Динамическое программирование

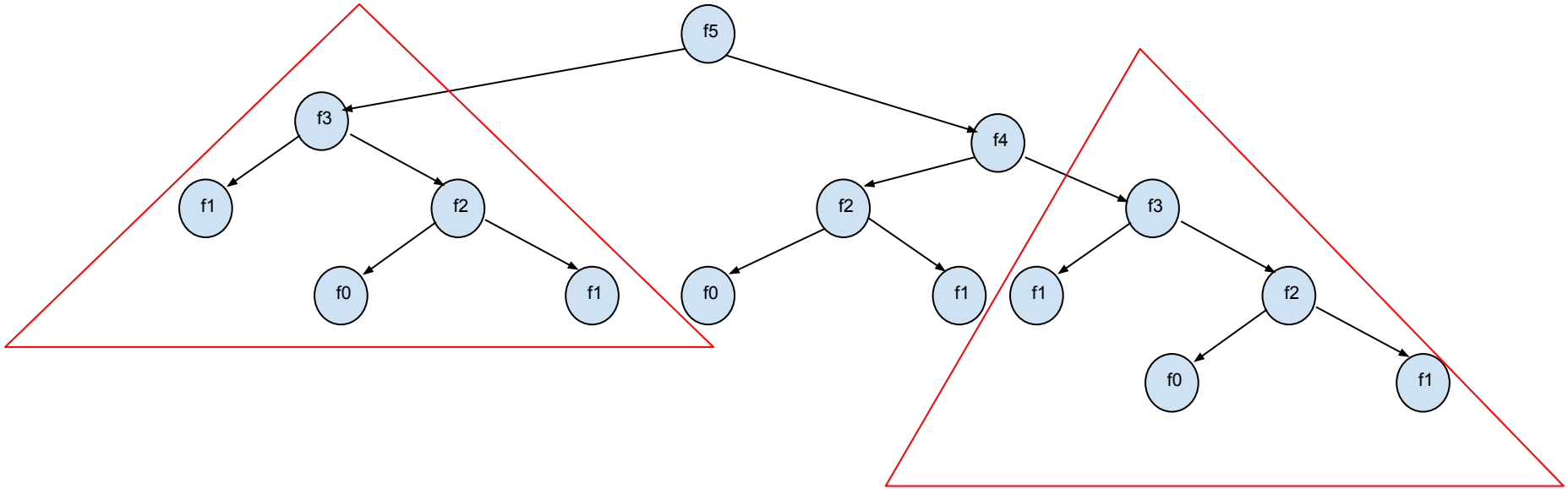
Преподаватель:
Макоян Артем Каренович

Числа Фибоначчи

Последовательность Фибоначчи задается правилом f(n) = f(n-1) + f(n-2), f(0)=0, f(1)=1. Необходимо посчитать, чему равно f(n).

Рекурсивный подход:

```
1   int f(int n) {
2       if (n == 0) {
3           return 0;
4       }
5       if (n == 1) {
6           return 1;
7       }
8       return f(n - 2) + f(n - 1);
9   }
10
```


f5
f3
f4
f1
f2
f2
f3
f0
f1
f0
f1
f1
f2
f0
f1

Числа Фибоначчи

Будем запоминать состояния в массиве f: f[i] = i-ое число Фибоначчи.

```
1   int f(int n) {
2       vector <int> f(n + 1);
3       f[0] = 0;
4       f[1] = 1;
5       for (int i = 2; i <= n; ++i) {
6           f[i] = f[i - 1] + f[i - 2];
7       }
8       return f[n];
9   }
10
```

Динамическое программирование

Идея: мы считаем значение для n, используя предыдущие

- **Состояние динамики:** число i - номер числа Фибоначчи
- **Значение динамики:** число f[i] - значение числа Фибоначчи
- **Переход динамики:** f[i] = f[i - 1] + f[i - 2] - правило пересчета нового значения динамики
- **База динамики:** f[0] = 0, f[1] = 1 - начальные значения
- **Обход динамики**: идем слева направо - то, в каком порядке мы считаем значения
- **Ответ:** f[n] - как мы получаем ответ

Уменьшение числа состояний.

Посчитать количество двоичных последовательностей длины n, в которых нет 2-ух рядом стоящих единиц.

Полный перебор: перебираем все двоичные последовательности длины n и проверяем в каких нету 2-ух подряд идущих единиц - O(2^n * n)

Уменьшение числа состояний.

Слишком много состояний (2^n).

Введем новые **состояния**: (n, c), где c = 0 или 1, а n - целое неотрицательное.

Значение динамики: dp(n, c) = количество чисел длины n, которые заканчиваются на c и в которых никакие 2 единицы не идут рядом.

Пересчет: dp(n, 0) = dp(n - 1, 0) + dp(n - 1, 1); dp(n, 1) = dp(n - 1, 0);

Итого: O(n).

Задача о кузнечике

Есть поляна - n подряд идущих клеток. Кузнечик сидит на 1-ой клетке, ему надо добраться до последней. Он может прыгать либо на 1 клетку вправо, либо на 2. Сколько существует способов добраться до последней клетки?

- **Состояние динамики:** число i - номер клетки
- **Значение динамики:** число способов добраться до клетки i
- **Переход динамики:** в клетку i мы можем попасть из i-1 и i-2, a[i] = a[i-1]+a[i-2]
- **База динамики:** a[1] = a[2] = 1
- **Обход динамики:** слева направо, циклом for
- **Ответ:** a[n]

Задача о кузнечике

Теперь пусть в некоторые клетки попадать нельзя (там лягушка!)

В таких клетках **значение динамики** будет a[i] = 0. В остальных клетках оно не изменится.

Тогда a[i] = a[i - 1] + a[i - 2], если i не заблокирована лягушкой и a[i] = 0 иначе.

Задача о кузнечике

Теперь кузнечик может прыгать вправо на k1, k2 … km клеток.

Новый **переход динамики:** a[i] = a[i - k1] + a[i - k2] + … + a[i - km] (нужно учитывать только те слагаемые, для которых i - kj >= 0).

Для заблокированных клеток решение тоже работает.

```cpp
int f(int n, vector <int> jump, vector <bool> close) {
    // close[i] = true, если клетка открыта, иначе close[i] = false
    // jump - вектор длин, на которые может прыгать кузнечик
    // Нумерация клеток от 0 до n - 1
    vector <int> dp(n, 0);
    if (!close[0]) {
        dp[0] = 1;
    }
    for (int i = 1; i < n; ++i) {
        if (close[i]) {
            dp[i] = 0;
        } else {
            for (auto len : jump) {
                if (i >= len) {
                    dp[i] += dp[i - len];
                }
            }
        }
    }
    return dp[n - 1];
}
```

Задача о банкомате

В стране X есть купюры номиналом x1, x2 … xm. Можно ли с их помощью набрать ровно n рублей (каждую купюру можно использовать неограниченное число раз)?

- **Состояние динамики:** число i - количество рублей
- **Значение динамики:** 1, если набрать можно, 0 - если нельзя
- **Переход динамики:** если dp[i - x1] = 1 ИЛИ dp[i - x2] = 1 … ИЛИ dp[i - xm] = 1, то тогда мы можем набрать и i (добавив xj к i-xj), то есть dp[i] = 1, иначе dp[i] = 0
- **База динамики:** a[0] = 1, так как мы можем набрать 0, ничего не взяв
- **Обход динамики:** слева-направо
- **Ответ:** если a[n] = 1, то можно, если a[n] = 0, то нельзя

Задача о банкомате

А теперь мы хотим узнать, какими именно купюрами мы можем набрать n рублей

Значение динамики: dp[i] = -1, если набрать нельзя, dp[i] = x, если последней купюрой мы должны взять x (то есть dp[i - x] не равно -1)

База: dp[0] = 0 - отличительный знак, так как мы ничего не брали, чтобы получить 0. Значит, и последней купюры нет.

```cpp
vector <int> f(int n, vector <int> x) {
    // n - сумма, которую хотим набрать
    // x - доступные купюры
    vector <int> dp(n + 1, -1);
    dp[0] = 0;
    for (int i = 1; i < n + 1; ++i) {
        for (auto bill : x) {
            if (i >= bill && dp[i - bill] != -1) {
                dp[i] = bill;
                break;
            }
        }
    }
    if (dp[n] == -1) {
        // Набрать нельзя - возвращаем пустой вектор
        return {};
    } else {
        vector <int> res;
        while (n > 0) {
            res.push_back(dp[n]);
            n -= dp[n];
        }
        // Возвращаем вектор купюр
        return res;
    }
}
```

Задача о рюкзаке

Есть N предметов с весами w1 … wN и стоимостями c1 … cN. Есть рюкзак, вмещающий вес M. Нужно сложить в рюкзак предметы максимальной стоимости, чтобы он не порвался.

Полный перебор: O(2^n * n).

Уменьшим количество состояний.

Задача о рюкзаке

- **Состояние**: (n, m), n - количество предметов (0 <= n <= N), m - вес (0 <= m <= M)
- **Значение**: dp[n][m] - ответ на задачу, если нужно набрать вес **ровно** m из предметов с номерами от 1 до n
- **Переход**: пусть мы набрали вес ровно m с максимальной стоимостью. Тогда мы либо не взяли предмет номер n (тогда dp[n - 1][m] - суммарная стоимость предметов), либо взяли (тогда dp[n - 1][m - wn] + cn - суммарная стоимость предметов. Итого: dp[n][m] = max(dp[n - 1][m], dp[n - 1][m - wn] + cn)
- **База**: dp[0][0] = 0, dp[0][m] = INF (m > 0).
- **Пересчет**: for (n = 1 … N), а внутри for (m = 0 … M)
- **Ответ**: максимум dp[N][m] для всех 0 <= m <= M

Задача о рюкзаке

Пусть N = 4, M = 9 и у нас есть предметы:

№	1	2	3	4
w	1	2	2	7
c	3	2	3	5

Тогда получаем двумерную таблицу:

n\m	0	1	2	3	4	5	6	7	8	9
0	0	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
1	0	3	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
2	0	3	2	5	-∞	-∞	-∞	-∞	-∞	-∞
3	0	3	3	6	5	8	-∞	-∞	-∞	-∞
4	0	3	3	6	5	8	-∞	5	8	8

Задача о рюкзаке. Восстановление ответа.

Вспомним пересчет динамики: dp[n][m] = max(dp[n - 1][m], dp[n - 1][m - wn] + cn)

Тогда мы можем переходить по “стрелочкам” (она ведет из текущего состояния в то, из которого мы пересчитались) из последней строки в нулевую, если перешли “вверх” (на {0, -1}), то не берем предмет, иначе (перешли на {-wn, -1}) - берем

n\m	0	1	2	3	4	5	6	7	8	9
0	0	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
1	0	3	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
2	0	3	2	5	-∞	-∞	-∞	-∞	-∞	-∞
3	0	3	3	6	5	8	-∞	-∞	-∞	-∞
4	0	3	3	6	5	8	-∞	5	8	8

```cpp
const int INF = 1e9;

vector <int> f(int n, int m, vector <int> w, vector <int> c) {
    // n - количество предметов
    // m - вес, который нужно набрать
    // w[i] - вес i-ого предмета
    // c[i] - стоимость i-ого предмета
    // Нумерация предметов с 1, то есть в w[0] и c[0] находятся фиктивные (ненужные) значения
    vector <vector <int> > dp(n + 1, vector <int> (m + 1, -INF));
    dp[0][0] = 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= m; ++j) {
            dp[i][j] = dp[i - 1][j];
            if (j >= w[i] && dp[i][j] < dp[i - 1][j - w[i]] + c[i]) {
                dp[i][j] = dp[i - 1][j - w[i]] + c[i];
            }
        }
    }
    vector <int> res;
    while (n > 0) {
        if (dp[n][m] != dp[n - 1][m]) {
            // Значит стрелочка наискосок
            res.push_back(n);
            m -= w[n];
        }
        n--;
    }
    reverse(res.begin(), res.end());
    return res;
}
```

Задача о рюкзаке. Вариации.

- Рюкзак должен вмещать **ровно** M веса
- Без стоимостей, нужно набрать вес ровно M
- Без стоимостей, набрать вес ровно M, выбрав наименьшее количество предметов
- Изначальная задача, но при равных стоимостях минимизировать количество