ITIVITI

LET IT TRADE

Алгоритмы и структуры данных

Акатьев Никита Львович
кто это?

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

УНИВЕРСИТЕТ ИТМО

ITIVITI

LET IT TRADE

Динамическое программирование

УНИВЕРСИТЕТ ИТМО

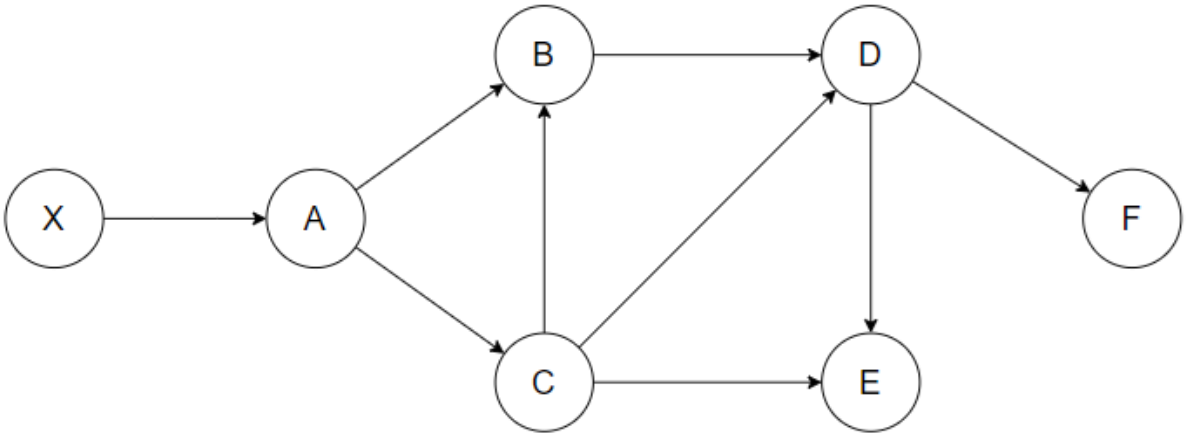
Немного истории

"I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, "programming" I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is its impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. Its impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

Richard Bellman, *Eye of the Hurricane: an autobiography*, 1984.

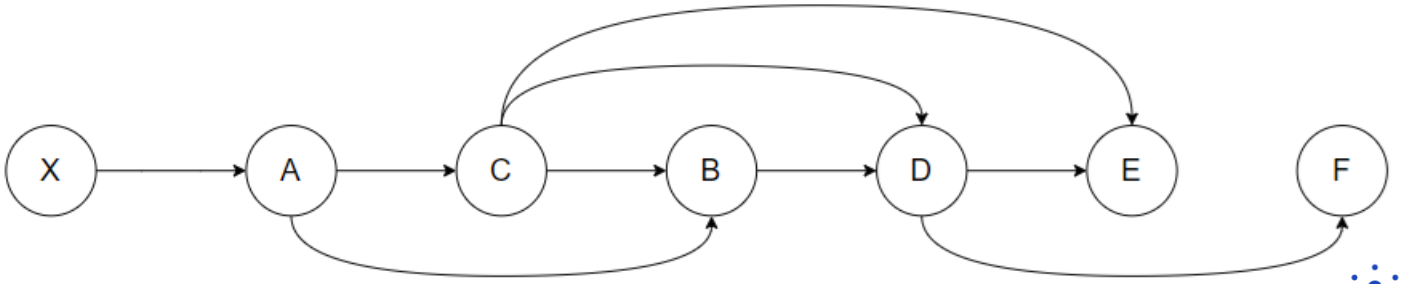
Поиск кратчайших путей

- Задача тривиальна для ориентированных ациклических графов
  - dist(X, X) = 0
  - dist(X, D) = min(dist(X, B) + w(BD), dist(X, C) + w(CD))
- В каком порядке считать значения dist(X, ·) ?



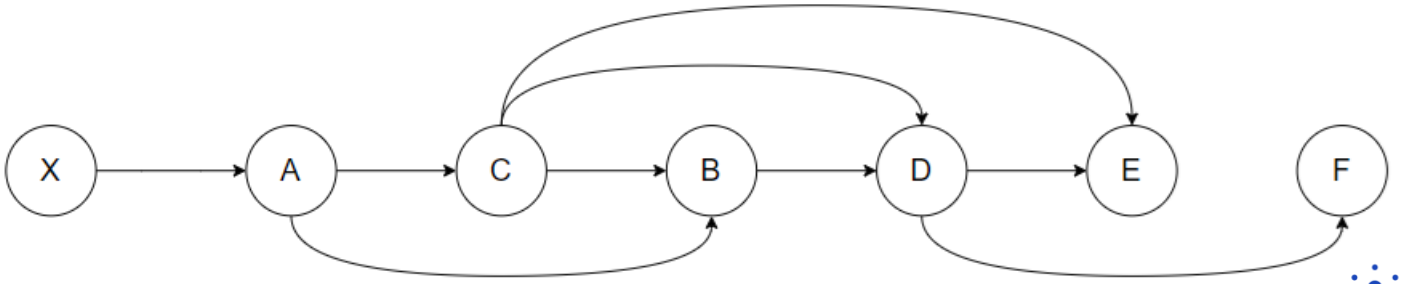
Топологический порядок обхода

- DAG можно обходить в топологическом порядке
  - Двигаясь по ребрам, мы никогда не повернем назад
  - Для каждой вершины гарантируется, что dist(X, ·) для всех входящих в нее вершин уже посчитан
  - Каждое ребро будет пройдено ровно один раз



Топологический порядок обхода

- Можно решить сразу спектр задач
  - Вместо кратчайшего пути ищем путь наибольшей длины
  - Наименьшее / наибольшее количество вершин
  - Путь с самым легким / самым тяжелым ребром
- Более общее: каждая вершина – это некоторая задача, решение которой зависит от подзадач (вершин, связанных с ней ребрами)



ДП: Граф подзадач

ITIVITI

LET IT TRADE

- Числа Фибоначчи:
  - $F_0 = 0, F_1 = 1$
  - $F_n = F_{n-2} + F_{n-1}, n \geq 2$

boss make a dollar,
I make a dime,
that's why my algorithms
run in exponential time

3:36 PM · May 8, 2021 · Twitter Web App

502 Retweets **16** Quote Tweets **3,980** Likes

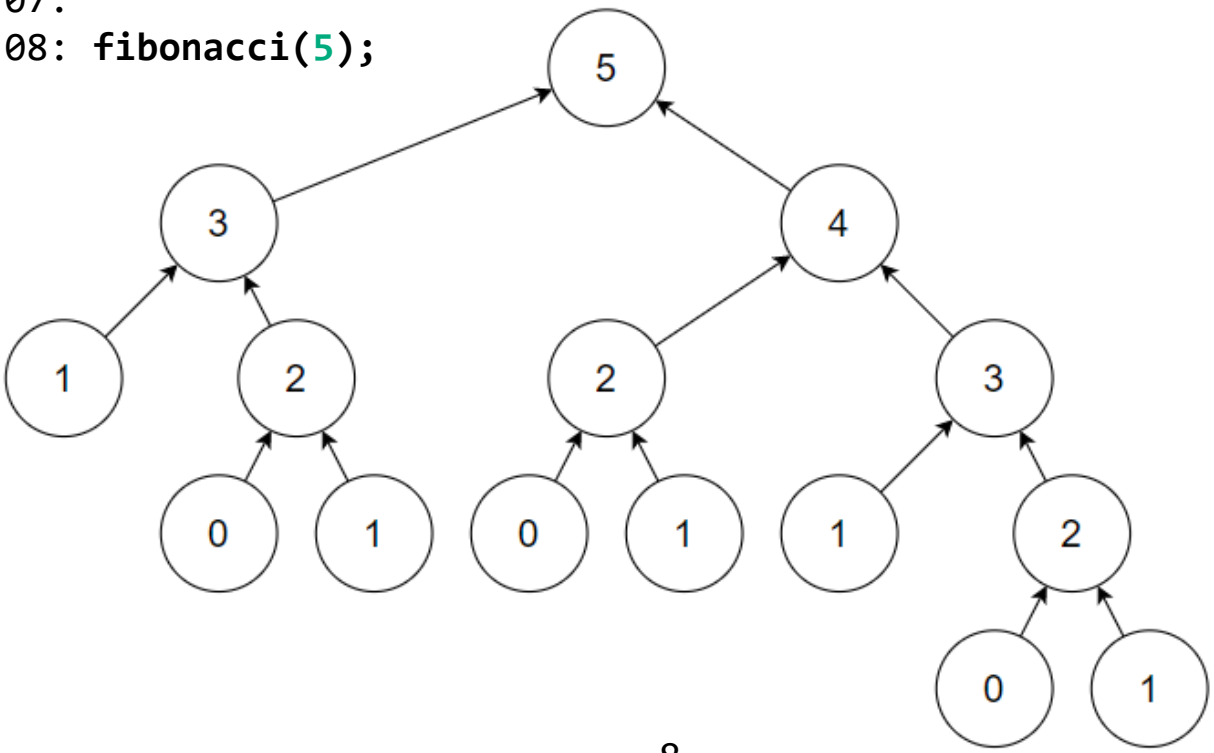
```
01: int fibonacci(int n)
02: {
03:    if (n <= 1)
04:       return n;
05:    return fibonacci(n-2) + fibonacci(n-1);
06: }
```

- Как выглядит граф подзадач?

УНИВЕРСИТЕТ ИТМО

ДП: Граф подзадач

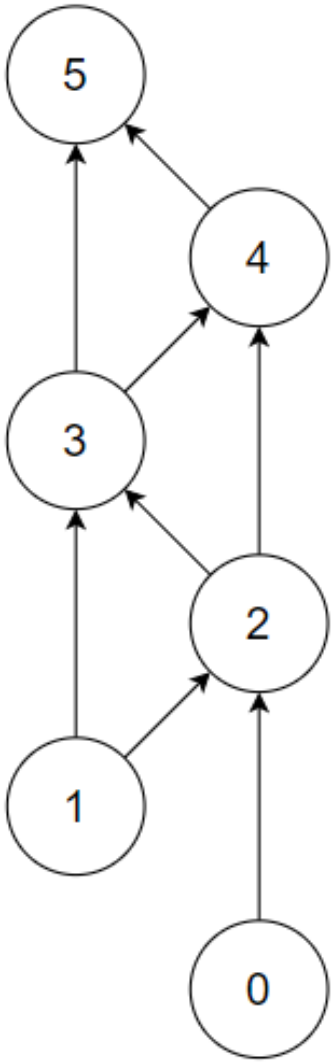
```
01: int fibonacci(int n)
02: {
03:     if (n <= 1)
04:         return n;
05:     return fibonacci(n-2) + fibonacci(n-1);
06: }
07:
08: fibonacci(5);
```



ДП: Граф подзадач

- Числа Фибоначчи. Способ 1
  - Рекурсивно с запоминанием, сверху вниз

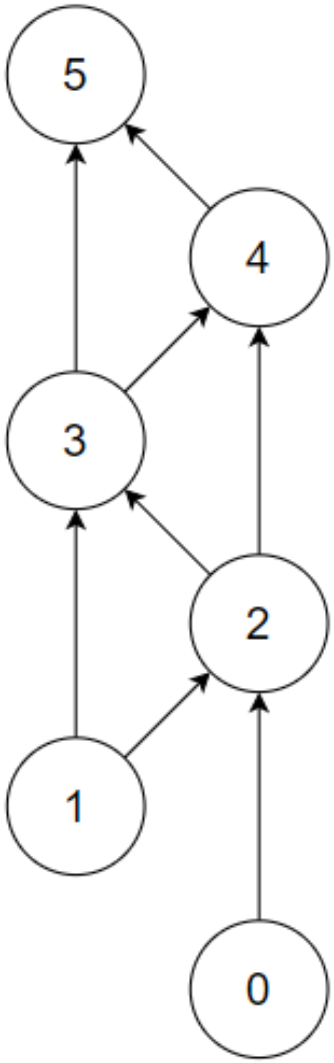
```
01: int memo[MAX];
02:
03: void initialize() {
04:   for (int i = 0; i < MAX; ++i)
05:     memo[i] = -1;
06: }
07:
08: int fibonacci(int n) {
09:   if(memo[n] == -1) {
10:     if (n <= 1)
11:       memo[n] = n;
12:     else
13:       memo[n] = fibonacci(n-2) + fibonacci(n-1);
14:   }
15:   return memo[n];
16: }
```



ДП: Граф подзадач

- Числа Фибоначчи. Способ 2
  - Заполнение таблицы, снизу вверх

```
01: int fibonacci(int n)
02: {
03:   int table[MAX] = {0, 1};
04:   int i = 2;
05:   while (i <= n) {
06:     table[i] = table[i-2] + table[i-1];
07:     i++;
08:   }
09:   return table[n];
10: }
```



- Как переиспользовать таблицу при многократных запусках?
- Как не использовать таблицу?

Динамическое программирование

ITIVITI

- Часто используется для поиска *оптимального* решения
  - Ключевые слова: кратчайший, наименьший, наибольший, …
- Задача разбивается на подзадачи
  - Оптимальное решение задачи состоит из оптимальных решений подзадач
  - Подзадачи пересекаются – одна и та же подзадача может быть составляющей нескольких задач

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Динамическое программирование

ITIVITI

- Между задачами есть определенные зависимости (рекуррентные соотношения)
  - Граф зависимостей ациклический
  - Истоки – тривиальные задачи
  - Конечная задача – сток
- Подзадачи решаются в топологическом порядке, ответ на каждую подзадачу запоминается

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Динамическое программирование

ITIVITI

LET IT TRADE

- Компания продает бревна разной длины по разной цене:

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	15	17	20

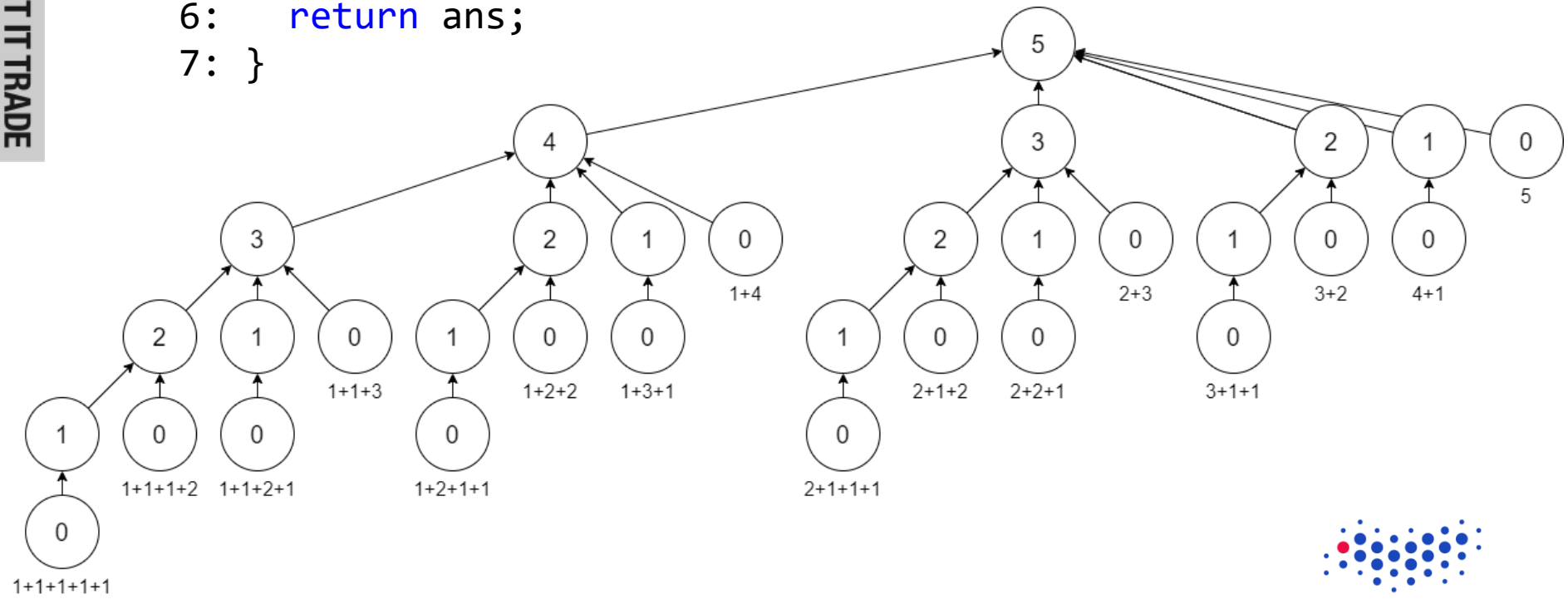
- Какую максимальную выручку компания может получить, разрезая на части бревно и продавая их по отдельности?
  - Полный перебор?

УНИВЕРСИТЕТ ИТМО

Динамическое программирование

ITIVITI

```
1: int maxprice(int * price, int len) {
2:   int ans = 0;
3:   for (int i = 1; i <= len; i++) {
4:     ans = max(ans, maxprice(price, len - i) + price[i]);
5:   }
6:   return ans;
7: }
```



LET IT TRADE

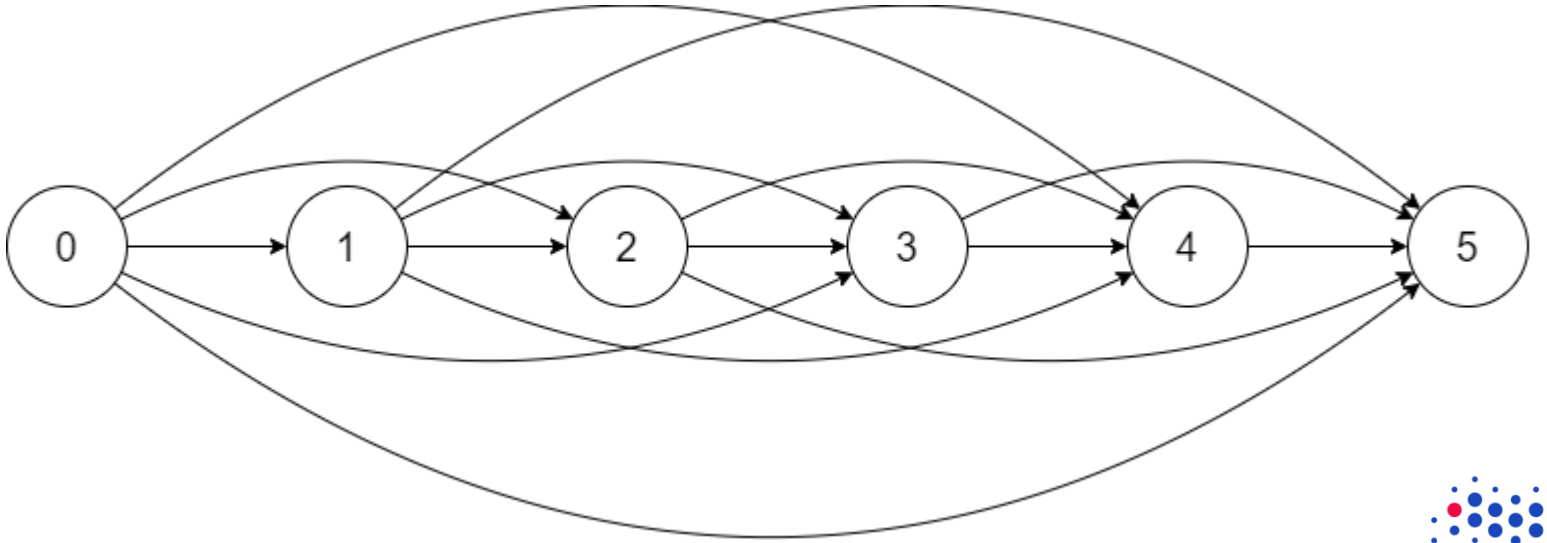
УНИВЕРСИТЕТ ИТМО

Динамическое программирование

ITIVITI

- gain[*N*] = max(gain[*N* – *i*] + price[*i*]) среди всех *i* <= *N*

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	15	17	20
gain								



LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Динамическое программирование

ITIVITI

- gain[*N*] = max(gain[*N* – *i*] + price[*i*]) среди всех *i* <= *N*

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	15	17	20
gain	*i* = 1	*i* = 2	*i* = 3					
	1	5	8					

- gain[4]:
  - *i* = 1: gain[3] + price[1] = 8 + 1 = 9
  - *i* = 2: gain[2] + price[2] = 5 + 5 = **10**
  - *i* = 3: gain[1] + price[3] = 1 + 8 = 9
  - *i* = 4: gain[0] + price[4] = 0 + 9 = 9

LET IT TRADE

ДП: Восстановление решения

ITIVITI

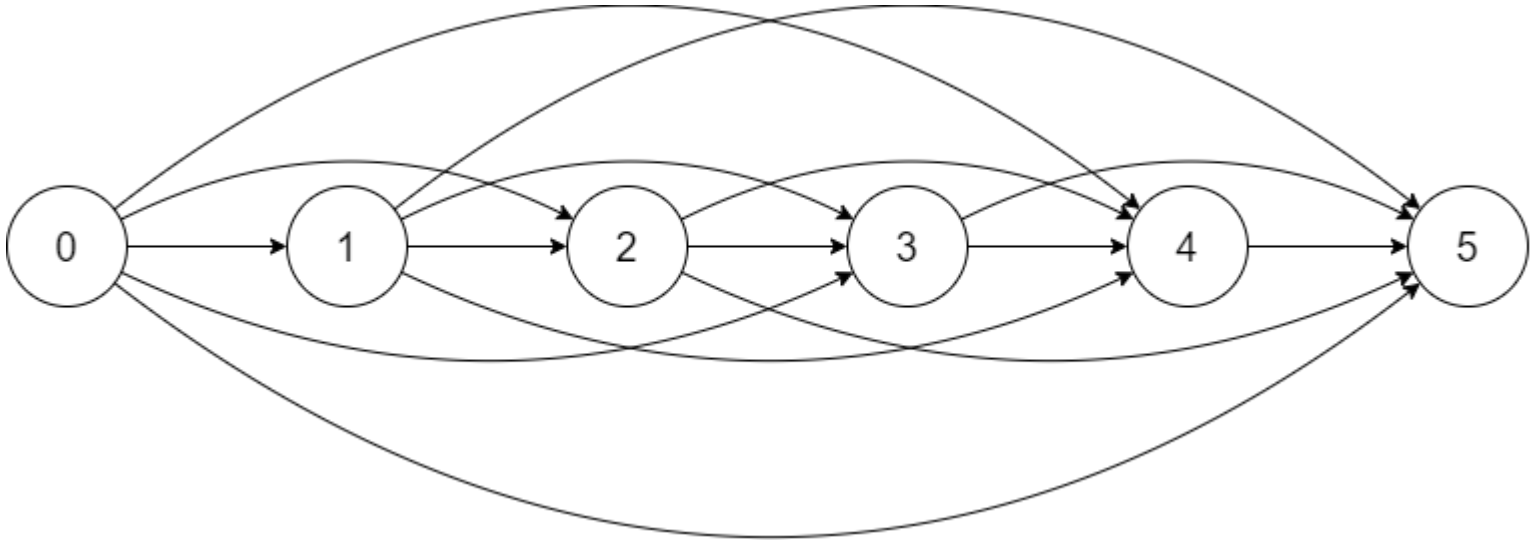
LET IT TRADE

- Как восстановить оптимальное решение?

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	15	17	20
gain	$i = 1$	$i = 2$	$i = 3$	$i = 2$	$i = 2$	$i = 3$	$i = 2$	$i = 2$
	1	5	8	10	13	16	18	21

- Можно получить из значений i:
  - От бревна длины 8 взяли $i = 2$, осталось 6
  - От бревна длины 6 взяли $i = 3$, осталось 3
  - От бревна длины 3 взяли $i = 3$, осталось 0
- Ответ: gain[8] = p[2] + p[3] + p[3] = 5 + 8 + 8 = **21**

ДП: Асимптотика



- Сложность алгоритма – построим граф подзадач
  - Пространственная сложность – кол-во вершин
  - Временная сложность построения решения – кол-во ребер
  - Временная сложность восстановления решения – кол-во вершин

ДП: Порядок обхода

ITIVITI

LET IT TRADE

- В заданной строке s найти подстроку-палиндром максимальной длины
- $p[i][j]$ – подстрока с позиции i (включ.) и до позиции j (исключ.)
  - $p[i][i]$ и $p[i][i+1]$ – палиндром для любого i
  - $p[i][j]$ – палиндром, если $p[i+1][j-1]$ палиндром и $s[i] == s[j-1]$
- Чтобы найти палиндром максимальной длины, требуется обойти все пары i, j такие, что $i <= j$
  - В каком порядке?

ДП: Порядок обхода

- $p[i][j]$ зависит от $p[i+1][j-1]$:

p	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5
i = 0	true	true				
i = 1		true	true			
i = 2			true	true		
i = 3				true	true	
i = 4					true	true

- Обход по строчкам:

```
for (int i = LEN-1; i >= 0; i--)
  for (int j = 0; j <= LEN; j++) { ... }
```

- По столбцам:

```
for (int j = 0; j <= LEN; j++)
  for (int i = 0; i < LEN; i++) { ... }
```

ДП: Использование памяти

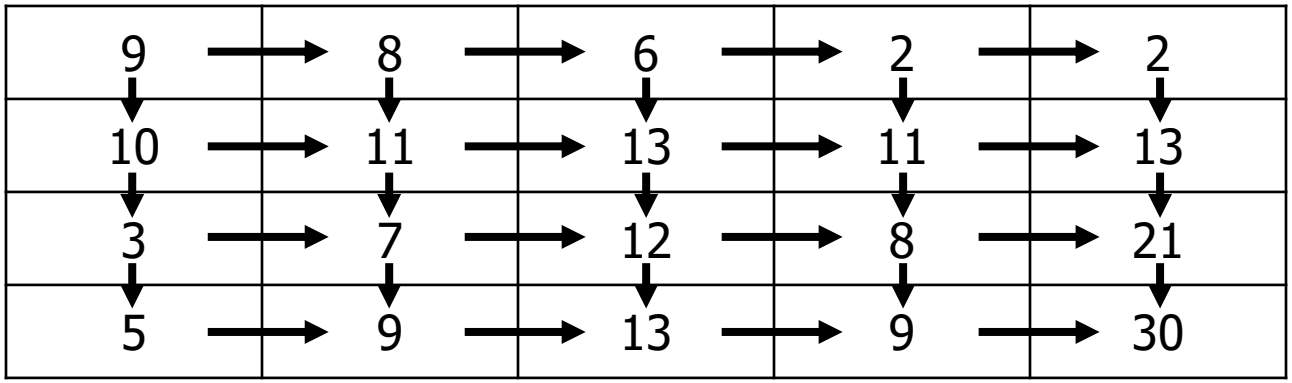
- «Черепашка»:

9	8	6	2	2
10	11	13	11	13
3	7	12	8	21
5	9	13	9	30

- Передвигаясь по клеткам шагами вправо или вниз, найти путь из верхнего левого угла в правый нижний с максимальной суммой ячеек
  - Рекуррентная формула:
    $s[i][j] = \max(s[i-1][j], s[i][j-1]) + a[i][j]$

ДП: Использование памяти

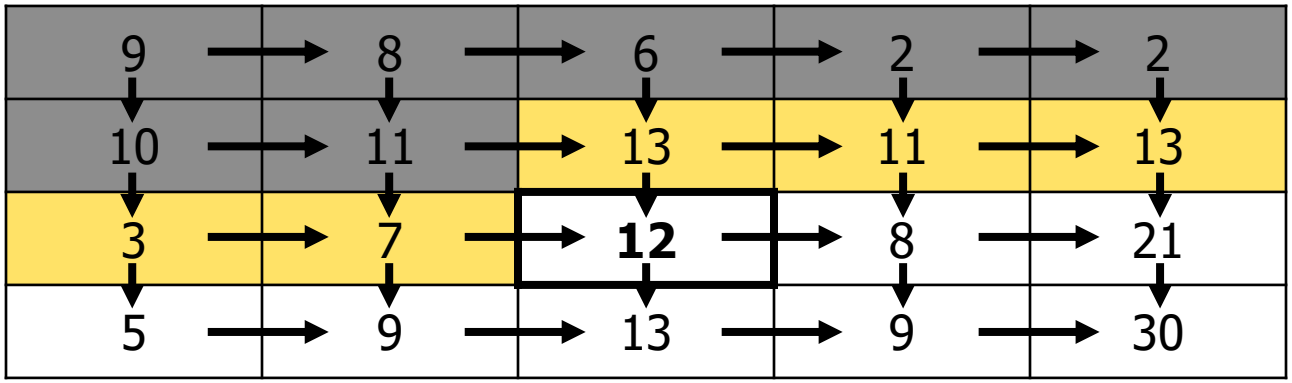
- Граф подзадач:



- Оптимальный путь обхода?
- Время: $O(MN)$
- Память: $O(MN)$
  - Можем ли мы использовать меньше памяти?

ДП: Использование памяти

- Порядок обхода – построчно, слева направо:



- В каждый момент задачи можно разделить на три группы:
  - Решенные, от которых не зависит ни одна нерешенная
  - Решенные, от которых зависит хотя бы одна нерешенная
  - Еще не решенные
- Первую группу в памяти хранить не требуется

ДП: Использование памяти

ITIVITI

- Решение 1. Сохраняем в табличку все

```
01: int solve(int a[N][M])
02: {
03:   int s[N][M];
04:   s[0][0] = a[0][0];
05:   for (int j = 1; j < M; j++)
06:     s[0][j] = s[0][j-1] + a[0][j];
07:
08:   for (int i = 1; i < N; i++) {
09:     s[i][0] = s[i-1][0] + a[i][0];
10:     for (int j = 1; j < M; j++)
11:       s[i][j] = max(s[i-1][j], s[i][j-1]) + a[i][j];
12:   }
13:
14:   return s[N-1][M-1];
15: }
```

- Недостаток: используется много памяти
- Преимущество: можем восстановить путь

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

ДП: Использование памяти

- Решение 2. Сохраняем только необходимое

```
01: int solve(int a[N][M])
02: {
03:   int s[M];
04:   s[0] = a[0][0];
05:   for (int j = 1; j < M; j++)
06:     s[j] = s[j-1] + a[0][j];
07:
08:   for (int i = 1; i < N; i++) {
09:     s[0] = s[0] + a[i][0];
10:     for (int j = 1; j < M; j++)
11:       s[j] = max(s[j], s[j-1]) + a[i][j];
12:   }
13:
14:   return s[M-1];
15: }
```

- Дополнительная память – $O(M)$

Динамическое программирование

ITIVITI

- Перебор с запоминанием
- Используется для поиска оптимальных решений
  - Оптимальное решение задачи состоит из оптимальных решений подзадач
  - Подзадачи часто пересекаются => запоминаем их решения, чтобы не решать много раз
  - Дополнительная информация о структуре решения позволит быстро восстановить его после построения
- Удобно представлять задачу в виде графа подзадач
  - Временная сложность пропорциональна кол-ву ребер
  - Пространственная сложность пропорциональна кол-ву вершин (но можно лучше!)

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

ДП: Другие примеры

ITIVITI

- Проверка орфографии – расстояние Левенштейна
- Анализ цепочек ДНК
  - Алгоритм Нидлмана-Вунша для выравнивания последовательностей
  - Максимальная общая подстрока
- Вычислительная математика
  - Оптимизация порядка перемножения матриц
  - Метод неопределенных коэффициентов
- Алгоритмы на графах – алгоритм Флойда-Уоршелла, алгоритм Беллмана-Форда
- Задача о рюкзаке (куче камней)

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Расстояние Левенштейна

- Определен набор операций над строкой:
  - Вставка одного символа
  - Удаление одного символа
  - Замена символа на другой
- Какое минимальное количество операций требуется совершить, чтобы превратить строку S_1 в строку S_2?
- ДП: рассмотрим все способы, которыми можно «поправить» последний символ
  - Если последний символ правильный, ничего поправлять не нужно

Расстояние Левенштейна

- $S_1 = a_1a_2a_3\ldots a_{n-1}\mathbf{a_n}$, $S_2 = b_1b_2b_3\ldots b_{m-1}\mathbf{b_m}$:
  - Преобразуем $a_1\ldots a_n$ в $b_1\ldots b_{m-1}$, затем вставим b_m
  - Удалим a_n, затем преобразуем $a_1\ldots a_{n-1}$ в $b_1\ldots b_m$
  - Преобразуем $a_1\ldots a_n$ в $b_1\ldots b_{m-1}a_n$, затем поменяем a_n на b_m
- «костоло**м**» → «колоко**л**»:
  - «костолом» → … → «колоко» → «колокол»
  - «костолом» → «костоло» → … → «колокол»
  - «костолом» → … → «колоком» → «колокол»

Расстояние Левенштейна

D	""	к	о	с	т	о	л	о	м
""	0								
к									
о									
л									
о									
к									
о									
л									

- Если $S_1[i] = S_2[j]$: $D[i][j] = D[i-1][j-1]$
- Иначе: $D[i][j] = \min(D[i-1][j], D[i][j-1], D[i-1][j-1]) + 1$

Расстояние Левенштейна

D	""	к	о	с	т	о	л	о	м
""	0	1	2	3	4	5	6	7	8
к	1	0	1	2	3	4	5	6	7
о	2	1	0	1	2	3	4	5	6
л	3	2	1	1	2	3	3	4	5
о	4	3	2	2	2	2	3	3	4
к	5	4	3	3	3	3	3	4	4
о	6	5	4	4	4	3	4	3	4
л	7	6	5	5	5	4	3	4	**4**

- Восстановим решение: если $S_1[i] \neq S_2[j]$, выбираем минимальный из $D[i-1][j]$, $D[i][j-1]$, $D[i-1][j-1]$

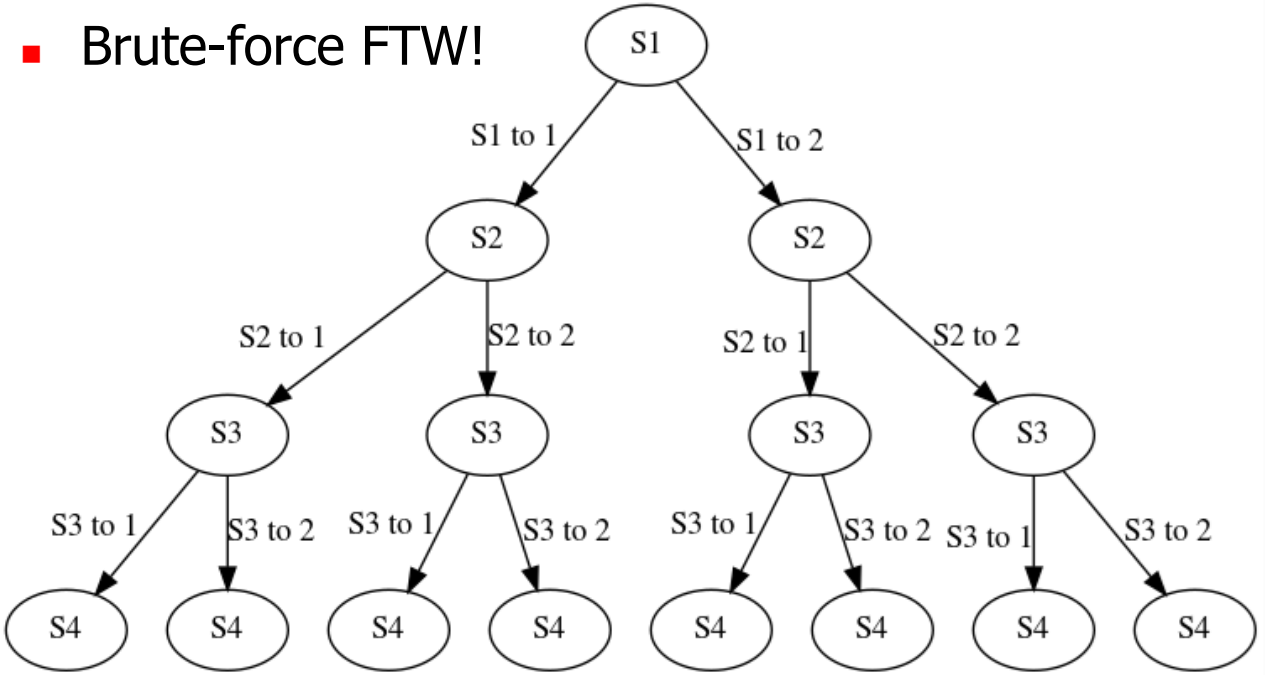
Расстояние Левенштейна

D	""	к	о	с	т	о	л	о	м
""	**0**	1	2	3	4	5	6	7	8
к	1	**0**	1	2	3	4	5	6	7
о	2	1	**0**	1	2	3	4	5	6
л	3	2	1	**1**	**2**	3	3	4	5
о	4	3	2	2	2	**2**	3	3	4
к	5	4	3	3	3	3	**3**	4	4
о	6	5	4	4	4	3	4	**3**	4
л	7	6	5	5	5	4	3	4	**4**

«ко**с**толом» → «ко**л**толом» (замена)
«кол**т**олом» → «кололом» (удаление)
«колo**л**ом» → «колo**к**ом» (замена)
«колоко**м**» → «колоко**л**» (замена)

Куча камней

- Timus 1005. Куча камней
  - N камней целочисленного веса – S_1, S_2, S_3, …, S_N. Распределить камни по двум кучам так, чтобы разность весов куч была минимальной
  - Brute-force FTW!



ITIVITI

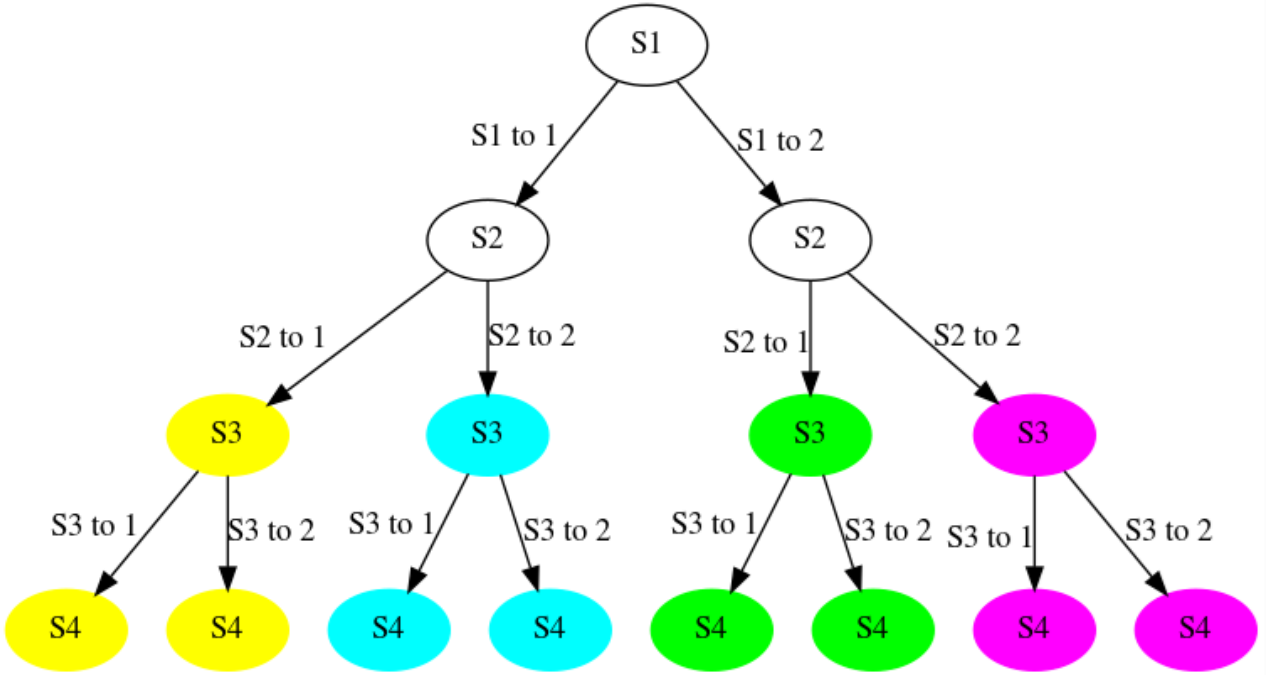
LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО

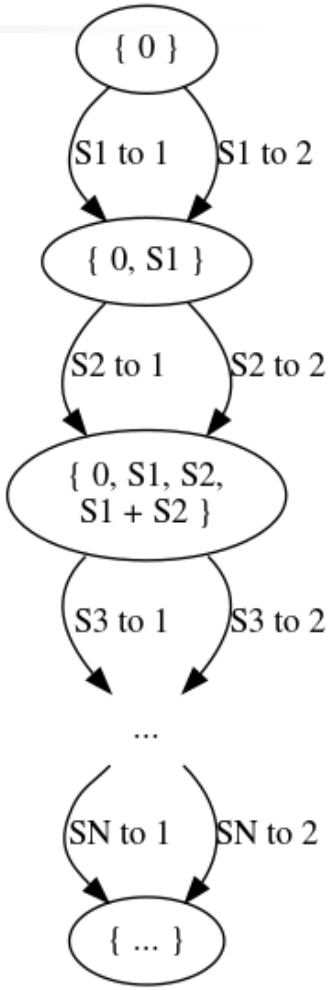
Куча камней

- Полный перебор определяет для одной из куч все возможные веса, которые можно получить из данных камней
- «Перебор с запоминанием»: запомним, какие веса можно получить с помощью камней S_1, S_2, …, S_{k-1}



Куча камней

- Пусть A_k – множество весов кучи №1, которые можно получить из камней S_1, S_2, …, S_k
- Сформируем A_k рекуррентно:
  - По сравнению с A_{k-1}, вес кучи либо не изменится, либо увеличится на S_k
  - $\{s\} \rightarrow \{s, s + S_k\}$ для каждого s в A_{k-1}
  - $A_k = A_{k-1} \cup \{s + S_k : \forall s \in A_{k-1}\}$
- Вариант реализации множеств: массив $A[N + 1][S + 1]$, где $S = S_1 + S_2 + \ldots + S_N$
  - A[k][s] = 1, если существует набор камней из S_1, S_2, …, S_k такой, что их сумма равна s; иначе A[k][s] = 0



Куча камней

```
A[k][s] = A[k – 1][s – S[k]] || A[k – 1][s]
```

	0	1	2	…	S_1	…	S_2	…	$S_1 + S_2$	…	S - 1	S	Представление
0	1	0	0	…	0	…	0	…	0	…	0	0	{0}
1	1	0	0	…	1	…	0	…	0	…	0	0	{0, S_1}
2	1	0	0	…	1	…	1	…	1	…	0	0	{0, S_1, S_2, $S_1 + S_2$}
…	…	…	…	…	…	…	…	…	…	…	…	…	…
N	1	0	1	…	1	…	1	…	1	…	0	1	{ … }

- Решение задачи:
  - Ищем такое M_1, что $A[N][M_1] = 1$ и $|S/2 – M_1|$ минимальное из всех
  - $M_2 = S – M_1$; отсюда ответ = $|M_1 – M_2|$.

Классический рюкзак

```
A[k][s] = max(A[k – 1][s – S[k]] + P[k], A[k – 1][s])
```

	0	1	2	…	S_1	…	S_2	…	$S_1 + S_2$	…	S - 1	S	Представление
0	0	0	0	…	0	…	0	…	0	…	0	0	{0}
1	0	0	0	…	4	…	0	…	0	…	0	0	{0, S_1}
2	0	0	0	…	4	…	5	…	9	…	0	0	{0, S_1, S_2, $S_1 + S_2$}
…	…	…	…	…	…	…	…	…	…	…	…	…	…
N	0	0	3	…	7	…	5	…	12	…	0	P	{ … }

- Какие предметы можно положить в рюкзак размера T, чтобы их суммарная стоимость была максимальной?
  - Ответ – ищем максимальное A[N][s] для $0 \le s \le T$
  - Восстановление решения?

Литература

ITIVITI

LET IT TRADE

При подготовке к лекции использовались:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms (4th edition, 2022)
- S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani. Algorithms (2006)
- С. М. Акулов, О. А. Пестов. Динамическое программирование. 2-е издание (электронное)
- https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование

Что дальше?

ITIVITI

LET IT TRADE

- Почитать:
  - «Большая книга алгоритмов» от MIT: Cormen, Leiserson, Rivest, Stein “Introduction to Algorithms”
  - Sedgewick “Algorithms” – глубокий разбор ключевых алгоритмов, обзор применений в индустрии
  - Конспекты студентов КТ ИТМО по АСД
- Посмотреть:
  - David Galles и его визуализации
- Порешать:
  - CodeForces, ACMP, LeetCode

УНИВЕРСИТЕТ ИТМО

ITIVITI

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО