vee f(k - b) \vee f(k - c)$.

Кузнечик — может или нет

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- Пример: $a = 2$, $b = 3$, $c = 5$.
- Может ли кузнечик добраться до клетки n?

Псевдокод:

```
f[1] := true
for k := 2, ..., n:
  f[k] := false
  if k >= a and f[k - a]:  f[k] := true
  if k >= b and f[k - b]:  f[k] := true
  if k >= c and f[k - c]:  f[k] := true
```

Кузнечик — как можно быстрее

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Кузнечик — как можно быстрее

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Обобщим задачу: для каждой клетки k выясним, за какое минимальное число прыжков кузнечик может до неё добраться.

- База: $f(1) = 0$.
- Пересчёт: $f(k) = 1 + \min(f(k-a), f(k-b), f(k-c))$.

Кузнечик — как можно быстрее

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Псевдокод:

```
f[1] := 0
for k := 2, ..., n:
  f[k] := infinity
  if k >= a:  f[k] := min (f[k], 1 + f[k - a])
  if k >= b:  f[k] := min (f[k], 1 + f[k - b])
  if k >= c:  f[k] := min (f[k], 1 + f[k - c])
```

Кузнечик — ямы

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Кузнечик — ямы

- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Обобщим задачу: для каждой клетки k выясним, за какое минимальное число прыжков кузнечик может до неё добраться.

- База: $f(1) = 0$.
- Пересчёт: $f(k) = 1 + \min(f(k-a), f(k-b), f(k-c))$.
- Если $h_k = \texttt{true}$ (hole, яма в клетке k), то пусть $f(k) = \infty$.

Кузнечик — ямы

- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?

Псевдокод:

```
f[1] := 0
for k := 2, ..., n:
  f[k] := infinity
  if not h[k]:
    if k >= a:  f[k] := min (f[k], 1 + f[k - a])
    if k >= b:  f[k] := min (f[k], 1 + f[k - b])
    if k >= c:  f[k] := min (f[k], 1 + f[k - c])
```

Кузнечик — сок

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- В некоторых клетках — ямы, туда попадать нельзя.
- В каждой клетке есть неотрицательное количество сока.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Из всех таких маршрутов выберите тот, на котором кузнечик соберёт как можно больше сока.

Кузнечик — сок

- В некоторых клетках — ямы, туда попадать нельзя.
- В каждой клетке есть неотрицательное количество сока.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Из всех таких маршрутов выберите тот, на котором кузнечик соберёт как можно больше сока.

Обобщим задачу. Для каждой клетки k найдём пару: минимальное количество прыжков до неё, максимальное количество сока при таком количестве прыжков.

- База: $f(1) = \{0, j_1\}$; пусть j_k (juice) — количество сока в клетке k.
- Пересчёт: $f(k) = \{1, j_k\} + \text{best}(f(k-a), f(k-b), f(k-c))$.
- Если $h_k =$ `true` (hole, яма в клетке k), то пусть $f(k) = \{\infty, 0\}$.

Кузнечик — сок

- В некоторых клетках — ямы, туда попадать нельзя.
- В каждой клетке есть неотрицательное количество сока.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Из всех таких маршрутов выберите тот, на котором кузнечик соберёт как можно больше сока.

Псевдокод:

```
f[1] := {0, -j[1]}
for k := 2, ..., n:
  f[k] := {infinity, 0}
  if not h[k]:
    if k >= a:  f[k] := min (f[k], {1, -j[k]} + f[k - a])
    if k >= b:  f[k] := min (f[k], {1, -j[k]} + f[k - b])
    if k >= c:  f[k] := min (f[k], {1, -j[k]} + f[k - c])
```

Восстановление ответа

- Есть лента из клеток, по которой прыгает кузнечик.
- Он начинает в клетке 1.
- Из любой клетки x он может прыгнуть в клетки $x + a$, $x + b$ и $x + c$.
- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Как именно нужно прыгать кузнечику для этого?

Восстановление ответа

- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Как именно нужно прыгать кузнечику для этого?

Обобщим задачу: для каждой клетки k выясним, за какое минимальное число прыжков кузнечик может до неё добраться.

- База: $f(1) = 0$.
- Пересчёт: $f(k) = 1 + \min(f(k-a), f(k-b), f(k-c))$.

Восстановление ответа

- В некоторых клетках — ямы, туда попадать нельзя.
- За какое минимальное число прыжков кузнечик может добраться до клетки n?
- Как именно нужно прыгать кузнечику для этого?

Псевдокод:

```
f[1] := 0
for k := 2, ..., n:
  f[k] := infinity
  if not h[k]:
    if k >= a:  f[k] := min (f[k], 1 + f[k - a])
    if k >= b:  f[k] := min (f[k], 1 + f[k - b])
    if k >= c:  f[k] := min (f[k], 1 + f[k - c])
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

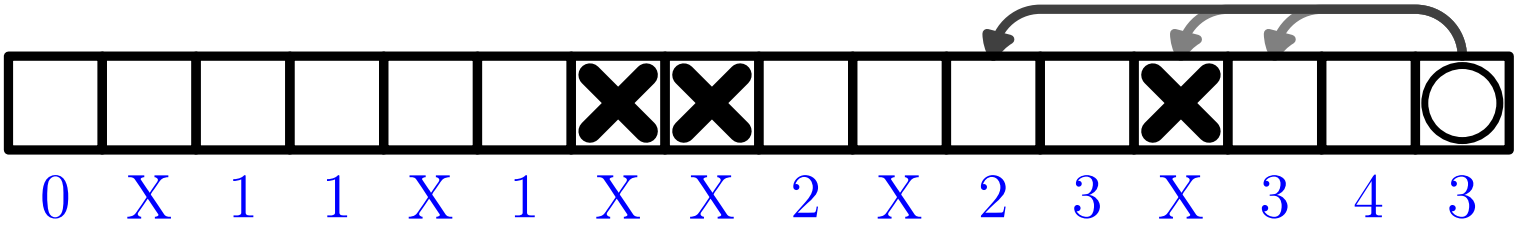
0 X 1 1 X 1 X X 2 X 2 3 X 3 4 3

Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

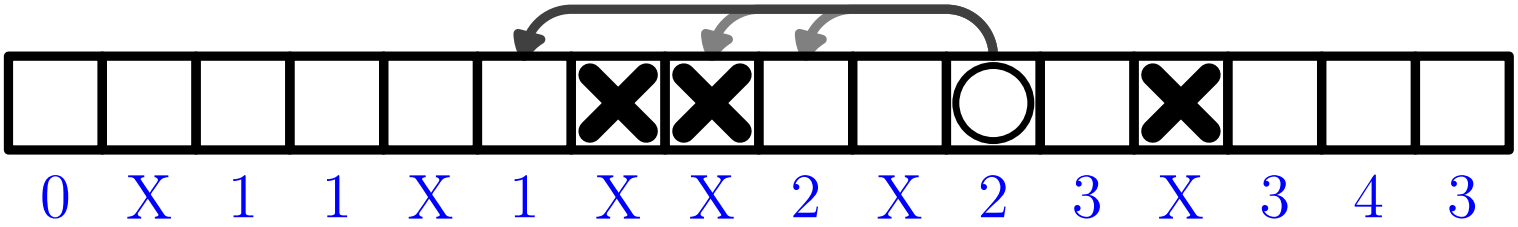


Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

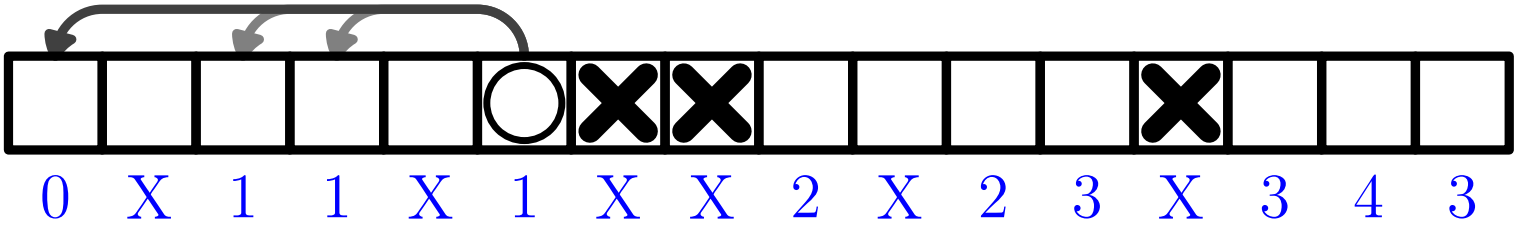


Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.



Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — обратные рёбра

Будем восстанавливать путь с конца: стоя в клетке x, смотреть, из какой клетки могло получиться значение $f(x)$.

0 X 1 1 X 1 X X 2 X 2 3 X 3 4 3

Псевдокод:

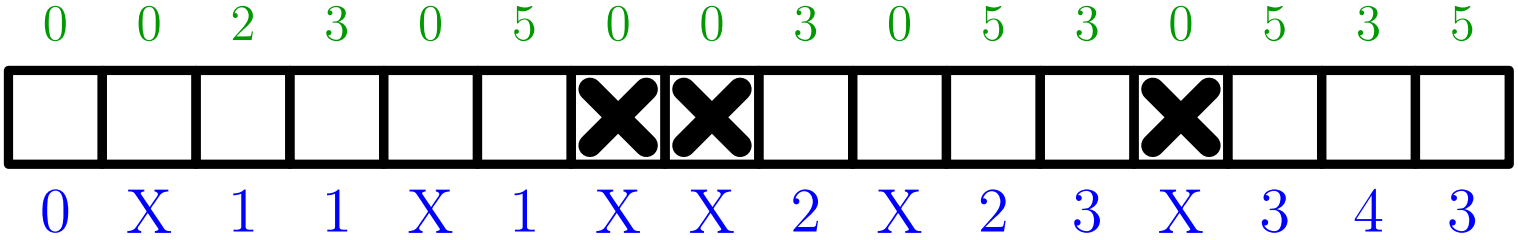
```
x := n
steps.add (x)
while x > 1:
  if      x >= a and f[x] = 1 + f[x - a]:  x -= a
  else if x >= b and f[x] = 1 + f[x - b]:  x -= b
  else if x >= c and f[x] = 1 + f[x - c]:  x -= c
  steps.add (x)
reverse (steps)
```

Восстановление ответа — хранение предков

Для каждой клетки k не только выясним, за какое минимальное число прыжков кузнечик может до неё добраться, но и запомним, каким прыжком он туда попал.

Восстановление ответа — хранение предков

Для каждой клетки k не только выясним, за какое минимальное число прыжков кузнечик может до неё добраться, но и запомним, каким прыжком он туда попал.

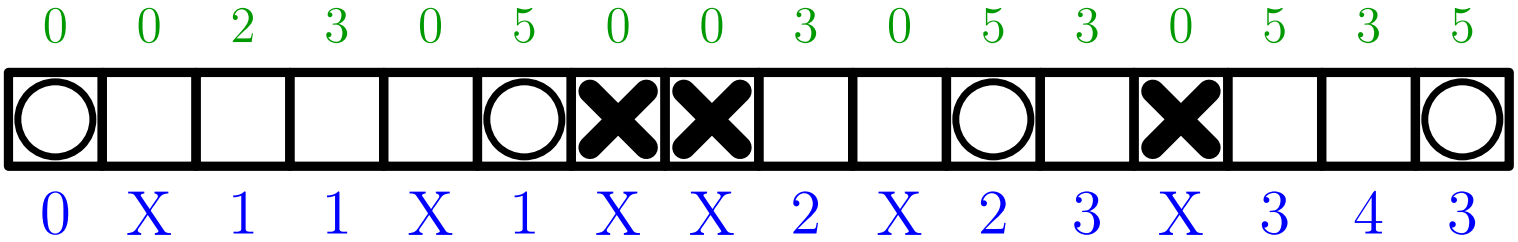


Псевдокод:

```
f[1] := 0
for k := 2, ..., n:
  f[k] := infinity
  if not h[k]:
    if k >= a and f[k] > 1 + f[k - a]:
      p[k] := a,  f[k] := 1 + f[k - a]
    if k >= b and f[k] > 1 + f[k - b]:
      p[k] := b,  f[k] := 1 + f[k - b]
    if k >= c and f[k] > 1 + f[k - c]:
      p[k] := c,  f[k] := 1 + f[k - c]
```

Восстановление ответа — хранение предков

Для каждой клетки k не только выясним, за какое минимальное число прыжков кузнечик может до неё добраться, но и запомним, каким прыжком он туда попал.



Будем восстанавливать путь с конца: стоя в клетке x, сдвигаться назад на величину прыжка p_x.
Псевдокод:

```
x := n
steps.add (x)
while x > 1:
  x -= p[x]
  steps.add (x)
reverse (steps)
```

Содержание

1 Вводная информация
- Ссылка
- Структура курса
- Структура занятия
- Как получить зачёт
- Проверка решений
- Чем пользоваться

2 Одномерные задачи
- Числа Фибоначчи
- Лестница
- Динамика и индукция
- Анти-лестница
- Кузнечик
- Восстановление ответа

3 Двумерные задачи
- Число сочетаний
- Наибольшая общая подпоследовательность
- Редакционное расстояние
- Строка и шаблон

Число сочетаний

Постановка задачи:

- Есть n различных объектов.
- Выбираем из них k различных (порядок не важен).
- Сколько способов это сделать? Обозначим как C_n^k.

Объекты можно просто пронумеровать числами от 1 до n.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

1	2	3	4	Сочетание
1	2	3	4	1, 2
1	2	3	4	1, 3
1	2	3	4	1, 4
1	2	3	4	2, 3
1	2	3	4	2, 4
1	2	3	4	3, 4

Число сочетаний

Постановка задачи:

- Есть n различных объектов.
- Выбираем из них k различных (порядок не важен).
- Сколько способов это сделать? Обозначим как C_n^k.

Объекты можно просто пронумеровать числами от 1 до n.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

1	2	3	4	Сочетание
1	2	3	4	1, 2
1	2	3	4	1, 3
1	2	3	4	1, 4
1	2	3	4	2, 3
1	2	3	4	2, 4
1	2	3	4	3, 4

Число сочетаний

Вычисление C_n^k — комбинаторное решение:

- Выбираем первый объект n способами.
- Выбираем второй объект из оставшихся $n-1$ способом.
- ...
- Выбираем k-й объект из оставшихся $n-k+1$ способом.
- Каждое сочетание получилось $k!$ раз в различных порядках.
- Поэтому поделим ответ на $k! = 1 \cdot 2 \cdot 3 \cdots k$.

Получаем, что

$$\begin{aligned} C_n^k &= \frac{n \cdot (n-1) \cdots (n-k+1)}{1 \cdot 2 \cdots k} \\ &= \frac{1 \cdot 2 \cdots (n-k)}{1 \cdot 2 \cdots (n-k)} \times \frac{(n-k+1) \cdot (n-k+2) \cdots n}{1 \cdot 2 \cdots k} \\ &= \frac{n!}{k! \times (n-k)!}. \end{aligned}$$

Число сочетаний

Вычисление C_n^k — комбинаторное решение:

- Выбираем первый объект n способами.
- Выбираем второй объект из оставшихся $n-1$ способом.
- ...
- Выбираем k-й объект из оставшихся $n-k+1$ способом.
- Каждое сочетание получилось $k!$ раз в различных порядках.
- Поэтому поделим ответ на $k! = 1 \cdot 2 \cdot 3 \cdots k$.

Итоговая формула:

$$C_n^k = \frac{n!}{k! \times (n-k)!}$$

Пример: $n = 4$, $k = 2$. Получаем

$$C_4^2 = \frac{1 \cdot 2 \cdot 3 \cdot 4}{1 \cdot 2 \times 1 \cdot 2} = \frac{24}{4} = 6.$$

Число сочетаний

Вычисление C_n^k — сведéние к подзадачам:

- Рассмотрим объект с номером n.
- Если мы выберем его, из $n-1$ оставшегося объекта нужно будет выбрать ещё $k-1$.
- Если мы не выберем его, из $n-1$ оставшегося объекта нужно будет выбрать ещё k.

Получаем, что

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$

База: $C_0^0 = 1$, $C_n^k = 0$ при $k < 0$ или $k > n$.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

Число сочетаний

n \ k	−1	0	1	2	3	4
0	0	1	0	0	0	0
1	0	?	?	0	0	0
2	0	?	?	?	0	0
3	0	?	?	?	?	0
4	0	?	?	?	?	?

Получаем, что

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$

База: $C_0^0 = 1$, $C_n^k = 0$ при $k < 0$ или $k > n$.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

Число сочетаний

n \ k	−1	0	1	2	3	4
0	0	1	0	0	0	0
1	0	1	1	0	0	0
2	0	1	2	1	0	0
3	0	1	3	3	1	0
4	0	1	4	6	4	1

Получаем, что

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$

База: $C_0^0 = 1$, $C_n^k = 0$ при $k < 0$ или $k > n$.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

Число сочетаний

n \ k	−1	0	1	2	3	4
0	0	1	0	0	0	0
1	0	1	1	0	0	0
2	0	1	2	1	0	0
3	0	1	3	3	1	0
4	0	1	4	6	4	1

Получаем, что

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$

База: $C_0^0 = 1$, $C_n^k = 0$ при $k < 0$ или $k > n$.
Пример: $n = 4$, $k = 2$. Получаем $C_4^2 = 6$.

Число сочетаний

n \ k	−1	0	1	2	3	4
0	0	1	0	0	0	0
1	0	1	1	0	0	0
2	0	1	2	1	0	0
3	0	1	3	3	1	0
4	0	1	4	6	4	1

Псевдокод:

```
for n := 0, ..., maxN:
  c[n][0] := 1
  for k := 1, ..., n:
    c[n][k] := c[n - 1][k - 1] + c[n - 1][k]
```

Наибольшая общая подпоследовательность

Определения:

- Подпоследовательность последовательности $p_1, p_2, \ldots, p_k$ (будем обозначать как $p_{1..k}$) — это последовательность $p_{i_1}, p_{i_2}, \ldots, p_{i_r}$ для индексов $1 \leqslant i_1 < i_2 < \ldots < i_r \leqslant k$.
- Общая подпоследовательность $a_{1..m}$ и $b_{1..n}$ — это последовательность, являющаяся подпоследовательностью и для a, и для b.

Постановка задачи:

- Есть две последовательности $a_{1..m}$ и $b_{1..n}$.
- Требуется найти общую подпоследовательность максимальной длины.

Из чего именно состоят последовательности — не важно.
Мы будем рассматривать последовательности целых чисел.

Наибольшая общая подпоследовательность

Пример:

- $a = (1, 2, 4, 6, 3)$, $m = 5$
- $b = (5, 1, 4, 2, 6, 3, 2)$, $n = 7$

Наибольшая общая подпоследовательность

Пример:

- $a = (1, 2, 4, 6, 3)$, $m = 5$
- $b = (5, 1, 4, 2, 6, 3, 2)$, $n = 7$
- Наибольшая общая подпоследовательность:
  $p = (1, 2, 6, 3)$, её длина равна 4.
  Как подпоследовательность a: $p = (a_1, a_2, a_4, a_5)$.
  Как подпоследовательность b: $p = (b_2, b_4, b_5, b_6)$.

Наибольшая общая подпоследовательность

Пример:

- $a = (1, 2, 4, 6, 3)$, $m = 5$
- $b = (5, 1, 4, 2, 6, 3, 2)$, $n = 7$
- Наибольшая общая подпоследовательность:
  $p = (1, 2, 6, 3)$, её длина равна 4.
  Как подпоследовательность a: $p = (a_1, a_2, a_4, a_5)$.
  Как подпоследовательность b: $p = (b_2, b_4, b_5, b_6)$.
- Другой оптимальный ответ:
  $q = (1, 4, 6, 3)$, её длина также равна 4.
  Как подпоследовательность a: $q = (a_1, a_3, a_4, a_5)$.
  Как подпоследовательность b: $q = (b_2, b_3, b_5, b_6)$.

Наибольшая общая подпоследовательность

Сводим к следующим подзадачам:

- Даны префиксы последовательностей a и b: $a_{1..i}$ и $b_{1..j}$.
- Найдём их наибольшую общую подпоследовательность.

Решение подзадачи для $a_{1..i}$ и $b_{1..j}$:

- Пусть оптимальное решение непусто (иначе всё очевидно).
- Рассмотрим последнее число в нём.
- Это число равно какому-то a_x $(1 \leqslant x \leqslant i)$.
- Это число также равно какому-то b_y $(1 \leqslant y \leqslant j)$.
- Значит, для получения оптимального решения нужно решить подзадачу для префиксов $a_{1..x-1}$ и $b_{1..y-1}$ и приписать к полученному решению это число (оно равно a_x и b_y).
- Если пар (x, y) несколько, ...

Наибольшая общая подпоследовательность

Сводим к следующим подзадачам:

- Даны префиксы последовательностей a и b: $a_{1..i}$ и $b_{1..j}$.
- Найдём их наибольшую общую подпоследовательность.

Решение подзадачи для $a_{1..i}$ и $b_{1..j}$:

- Пусть оптимальное решение непусто (иначе всё очевидно).
- Рассмотрим последнее число в нём.
- Это число равно какому-то a_x $(1 \leqslant x \leqslant i)$.
- Это число также равно какому-то b_y $(1 \leqslant y \leqslant j)$.
- Значит, для получения оптимального решения нужно решить подзадачу для префиксов $a_{1..x-1}$ и $b_{1..y-1}$ и приписать к полученному решению это число (оно равно a_x и b_y).
- Если пар (x, y) несколько, ...

Наибольшая общая подпоследовательность

Сводим к следующим подзадачам:

- Даны префиксы последовательностей a и b: $a_{1..i}$ и $b_{1..j}$.
- Найдём их наибольшую общую подпоследовательность.

Решение подзадачи для $a_{1..i}$ и $b_{1..j}$:

- Пусть оптимальное решение непусто (иначе всё очевидно).
- Рассмотрим последнее число в нём.
- Это число равно какому-то a_x $(1 \leqslant x \leqslant i)$.
- Это число также равно какому-то b_y $(1 \leqslant y \leqslant j)$.
- Значит, для получения оптимального решения нужно решить подзадачу для префиксов $a_{1..x-1}$ и $b_{1..y-1}$ и приписать к полученному решению это число (оно равно a_x и b_y).
- Если пар (x, y) несколько, найдём лучшее из решений для всех таких пар.

Наибольшая общая подпоследовательность

Сводим к следующим подзадачам:

- Даны префиксы последовательностей a и b: $a_{1..i}$ и $b_{1..j}$.
- Найдём их наибольшую общую подпоследовательность.

Решение подзадачи для $a_{1..i}$ и $b_{1..j}$:

- Пусть оптимальное решение непусто (иначе всё очевидно).
- Рассмотрим последнее число в нём.
- Это число равно какому-то a_x $(1 \leqslant x \leqslant i)$.
- Это число также равно какому-то b_y $(1 \leqslant y \leqslant j)$.
- Значит, для получения оптимального решения нужно решить подзадачу для префиксов $a_{1..x-1}$ и $b_{1..y-1}$ и приписать к полученному решению это число (оно равно a_x и b_y).
- Если пар (x, y) несколько, можно рассматривать только максимальные x и y.

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	2	0	0	0	0	2	0	0	2
4	3	0	0	0	2	0	0	0	0
6	4	0	0	0	0	0	3	0	0
3	5	0	0	0	0	0	0	4	0

Пример:

- $a = (1, 2, 4, 6, 3)$, $m = 5$
- $b = (5, 1, 4, 2, 6, 3, 2)$, $n = 7$

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	2	0	0	0	0	2	0	0	2
4	3	0	0	0	2	0	0	0	0
6	4	0	0	0	0	0	3	0	0
3	5	0	0	0	0	0	0	4	0

- $f(i,j)$ — ответ для $a_{1..i}$ и $b_{1..j}$ при условии, что $a_i = b_j$
- База: $f(i,0) = f(0,j) = 0$
- Ответ: $\max\limits_{i=0..m} \max\limits_{j=0..n} f(i,j)$
- Асимптотика: $O(m^2n^2)$ времени, $O(mn)$ памяти

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	2	0	0	0	0	2	0	0	2
4	3	0	0	0	2	0	0	0	0
6	4	0	0	0	0	0	3	0	0
3	5	0	0	0	0	0	0	4	0

```
for i := 1, ..., m:
  for j := 1, ..., n:
    if a[i] = b[j]:
      for u := 0, ..., i - 1:
        for v := 0, ..., j - 1:
          f[i][j] := max (f[i][j], f[u][v] + 1)
```

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1
2	2	0	0	1	1	2	2	2	2
4	3	0	0	1	2	2	2	2	2
6	4	0	0	1	2	2	3	3	3
3	5	0	0	1	2	2	3	4	4

Пример:

- $a = (1, 2, 4, 6, 3)$, $m = 5$
- $b = (5, 1, 4, 2, 6, 3, 2)$, $n = 7$

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1
2	2	0	0	1	1	2	2	2	2
4	3	0	0	1	2	2	2	2	2
6	4	0	0	1	2	2	3	3	3
3	5	0	0	1	2	2	3	4	4

- $g(i,j)$ — лучший из ответов в прямоугольнике $[0..i] \times [0..j]$
- База: $g(i,0) = g(0,j) = 0$
- Ответ: $g(m,n)$
- Асимптотика: $O(m^2n^2)$ времени (???), $O(mn)$ памяти

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1
2	2	0	0	1	1	2	2	2	2
4	3	0	0	1	2	2	2	2	2
6	4	0	0	1	2	2	3	3	3
3	5	0	0	1	2	2	3	4	4

- $g(i,j)$ — лучший из ответов в прямоугольнике $[0..i] \times [0..j]$
- Все решения, в которых последнее число не является одновременно a_i и b_j, уже учтены либо в прямоугольнике $[0..i] \times [0..j-1]$, либо в прямоугольнике $[0..i-1] \times [0..j]$
- Асимптотика: $O(mn)$ времени, $O(mn)$ памяти

Наибольшая общая подпоследовательность

	b_j		**5**	**1**	**4**	**2**	**6**	**3**	**2**
a_i	$i \setminus j$	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1
2	2	0	0	1	1	2	2	2	2
4	3	0	0	1	2	2	2	2	2
6	4	0	0	1	2	2	3	3	3
3	5	0	0	1	2	2	3	4	4

```
for i := 1, ..., m:
  for j := 1, ..., n:
    g[i][j] := max (g[i][j - 1], g[i - 1][j])
    if a[i] = b[j]:
      g[i][j] := max (g[i][j], g[i - 1][j - 1] + 1)
```

Редакционное расстояние

Определение:

- Редакционное расстояние (или расстояние Левенштейна) $d(s, t)$ между двумя строками s и t — это минимальное количество добавлений, удалений и замен символов, которые нужно сделать с s, чтобы получить t.

Пример:

- $d(\texttt{gets}, \texttt{goat}) = 3$: $\texttt{gets} \xrightarrow{-\texttt{s}} \texttt{get} \xrightarrow{\texttt{e}\to\texttt{o}} \texttt{got} \xrightarrow{+\texttt{a}} \texttt{goat}$.
  Другой способ: $\texttt{gets} \xrightarrow{\texttt{e}\to\texttt{o}} \texttt{gots} \xrightarrow{\texttt{t}\to\texttt{a}} \texttt{goas} \xrightarrow{\texttt{s}\to\texttt{t}} \texttt{goat}$.

Постановка задачи:

- По двум данным строкам требуется найти редакционное расстояние между ними.

Замечание:

- Строки s и t равноправны: к каждому действию есть обратное. Если выполнить обратные действия в обратном порядке, из t получится s.

Редакционное расстояние

Сводим к следующим подзадачам:

- Даны префиксы (начала) строк s и t: $s_{1..i}$ (префикс строки s длины i) и $t_{1..j}$ (префикс строки t длины j).
- Найдём редакционное расстояние между ними.

Решение подзадачи для $s_{1..i}$ и $t_{1..j}$:

- Если $s_i = t_j$, перейдём к решению подзадачи для $s_{1..i-1}$ и $t_{1..j-1}$.
- В противном случае оптимальная последовательность действий непуста, и либо с s_i, либо с t_j случилось хотя бы одно действие.
- Рассмотрим какое-нибудь из этих действий.
- Если это удаление, то это удаление символа s_i. Делаем его и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j}$.
- Если это добавление, то это добавление символа t_j. Делаем его и переходим к подзадаче для $s_{1..i}$ и $t_{1..j-1}$.
- Если это замена, то это замена символа s_i на символ t_j. Делаем её и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j-1}$.

Редакционное расстояние

Сводим к следующим подзадачам:

- Даны префиксы (начала) строк s и t: $s_{1..i}$ (префикс строки s длины i) и $t_{1..j}$ (префикс строки t длины j).
- Найдём редакционное расстояние между ними.

Решение подзадачи для $s_{1..i}$ и $t_{1..j}$:

- Если $s_i = t_j$, перейдём к решению подзадачи для $s_{1..i-1}$ и $t_{1..j-1}$.
- В противном случае оптимальная последовательность действий непуста, и либо с s_i, либо с t_j случилось хотя бы одно действие.
- Рассмотрим какое-нибудь из этих действий.
- Если это удаление, то это удаление символа s_i. Делаем его и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j}$.
- Если это добавление, то это добавление символа t_j. Делаем его и переходим к подзадаче для $s_{1..i}$ и $t_{1..j-1}$.
- Если это замена, то это замена символа s_i на символ t_j. Делаем её и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j-1}$.

Редакционное расстояние

Сводим к следующим подзадачам:

- Даны префиксы (начала) строк s и t: $s_{1..i}$ (префикс строки s длины i) и $t_{1..j}$ (префикс строки t длины j).
- Найдём редакционное расстояние между ними.

Решение подзадачи для $s_{1..i}$ и $t_{1..j}$:

- Если $s_i = t_j$, перейдём к решению подзадачи для $s_{1..i-1}$ и $t_{1..j-1}$.
- В противном случае оптимальная последовательность действий непуста, и либо с s_i, либо с t_j случилось хотя бы одно действие.
- Рассмотрим какое-нибудь из этих действий.
- Если это удаление, то это удаление символа s_i. Делаем его и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j}$.
- Если это добавление, то это добавление символа t_j. Делаем его и переходим к подзадаче для $s_{1..i}$ и $t_{1..j-1}$.
- Если это замена, то это замена символа s_i на символ t_j. Делаем её и переходим к подзадаче для $s_{1..i-1}$ и $t_{1..j-1}$.

Редакционное расстояние

Пример:

- $s =$ `gets`
- $t =$ `goat`

	t_j		g	o	a	t
s_i	$i \setminus j$	0	1	2	3	4
	0	0	1	2	3	4
g	1	1	0	1	2	3
e	2	2	1	1	2	3
t	3	3	2	2	2	2
s	4	4	3	3	3	3

- База: $d(i, 0) = i$, $d(0, j) = j$
- Ответ: $d(m, n)$, где m — длина строки s, а n — длина строки t
- Асимптотика: $O(mn)$ времени, $O(mn)$ памяти

Редакционное расстояние

Пример:

- $s =$ gets
- $t =$ goat

	t_j		g	o	a	t
s_i	$i \setminus j$	0	1	2	3	4
	0	0	1	2	3	4
g	1	1	0	1	2	3
e	2	2	1	1	2	3
t	3	3	2	2	2	2
s	4	4	3	3	3	3

Псевдокод:

```
for i := 1, ..., m:
  for j := 1, ..., n:
    if s[i] = t[j]:
      d[i][j] := d[i - 1][j - 1]
    else:
      d[i][j] := 1 + min (d[i - 1][j    ],
                          d[i - 1][j - 1],
                          d[i    ][j - 1])
```

Строка и шаблон

Определения:

- Шаблон — это строка, которая может содержать специальные символы ‘?’ и ‘*’.
- Обычному символу шаблона можно поставить в соответствие ровно один такой же символ строки.
- Специальному символу ‘?’ можно поставить в соответствие ровно один любой символ строки.
- Специальному символу ‘*’ можно поставить в соответствие любую (в том числе и пустую) последовательность любых символов строки.
- Строка s длины m соответствует шаблону p длины n, если она разбивается на n частей так, что i-я часть строки соответствует i-му символу шаблона.

Строка и шаблон

Постановка задачи:

- По данным s и p нужно определить, соответствует ли строка s шаблону p.

Пример:

- $s =$ `button`, $p =$ `?u*t*n`
- Соответствие:

```
b  u  t  t  o  n
?  u  *  t  *  n
```

- Другой способ:

```
b  u     t  to  n
?  u  *  t   *   n
```

Строка и шаблон

Сводим к следующим подзадачам:

- Даны префиксы (начала) строки s и шаблона t: $s_{1..i}$ (префикс строки s длины i) и $p_{1..j}$ (префикс шаблона p длины j).
- Выясним, есть ли между ними соответствие.

Решение подзадачи для $s_{1..i}$ и $p_{1..j}$:

- Если p_j — обычный символ, сравним его с s_i и в случае равенства перейдём к решению подзадачи для $s_{1..i-1}$ и $p_{1..j-1}$.
- Если $p_j = ?$, перейдём к решению подзадачи для $s_{1..i-1}$ и $p_{1..j-1}$.
- Если $p_j = *$, соответствие есть тогда и только тогда, когда соответствие есть хотя бы в одной из подзадач для $s_{1..k}$ и $p_{1..j-1}$ при $0 \leqslant k \leqslant i$.

Замечание:

- Чтобы база была простой, допишем к обеим строкам спереди одинаковый специальный символ, например, #.

Строка и шаблон

Сводим к следующим подзадачам:

- Даны префиксы (начала) строки s и шаблона t: $s_{1..i}$ (префикс строки s длины i) и $p_{1..j}$ (префикс шаблона p длины j).
- Выясним, есть ли между ними соответствие.

Решение подзадачи для $s_{1..i}$ и $p_{1..j}$:

- Если p_j — обычный символ, сравним его с s_i и в случае равенства перейдём к решению подзадачи для $s_{1..i-1}$ и $p_{1..j-1}$.
- Если $p_j = ?$, перейдём к решению подзадачи для $s_{1..i-1}$ и $p_{1..j-1}$.
- Если $p_j = *$, соответствие есть тогда и только тогда, когда соответствие есть хотя бы в одной из подзадач для $s_{1..k}$ и $p_{1..j-1}$ при $0 \leqslant k \leqslant i$.

Замечание:

- Чтобы база была простой, допишем к обеим строкам спереди одинаковый специальный символ, например, #.

Строка и шаблон

	s_i	#	b	u	t	t	o	n
p_j	$j \setminus i$	1	2	3	4	5	6	7
#	1	1	0	0	0	0	0	0
?	2	0	1	0	0	0	0	0
u	3	0	0	1	0	0	0	0
*	4	0	0	1	1	1	1	1
t	5	0	0	0	1	1	0	0
*	6	0	0	0	1	1	1	1
n	7	0	0	0	0	0	0	1

- База: $f(0,0) = 1$, $f(i,0) = 0$ при $i > 0$, $f(0,j) = 0$ при $j > 0$
- Ответ: $f(m,n)$
- Асимптотика: $O(m^2 n)$ времени (???), $O(mn)$ памяти

Строка и шаблон

Псевдокод:

```
for i := 1, ..., m:
  for j := 1, ..., n:
    if p[j] = '*':
      f[i][j] := 0
      for k := 0, ..., i:
        f[i][j] := f[i][j] or f[k][j - 1]
    else if p[j] = '?':
      f[i][j] := f[i - 1][j - 1]
    else:
      f[i][j] := f[i - 1][j - 1] and (s[i] = p[j])
```

- База: $f(0,0) = 1$, $f(i,0) = 0$ при $i > 0$, $f(0,j) = 0$ при $j > 0$
- Ответ: $f(m,n)$
- Асимптотика: $O(m^2n)$ времени (???), $O(mn)$ памяти

Строка и шаблон

	s_i	#	b	u	t	t	o	n
p_j	$j \setminus i$	1	2	3	4	5	6	7
#	1	1	0	0	0	0	0	0
?	2	0	1	0	0	0	0	0
u	3	0	0	1	0	0	0	0
*	4	0	0	1	1	1	1	1
t	5	0	0	0	1	1	0	0
*	6	0	0	0	1	1	1	1
n	7	0	0	0	0	0	0	1

При $p_j =$ *:

- $f(i,j) = f(0,j-1) \vee \ldots \vee f(i-1,j-1) \vee f(i,j-1)$
- $f(i-1,j) = f(0,j-1) \vee \ldots \vee f(i-1,j-1)$
- Значит, $f(i,j) = f(i-1,j) \vee f(i,j-1)$

Строка и шаблон

Псевдокод:

```
for i := 1, ..., m:
  for j := 1, ..., n:
    if p[j] = '*':
      f[i][j] := f[i - 1][j] or f[i][j - 1]
    else if p[j] = '?':
      f[i][j] := f[i - 1][j - 1]
    else:
      f[i][j] := f[i - 1][j - 1] and (s[i] = p[j])
```

- База: $f(0,0) = 1$, $f(i,0) = 0$ при $i > 0$, $f(0,j) = 0$ при $j > 0$
- Ответ: $f(m,n)$
- Асимптотика: $O(mn)$ времени, $O(mn)$ памяти

Вопросы?

Вопросы?