Построение и анализ алгоритмов

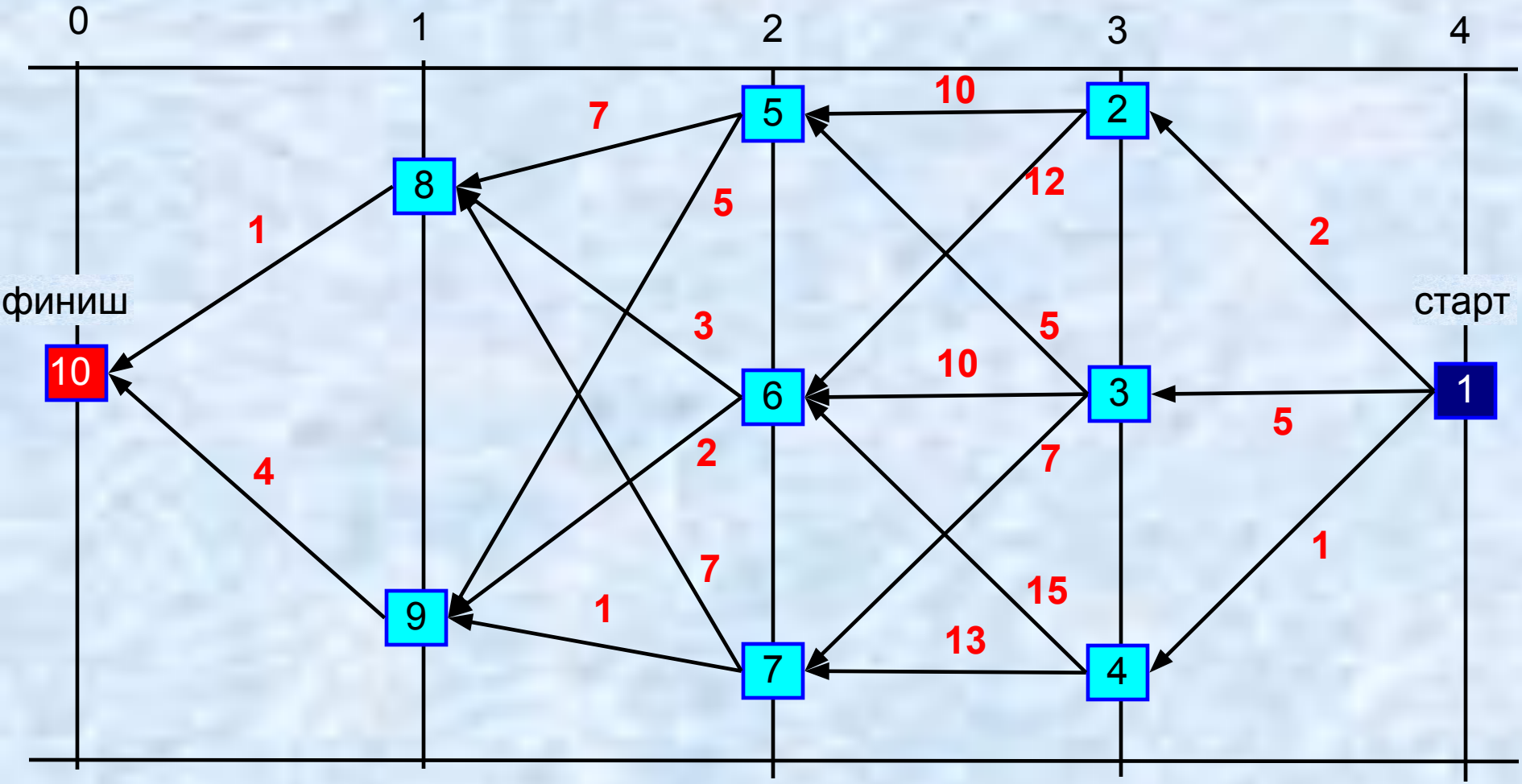
Лекция 3

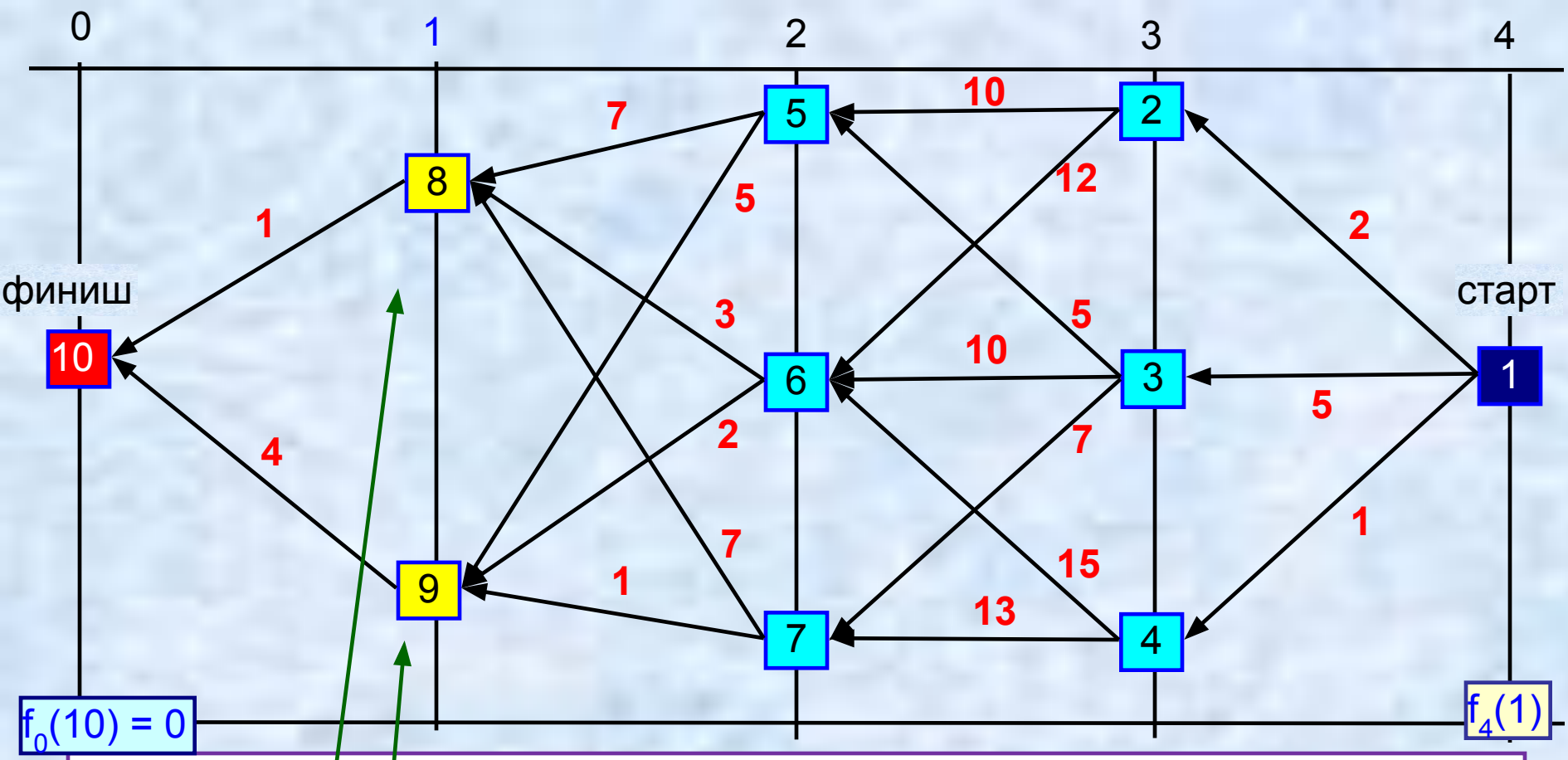
Динамическое программирование

Динамическое программирование

Пример 1:

путь минимальной стоимости в слоистой сети (дорог)





Пусть $f_n(s)$ – стоимость пути от вершины s до финиша на отрезке из n последних шагов (т.е. s — из слоя n).

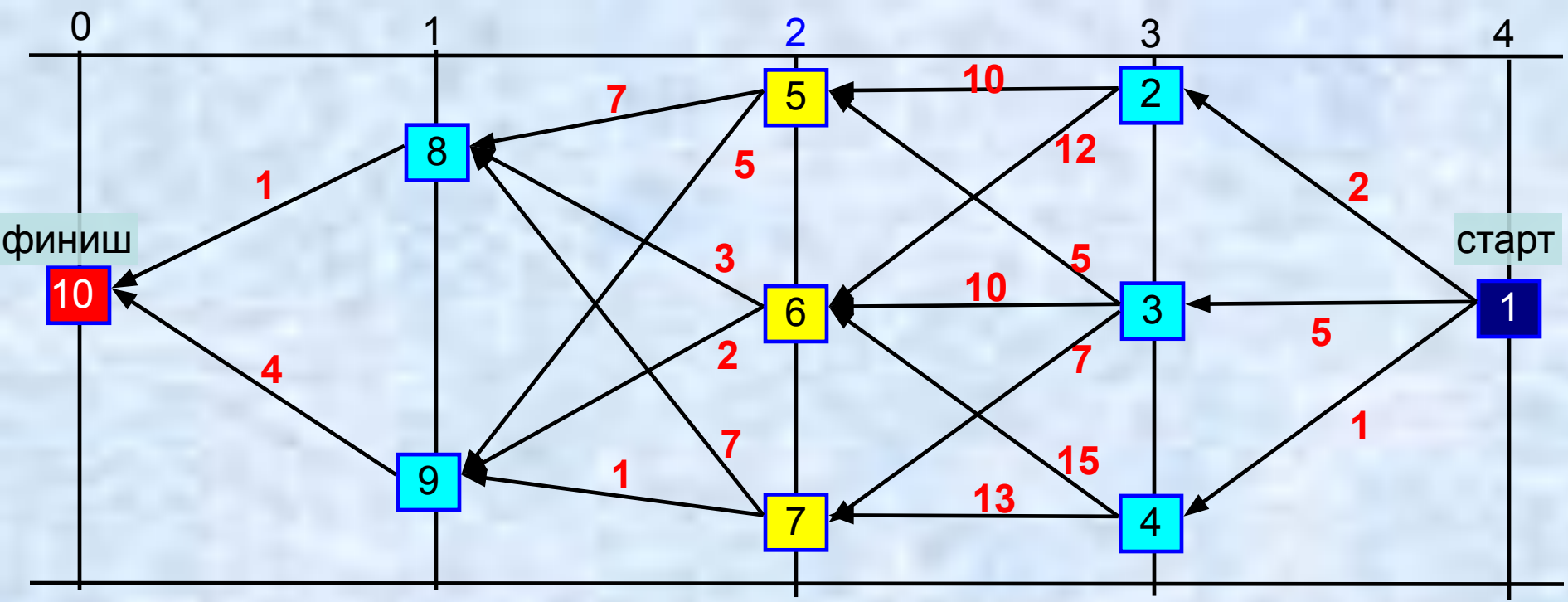
Требуется найти $f_4(1)$.

Ясно, что $f_0(10) = 0$, (0-й слой)

$f_1(8) = 1$, $f_1(9) = 4$. (1-й слой)

Таблица 1

вершина	8	9
следующая	10	10
стоимость	1	4



2-й слой

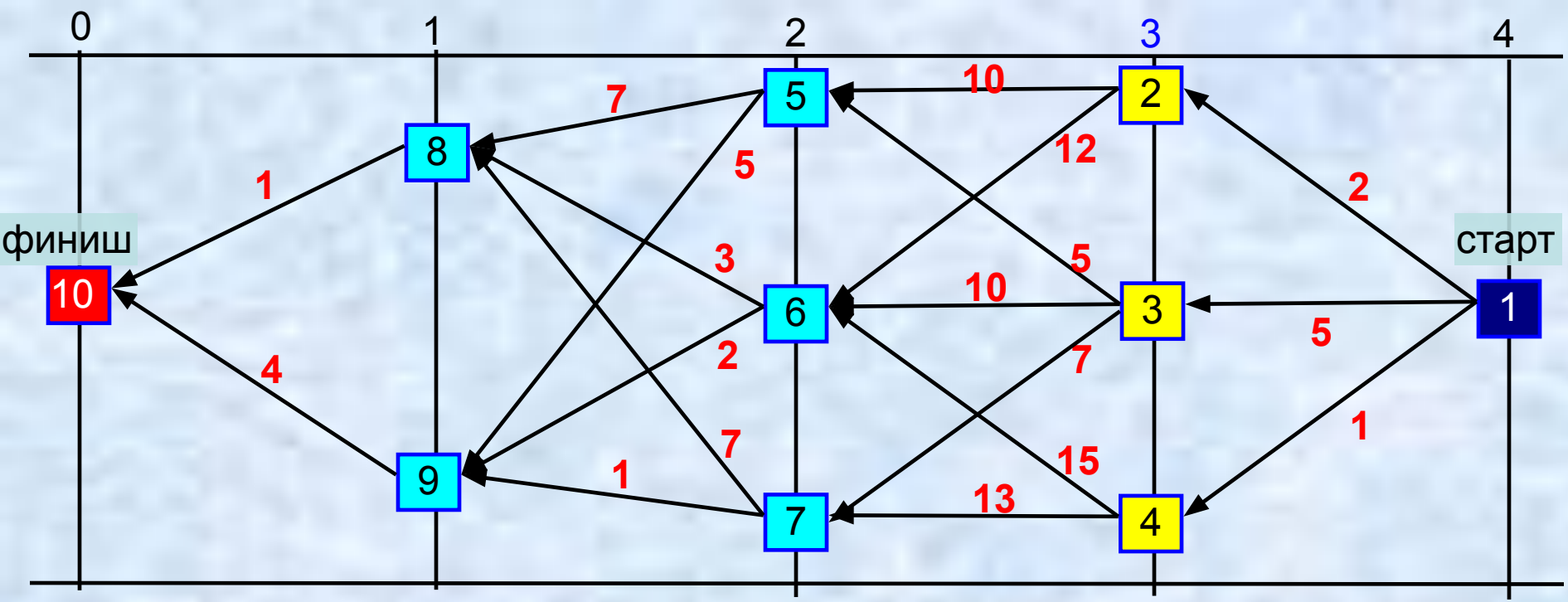
$f_2(5)= \min \{ C_{5,8}+f_1(8) , C_{5,9}+f_1(9) \} = \min \{\underline{7+1}, 5+4\} = 8.$

$f_2(6)= \min \{ C_{6,8}+f_1(8) , C_{6,9}+f_1(9) \} = \min \{\underline{3+1}, 2+4\} = 4.$

$f_2(7)= \min \{ C_{7,8}+f_1(8) , C_{7,9}+f_1(9) \} = \min \{7+1, \underline{1+4}\} = 5.$

Таблица 2

вершина	5	6	7
следующая	8	8	9
стоимость	8	4	5



3-й слой

$f_3(2)= \min \{ C_{2,5}+f_2(5) , C_{2,6}+f_2(6) \} = \min \{10+8, \underline{12+4}\} = 16.$

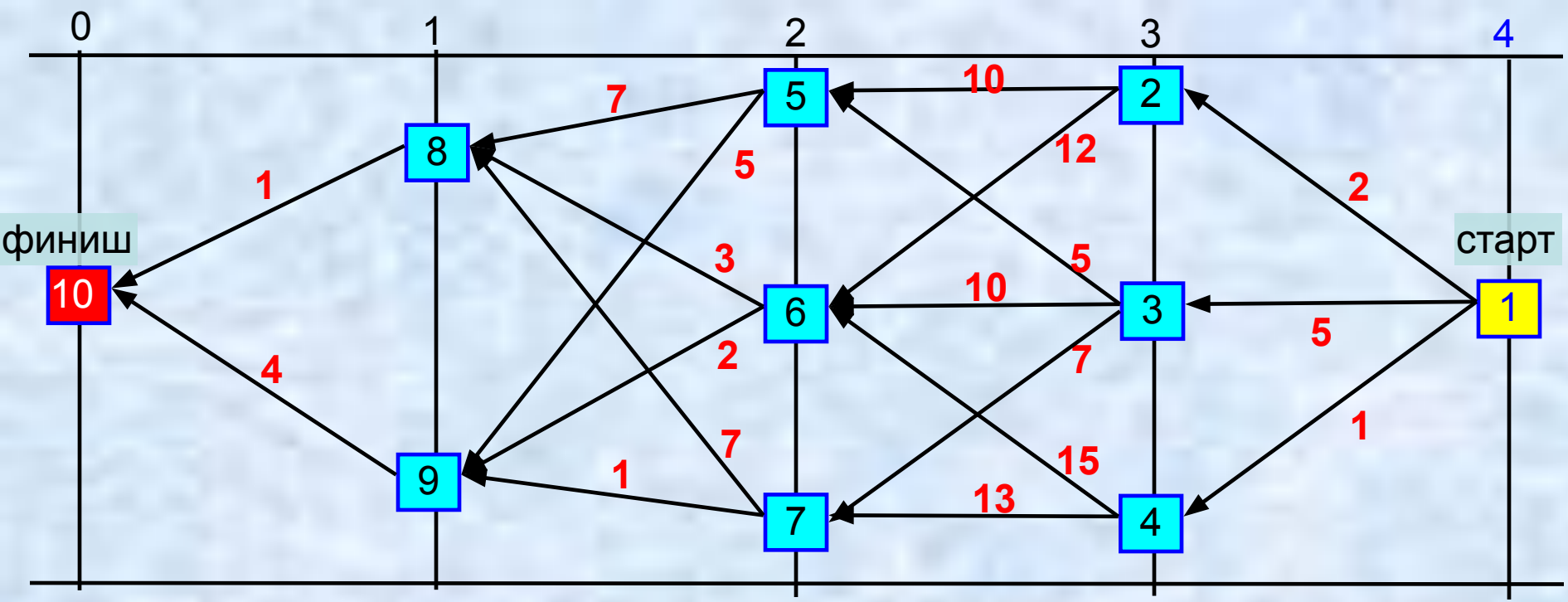
$f_3(4)= \min \{ C_{4,6}+f_2(6) , C_{4,7}+f_2(7) \} = \min \{15+8, \underline{13+5}\} = 18.$

$f_3(3)= \min \{ C_{3,5}+f_2(5) , C_{3,6}+f_2(6), C_{3,7}+f_2(7) \} =$

$= \min \{ 5+8, 10+4, \underline{7+5} \} = 12.$

Таблица 3

вершина	2	3	4
следующая	6	7	7
стоимость	16	12	18



4-й слой (последний)

$f_4(1) = \min \{ C_{1,2}+f_3(2) , C_{1,3}+f_3(3), C_{1,4}+f_3(4), \} =$

$= \min \{2+16, \underline{5+12,} 1+18\} = 17.$

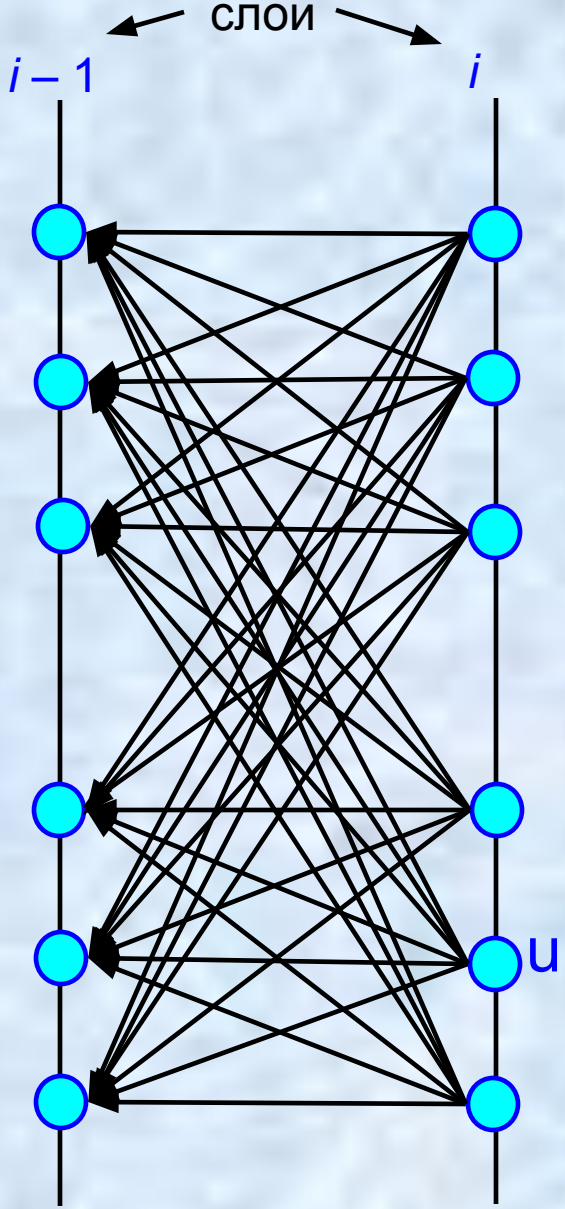
Т.о. путь 1 – 3 – 7 – 9 – 10

(восстанавливается по таблицам)

Стоимость = 17 = 5+7+1+4

Таблица 4

вершина	1
следующая	3
стоимость	17



В общем случае

$$\forall u \in I_i :$$

$$f_i(u) = \min \{ f_{i-1}(v) + C_{u,v} \mid v \in I_{i-1} \},$$

где I_i – множество вершин в слое *i*.

Таблица *i*

Вершина u	*	*	*	*	*	*
Следующая v	*	*	*	*	*	*
Стоимость f	*	*	*	*	*	*

Основные особенности метода ДП

1. Рекуррентное соотношение

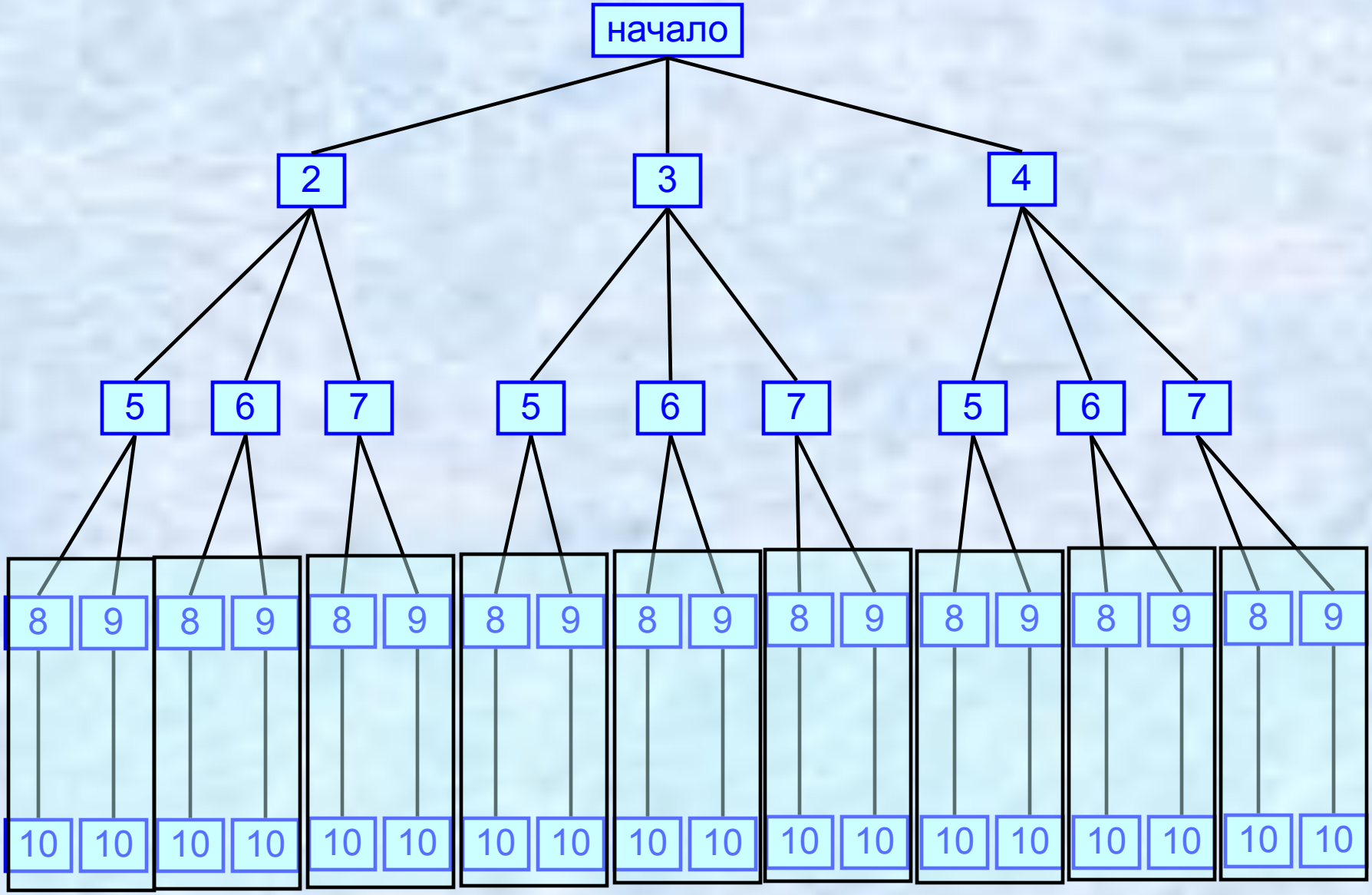
$f_i(u) = \min\{ f_{i-1}(v) + C_{u,v} \mid v \in I_{i-1} \},\ i \in 1..n,\ f_0(u)=0$

Большая задача решается на основе решений меньших задач

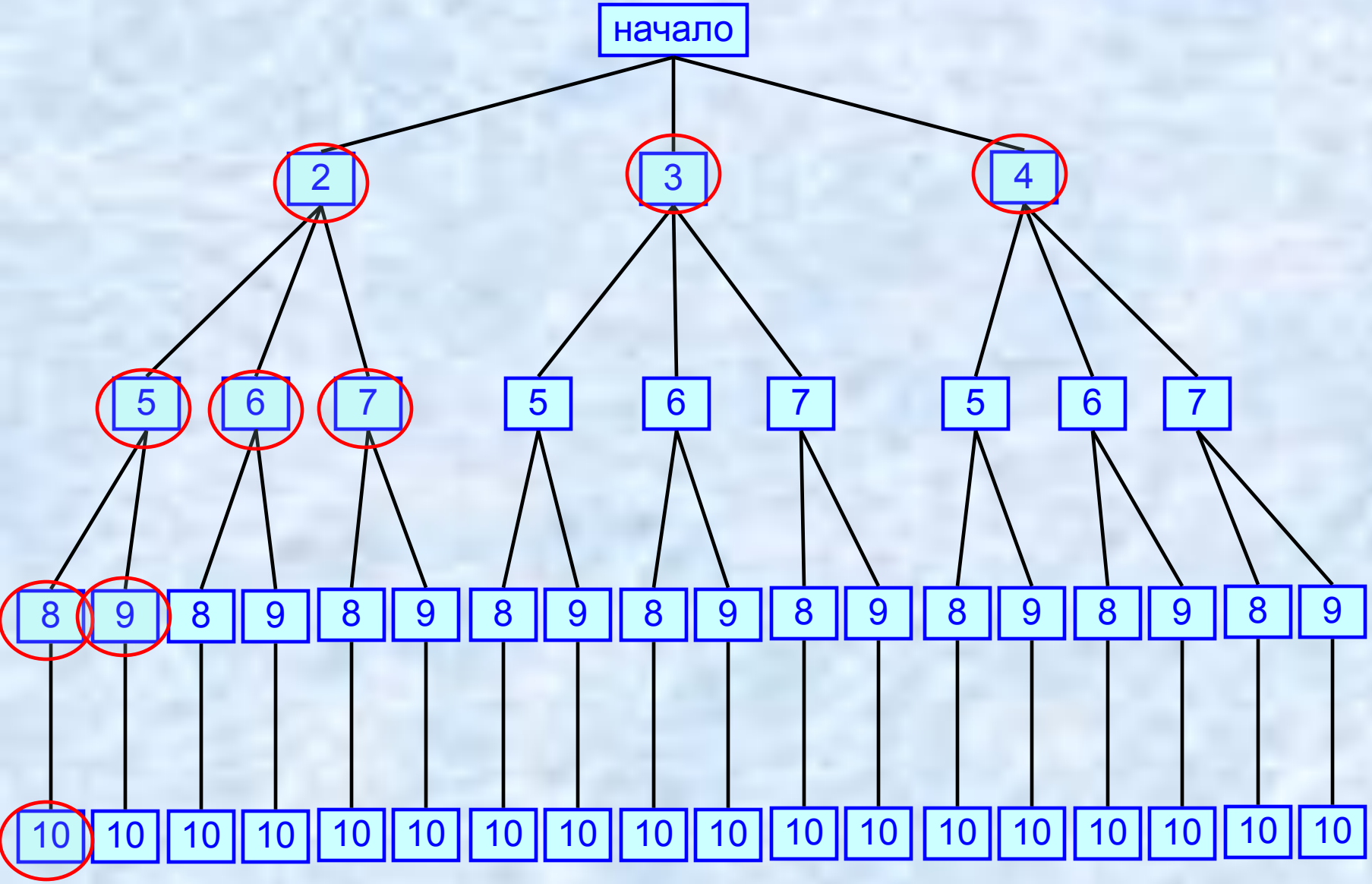
2. Хранение таблиц
3. Принцип оптимальности:

   Часть (например, $f_{i-1}(v)$) *оптимального решения* $f_i(u)$ должна быть *оптимальна*

Решение методом ветвей и границ



Решение методом ветвей и границ



Динамическое программирование. Пример 2: Задача о порядке перемножения матриц

Рассмотрим произведение матриц

$$M_1 \times M_2 \times M_3 \times \ldots \times M_{n-1} \times M_n.$$

Каждая матрица M_i имеет размер $r_{i-1} \times r_i$.

$$M_1(r_0;r_1) \times M_2(r_1;r_2) \times M_3(r_2;r_3) \times \ldots \times M_{n-1}(r_{n-2};r_{n-1}) \times M_n(r_{n-1};r_n).$$

Вычисление произведения двух матриц – размер первой $n \times p$ и размер второй $p \times m$ – требует $n\ p\ m$ умножений их элементов:

$$C(n;m) = A(n;p) \times B(p;m)$$

$$c_{ij} = \sum_{k=1..p} (a_{ik} * b_{kj}) \quad \text{для } i \in 1..n,\ j \in 1..m$$

Задача о порядке перемножения матриц

Общее количество элементарных операций умножения, требуемое при вычислении произведения цепочки матриц, *зависит от порядка*, в котором производятся попарные умножения матриц.

Требуется найти такой *порядок* перемножения матриц, который минимизирует общее количество элементарных операций умножения.

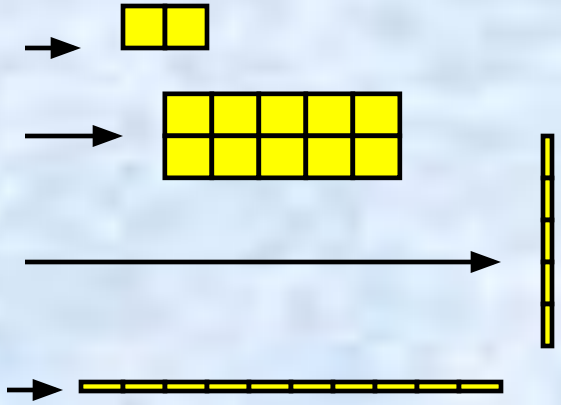
Пример: $M_1 \times M_2 \times M_3 \times M_4$,

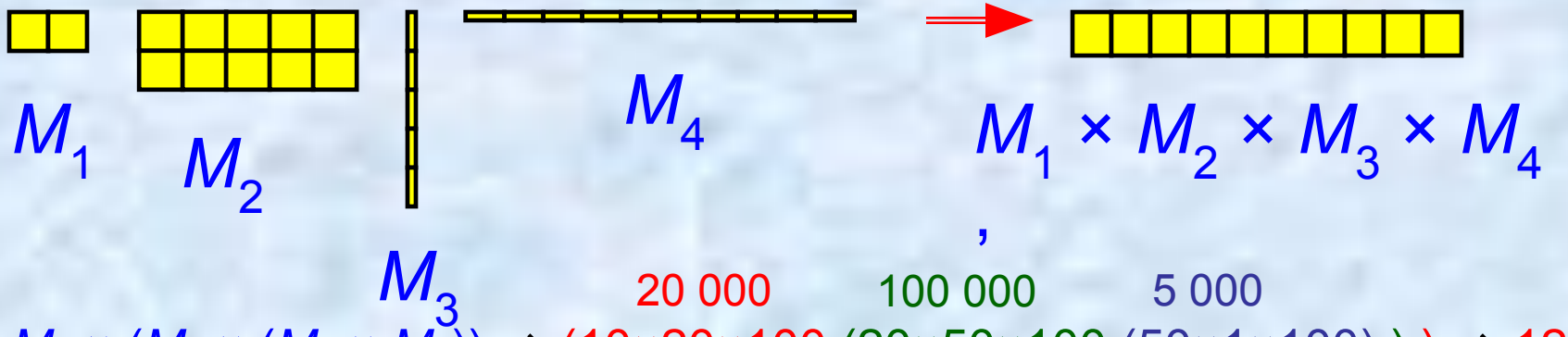
где *размер* $(M_1) = 10 \times 20$,

размер $(M_2) = 20 \times 50$,

размер $(M_3) = 50 \times 1$,

размер $(M_3) = 1 \times 100$.





,

1) $M_1 \times (M_2 \times (M_3 \times M_4)) \Rightarrow$ (10×20×100 (20×50×100 (50×1×100))) ⇒ 125 000

(20 000 — 100 000 — 5 000)

2) $(M_1 \times (M_2 \times M_3)) \times M_4 \Rightarrow$ ((10×20×1 (20×50×1)) 10×1×100) ⇒ 2 200

(200 — 1 000 — 1 000)

Рекуррентное соотношение

Пусть m_{ij} – *оптимальное* количество умножений, требуемое для вычисления произведения цепочки матриц $M(i, j) = M_i \times M_{i+1} \times \ldots \times M_{j-1} \times M_j$,

где $1 \le i \le j \le n$.

Очевидно, что $M(i, i) = M_i$ и $m_{ii} = 0$,

а m_{1n} – соответствует решению задачи для исходной цепочки $M(1, n)$.

При $1 \le i < j \le n$ справедливо :

$$m_{ij} = \mathrm{Min}\{m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j \mid i \le k < j\}.$$

$$M(i, j) = M(i, k) \times M(k+1, j), \quad i \le k < j$$

$r_{i-1} \times r_j \qquad r_{i-1} \times r_k \quad r_k \times r_j$

1) Заметим, что в правой части равенства разности индексов $k - i$ и $j - k - 1$ у слагаемых m_{ik} и $m_{k+1, j}$ меньше, чем разность индексов $j - i$ в m_{ij} ($k-i < j-i$ и $j-k-1 < j-i$).

Таким образом, рекуррентное соотношение следует решать, начиная с $m_{ii} = 0$ и последовательно увеличивая разность индексов $j - i$ до тех пор, пока не получим m_{1n}.

2) Удобно представлять результаты вычислений в виде таблицы.

В этой таблице строка с номером l состоит из ячеек $T(i, j)$, индексы которых связаны соотношением $j - i = l$.

Т.е. $j = i + l$ и $T(i, j) = T(i, i + l)$.

В ячейках таблицы $T(i, j)$ хранятся вычисленные значения m_{ij} и те значениея $q_{ij} = k$ в диапазоне $i \le k < j$, при которых был получен

$$\text{Min}\{ m_{ik} + m_{k+1,\,j} + r_{i-1} \times r_k \times r_j \}.$$

$l = 0$	$T(1, 1)$	$T(2, 2)$	$T(3, 3)$	$T(4, 4)$	…	$T(n-1, n-1)$	$T(n, n)$
$l = 1$	$T(1, 2)$	$T(2, 3)$	$T(3, 4)$	…	$T(n-2, n-1)$	$T(n-1, n)$	
$l = 2$	$T(1, 3)$	$T(2, 4)$	…	$T(n-3, n-1)$	$T(n-2, n)$		
…	…	…	…	…			
$l = n-3$	$T(1, n-2)$	$T(2, n-1)$	$T(3, n)$				
$l = n-2$	$T(1, n-1)$	$T(2, n)$					
$l = n-1$	$T(1, n)$						

Алгоритм вычисляет оптимальное значение m_{1n} и заполняет таблицу T по строкам *сверху вниз*:

```
for (i = 1; i < n; i++) m[i, i] = 0; {заполнение первой строки}
for l := 1 to n –1 do {по строкам}
    for i := 1 to n – l do {по ячейкам строки L}
    begin
        j := i + l;
        {заполнение T(i, j):}
        m[i, j] := +∞;
        for k := i to j – 1 do
        begin
            s := m[i, k] + m [k +1,j] + r_{i –1} * r_k * r_j;
            if s < m[i, j] then
            begin   m[i, j] := s;
                q[i, j] := k
            end { if }
        end { for k }
    end { for i }
```

Алгоритм вычисляет оптимальное значение m_{1n} и заполняет таблицу T по строкам *сверху вниз*:

```
for (i = 1; i < n; i++) m[i][i] = 0; //заполнение первой строки табл.
for (L = 1; L < n; L++) //по строкам
 for (i = 1; i < n-L+1; i++) {//по ячейкам строки L
      j = i + L;
      // заполнение T(i, j):
      m[i][j] = +∞;
      for (k = i; k < j; k++) {
           s = m[i][k] + m[k+1][j] + r(i-1) * r(k) * r(j);
           if (s < m[i][j]){
                m[i][j] = s;
                q[i][j] = k;
           }
      }
 }
```

Характеристики алгоритма

Алгоритм требует:

- порядка $n^2/2$ элементов памяти для хранения таблицы
- около $n^3/3$ выполнений тела внутреннего цикла.

Пример см. далее

Пример вычисления M1 × M2 × M3 × M4 (см. слайд 13)

Для заполнения строки таблицы при $l = 1$ вычислим последовательно

$m_{1,2} = m_{1,1} + m_{2,2} + r_0 \times r_1 \times r_2 = 10 \times 20 \times 50 = 10\,000,$

$m_{2,3} = m_{2,2} + m_{3,3} + r_1 \times r_2 \times r_3 = 20 \times 50 \times 1 = 1000,$

$m_{3,4} = m_{3,3} + m_{4,4} + r_2 \times r_3 \times r_4 = 50 \times 1 \times 100 = 5000.$

Здесь фактически минимум находить не требуется, так как тело цикла по k выполняется лишь один раз (при $k = i$).

Заполненная строка таблицы есть

$l = 1$	$m_{1,2} = 10\,000$ $q_{1,2} = 1$	$m_{2,3} = 1000$ $q_{2,3} = 2$	$m_{3,4} = 5000$ $q_{3,4} = 3$

$$m_{ij} = \text{Min}\,\{m_{ik} + m_{k+1,\,j} + r_{i-1} \times r_k \times r_j \mid i \le k < j\}.$$

Строка таблицы при $L = 2$

$m_{1,3} = \text{Min}\{m_{1k} + m_{k+1,3} + r_0 \times r_k \times r_3 \mid k = 1, 2\} =$

$= \text{Min}\{m_{1,1} + m_{2,3} + r_0 \times r_1 \times r_3 , m_{1,2} + m_{3,3} + r_0 \times r_2 \times r_3\} =$

$= \text{Min}\{0 + 1000 + 200, 10\,000 + 0 + 500\} =$

$= \text{Min}\{\underline{1200}, 10\,500\} = 1200$ (при $k = 1$),

$m_{2,4} = \text{Min}\{m_{2k} + m_{k+1,4} + r_1 \times r_k \times r_4 \mid k = 2, 3\} =$

$= \text{Min}\{m_{2,2} + m_{3,4} + r_1 \times r_2 \times r_4 , m_{2,3} + m_{4,4} + r_1 \times r_3 \times r_4\} =$

$= \text{Min}\{0 + 5000 + 100\,000, 1000 + 0 + 2000\} =$

$= \text{Min}\{105\,000, \underline{3000}\} = 3000$ (при $k = 3$)

$l = 1$	$m_{1,2} = 10\,000$ $q_{1,2} = 1$	$m_{2,3} = 1000$ $q_{2,3} = 2$	$m_{3,4} = 5000$ $q_{3,4} = 3$
$l = 2$	$m_{1,3} = 1200$ $q_{1,3} = 1$	$m_{2,4} = 3000$ $q_{2,4} = 3$	

$$m_{ij} = \text{Min}\{m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j \mid i \le k < j\}.$$

Последняя строка таблицы (из одной ячейки) $T(1, 4)$:

$$m_{1,4} = \text{Min}\{ m_{1k} + m_{k+1,4} + r_0 \times r_k \times r_4 \mid k = 1, 2, 3\} =$$

$$= \text{Min}\{ m_{1,1} + m_{2,4} + r_0 \times r_1 \times r_4,$$
$$m_{1,2} + m_{3,4} + r_0 \times r_2 \times r_4,$$
$$m_{1,3} + m_{4,4} + r_0 \times r_3 \times r_4 \} =$$
$$= \text{Min}\{0 + 3000 + 20\,000,$$
$$10\,000 + 5000 + 50\,000,$$
$$1200 + 0 + 1000\} =$$
$$= \text{Min}\{23\,000, 65\,000, \underline{2200}\} = 2200 \text{ (при } k = 3\text{)}.$$

$$m_{ij} = \text{Min}\{m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j \mid i \le k < j\}.$$

Вся таблица вычислена и имеет вид

$l = 0$	$m_{1,1} = 0$ $q_{1,1} = 1$	$m_{2,2} = 0$ $q_{2,2} = 2$	$m_{3,3} = 0$ $q_{3,3} = 3$	$m_{4,4} = 0$ $q_{4,4} = 4$
$l = 1$	$m_{1,2} = 10\,000$ $q_{1,2} = 1$	$m_{2,3} = 1000$ $q_{2,3} = 2$	$m_{3,4} = 5000$ $q_{3,4} = 3$	
$l = 2$	$m_{1,3} = 1200$ $q_{1,3} = 1$	$m_{2,4} = 3000$ $q_{2,4} = 3$		
$l = 3$	$m_{1,4} = 2200$ $q_{1,4} = 3$			

$$(M1 \times (M2 \times M3)) \times M4$$

В общем случае *порядок* перемножения матриц легко определить рекурсивно. Пусть имеется функция перемножения двух матриц

func *Mult (A, B: Matrix): Matrix.*

«Набросок» функции перемножения цепочки матриц:

```
func MatrixSeqMult ( i, j: Index): Matrix;     {i ≤ j}
    global q: Tab_q; {указывает, как перемножать}
    var k: Index;     var  A, B: Matrix;
begin
    if i < j then
    begin
        k := q[i, j];
        A := MatrixSeqMult ( i, k);
        B := MatrixSeqMult ( k +1, j);
        Return Mult(A, B)
    end
    else {i = j} Return M_i
end {MatrixSeqMult}
```

«Набросок» функции перемножения цепочки матриц:

```
// Псевдокод
Matrix MatrixSeqMult ( int i, int j)     //    i <= j
// Tab_q q;
{   int k;     Matrix A;   Matrix B;
    if (i < j) {
     k = q[i][j]; //эти значения k обеспечат
                  //оптимальный порядок
     A = MatrixSeqMult ( i, k);
     B = MatrixSeqMult ( k +1, j);
     return Mult(A, B);
    }
    else // i = j
     return M[i]
}
```

В общем случае *порядок* перемножения матриц легко определить рекурсивно.

Используется функция перемножения двух матриц
Matrix Mult (Matrix A, Matrix B);

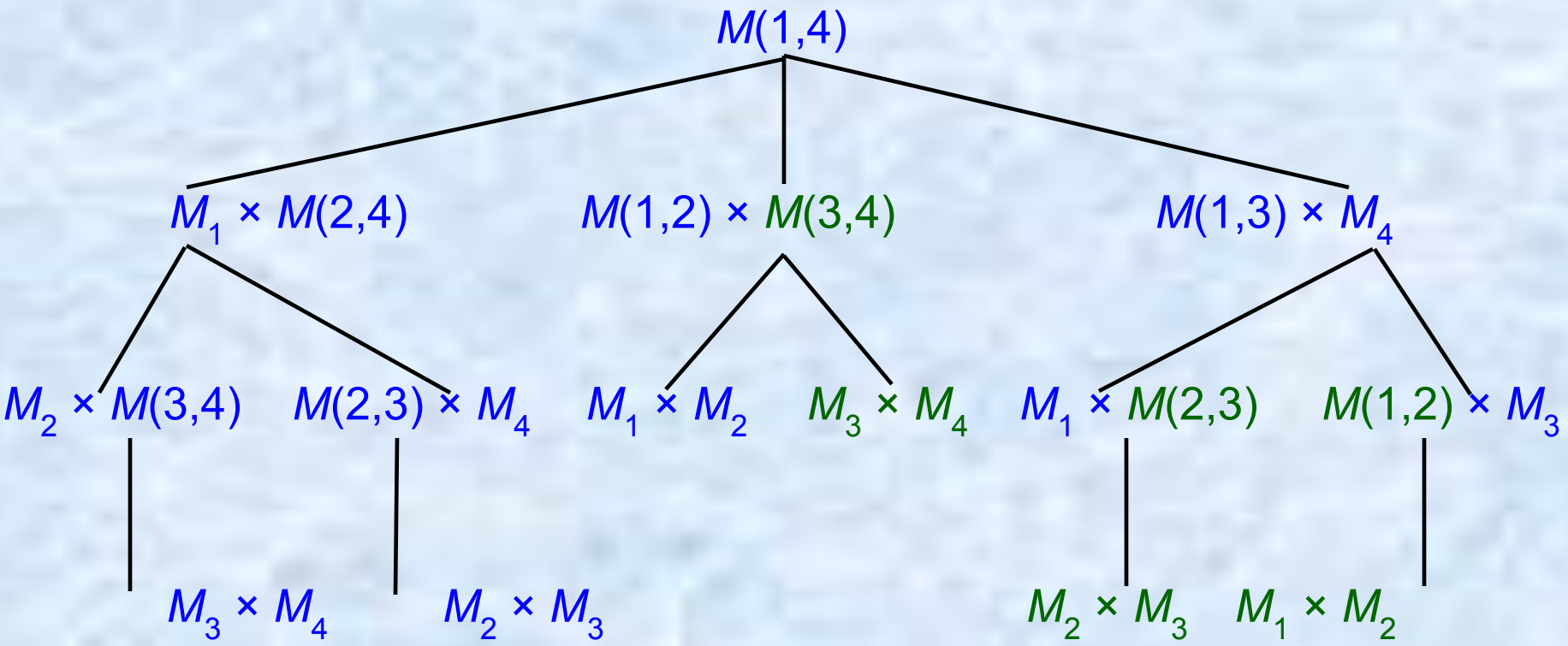
Одна их входных матриц M_i

Полезно сравнить решение, полученное методом ***динамического программирования***, с решением методом ***ветвей и границ***.

В рассмотренном примере возможны следующие 5 вариантов перемножения матриц $M_1 \times M_2 \times M_3 \times M_4$, а именно:

$$M_1 \times (M_2 \times (M_3 \times M_4)),$$
$$M_1 \times ((M_2 \times M_3) \times M_4),$$
$$(M_1 \times M_2) \times (M_3 \times M_4),$$
$$(M_1 \times (M_2 \times M_3)) \times M_4,$$
$$((M_1 \times M_2) \times M_3) \times M_4.$$

Дерево перебора в методе ветвей и границ



В методе динамического программирования повторных вычислений не делается.

Вычисления проводятся так, как будто дерево сканируется снизу вверх, а результаты вычислений сохраняются в таблице и далее используются.

Оценка количества узлов дерева

Оценить количество узлов дерева в общем случае можно подсчётом всех возможных *вариантов расстановок скобок в произведении* матриц.

Пусть p_n – число вариантов расстановок скобок в произведении n сомножителей (включая самые внешние скобки).

Например, для трёх сомножителей abc имеем два варианта $(a(bc))$ и $((ab)c)$, а следовательно, $p_3 = 2$.

В общем случае, считая, что «последнее» по порядку умножение может оказаться на любом из $n-1$ мест, запишем следующее рекуррентное соотношение:

$$p_n = p_1 p_{n-1} + p_2 p_{n-2} + \ldots + p_{n-2} p_2 + p_{n-1} p_1.$$

Начальное условие $p_1 = 1$. Далее

$$p_2 = p_1 p_1 = 1,$$
$$p_3 = p_1 p_2 + p_2 p_1 = 2,$$
$$p_4 = p_1 p_3 + p_2 p_2 + p_3 p_1 = 5.$$

Оказывается [7, с. 393], что решением этого рекуррентного уравнения являются так называемые

числа Каталана $p_n = C_{n-1}$,

где $C_k = (2k \mid k) / (k+1)$,

а запись $(n \mid m)$ обозначает биномиальный коэффициент

$$(n \mid m) = n!/(m!\,(n-m)!).$$

См. также 1.6.10 и 1.7.4 в книге

При больших значениях k удобно использовать формулу Стирлинга

$$k! \approx (2\pi)^{1/2} k^{k+\frac{1}{2}} e^{-k}$$

Тогда для чисел Каталана при больших значениях n справедливо

$$C_n \approx 4^n / n\sqrt{\pi n}$$

т. е. число узлов в дереве перебора есть экспоненциальная функция от n.

Несколько первых чисел Каталана

n	0	1	2	3	4	5	6	7	8	9	10
C_n	1	1	2	5	14	42	132	429	1430	4862	16 796

Ср. C_{n-1} и $(n^3 - n)/3$

Например, при $n = 10$

n	6	7	8	9	10
C_{n-1}	42	132	429	1430	4862
$(n^3 - n)/3$	70	112	168	240	330

Пример 3.
Оптимальные деревья поиска

Ранее при рассмотрении БДП, как правило, предполагалось, что для поиска различные ключи предъявляются **с равной вероятностью**.

Пусть теперь заранее известно, что некоторые ключи предъявляются чаще других.

Тогда расположение «частых» ключей ближе к корню дерева сократит время их поиска и, возможно, среднее время поиска (по разным предъявлениям ключей).

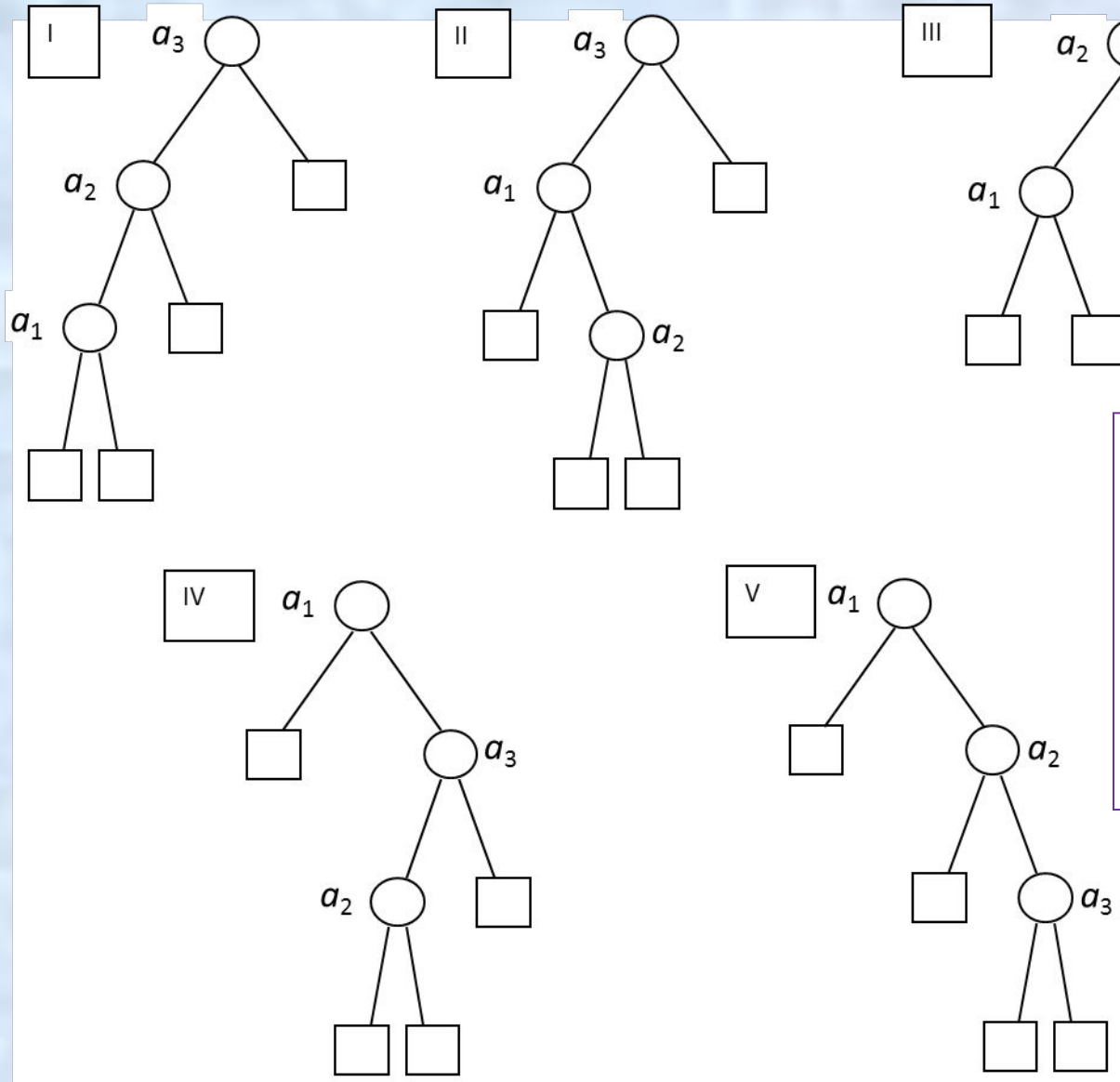


Рис. 2.9. Все различные расширенные БДП из трех элементов $a_1 < a_2 < a_3$

Пример дерева поиска из трёх элементов $a_1 < a_2 < a_3$.

Заданы вероятности предъявления элемента x для поиска: $P(x = a_1) = \alpha$; $P(x = a_2) = \beta$; $P(x = a_3) = \gamma$.

Дерево	Стоимость	Варианты значений α, β и γ			
		α = 1/3 β = 1/3 γ = 1/3	α = 3/6 β = 2/6 γ = 1/6	α = 4/7 β = 2/7 γ = 1/7	α = 5/8 β = 2/8 γ = 1/8
		Значения стоимости при заданных α, β и γ			
I	3α + 2β + γ	6/3	14/6	17/7	20/8
II	2α + 3β + γ	6/3	13/6	15/7	17/8
III	2α + β + 2γ	***5/3***	***10/6***	12/7	14/8
IV	α + 3β + 2γ	6/3	11/6	12/7	13/8
V	α + 2β + 3γ	6/3	***10/6***	***11/7***	***12/8***

Среднее (по всем предъявлениям x) число сравнений (**стоимость**) в случаях успешного поиска как функция переменных α, β и γ,

Постановка задачи

Поиск будет осуществляться среди набора данных

$a_1, a_2, \ldots, a_{n-1}, a_n$.

Пусть последовательность упорядочена:

$a_1 < a_2 < \ldots < a_{n-1} < a_n$.

$A_1, \ldots, A_n$- события, соответствующие вариантам успешных исходов поиска,

т. е. $A_i : (x = a_i)$ для $i \in 1..n$,

$B_0, \ldots, B_n$ - события, соответствующие вариантам неудачных исходов поиска,

т. е. $B_i : (a_i < x < a_{i+1})$ для $i \in 0..n$.

Здесь для упрощения записи событий B_0 и B_n добавлены фиктивные элементы $a_0 = -\infty$ и $a_{n+1} = +\infty$, которые не должны использоваться в алгоритме.

Постановка задачи (продолжение)

Все эти $2n + 1$ событий (исходов поиска)

$$(A_i)_1^n \quad (B_i)_0^n$$

могут быть упорядочены:

$$B_0 < A_1 < B_1 < A_2 < \ldots < B_{n-1} < A_n < B_n.$$

Заданы вероятности (или частоты) этих событий:

$p_i = P(A_i)$ для $i \in 1..n$, и $q_i = P(B_i)$ для $i \in 0..n$.

При этом $\Sigma_{i \in 1..n} p_i + \Sigma_{i \in 0..n} q_i = 1$.

События A_i соответствуют внутренним узлам расширенного дерева поиска, а события B_i – внешним узлам (листьям) расширенного дерева поиска.

Постановка задачи (продолжение)

Тогда среднее число (математическое ожидание) сравнений при поиске можно записать в виде

$$C_{0,n} = \sum_{i=1}^{n} p_i \, (l(A_i)+1) + \sum_{i=0}^{n} q_i \, l(B_i),$$

где $l(x)$ – уровень узла x (или длина пути от корня до узла x) в БДП.

Здесь уровень узла определён так, что l (корень) = 0.

Итак, задача состоит в том, чтобы по заданным *весам*

$$\{p_i\}_1^n \text{ и } \{q_i\}_0^n$$

построить БДП, минимизирующее значение $C_{0,n}$.

Постановка задачи (продолжение)

Такое дерево называют *оптимальным* БДП.

Есть ли сходство этой задачи с задачей построения оптимального префиксного кода ?

Напоминание

Задача

построения **оптимального префиксного кода** есть задача минимизации функции

$$L = \Sigma_{i=1..n} w_i l_i$$

целочисленных положительных переменных $(l_i)_1^n$ при заданном наборе $(w_i)_1^n$

и при условии (здесь не формализованном) выполнения свойства префиксности кода.

Набор переменных $(l_i)_1^n$, минимизирующий L, определяет структуру дерева (кода).

Итак, …

Есть ли сходство этой задачи с задачей построения оптимального префиксного кода ?

В чём сходство, в чём различие?

Ответ.

Очевидное решение поставленной задачи состоит в переборе всех структурно различных бинарных деревьев с n узлами и выборе дерева с наименьшей *стоимостью* $C_{0,n}$.

Однако поскольку (см. лекции про БДП) число b_n структурно различных бинарных деревьев с n узлами есть

$$b_n = \frac{4^n}{\sqrt{\pi} n^{3/2}}$$

, то этот способ вряд ли будет иметь практическую ценность.

Оказывается, приемлемое по количеству вычислений решение данной задачи может быть получено **методом динамического программирования.**

Решение поставленной задачи методом динамического программирования на следующей лекции.

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ