

Динамическое программирование.

Лекция 7

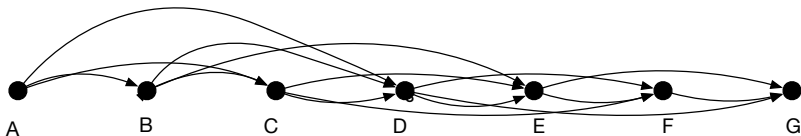
План лекции

- ▶ Задача о количестве маршрутов
- ▶ Задача о возрастающей подпоследовательности наибольшей длины
- ▶ Рекурсия как база динамического программирования
- ▶ Декомпозиция задач
- ▶ Восстановление решения
- ▶ Динамическое программирование и игры
- ▶ Уход от рекурсии. Восходящее решение
- ▶ Использование отображений для решения задач методом динамического программирования.
- ▶ Этапы решения задачи методом динамического программирования. Применимость динамического программирования
- ▶ Многомерные варианты

Задача о количестве маршрутов

Задача о количестве маршрутов

- ▶ Классическая задача. Требуется найти количество маршрутов из пункта A в пункт G .



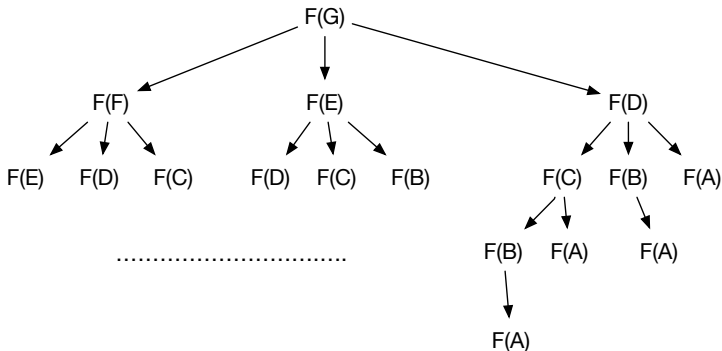
- ▶ Если обозначить число маршрутов из A до i как $F(i)$, то:
- ▶ $F(G) = F(F) + F(E) + F(D)$
- ▶ Аналогично:
- ▶ $F(F) = F(E) + F(D) + F(C)$
- ▶ $F(E) = F(D) + F(C) + F(B)$
- ▶ $F(D) = F(C) + F(B) + F(A)$
- ▶ $F(C) = F(B) + F(A)$
- ▶ $F(B) = F(A)$
- ▶ $F(A) = 1$

Задача о количестве маршрутов

- ▶ Это — рекурсивная задача.
- ▶ Каждая задача разбивается на подзадачи.
- ▶ Имеется операция консолидации результата.
- ▶ Каждая из подзадач решается аналогично основной, но несколько проще, чем главная.
- ▶ Имеются подзадачи, не требующие рекурсии.

Задача о количестве маршрутов

- Дерево рекурсии для такой задачи велико.



- Подзадачи частично совпадают.
- Это — задача динамического программирования.

Условия появления задачи *динамического программирования*:

- ▶ Задача может быть разбита на произвольное количество подзадач.
- ▶ Решение полной задачи зависит только от решения подзадач.
- ▶ Каждая из подзадач по какому-либо критерию *немного* проще основной задачи.
- ▶ Часть подзадач и подзадач у подзадач совпадает.

Два основных способа решения:

- ▶ Рекурсивное решение с запоминанием результатов.
- ▶ Восходящее решение в правильном порядке.

Условия применения метода *разделяй и властвуй*:

- ▶ Задача может быть разбита на произвольное количество b подзадач.
- ▶ Решение полной задачи зависит только от решения подзадач.
- ▶ Каждая из подзадач проще основной задачи не менее, чем в b раз.

Основной способ решения — рекурсия или эквивалентная ей итерация.

Принцип Беллмана

- ▶ Беллман, 1940-е годы, принцип оптимальности:
«Каковы бы ни были первоначальное состояние и решение, последующие решения должны составлять оптимальное поведение относительно уже решённого состояния.»
- ▶ Арис:
«Если вы не научились использовать наилучшим образом то, что у вас есть, вы не научитесь использовать то, что у вас будет»

Уравнение Беллмана

- ▶ Пусть имеется управляемая система.
- ▶ S — её текущее состояние.
- ▶ $W_i = f_i(S, x_i)$ — функция выигрыша(стоимости) при использовании управления x на i -м шаге.
- ▶ $S' = \varphi_i(S, x_i)$ — состояние, в которое переходит система при воздействии x

Независимо от значения S нужно выбрать управление на этом шаге так, чтобы выигрыш на данном шаге плюс оптимальный выигрыш на всех последующих шагах был максимальным.

$$W_i(S) = \max_{x_i} \{f_i(S, x_i) + W_{i+1}(\varphi_i(S, x_i))\}$$

Уравнение Беллмана для задачи о количестве маршрутов

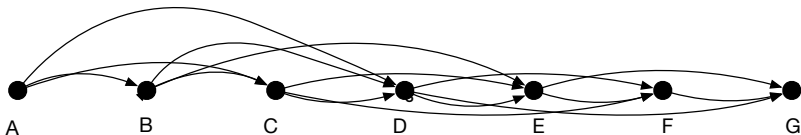
Пусть A — матрица смежности,

$$A[i, j] = \begin{cases} 1, & \text{если имеется прямой путь из } i \text{ в } j \\ 0, & \text{если не имеется прямого пути из } i \text{ в } j. \end{cases}$$

Тогда:

$$F(x) = \sum_{i=1}^k F(i) \cdot A[i, x]$$

Дорожный граф



Его матрица смежности

$$A = \begin{matrix} & \begin{matrix} A & B & C & D & E & F & G \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \end{matrix} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

Задача о возрастающей
подпоследовательности
наибольшей длины.

Задача о возрастающей подпоследовательности наибольшей длины

- ▶ Имеется последовательность чисел a_1, a_2, \dots, a_n .
- ▶ Подпоследовательность $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ называется возрастающей, если

$$1 \leq i_1 < i_2 < \dots < i_k \leq n$$

и

$$a_{i_1} < a_{i_2} < \dots < a_{i_k}$$

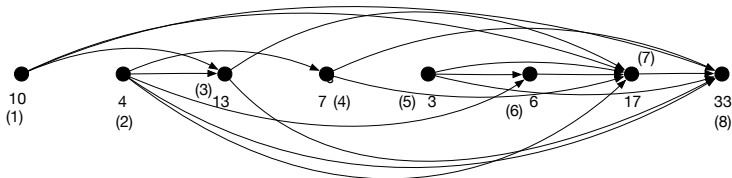
- ▶ Требуется найти максимальную длину возрастающей подпоследовательности.

Пример: $a_i = \{10, 4, 13, 7, 3, 6, 17, 33\}$

Одна из возрастающих подпоследовательностей есть $\{4, 7, 17, 33\}$.

Задача о возрастающей подпоследовательности

- ▶ Соединим направленными рёбрами элементы, которые могут быть друг за другом в подпоследовательности.



- ▶ Задача — найти не количество путей, а длину наибольшего пути.

Задача о возрастающей последовательности

- ▶ В задаче на количество путей консолидация подзадач имела вид

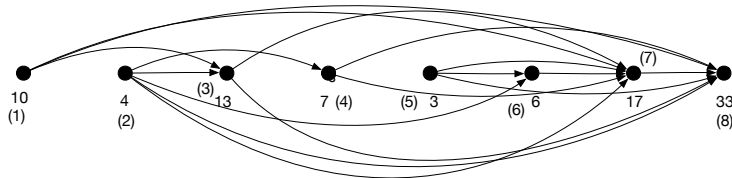
$$R_i = \sum_{j=1}^{N_{R_{i-1}}} R_{j-1}$$

- ▶ В этой задаче длина наибольшего пути к вершине i есть максимум из наибольших путей к предыдущим вершинам плюс единица

$$L_i = 1 + \max(L_{i-1,j}, j = 1, N_{L_{i-1}})$$

Задача о возрастающей последовательности

► Для



$$L_8 = 1 + \max(L_1, L_2, L_3, L_4, L_5, L_6, L_7)$$

$$L_7 = 1 + \max(L_1, L_2, L_3, L_4, L_5, L_6)$$

$$L_6 = 1 + \max(L_2, L_5)$$

...

Задача о возрастающей последовательности

- ▶ Рекурсивно эта задача решается следующей программой:

```
int f(int a[], int N, int k) { // k - номер элемента
    int m = 0;
    for (int i = 0; i < k-1; i++) { // для всех слева
        if (a[i] < a[k]) { // есть путь?
            int p = f(a,N,i); // его длина
            if (p > m) m = p; // m = max(m,p)
        }
    }
    return m+1;
}
```

Задача о возрастающей последовательности

- ▶ Для последовательности $\{1, 4, 2, 5, 3\}$ решение будет таким:

- ▶ $f_5 = 1 + \max(f_1, f_3)$
- ▶ $f_4 = 1 + \max(f_1, f_2, f_3)$
- ▶ $f_3 = 1 + \max(f_1)$
- ▶ $f_2 = 1 + \max(f_1)$
- ▶ $f_1 = 1$
- ▶ Возвращаемся назад
- ▶ $f_2 = 1 + \max(f_1) = 2$
- ▶ $f_3 = 1 + \max(f_1) = 2$
- ▶ $f_4 = 1 + \max(f_1, f_2, f_3) = 3$
- ▶ $f_5 = 1 + \max(f_1, f_3) = 3$

Решение есть $\max(f_1, f_2, f_3, f_4, f_5) = 3$

Задача о возрастающей последовательности

- ▶ Чтобы повторно не решать решённые подзадачи вводим массив с размером N , хранящий значения вычисленных функций. Начальные значения его равны нулю.

```
int f(int a[], int N, int k, int c[]) {  
    if (c[k] != 0) return c[k]; // Уже решали для k  
    int m = 0;  
    for (int i = 0; i < k-1; i++) { // для всех слева  
        if (a[i] < a[k]) { // есть путь?  
            int p = f(a,N,i); // его длина  
            if (p > m) m = p; // m = max(m,p)  
        }  
    }  
    return c[k] = m+1; // заносим в кэш и возвращаем  
}
```

Рекурсия как база динамического программирования.

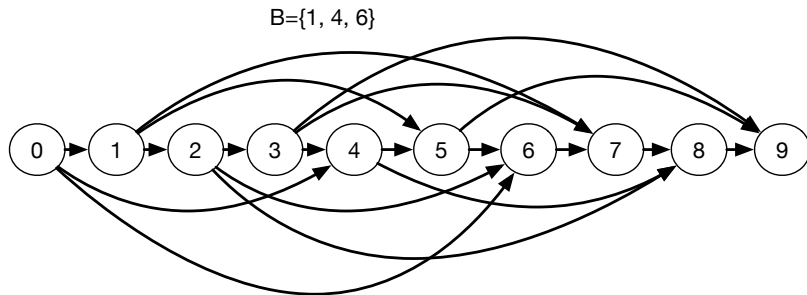
Рекурсия как база динамического программирования

- ▶ Вернёмся к задаче о банкомате.
- ▶ Пусть в банкомате имеется неограниченное количество банкнот (b_1, b_2, \dots, b_n) заданных номиналов.
- ▶ Задача заключается в том, чтобы выдать требуемую сумму денег наименьшим количеством банкнот.

Задача о банкомате

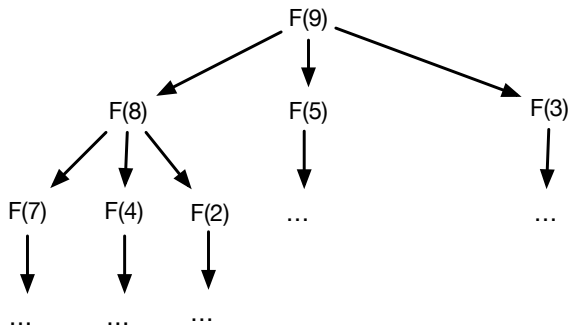
- ▶ Жадное решение имеется только для некоторого набора b_i .
- ▶ Например, при $b = \{1, 6, 10\}$ и $x = 12$ жадное решение даст ответ 3 ($10 + 1 + 1$), хотя существует более оптимальное решение 2 ($6 + 6$).
- ▶ Для набора $b = \{6, 10\}$ жадный алгоритм решения не найдёт.

Задача о банкомате: граф



Сведение задачи к задаче о количестве маршрутов.

Задача о банкомате: дерево рекурсии



Задача о банкомате: уравнение Беллмана

$$f(x) = \begin{cases} \min_{i=1,n} \{f(x - b_i)\} + 1, & \text{если } x > 0 \\ 0, & \text{если } x = 0 \\ \infty, & \text{если } x < 0 \end{cases}$$

Задача о банкомате: простая рекурсия

$$f(x) = \begin{cases} \min_{i=1,n} \{f(x - b_i)\} + 1, & \text{если } x > 0 \\ 0, & \text{если } x = 0 \\ \infty, & \text{если } x < 0 \end{cases}$$

```
const int GOOGOL = 999999999;
int f(int x, int *b, int n) {
    if (x < 0) return GOOGOL;
    if (x == 0) return 0;
    int min = GOOGOL;
    for (int i = 0; i < n; i++) {
        int r = f(x - b[i], b, n);
        if (r < min) min = r;
    }
    return min + 1;
}
```

Задача о банкомате: борьба с излишней рекурсией

- ▶ Глубина рекурсии не превосходит $\frac{x}{\min_{i=1,n} b_i}$
- ▶ Количество рекурсивных вызовов растёт по экспоненте.
- ▶ Если при $b = \{1, 4, 6\}$ и $x = 30$ количество рекурсивных вызовов 285709, то для $x = 31$ оно уже 418729, а для $x = 40$ оно равно 597124768.

Задача о банкомате: борьба с излишней рекурсией

- ▶ Нам помогает сохранение промежуточных результатов.

```
const int GOOGOL = 999999999;
int f(int x, int *b, int n, int *cache) {
    if (x < 0) return GOOGOL;
    if (x == 0) return 0;
    if (cache[x] >= 0) return cache[x];
    int min = GOOGOL;
    for (int i = 0; i < n; i++) {
        int r = f(x - b[i], b, n, cache);
        if (r < min) min = r;
    }
    return cache[x] = min + 1;
}
```

Задача о банкомате: борьба с излишней рекурсией

- ▶ Массив *cache*, передаваемый в функцию решения должен быть подготовлен.
- ▶ Мы должны узнать, решалась ли эта задача ранее.
- ▶ Инициализация массива проводится элементами, которые не могут быть результатом решения задачи.
- ▶ Размер массива должен соответствовать x .

Декомпозиция задачи

Декомпозиция задачи

- ▶ Для задачи о количестве путей до точки i подзадачей было определение количества путей до точек, находящихся в одном шаге от i .
 1. При условии отсутствия замкнутых маршрутов размер подзадачи всегда был несколько меньше размера задачи.
 2. Подзадача решалась тем же методом, что и задача.
- ▶ Наличие таких условий — подозрение на то, что задачу можно решить методом динамического программирования.

Декомпозиция задачи

- ▶ Общая схема динамического программирования:
 1. Можно выделить множество подзадач
 2. Имеется порядок на подзадачах
 3. Имеется рекуррентное соотношение решения задачи через решения подзадач
 4. Рекуррентное соотношение есть рекурсивная функция с целочисленными аргументами

Декомпозиция задачи

При динамическом программировании:

1. принципиально исключаются повторные вычисления в любой рекурсивной функции если есть возможность запоминать значения функции для аргументов, меньших текущего;
2. снижает время выполнения рекурсивной функции до времени, порядок которого равен сумме времён выполнения всех функций с аргументом, меньших текущего, если затраты на рекурсивный вызов постоянны.

Декомпозиция задачи

- ▶ Правильная декомпозиция задачи — ключ к её решению.
- ▶ Вернёмся к задаче о рюкзаке.
- ▶ Имеется рюкзак объёма V и N предметов весом H_i и стоимостью C_i . Найти комбинацию предметов, имеющую наибольшую суммарную стоимость, которые можно унести.
- ▶ Что есть подзадача меньшего размера?

Декомпозиция задачи

- ▶ Правильная декомпозиция задачи — ключ к её решению.
- ▶ Вернёмся к задаче о рюкзаке.
- ▶ Имеется рюкзак объёма V и N предметов весом H_i и стоимостью C_i . Найти комбинацию предметов, имеющую наибольшую суммарную стоимость, которые можно унести.
- ▶ Что есть подзадача меньшего размера?
- ▶ Наполнение рюкзака меньшего размера?

Декомпозиция задачи

- ▶ Правильная декомпозиция задачи — ключ к её решению.
- ▶ Вернёмся к задаче о рюкзаке.
- ▶ Имеется рюкзак объёма V и N предметов весом H_i и стоимостью C_i . Найти комбинацию предметов, имеющую наибольшую суммарную стоимость, которые можно унести.
- ▶ Что есть подзадача меньшего размера?
- ▶ Наполнение рюкзака меньшего размера?
- ▶ Наполнение рюкзака меньшим количеством предметов?

Декомпозиция задачи

- ▶ Требуется ответ на вопросы:
 1. какие ресурсы потребуются для запоминания результатов подзадач?
 2. если имеется решение подзадач, можно ли на основе этого получить решение задачи?
- ▶ Для подзадачи «рюкзак меньшего размера»
 1. размер памяти для результатов есть размер рюкзака
 2. если мы знаем результаты для всех меньших рюкзаков V_k , то поможет ли это решить задачу?
- ▶ Для подзадачи «рюкзак с меньшим количеством предметов»
 1. размер памяти для результатов есть количество предметов
 2. если мы знаем результаты для всех подзадач с меньшим количеством предметов, то поможет ли это решить задачу?

Декомпозиция задачи

- ▶ Задача про банкомат с неограниченным запасом купюр каждого номинала идентична задаче про рюкзак и магазин с неограниченным количеством товара каждой номенклатуры.
- ▶ Если количество предметов ограничено, то входные данные задачи — множество предметов, которые можно взять.
- ▶ Решение задачи про банкомат здесь не подходит — после выбора одного из предметов множество для выбора изменяется.

Декомпозиция задачи

Декомпозиция по размеру рюкзака.

1. Пусть аргументами подзадачи будут оставшееся множество предметов S и оставшееся место в рюкзаке L .
2. w — веса предметов, c — их стоимости.

$$f(S, L) = \begin{cases} \max_{e \in S} (f(S - e, L - w(e)) + c(e)), & \text{если } L > 0 \\ 0, & \text{если } L = 0 \\ -\infty, & \text{если } L < 0 \end{cases}$$

Каков размер пространства аргументов этой задачи?

$$D = O(2^N \cdot L_0) = O(2^N)$$

Декомпозиция задачи

Декомпозиция по количеству предметов.

Пусть мы берёмся за решение задачи с $K + 1$ предметами, зная решения всех задач с K предметами.

1. Аргументы подзадачи есть номер K и оставшееся место в рюкзаке L .

Если у нас есть решение задачи для K предметов, то $(K + 1)$ -й предмет мы можем либо взять, либо его не брать.

$$f(K, L) = \begin{cases} \max(f(L - w_K, K - 1) + c_K, f(L, K - 1)) & \text{если } L > 0 \\ 0, & \text{если } L = 0 \\ -\infty, & \text{если } L < 0 \end{cases}$$

Каков размер пространства аргументов этой задачи?

$$D = O(N \cdot L_0)$$

Восстановление решения

Задача о банкомате: нахождение банкнот

- ▶ Мы искали решение задачи минимизации.
- ▶ До сих пор нас интересовал только ответ, значение минимума.
- ▶ Мы его получили.
- ▶ Мы не получили, какие именно банкноты требуется выдать.

Задача о банкомате: нахождение банкнот

- ▶ Мы искали решение задачи минимизации.
- ▶ До сих пор нас интересовал только ответ, значение минимума.
- ▶ Мы его получили.
- ▶ Мы не получили, какие именно банкноты требуется выдать.
- ▶ Имеется несколько подходов к получению детального решения:
 - ▶ Каждый раз при нахождении решения подзадачи мы сохраняем историю его получения.
 - ▶ Зная ответ, мы повторяем решение, воссоздавая историю получения.

Задача о банкомате: восстановление решения

Первый способ: сохранять цепочку вызовов.

- ▶ Потребуется: иметь список банкнот для каждого промежуточного решения.
- ▶ Всего промежуточных решений может быть N
- ▶ Каждое из решений имеет различную длину.
- ▶ Потребуется N векторов.

Задача о банкомате: сохраняем список

```
const int GOOGOL = 999999999;
int f(int x, int *b, int n,
    int *cache, vector<vector<int> > &solution) {
    if (x < 0) return GOOGOL;
    if (x == 0) return 0;
    if (cache[x] >= 0) return cache[x];
    int min = GOOGOL, best = -1;
    for (int i = 0; i < n; i++) {
        int r = f(x - b[i], b, n, cache, solution);
        if (r < min) {
            min = r; best = b[i];
        }
    }
    solution[x] = solution[x - best];
    solution[x].push_back(best);
    return cache[x] = min + 1;
}
```

Задача о банкомате: восстанавливаем решение

- ▶ Зная ответ и имея кэш-таблицу, можно восстановить решение.
- ▶ Предположим, что мы знаем, что точный ответ при заданных начальных значениях есть 7.
- ▶ Тогда возникает вопрос: какой предыдущий ход мы сделали, чтобы попасть в заключительную позицию?
- ▶ Введём термин *ранг* для обозначения наименьшего числа ходов, требуемого для достижения текущей позиции из начальной.
- ▶ Тогда каждый ход решения всегда переходит в позицию с рангом, большим строго на единицу.

Задача о банкомате: восстанавливаем решение

- ▶ Восстановление решения по таблице ответов:
 1. Заключительная позиция имеет ранг k .
 2. Делаем позицию текущей.
 3. Если текущая позиция имеет ранг 0, то это — начальная позиция и алгоритм завершён.
 4. Рассматриваем все позиции, ведущие в текущую и выбираем из них произвольную с рангом $k - 1$.
 5. Запоминаем ход, приведший к из позиции ранга $k - 1$ в ранг k .
 6. Понижаем ранг - $k \rightarrow k - 1$ и переходим к 2.

Задача о банкомате: восстанавливаем решение

```
vector<int> buildSolution(int x, int *b, int n, int *cache)
{
    vector<int> ret;
    for (int k = cache[x]; k >= 0; k--) {
        for (int i = 0; i < n; i++) {
            int r = x - b[i];
            if (r >= 0 && cache[r] == k-1) {
                x = r;
                ret.push_back(b[i]);
                break;
            }
        }
    }
    return ret;
}
```

Решение задачи о банкомате: восстанавливаем решение

Третья возможность восстановить решение — добавить ещё один кэш, сохраняющий номинал лучшей банкноты при лучшем решении.

```
const int GOOGOL = 999999999;
int f(int x, int *b, int n,
    int *cache, int *bestnote) {
    if (x < 0) return GOOGOL;
    if (x == 0) return 0;
    if (cache[x] >= 0) return cache[x];
    int min = GOOGOL, best = -1;
    for (int i = 0; i < n; i++) {
        int r = f(x - b[i], b, n, cache, bestnote);
        if (r < min) {
            min = r;
            bestnote[x] = r;
        }
    }
    return cache[x] = min + 1;
}
```

Решение задачи о банкомате: восстанавливаем решение

Теперь для того, чтобы получить нужные для размена банкноты для суммы в x , достаточно пробежаться по кэшу банкнот:

```
while (x > 0) {  
    // вывод bestcache[x];  
    x -= bestcache[x];  
}
```

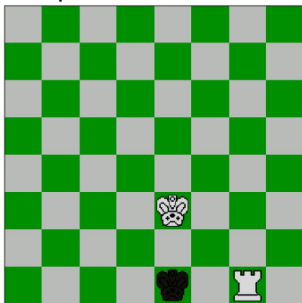
Динамическое программирование и игры.

Динамическое программирование и игры

- ▶ Шахматы — конечная игра с наличием цели.
- ▶ Деревья игры заканчиваются либо в позициях с оценкой $+\infty$, если выигрывают белые, либо с оценкой 0, если заключительная позиция — ничья, либо $-\infty$, если выигрывают чёрные.
- ▶ В конкретно заданной позиции результат обоюдно лучшей игры предопределён.
- ▶ Сложность лишь заключается в большом размере дерева игры.

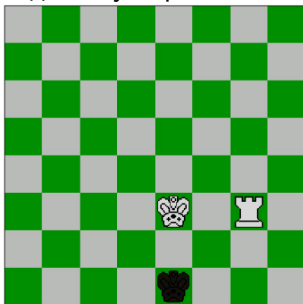
Динамическое программирование и игры

- ▶ Назовём позициями ранга 0 те позиции, в которых игра завершилась с каким-либо результатом.



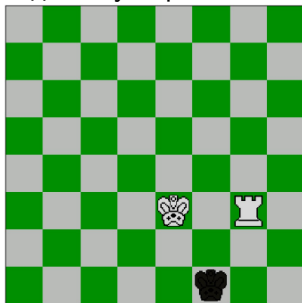
Динамическое программирование и игры

- Позиции ранга один — те позиции, которые при очередном ходе могут привести к позициям ранга 0.



Динамическое программирование и игры

- ▶ Позиции ранга два — те позиции, которые при очередном ходе могут привести к позициям ранга 1.



Динамическое программирование и игры

- ▶ Шахматы — одна из игр, решаемых динамическим программированием.
- ▶ Существует большое множество позиций, особенно с небольшим количеством фигур, для которых известен их ранг.
- ▶ Для всех позиций до 7 фигур включительно известен их ранг.
- ▶ Например, известны позиции с рангом 1097.



- ▶ На 549-м ходу белые ставят мат.

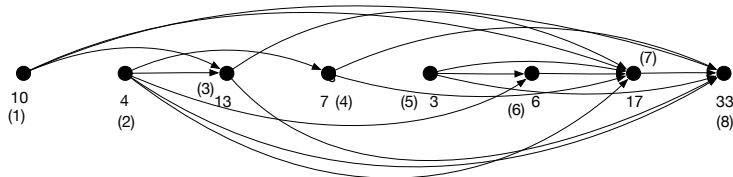
Динамическое программирование и игры

- ▶ Табличный подход изобрёл Кен Томпсон в 1970-х годах.
- ▶ Эффективную реализацию с помощью динамического программирования — Новосибирский математик Евгений Налимов, таблицы с рангом позиций называются таблицами Налимова.
- ▶ Размер таблиц с 7-ю фигурами составляет 140 тибибайт. Расчёт таблиц производился в 2012 году на компьютере «Ломоносов» ВМК МГУ.
- ▶ Наилучшая игра обоих сторон заключается в том, чтобы каждым очередным ходом переходить в позицию с тем же результатом и рангом, меньшим ровно на единицу.

Уход от рекурсии. Восходящее решение.

Восходящее решение

- ▶ Вернёмся к задаче о возрастающей подпоследовательности.

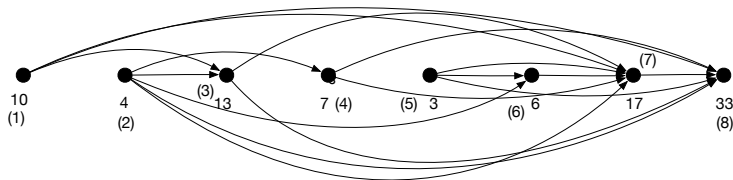


- ▶ При нисходящем решении нам требуется находить максимум из всех значений функций от $F(1)$ до $F(N)$, для чего рекурсивный вызов должен опуститься от $F(N)$ до $F(1)$.
- ▶ Для больших N уровень рекурсивных вызовов может превысить разумные рамки (размер стека в программах ограничен).

Восходящее решение

- ▶ При восходящем решении мы пробуем решать подзадачи *до того*, как они будут поставлены.
- ▶ Решая задачи в возрастающем порядке мы достигаем следующих целей:
 1. Гарантируется, что все задачи, решаемые позже, будут зависеть от ранее решённых.
 2. Не потребуются рекурсивные вызовы.
- ▶ Обязательное условие: монотонность аргументов при решении подзадач, если $f(k)$ — подзадача для $f(n)$, то $M(k) < M(n)$, где $M(x)$ есть некая метрика сложности решения.

Восходящее решение



► Для данной задачи порядок решения будет таким:

► $F(1) = 1$

► $F(2) = 1$

► $F(3) = \max(F(1), F(2)) + 1 = 2$

► ...

► $F(8) = \max(F(1), F(2), F(3), F(4), F(5), F(6), F(7)) + 1$

После получения значений $F(i)$ не требуется вызывать функцию, можно использовать уже вычисленное значение.

Восходящее решение

- ▶ Восходящее решение не всегда оправдано.
- ▶ Пусть решение задачи определяется таким образом:

$$F(n) = \max(F((n+1)/2), F(n/3)) + 1$$

$$F(0) = F(1) = 1$$

- ▶ Это описывает какую-то последовательность.
- ▶ При нисходящем способе для $F(8)$ мы получаем:
- ▶ $F(8) = \max(F(4), F(2)) + 1$
- ▶ $F(4) = \max(F(2), F(1)) + 1$
- ▶ $F(2) = \max(F(1), F(0)) + 1 = 2$
- ▶ $F(4) = 3$
- ▶ $F(8) = 4$

Восходящее решение

- ▶ При попытке решить задачу восходящим решением мы получим:
- ▶ $F(0)=F(1)=1$
- ▶ $F(2)=\max(F(1),F(0))+1=2$
- ▶ $F(3)=\max(F(2),F(1))+1=3$
- ▶ $F(4)=\max(F(2),F(1))+1=3$
- ▶ $F(5)=\max(F(3),F(1))+1=3$
- ▶ $F(6)=\max(F(3),F(2))+1=3$
- ▶ $F(7)=\max(F(4),F(2))+1=4$
- ▶ $F(8)=\max(F(4),F(3))+1=4$
- ▶ Значения $F(3),F(5),F(6),F(7)$ были вычислены зря.

Восходящее решение

- ▶ Поставленная задача более подходила под рекурсивный алгоритм с запоминанием промежуточных результатов (*memoizing*).
- ▶ Нисходящее решение $F(N)$ имеет сложность $O(\log N)$
- ▶ Восходящее решение $F(N)$ имеет сложность $O(N)$

Восходящее решение

- ▶ Рассмотрим задачу

$$F(0) = F(1) = F(2) = 1$$

$$F(n) = F(n - 1) + F(n - 3)$$

- ▶ При вычислении $F(100000)$ в нисходящем порядке размер стека будет равен 100000, что, возможно, приведёт к аварийному завершению программы.
- ▶ Здесь мы различаем понятия «алгоритм», который корректен и «программа», которая исполняется «исполнителем».
- ▶ Для хранения аргументов и локальных переменных каждого рекурсивного вызова потребуется всё возрастающее количество памяти.

Восходящее решение

- ▶ Последовательность будет выглядеть так:
- ▶ $\{1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129\}$
- ▶ $F(998) \approx 2.89273 \cdot 10^{165}$
- ▶ $F(999) \approx 4.23951 \cdot 10^{165}$
- ▶ $\lim_{n \rightarrow \infty} \frac{F(i+1)}{F(i)} \approx 1.46557$
- ▶ Для хранения 999-го элемента $F(999)$ потребуется по меньшей мере $\log_2 4.23951 \cdot 10^{165} \approx 559$ бит ≈ 69 байтов, не считая управляющей информации.

- ▶ Восходящее решение этой задачи обойдётся локальными переменными.

Восходящее решение vs нисходящее

- ▶ Восходящее решение по корректности эквивалентно нисходящему.
- ▶ Необходимо обеспечить вычисление в соответствующем порядке.
- ▶ При простом целочисленном аргументе вычисления можно проводить в порядке увеличения аргумента.
- ▶ Восходящее решение требует меньшего размера стека, но может решать задачи, ответ на которых не понадобится при решении главной задачи.

Использование отображений
при решении задач методом
динамического
программирования.

Использование отображений

- ▶ Пока решались задачи, требующие одного или двух целочисленных аргументов.
- ▶ Увы, не все задачи такие.

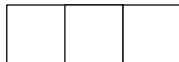
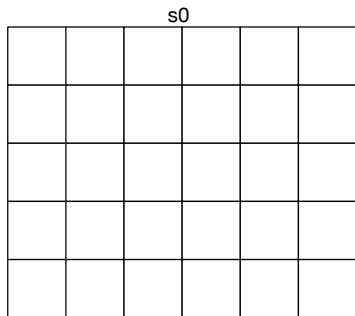
Использование отображений

Задача о покрытии.

- ▶ Имеется прямоугольник размером 5×6 .
- ▶ Сколькими способами его можно замостить фигурами 1×2 и 1×3 ? Симметрии и повороты различаются.

Использование отображений

- Задача о разбиении прямоугольника.

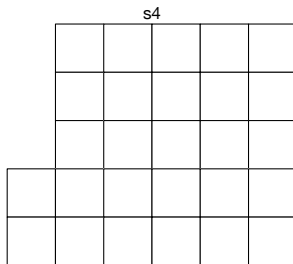
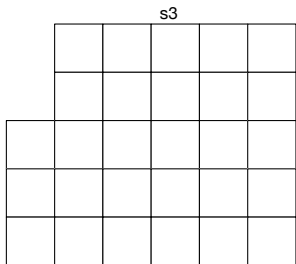
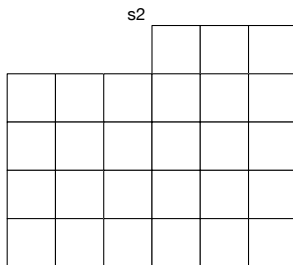
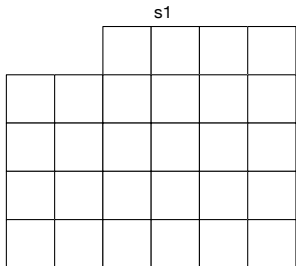


Использование отображений

- ▶ Один из методов решения задачи - динамическое программирование.
- ▶ Обозначим через $f(s)$ количество разбиений фигуры s на требуемые фрагменты
- ▶ Тогда $f(s_0) = f(s_1) + f(s_2) + f(s_3) + f(s_4)$, где s_i - подфигуры, получающиеся вычитанием одного из фрагментов, содержащих крайнюю левую верхнюю точку.

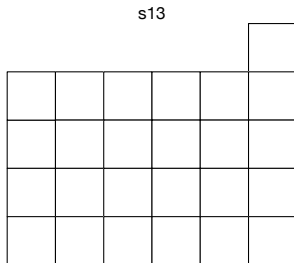
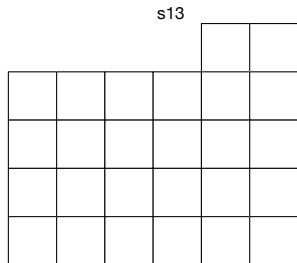
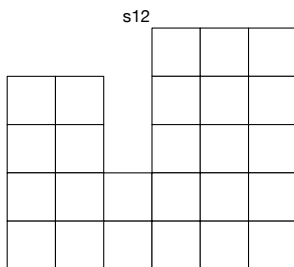
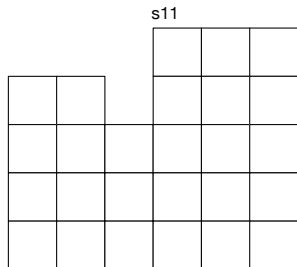
Использование отображений

► Возможные подзадачи.



Использование отображений

► $f(s_1) = f(s_{11}) + f(s_{12}) + f(s_{13}) + f(s_{14})$



Использование отображений

- При предложенном алгоритме решения одна подзадача может возникнуть при различных путях:

s'

1	1	2	2	3	3
4	4	5	5	6	6

s''

1	2	3	4	5	6
1	2	3	4	5	6

s'''

1	1	1	2	2	2
3	3	3	4	4	4

Использование отображений

- ▶ Решение таких подзадач не зависит от истории их получения.
- ▶ Если имеется отображение аргументов подзадачи (позиции) на результат, то задача может быть решена методом динамического программирования.

Использование отображений

- ▶ Что есть $f(s)$, если s не является числом?
- ▶ s есть объект, который мы должны использовать в виде ключа *key* в отображении.
- ▶ *value* в отображении есть значение, хранимое по этому ключу.
- ▶ Требуется создать *взаимно однозначное* соответствие объекта какому-то набору битов.
- ▶ В данной задаче можно пронумеровать все 30 элементов и каждому из номеров присвоить 1, если он присутствует и 0, если отсутствует.
- ▶ Размер множества возможных ключей есть 2^{30} , что слишком много для восходящего метода.

Использование отображений

- ▶ Воспользуемся изученной абстракцией отображения.
- ▶ Каждая из позиций является ключом в отображении.
- ▶ Задача решается нисходящим динамическим программированием с мемоизацией.
 1. При решении задачи из таблицы решений по ключу, соответствующему текущей позиции, извлекается значение.
 2. Если такого ключа нет, то производится полное решение и в отображение добавляется пара (ключ/полученное значение)
 3. Если ключ имеется, то результатом подзадачи будет значение по ключу.

Этапы решения задачи
методом динамического
программирования.

Этапы решения задачи

1. Определяется необходимость именно в динамическом программировании. При быстром уменьшении подзадач задача решается методом «разделяй и властвуй».
2. Определяется максимальный уровень рекурсии в главной задаче. Например, в задаче на покрытие прямоугольника максимальный уровень равен 15.
3. Для нетривиальных задач всегда разрабатывается рекурсивный метод решения задачи.
4. Разрабатывается метод отображения аргументов задачи в результат.
5. Реализуется процесс мемоизации в нисходящем методе.
6. Если максимальный уровень слишком большой, то нисходящий метод неприменим, но по результатам исследования реализуется восходящий метод.

Многомерные варианты.

Расстояние редактирования

- ▶ При исправлении слова в текстовом редакторе: три операции:
 - ▶ замена одной буквы на другую;
 - ▶ удаление лишней буквы;
 - ▶ вставка буквы.
- ▶ Больше операций редактирования — меньше похожи слова.
- ▶ Количество операций — *расстояние редактирования* или *расстояние Левенштейна*.
- ▶ Это — мера различия между двумя строками.

Расстояние редактирования

Сколько нужно операций, чтобы превратить слово СЛОН в слово ОГОНЬ?

Один из вариантов:

СЛОН \rightarrow СГОН \rightarrow СГОНЬ \rightarrow ОГОНЬ

Расстояние редактирования: ищем рекурсию

Вопросы:

- ▶ Задача ли это динамического программирования?
- ▶ Что является рекурсивной функцией, решающей данную задачу?
- ▶ Что есть аргументы функции?

Расстояние редактирования: ищем рекурсию

Зафиксируем входные строки — исходную строку s и строку назначения d (образец). s должна превратиться в d .

- ▶ Как найти более простые подзадачи? Подзадач для более коротких строк-префиксов.
- ▶ Что является мерой простоты? Длина строки.

Расстояние редактирования

- ▶ Пусть i — длина префикса s , а j — длина префикса d .
- ▶ Три варианта:
 - ▶ Заменить последний символ строки s на последний символ строки d .
 - ▶ Добавить символ к концу строки s .
 - ▶ Удалить последний символ строки s .

Расстояние редактирования

Последовательность переходов $\text{СЛОН} \rightarrow \text{СГОН} \rightarrow \text{СГОНЬ} \rightarrow \text{ОГОНЬ}$ не обладает свойством порядка на подзадачах.

Имеется способ с сохранением порядка на подзадачах:

1. В префиксах строк длины 1 С и О меняем букву С на О. $\text{СЛОН} \rightarrow \text{ОЛОН}$.
2. В префиксах длины 2 меняем Л на Г. $\text{ОЛОН} \rightarrow \text{ОГОН}$.
3. В префиксах длины 4 добавляем букву Ъ. $\text{ОГОН} \rightarrow \text{ОГОНЬ}$.

Расстояние редактирования: уравнение Беллмана

Введём функцию $F(i, j)$ как решение задачи для префиксов строк s и t длинами i и j .

$$F(i, j) = \min \begin{pmatrix} F(i-1, j-1), \text{ если } s_i = t_j \text{ или } F(i-1, j-1) + 1, \text{ если } s_i \neq t_j \\ F(i-1, j) + 1 \\ F(i, j-1) + 1 \end{pmatrix}$$

1. Различает случай совпадения последних символов.
2. Мы должны дописать символ в конец s .
3. Мы должны удалить последний символ из s .

Расстояние редактирования: мемоизация

- ▶ Аргументы — дискретные.
- ▶ Значений аргументов — плотные.
- ▶ Множество пар аргументов конечно и равно $|s| \times |d|$.
- ▶ Допускается восходящее решение без рекурсии заполнением таблицы по строкам.

Расстояние редактирования: пример решения

A	R	R	O	G	A	N	T	
S	U	R	R	O	G	A	T	E

Расстояние редактирования: пример решения

- ▶ Добавим один лишний левый столбец и одну лишнюю верхнюю строку к таблице результатов.
- ▶ Заполним их последовательно возрастающими числами.
- ▶ Пустая строка может превратиться в образец за число операций, равное длине образца.
- ▶ Непустая строка превращается в пустой образец за число операций, равное длине строки.

		S	U	R	R	O	G	A	T	E
	0	1	2	3	4	5	6	7	8	9
A	1									
R	2									
R	3									
O	4									
G	5									
A	6									
N	7									
T	8									

Расстояние редактирования: пример решения

Заполнение таблицы ведём по строкам. Значение в первой свободной ячейке зависит от трёх элементов таблицы:

		S	U	R	R	O	G	A	T	E
	0	1	2	3	4	5	6	7	8	9
A	1									
R	2									
R	3									
O	4									
G	5									
A	6									
N	7									
T	8									

Буквы S столбца и A строки, не совпадают → значение, полученное по диагональное стрелке, увеличивается на 1. По горизонтальной и вертикальной стрелке в ячейку приходят увеличенные на 1 числа из тех ячеек.

Расстояние редактирования: пример решения

А вот при совпадении букв штраф за замену отсутствует и в клеточку записывается число 6, значение, полученное по диагональной стрелке.

		S	U	R	R	O	G	A	T	E
	0	1	2	3	4	5	6	7	8	9
A	1	1	2	3	4	5	6	6		
R	2									
R	3									
O	4									
G	5									
A	6									
N	7									
T	8									

Расстояние редактирования: пример решения

Решение задачи — правый нижний элемент таблицы.

		S	U	R	R	O	G	A	T	E
	0	1	2	3	4	5	6	7	8	9
A	1	1	2	3	4	5	6	6	7	8
R	2	2	2	2	3	4	5	6	7	8
R	3	3	3	2	2	3	4	5	6	7
O	4	4	4	3	3	2	3	4	5	6
G	5	5	5	4	4	3	2	2	3	5
A	6	6	6	5	5	4	3	2	3	4
N	7	7	7	6	6	5	4	3	3	4
T	8	8	8	7	7	6	5	4	3	4

Расстояние редактирования

- ▶ Сложность по времени — $O(|s| \times |d|)$.
- ▶ Сложность по памяти — $O(|d|)$.

Задача о счастливых билетах

Билет состоит из N цифр от 0 до 9 и является счастливым, если сумма первой половины его цифр равна сумме второй половины. Найти количество счастливых билетов.

Задача о счастливых билетах

- ▶ Причём здесь динамическое программирование? Это ведь комбинаторная задача.
- ▶ Сведём задачу к другой. Пусть $N = 6$.

3	3	5	6	4	1
3	3	5	3	5	8

- ▶ Заменяем во второй половине числа все цифры на их дополнение до девяти.
- ▶ Количество таких чисел в точности равно количеству счастливых, так как отображение биективное.
- ▶ Исходный инвариант: $x_1 + x_2 + x_3 = x_4 + x_5 + x_6$
- ▶ Инвариант после отображения:
$$x_1 + x_2 + x_3 + (9 - x_1) + (9 - x_2) + (9 - x_3) = 3 \cdot 9$$
- ▶ Требуется найти количество N -значных чисел, сумма которых равна $9 \cdot \frac{N}{2}$

Задача о счастливых билетах

- ▶ Привычное решение задачи наталкивается на проблему: нам нужно найти не просто количество любых чисел от 0 до 9, сумма которых 27, нужно, чтобы таких чисел было именно 6.
- ▶ Количество таких чисел есть сумма количеств чисел:
 - ▶ первая цифра которых 0 и сумма пяти остальных равна 27;
 - ▶ первая цифра которых равна 1 и сумма пяти остальных равна 26;
 - ▶ ...
 - ▶ первая цифра которых равна 9 и сумма остальных пяти равна 18

Задача о счастливых билетах

- ▶ Нам удалось разбить задачу подзадачи меньшего ранга.
- ▶ Обозначим за $f(n, left)$ количество чисел, имеющих n знаков, сумма которых $left$.
- ▶ Тогда $f(6, 27) = f(5, 27) + f(5, 26) + f(5, 25) + f(5, 24) + f(5, 23) + f(5, 22) + f(5, 21) + f(5, 20) + f(5, 19) + f(5, 18)$
- ▶ Доопределим функцию $f(n, left)$ таким образом, что при $n > 0$ и $left < 0$ она возвращала 0 и $f(1, left) = 1$, если $0 \leq left \leq 9$ и $f(1, left) = 0$ в противном случае.
- ▶ Тогда
$$f(n, left) = \sum_{i=0}^9 f(n-1, left-i)$$
- ▶ Мы свели задачу к задаче динамического программирования, но двухмерной.

Задача о счастливых билетах

- ▶ Если задача двумерная, то можно использовать двумерную таблицу решений.
- ▶ Первый размер определяется размерностью задачи n
- ▶ Второй размер определяется максимальным значением $left = 9 \cdot \frac{n}{2}$
- ▶ Нерешённые подзадачи в таблице помечаются числом -1, так как ни одна из подзадач не может вернуть отрицательной число.

- ▶ Значения в таблице решений занимают последовательные ячейки, она не разрежена.
- ▶ Следовательно, задача допускает восходящее решение.
- ▶ Таблица заполняется, начиная от значения $n = 1$ и всех возможных *left* от 0 до 9.
- ▶ Максимальный уровень рекурсии равен n , это немного и восходящее решение, хотя и возможно, но не требуется.

Задача о вторичной структуре РНК

Задача из вычислительной биологии:

Имеется последовательность *оснований*

$$B = b_1 b_2 \dots b_n, b_i \in \{A, C, G, U\}.$$

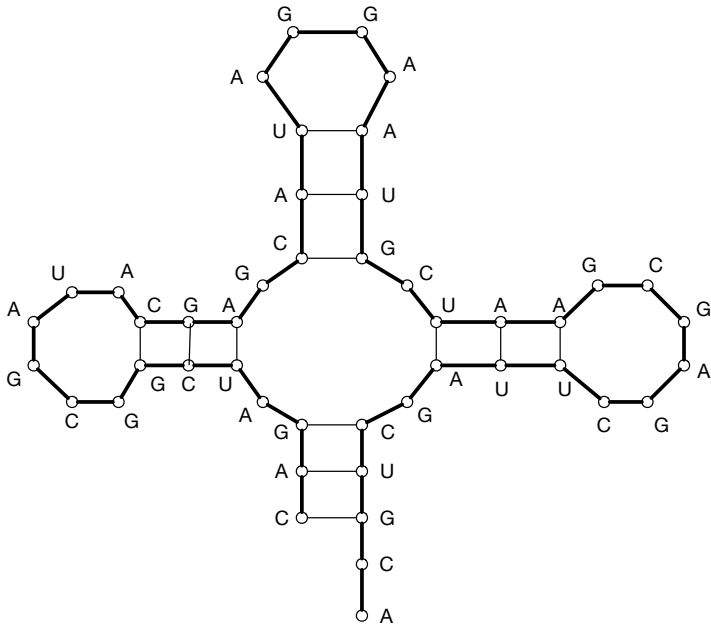
Каждое основание может образовывать пару не более, чем одним другим основанием.

$$A \leftrightarrow U$$

$$C \leftrightarrow G$$

Образуется вторичная структура

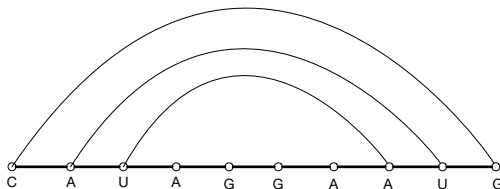
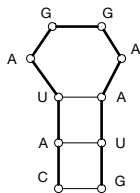
Задача о вторичной структуре РНК



Задача о вторичной структуре РНК

Условия, накладываемые на вторичную структуру.

1. **Отсутствие резких поворотов** Не существует пар (b_i, b_j) для которых $|i - j| \leq 4$.
2. **Состав пар** Возможны только пары (A, U) и (C, G) .
3. **Вхождение основания** Каждое основание входит не более, чем в одну пару.
4. **Отсутствие пересечений** Для двух произвольных пар (b_i, b_j) и (b_k, b_l) невозможно условие $i < k < j < l$



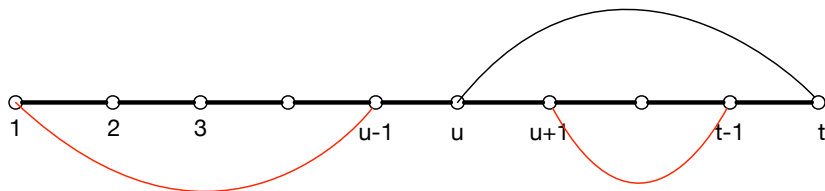
Задача о вторичной структуре РНК

- ▶ Задача ли это динамического программирования?
- ▶ Если да, то что есть «подзадача»?
- ▶ Если да, то что есть «консолидация»?

Задача о вторичной структуре РНК: попытка 1

- ▶ Пусть $F(k)$ — максимальное количество пар для вторичной структуры b_1, b_2, \dots, b_k
- ▶ Тогда $F(1) = F(2) = F(3) = F(4) = F(5) = 0$
- ▶ Пусть решены задачи для $F(1), F(2), \dots, F(t-1)$. Как решить задачу для $F(t)$?
- ▶ Возможны два варианта:
 - ▶ во вторичной структуре b_1, b_2, \dots, b_t t не участвует в паре.
 - ▶ t образует пару с каким-то элементом u , $u < t - 4$.
- ▶ Первый случай порождает подзадачу $F(t-1)$.
- ▶ А что во втором случае?

Задача о вторичной структуре РНК:попытка 1



Имеется запрет на пересечения \rightarrow нет пар, левый конец которых меньше u , а правый — больше u .

Возникает две подзадачи, первую мы решить можем, а вторую — нет.

Задача о вторичной структуре РНК: вторая попытка

Необходимость решения задач второго рода подсказывает: за целую задачу взять $F(k, l)$, то есть два параметра.

$$F(k, l) = \begin{cases} 0, & \text{если } k + 4 \leq l \\ \max(F(k, l - 1), 1 + \max_u (F(k, u - 1) + F(u + 1, l - 1))) & \text{иначе} \end{cases}$$

Восходящее решение: замечание: сначала решаются более *короткие* задачи.

Спасибо за внимание.

Следующая лекция —
Алгоритмы на графах.