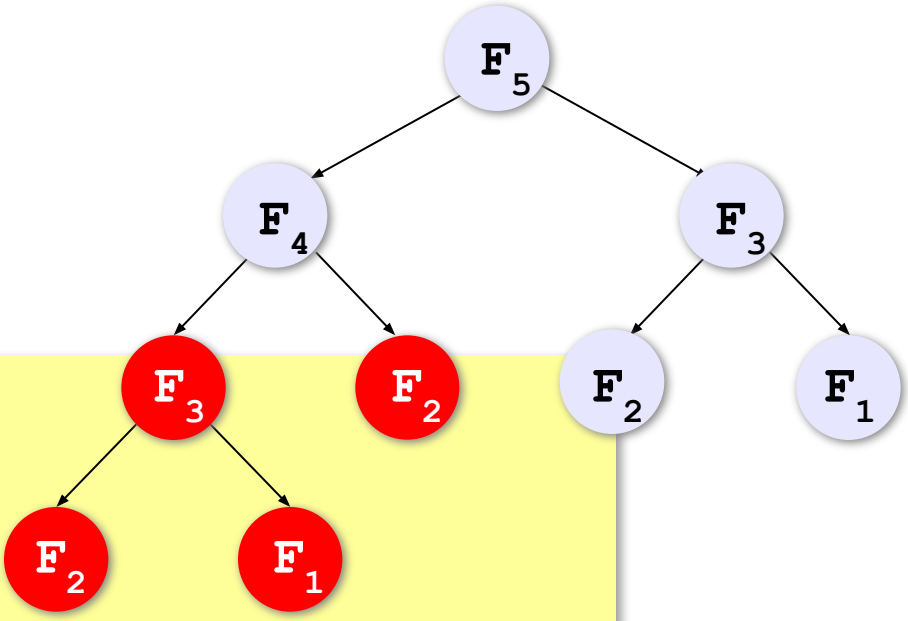
Динамическое программирование

Что такое динамическое программирование?

Числа Фибоначчи:

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \text{ при } n > 2$$




```
int Fib ( int N )
{
  if ( N<3 )
        return 1;
  else return Fib(N-1) + Fib(N-2);
}
```

повторное вычисление тех же значений

Запоминать то, что вычислено!

Динамическое программирование

F_1	F_2	F_3	F_4	F_5
1	1			

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \text{ при } n > 2$$

Объявление массива:

```
const int N=10;
int F[N+1]; // чтобы начать с 1
```

Заполнение массива:

```
F[1]=1; F[2]=1;
for ( i=3; i<=N; i++ )
  F[i]=F[i-1]+F[i-2];
```

F_{45}: рекурсия: **8 с**

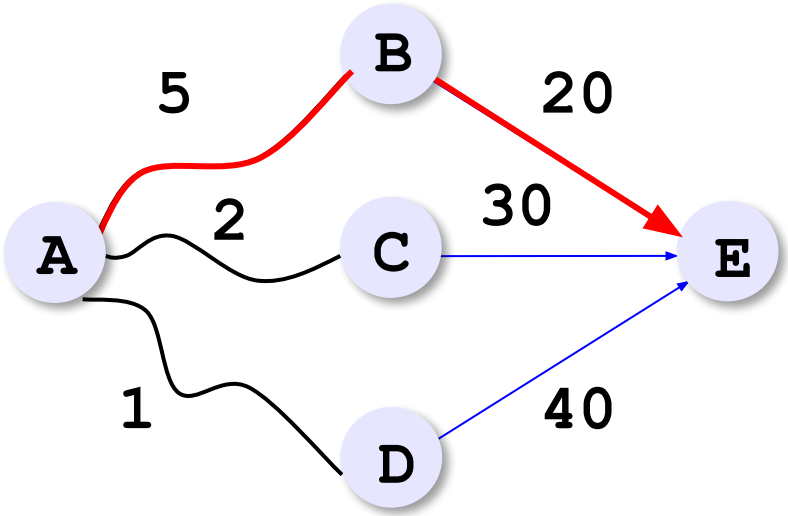
дин. программирование: **< 0,01 с**

? Можно ли обойтись без массива?

нужны только два последних!

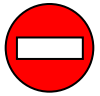
Динамическое программирование

Динамическое программирование – это способ решения сложных задач путем сведения их к более простым задачам того же типа.




```
ABE:  5 + 20 = 25
ACE:  2 + 30 = 32
ADE:  1 + 40 = 41
```


увеличение скорости

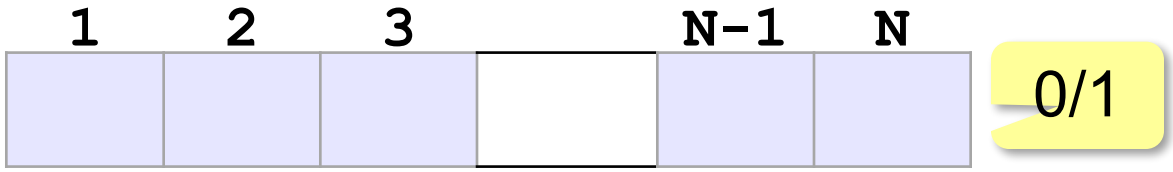

дополнительный расход памяти

Количество вариантов

Задача. Найти количество K_N цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд единиц.

Решение «в лоб»:

битовые цепочки



- построить все возможные цепочки
- проверить каждую на «правильность»

? Сколько возможных цепочек? 2^N

Сложность алгоритма $O(2^N)$

Количество вариантов

Задача. Найти количество K_N цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд единиц.

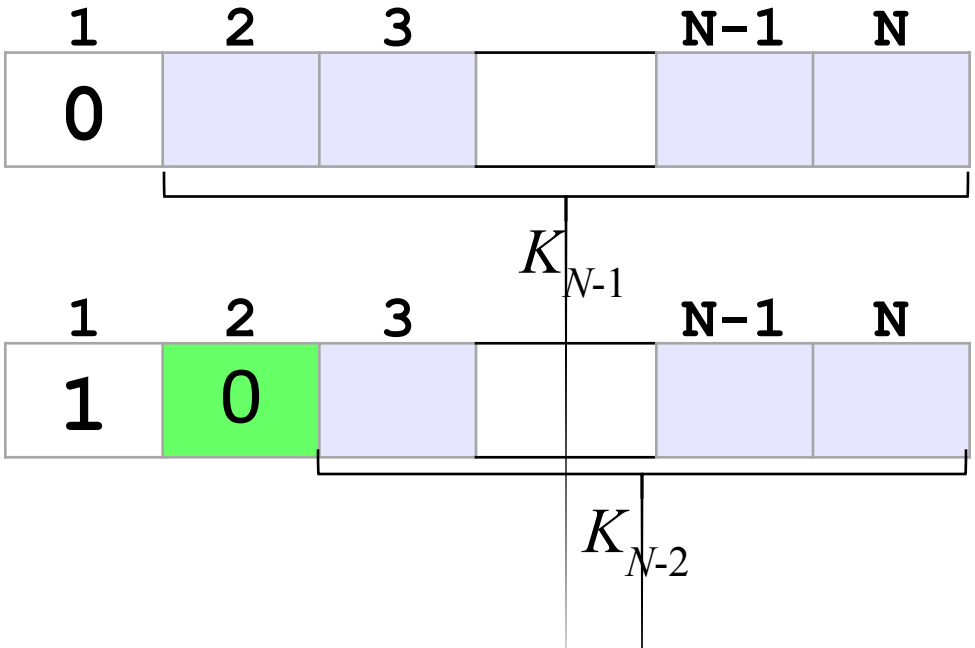
Простые случаи:

$N = 1$: **0** **1** $K_1 = 2$

$N = 2$: **00** **01** **10** $K_2 = 3$

$$K_N = K_{N-1} + K_{N-2} = F_{N+2}$$

Общий случай:



K_{N-1} «правильных» цепочек начинаются с нуля!

K_{N-2} «правильных» цепочек начинаются с единицы!

Оптимальное решение

Задача. В цистерне N литров молока. Есть бидоны объемом 1, 5 и 6 литров. Нужно разлить молоко в бидоны так, чтобы все бидоны были заполнены и количество используемых бидонов было **минимальным**.

Перебор?

при больших N – очень долго!

«Жадный алгоритм»?

$N = 10$: 10 = 6 + 1 + 1 + 1 + 1 $K = 5$

10 = 5 + 5 $K = 2$

! Не даёт оптимального решения!

Жадный алгоритм – это многошаговый алгоритм, в котором на каждом шаге принимается решение, лучшее в данный момент.

Оптимальное решение

K_N – минимальное число бидонов для N литров

Сначала выбрали бидон…

1 л: $K_N = 1 + K_{N-1}$

5 л: $K_N = 1 + K_{N-5}$

6 л: $K_N = 1 + K_{N-6}$

} min

Рекуррентная формула:

$$K_N = 1 + \min(K_{N-1}, K_{N-5}, K_{N-6}) \quad \text{при } N \geq 6$$

$$K_N = 1 + \min(K_{N-1}, K_{N-5}) \quad \text{при } N = 5$$

$$K_N = 1 + K_{N-1} \quad \text{при } N < 5$$

Оптимальное решение (бидоны)

$$K_N = 1 + \min(K_{N-1}, K_{N-5}, K_{N-6})$$

N	0	1	2	3	4	5	6	7	8	9	10
K_N	0	1	2	3	4	1	1	2	3	4	2
P	0	1	1	1	1	5	6	1	1	1	5

объём бидона, взятого последним

N	0	1	2	3	4	5	6	7	8	9	10
K_N	0	1	2	3	4	1	1	2	3	4	2
P	0	1	1	1	1	5	6	1	1	1	5

2 бидона 5 + 5

Похоже на алгоритм Дейкстры!

Задача о куче

Задача. Из камней весом p_i (i=1, …, N) набрать кучу весом ровно W или, если это невозможно, максимально близкую к W (но меньшую, чем W).

Решение «в лоб»:

1	2	3		N-1	N
1	0	0		1	0

камень взят

камень не взят

! Выбрать лучшую цепочку!

? Сколько возможных цепочек? 2^N

Сложность алгоритма $O(2^N)$

Задача о куче

Задача. Из камней весом p_i (i=1, …, N) набрать кучу весом ровно W или, если это невозможно, максимально близкую к W (но меньшую, чем W).

Идея: сохранять в массиве решения всех более простых задач этого типа (при меньшем количестве камней N и меньшем весе W).

Пример: W = 8, камни 2, 4, 5 и 7

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0								
3	5	0								
4	7	0								

w

базовые случаи

i

p_i

`T[i][w]` – оптимальный вес, полученный для кучи весом `w` из `i` первых по счёту камней.

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0								
4	7	0								

Добавляем камень с весом 4:

для `w < 4` ничего не меняется!

для `w ≥ 4`:

если его не брать: `T[2][w] = T[1][w]`

если его взять: `T[2][w] = 4 + T[1][w-4]`

? Какой вариант выбрать?

`max`

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0	0	2	2	4	5	6	7	7
4	7	0								

Добавляем камень с весом 5:

для `w < 5` ничего не меняется!

для `w ≥ 5`:

если его не брать: `T[3][w] = T[2][w]`

если его взять: `T[3][w] = 5 + T[2][w-5]`

`max`

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0	0	2	2	4	5	6	7	7
4	7	0	0	2	2	4	5	6	7	7

Добавляем камень с весом 7:

для `w < 7` ничего не меняется!

для `w ≥ 7`:

если его не брать: `T[4][w] = T[3][w]`

если его взять: `T[4][w] = 7 + T[3][w-7]`

max

Задача о куче

Добавляем камень с весом p_i:

для $w < p_i$ ничего не меняется!

для $w \geq p_i$:

если его не брать: `T[i][w] = T[i-1][w]`

если его взять: $T[i][w] = p_i + T[i-1][w-p_i]$

max

Рекуррентная формула:

при $w < p_i$: `T[i][w] = T[i-1][w]`

при $w \geq p_i$: `T[i][w] = max(T[i-1][w],`

p_i`+T[i-1][w-`p_i`])`

Задача о куче

? Какие камни нужно взять?

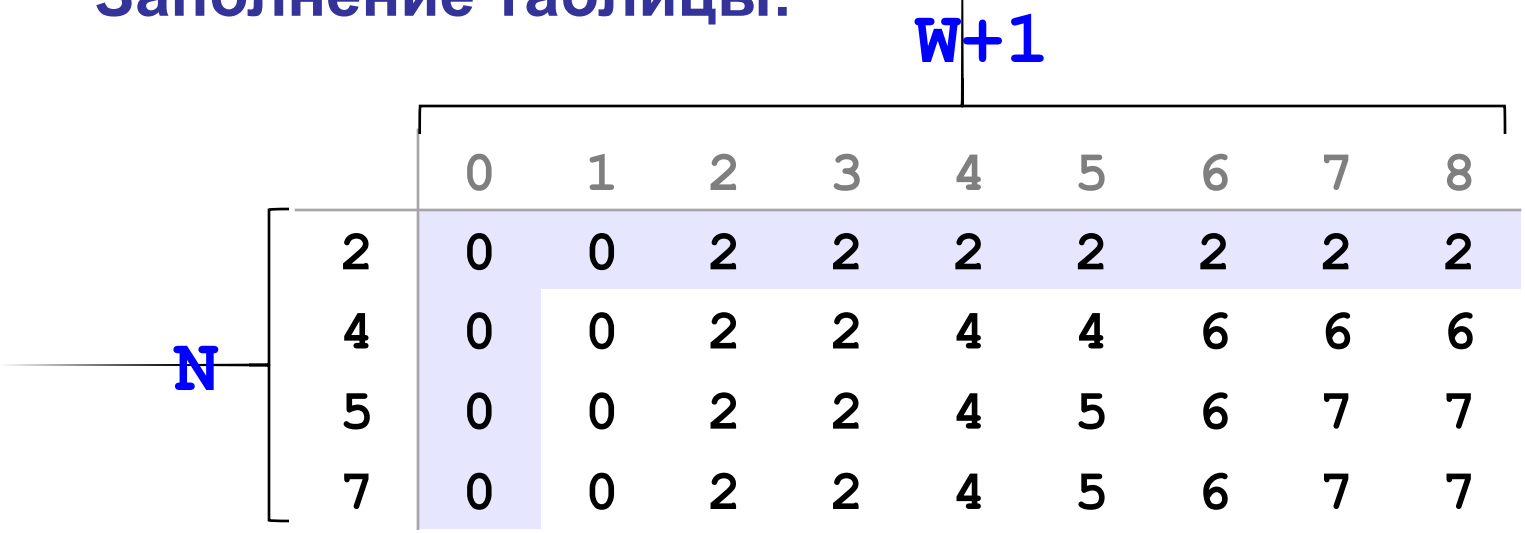
	0	1	2	3	4	5	6	7	8
2	0	0	2	2	2	2	2	2	2
4	0	0	2	2	4	4	6	6	6
5	0	0	2	2	4	5	6	7	7
7	0	0	2	2	4	5	6	7	7

Оптимальный вес 7 5 + 2

Задача о куче

? Какова сложность алгоритма?

Заполнение таблицы:



	0	1	2	3	4	5	6	7	8
2	0	0	2	2	2	2	2	2	2
4	0	0	2	2	4	4	6	6	6
5	0	0	2	2	4	5	6	7	7
7	0	0	2	2	4	5	6	7	7

! Сложность $O(N \cdot W)$!

псевдополиномиальный

Количество программ

Задача. У исполнителя Утроитель есть команды:

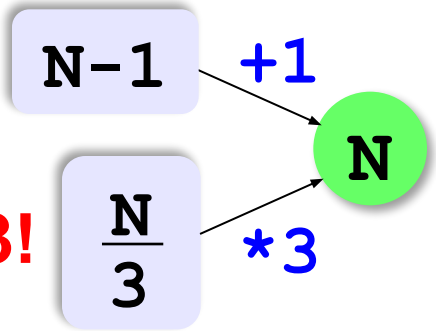
1. **прибавь 1**
2. **умножь на 3**

Сколько есть разных программ, с помощью которых можно из числа **1** получить число **20**?

Как решать, не выписывая все программы?

Количество программ

Как получить число N:



последняя команда

если делится на 3!

Рекуррентная формула:

$K_N = K_{N-1}$ если N не делится на 3

$K_N = K_{N-1} + K_{N/3}$ если N делится на 3

Количество программ

Рекуррентная формула:

$K_N = K_{N-1}$ если N не делится на 3

$K_N = K_{N-1} + K_{N/3}$ если N делится на 3

Заполнение таблицы:

N	1	2	3	4	5	6	7	8	9	10
K_N	1	1	2	2	2	3	3	3	5	5

одна пустая!

K_2+K_1

K_5+K_2

K_8+K_3

N	11	12	13	14	15	16	17	18	19	20
K_N	5	7	7	7	9	9	9	12	12	12

Количество программ

Только точки изменения:

20

N	1	3	6	9	12	15	18	21
K_N	1	2	3	5	7	9	12	15

Программа:

```
K[1] = 1;
for ( i = 2; i <= N; i ++ )
   {
   K[i] = K[i-1];
   if ( i % 3 == 0 )
      K[i] = K[i] + K[i/3];
   }
```

? Где ответ?

? Как объявить массив K?

Размен монет

Задача. Сколькими различными способами можно выдать сдачу размером W рублей, если есть монеты достоинством p_i (i=1, …, N)? В наборе есть монета достоинством 1 рубль ($p_1 = 1$).

Перебор?

при больших N и W – очень долго!

Динамическое программирование:

запоминаем решения всех задач меньшей размерности: для меньших значений W и меньшего числа монет N.

Размен монет

Пример: W = 10, монеты 1, 2, 5 и 10

		0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	1										
3	5	1										
4	10	1										

w

базовые случаи

i **p_i**

T[i][w] – количество вариантов для суммы **w** с использованием **i** первых по счёту монет.

Рекуррентная формула (добавили монету p_i):

при $w < p_i$: **T[i][w] = T[i-1][w]**

при $w \geq p_i$: **T[i][w] = T[i-1][w] + T[i][w-p_i]**

без этой монеты

все варианты размена остатка

Размен монет

Пример: W = 10, монеты 1, 2, 5 и 10

		0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	1	1	2	2	3	3	4	4	5	5	6
3	5	1	1	2	2	3	4	5	6	7	8	10
4	10	1	1	2	2	3	4	5	6	7	8	11

Рекуррентная формула (добавили монету p_i):

при $w < p_i$: `T[i,w] = T[i-1][w]`

при $w \geq p_i$: `T[i,w] = T[i-1][w] + T[i][w-`p_i`]`

Конец фильма

ПОЛЯКОВ Константин Юрьевич
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

ЕРЕМИН Евгений Александрович
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru