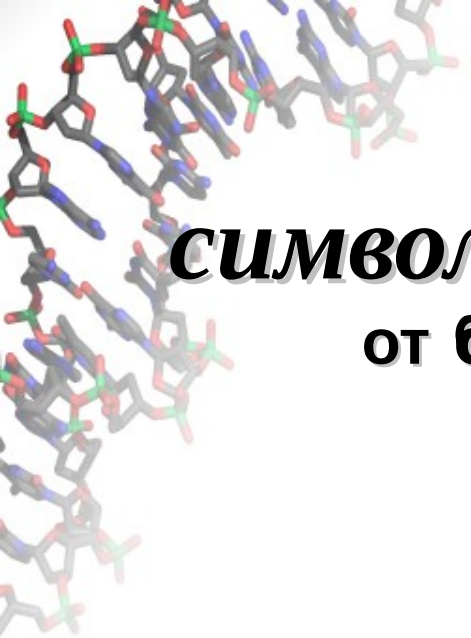


Анализ символьных последовательностей

от биоинформатики до лингвистики

М.А. Ройтберг

Занятие 11

Яндекс

29 апреля 2014

Прыжки на решётке

SPARSE DYNAMIC PROGRAMMING

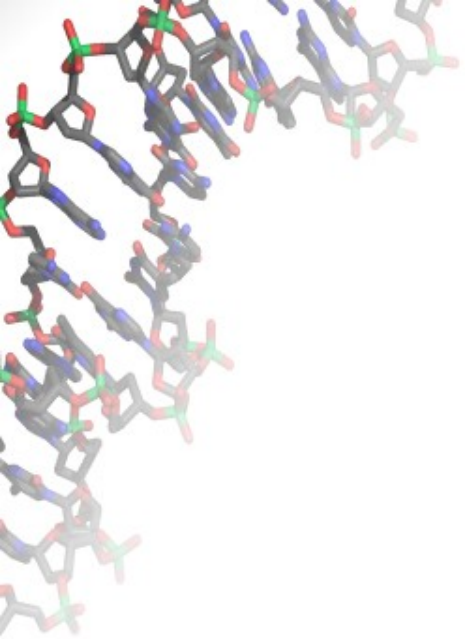
Разреженное динамическое программирование

Метод динамического программирования для разреженных матриц

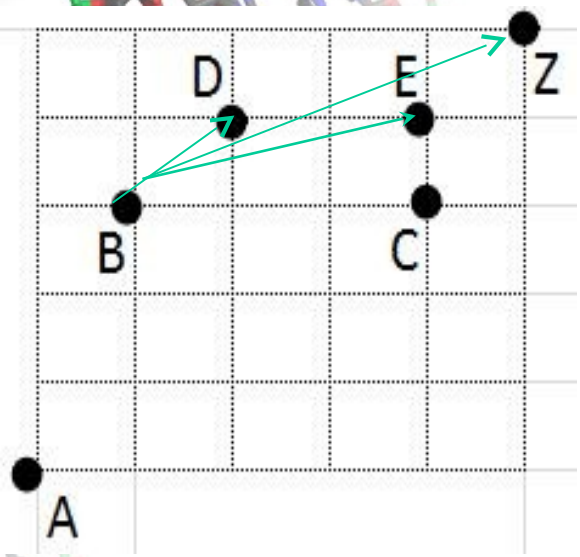
Eppstein, D., Galil Z., Giancarlo R., Italiano G

Sparse dynamic programming I: linear cost functions. //J. ACM. 1992. Vol. 39, P. 513 - 522

Sparse dynamic programming II: convex and concave cost functions //J. ACM. 1992. Vol. 39, P. 546 - 567



Введение



Постановка задачи

Решётка NxN .

Вершины в некоторых узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

$p(R_1, R_2)$ – штраф за переход из R_1 в R_2

Вес пути = Сумма весов – Сумма штрафов

Максимальный путь?

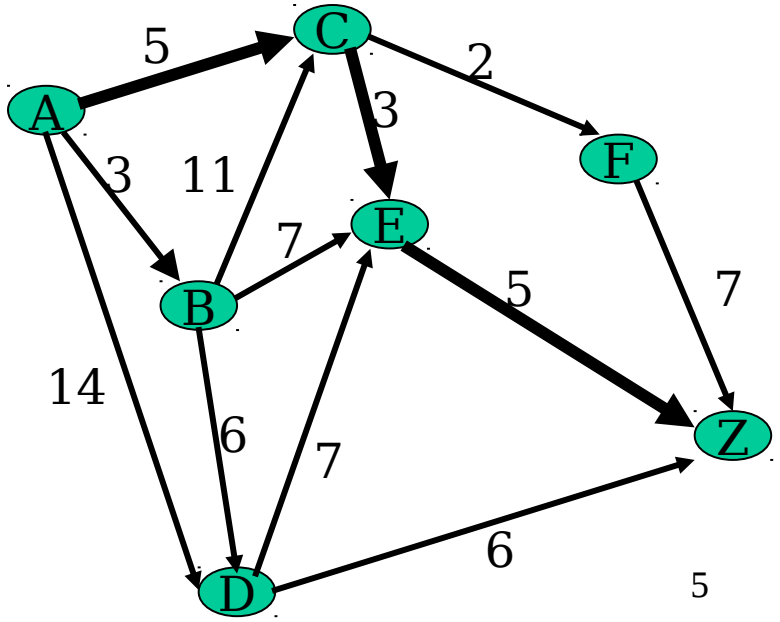
Примеры полуколец.

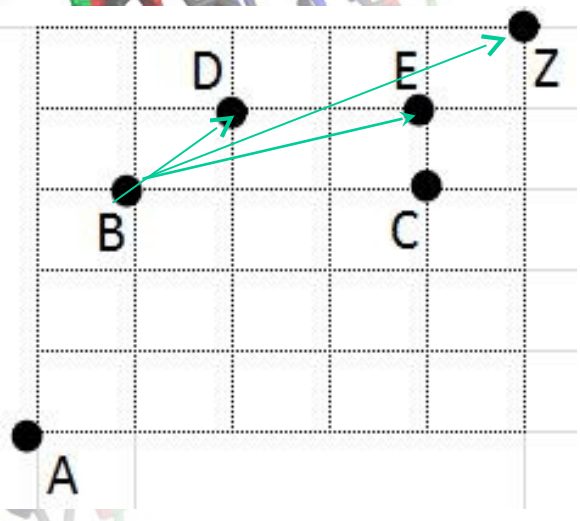
Первая операция – аналог сложения («целевая операция»), вторая – аналог умножения («соединяющая операция»):

- на числах: {+, x}, {min, +}; {max, min};
- на множествах: {∪, ∩}
- на множествах слов: {∪, •}
- на матрицах: {+, x}.

+ - «целевая» операция

*** - «соединительная» операция**





Постановка задачи

Решётка NxN .

Вершины в (некоторых) узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

$p(R_1, R_2)$ – штраф за переход из R_1 в R_2

$p(R_1, R_2) = f(dx, dy)$, где $dx = x_2 - x_1$; $dy = y_2 - y_1$

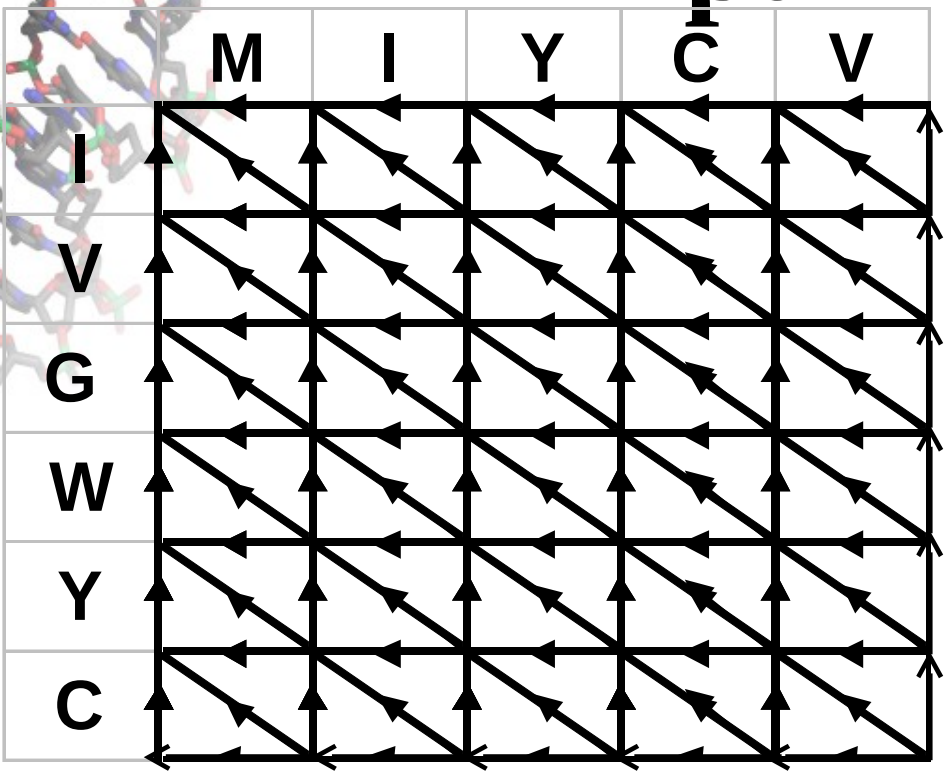
Вес пути = Сумма весов – Сумма штрафов

Вес ребра $(R_1, R_2) = w(R_2) - p(R_1, R_2)$

Вес пути = Сумма весов ребер

Максимальный путь ?

Графы на решетках и выравнивания



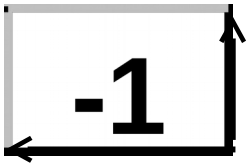
Порядок обхода вершин очевиден.

Вычислять ранги не нужно!

Веса ребер

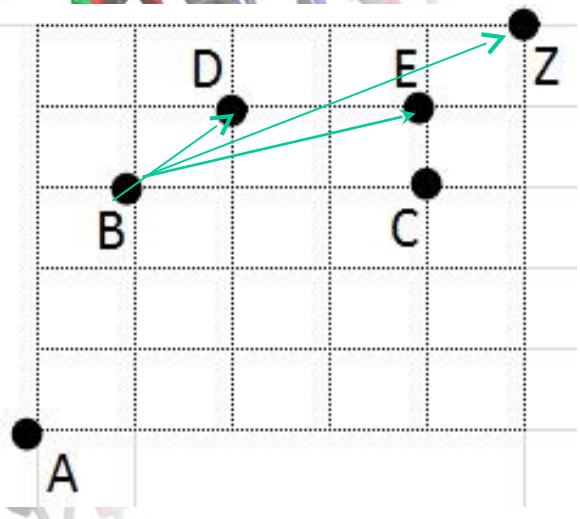
a) диагонали

	M	I	Y	C	V
I	2	4	-1	-1	3
V	2	3	-1	0	3
G	-4	-3	-4	-2	-3
W	-1	-2	-2	-1	-3
Y	0	-1	-1	0	-1
C	-1	-1	-1	12	0



b) вертикали

c) горизонтали



Постановка задачи

Решётка NxN .

Вершины в (некоторых) узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

$p(R_1, R_2)$ – штраф за переход из R_1 в R_2

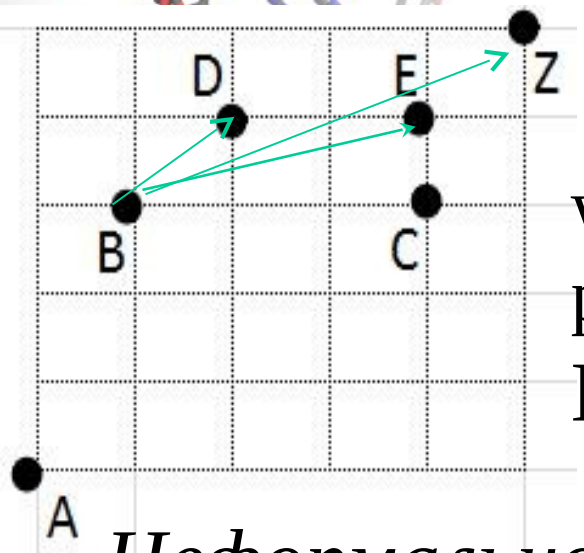
$p(R_1, R_2) = f(dx, dy)$, где $dx = x_2 - x_1$; $dy = y_2 - y_1$

Вес пути = Сумма весов – Сумма штрафов

Вес ребра $(R_1, R_2) = w(R_2) - p(R_1, R_2)$

Вес пути = Сумма весов ребер

Максимальный путь ?



Решётка NxN

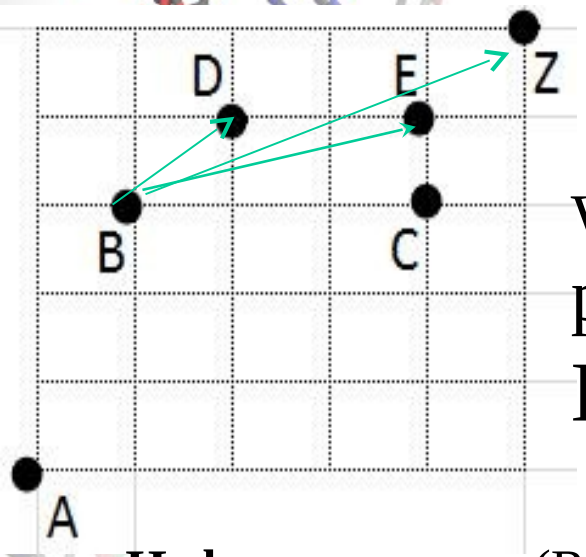
w(R) – вес вершины R

p(R_1, R_2) – штраф за переход из R_1 в R_2

Вес ребра (R_1, R_2) = w(R_2) –p(R_1, R_2)

Неформально: w(R) – вес (неточного) совпадения, соответствующего узлу решетки R

- p(R_1, R_2) – (примерный) вес выравнивания (слабо похожих) фрагментов последовательностей между узлами R_1, R_2.



Решётка NxN

w(R) – вес вершины R

p(R_1, R_2) – штраф за переход из R_1 в R_2

Вес ребра (R_1, R_2) = w(R_2) –p(R_1, R_2)

Неформально: w(R) – вес (неточного) совпадения, соответствующего узлу решетки R

- p(R1, R2) – (примерный) вес выравнивания (слабо похожих) фрагментов последовательностей между узлами R1, R2.

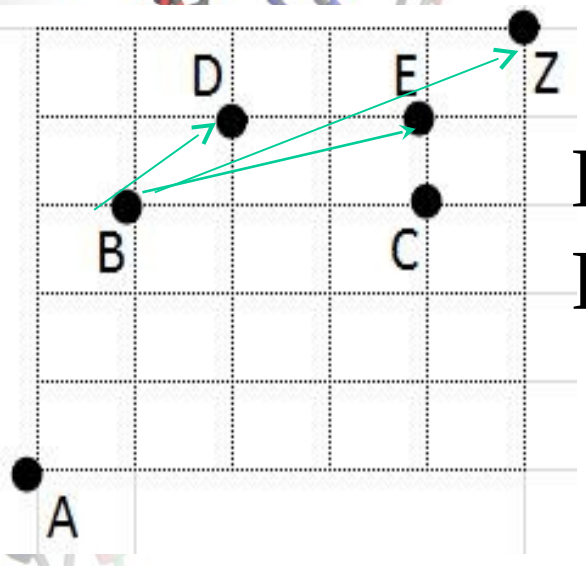
Уточнения

1.Разреженность.

Количество вершин M ~ **N**

Неформально: совпадений мало

*** Количество ребер ~ M^2



Решётка NxN

Вершины в узлах. К-во вершин ~ N

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2) \Leftrightarrow$

$\Leftrightarrow x_1 < x_2$ & $y_1 < y_2$

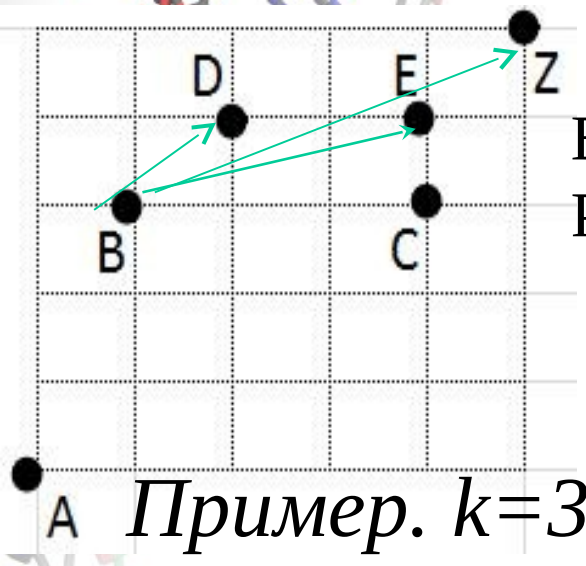
Важная вариация

Вершина – не сопоставление символов, а сопоставление сходных фрагментов длины k (k – параметр). Тогда меняется набор ребер и веса.

$k = 3$

x = AUG CUUAGCCUUA

y = AUGGCUUAGAUUUA



Решётка NxN

Вершины в узлах. К-во вершин ~ N

Ребро из R1(x1, y1) в R2(x2, y2) $\Leftrightarrow$

$\Leftrightarrow$ x1< x2 & y1< y2 (*)

Важная вариация

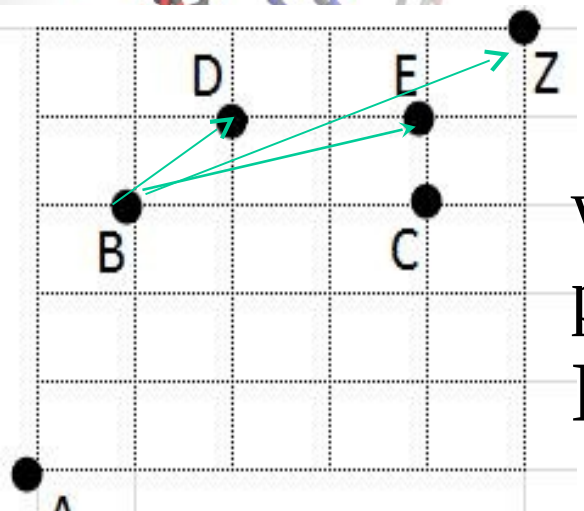
Пример. k=3

x = AUG CUUAGCCUUA

y = AUGGCUUAGAUUUA

1. Пятерка CUUAG это – 3 тройки. Условие (*) нарушается, но ребро из 1-й тройки в последнюю проводим.
2. Можно считать *k* разным в разных точках.
3. Наложение отрезков на разных диагоналях (на доске).

ДАЛЕЕ НЕ РАССМАТРИВАЕМ



Решётка NxN

w(R) – вес вершины R

p(R_1, R_2) – штраф за переход из R_1 в R_2

Вес ребра (R_1, R_2) = w(R_2) –p(R_1, R_2)

Неформально: w(R) – вес (неточного) совпадения, соответствующего узлу решетки R

- p(R1, R2) – (примерный) вес выравнивания (слабо похожих) фрагментов последовательностей между узлами R1, R2.

Уточнения

1.Разреженность.

Количество вершин M ~ **N**

2. *** Количество ребер ~ $\mathbf{M^2}$

!!! Тем не менее, хотим T ~ **M** **!!!**

Решётка NxN ; w(R) – вес вершины R
p(R1, R2) – штраф за переход из R1 в R2
Вес ребра (R1, R2) = w(R2) –p(R1, R2)

Уточнения

1. Количество вершин **M ~ N.** Количество ребер **~ M²**
2. **!!!** Тем не менее, хотим T **~ M !!!**

Идея (вспомогательная):

1) при оценке веса «дырки» p(R1, R2):

- НЕ УЧИТЫВАЕМ СИМВОЛЫ «непохожих» ФРАГИЕНТОВ;
- УЧИТЫВАЕМ только ДЛИНЫ ФРАГМЕНТОВ

Решётка NxN ; w(R) – вес вершины R
p(R1, R2) – штраф за переход из R1 в R2
Вес ребра (R1, R2) = w(R2) –p(R1, R2)

Уточнения

1. Количество вершин **M ~ N.** Количество ребер **~ M²**
2. **!!!** Тем не менее, хотим T **~ M !!!**

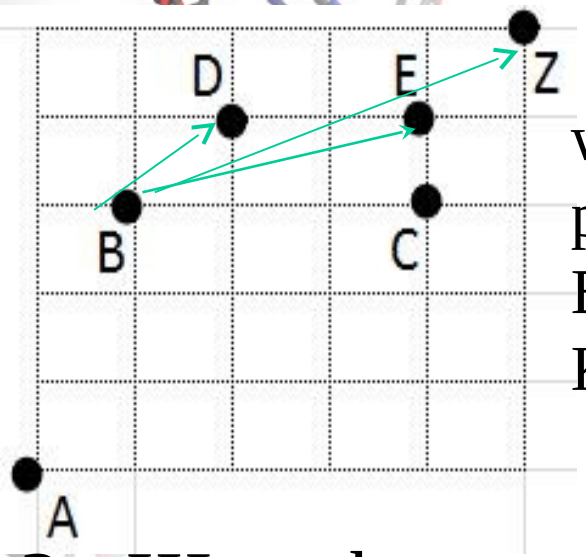
Идея (вспомогательная):

1) при оценке веса «дырки» p(R1, R2):

- НЕ УЧИТЫВАЕМ СИМВОЛЫ «непохожих» ФРАГИЕНТОВ;
- УЧИТЫВАЕМ только ДЛИНЫ ФРАГМЕНТОВ

2) Штрафы за переходы имитируют вес выравнивания

Что значит «имитируем вес выравнивания»?



Решётка NxN

w(R) – вес вершины R

p(R1, R2) – штраф за переход из R1 в R2

Вес ребра (R1, R2) = w(R2) –p(R1, R2)

Количество вершин ~ **N**

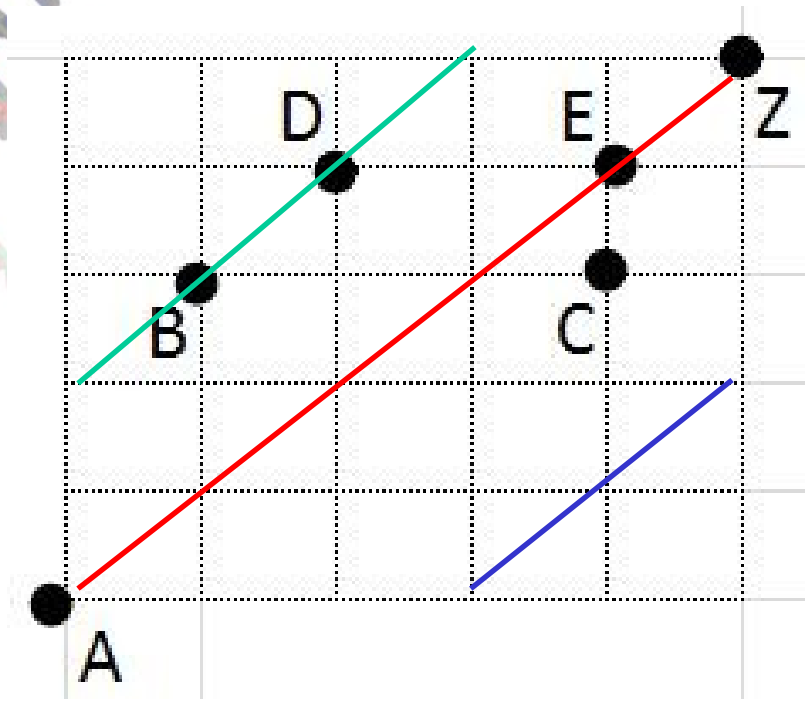
Уточнения

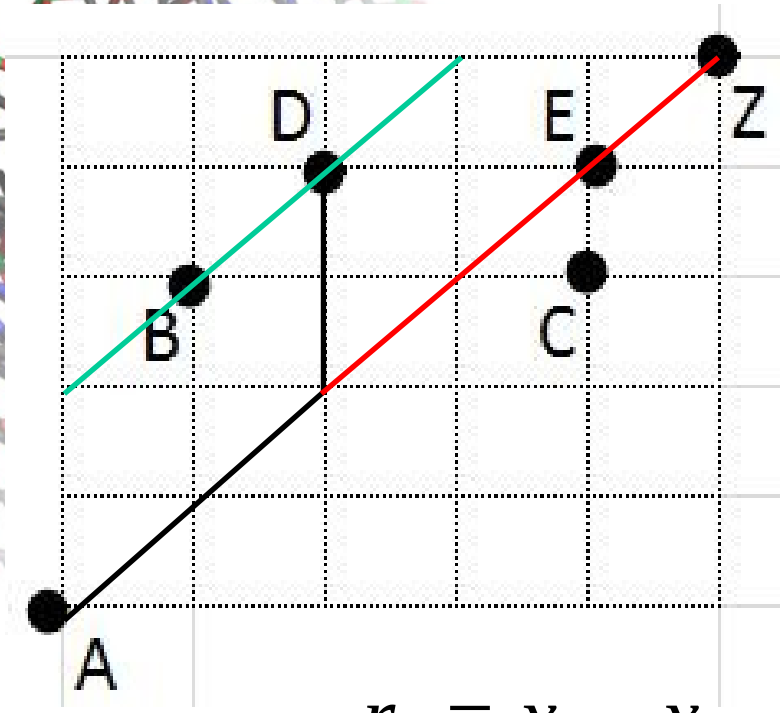
2. Штрафы за переходы имитируют вес выравнивания

Что значит «имитируем вес выравнивания»?

Диагонали:

y-x = 2 ***y-x=0*** ***y-x = -3***





Диагонали:

$y-x = 2$ $y-x=0$

$$d = y-x$$

$$x_A = 0 \quad y_A = 0 \quad d_A = 0$$

$$x_D = 2 \quad y_D = 4 \quad d_D = 2$$

$$\Delta d = |d_A - d_D| = 2$$

$$r_x = x_2 - x_1 - 1 \qquad r_y = y_2 - y_1 - 1$$

$$M = min(r_x, r_y)$$

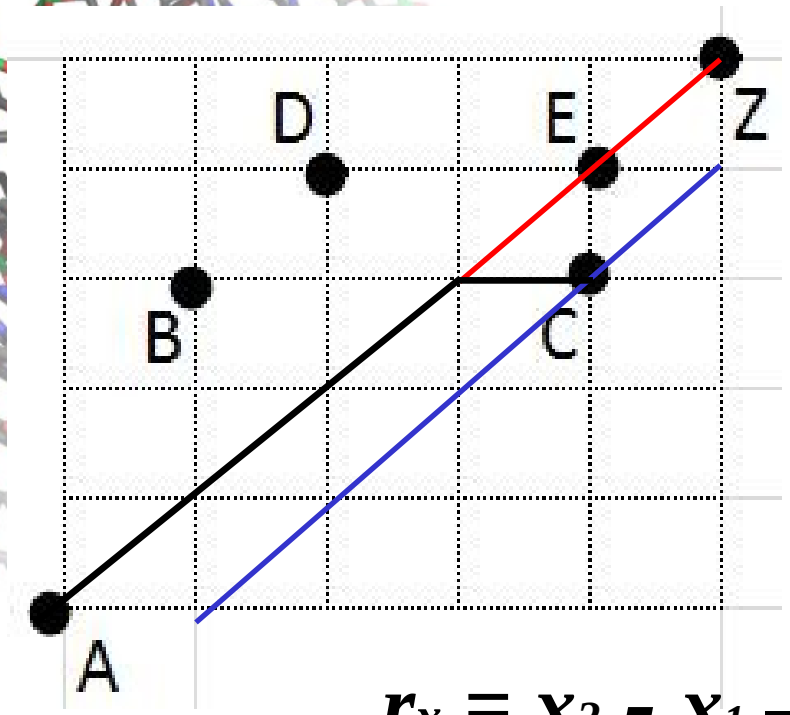
$$r_x(A, B) = 2 - 0 - 1 = 1 \qquad r_y(A, B) = 4 - 0 - 1 = 3$$

$$M = min(1, 3) = 1$$

M – количество «плохих» совпадений

Δd - длина удаленного фрагмента

```
Ax--D
AzzzD
```



Диагонали:

$y-x=0$ $y-x=-1$

$$d = y-x$$

$$x_A = 0 \quad y_A = 0 \quad d_A = 0$$

$$x_C = 4 \quad y_C = 3 \quad d_C = -1$$

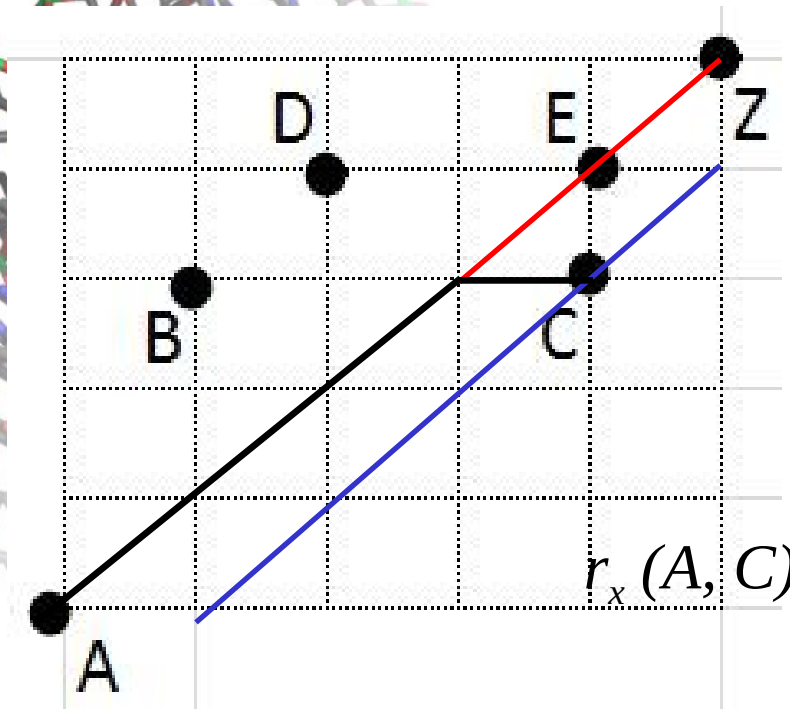
$$\Delta d = |d_A - d_B| = 1$$

$$r_x = x_2 - x_1 - 1 \qquad r_y = y_2 - y_1 - 1$$

$$M = \min(r_x, r_y)$$

$$r_x(A, C) = 4 - 0 - 1 = 3 \qquad r_y(A, C) = 3 - 0 - 1 = 2$$

$$M = \min(3, 2) = 2$$



$y-x=0$ $y-x=-1$

$d = y-x$

$x_A = 0 \quad y_A = 0 \quad d_A = 0$

$x_C = 4 \quad y_C = 3 \quad d_C = -1$

$\Delta d = |d_A - d_B| = 1$

$r_x = x_2 - x_1 - 1 \qquad r_y = y_2 - y_1 - 1$

$M = \min(r_x, r_y)$

$r_x(A, C) = 4 - 0 - 1 = 3 \qquad r_y(A, C) = 3 - 0 - 1 = 2$

$M = \min(3, 2) = 2$

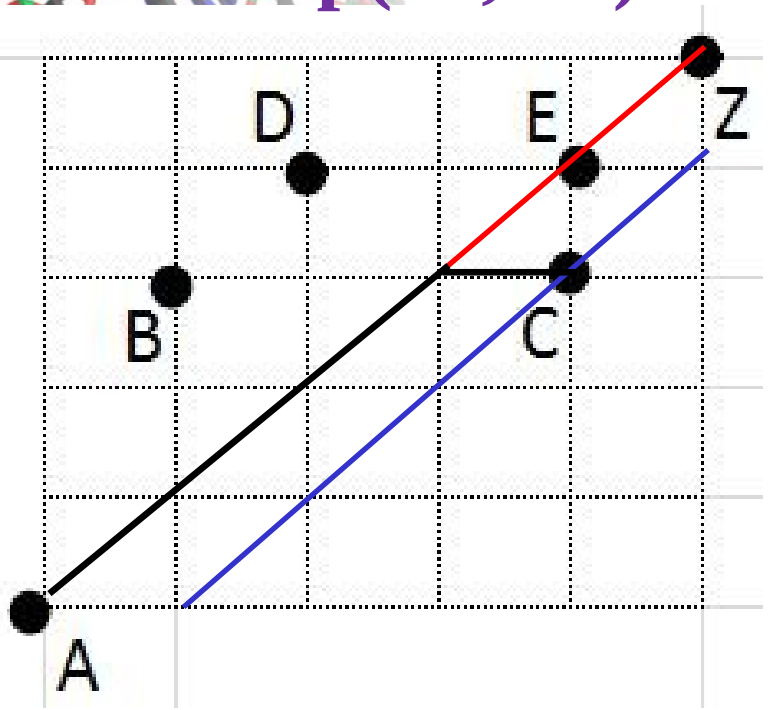
$p(R_1, R_2)$ – штраф за переход из R_1 в R_2

$b*M = b*\min(x_2-x_1-1, y_2-y_1-1)$ – штраф за несовпадение

$g(\Delta d) = g(|(y_1-x_1)-(y_2-x_2)|)$ – штраф за удаление

$$p(R_1, R_2) = b*M(R_1, R_2) + g(\Delta d\,(R_1, R_2))$$

$p(R_1, R_2)$ – штраф за переход из R_1 в R_2



$R_1 = (x_1, y_1)$; $R_2 = (x_2, y_2)$;

$d_1 = y_1 - x_1$; $d_2 = y_2 - x_2$;

$\Delta d\,(R_1, R_2) = |d_1 - d_2|$

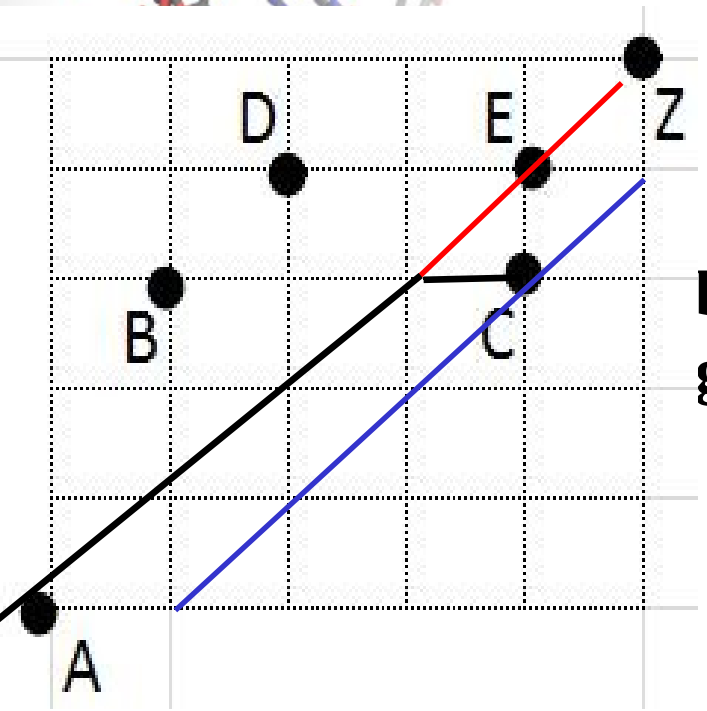
$$p(R_1, R_2) = b * M(R_1, R_2) + g(\Delta d\,(R_1, R_2))$$

b – средний вес сопоставления символов

Варианты штрафов за удаление:

1) $g(\Delta d) = a*(\Delta d) + c$ - *для простоты:* $g(\Delta d) = \Delta d$

2) f(dx, dy) = **log(dx+dy) – выпуклая функция**



Замечание 1.

(R1, R2) – штраф за переход из R1 в R2

b*M = b*min(x2-x1-1, y2-y1-1)

g(Δd) = g(|(y1-x1)-(y2-x2)|)

$p(R_1, R_2) = b*M(R_1, R_2) + g(\Delta d\,(R_1, R_2))$

(y1-x1) - (y2-x2) =

= (x2-x1-1) – (y2- y1-1)

p(R1, R2) =

= b*(x2-x1-1) + g((y2-x2) - (y1-x1)),

если (y2-x2) ≥ (y1-x1);

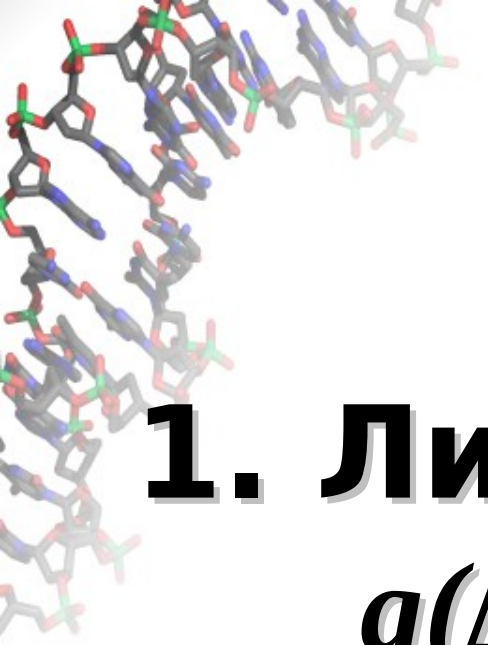
= b*(y2-y1-1) + g((y1-x1) - (y2-x2)),

если (y1-x1) > (y2-x2);

Замечание 2

E[i, j] – вес оптимального пути из A в (i, j)

$$E[i, j] = w(P) + \max\{ E[i', j'] - p((i', j'), (i, j)) \mid (i', j') < (i, j) \}$$



1. Линейные штрафы

$$g(\Delta d) = \Delta d = a*|d_1 - d_2|$$

(по М.Колупаеву)

Замечания 1, 2
E[i, j] – вес оптимального пути из A в P = (i, j)

$$p\,(i', j'), (i.\ j) =$$
$$= b*(i - i' - 1) + a*((j - i) - (j' - i)) + c,$$
если $(j - i) \geq (j' - i')$;
$$= b*(j - j' - 1) + a*(j' - i') - (j - i)) + c,$$
если $(j - i) < (j' - i')$;

$$E[i, j] = w(i, j) +$$

$$+ \max\{E[i', j'] - p((i', j'), (i.\ j)) \mid (i', j') < (i, j)\}$$

Замечания 1, 2
E[i, j] – вес оптимального пути из A в P = (i, j)

p (i’, j'), (i. j) =

= b*(i - i‘ - 1) + a*((j - i) - (j’- i’))+c если (j - i) ≥ (j’ – i’);

= b*(j – j’ - 1) + a*(j’ – i’) - (j - i))+c если (j - i) < (j’ – i’);

p (i’, j'), (i. j)

= **-b*(i’+1) –a*(j’ - i‘) + b*i+ a*(j - i) + c,**
если (j - i) ≥ (j’ – i’);

= **-b*(j’ + 1) + a*(j’ – i’) + b*j - a*(j - i) + c,**
если (j - i) < (j’ – i’);

p (i’, j'), (i. j) = p1(i’, j’) + p2(i, j)

E[i, j] = (w(i, j) - **p2(i, j)) +**

+ max{E[i’, j’] – **p1((i’, j') |** (i’, j’) < (i, j) }

Замечания 1, 2
E[i, j] – вес оптимального пути из A в P = (i, j)

p (i’, j'), (i. j) =

= b*(i - i‘ - 1) + a*((j - i) - (j’- i’)) если (j - i) ≥ (j’ – i’);

= b*(j – j’ - 1) + a*(j’ – i’) - (j - i)) если (j - i) < (j’ – i’);

p (i’, j'), (i. j)

= **-b*(i’+1) –a*(j’ - i‘) + b*i+ a*(j - i) +c**, если (j - i) ≥ (j’ – i’);

= **-b*(j’ + 1) + a*(j’ – i’) + b*j - a*(j - i)+c** , если (j - i) < (j’ – i’);

$$p(i', j'), (i. j) = p1(i', j') + p2(i, j)$$

$$w^*(i, j) = w(i, j) - p2(i, j); \qquad E^*[i, j] = E[i', j'] - p1((i', j')$$

$$E[i, j] = (w^*(i, j) + \max\{E^*[i', j'] \mid (i', j') < (i, j)\}$$

Идея алгоритма

- 1. Перебираем вершины по строкам; в строке – слева направо.
- 2.Для очередной вершины V(x, y):

- находим E_{Left} = max {E[i, j] + (**-b*(i'+1) –a*(j' - i')**)}

(i, j) – вершина; i < x; j < y; j – i ≥ (y-x)

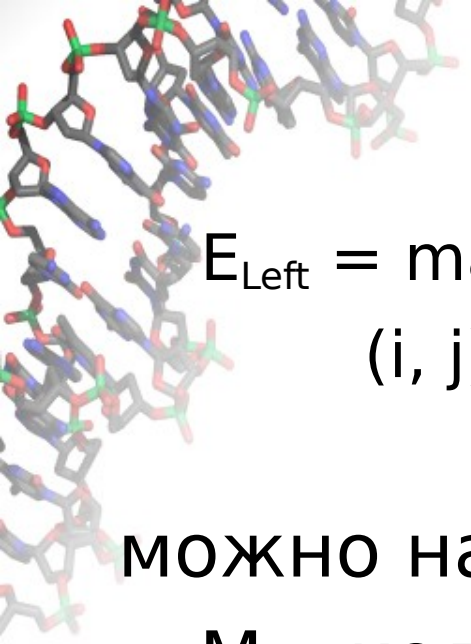
- находим E_{Right} = max {E[i, j] + (**-b*(i'+1) +a*(j' - i')**)}

(i, j) – вершина; i < x; j < y; j – i < (y-x)

- находим E[x, y] = max**{**E_{left} + **b*i+ a*(j - i) +c**;

E_{Right} + **b*i+ a*(j - i) +c}**

- устанавливаем ссылку на соответствующую максимуму вершину (i, j) < (x, y)



Еще одна идея:

$E_{Left} = \max \{E[i, j] + (\mathbf{-b*(i'+1) -a*(j' - i')})\}$

(i, j) – вершина; $i < x$; $j < y$; $j - i \geq (y-x)$

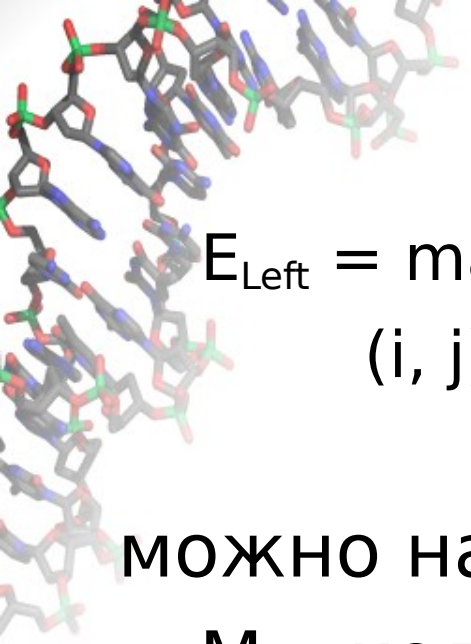
можно найти за время ~ log(M),

M – количество вершин, которые нужно просмотреть.

$E_{Right} = \max \{E[i, j] + (\mathbf{-b*(i'+1) +a*(j' - i')})\}$

(i, j) – вершина; $i < x$; $j < y$; $j - i < (y-x)$ –

- аналогично



Еще одна идея:

$E_{Left} = \max \{E[i, j] + (\mathbf{-b*(i'+1) -a*(j' - i')})\}$

(i, j) – вершина; $i < x$; $j < y$; $j - i \geq (y-x)$

можно найти за время ~ log(M),

M – количество вершин, которые нужно просмотреть.

Используем сбалансированные деревья !

Повторение: АВЛ-деревья (по М.Н.Курносову, http://www.mkurnosov.net и http://habrahabr.ru/post/150732/)

Двоичные деревья поиска

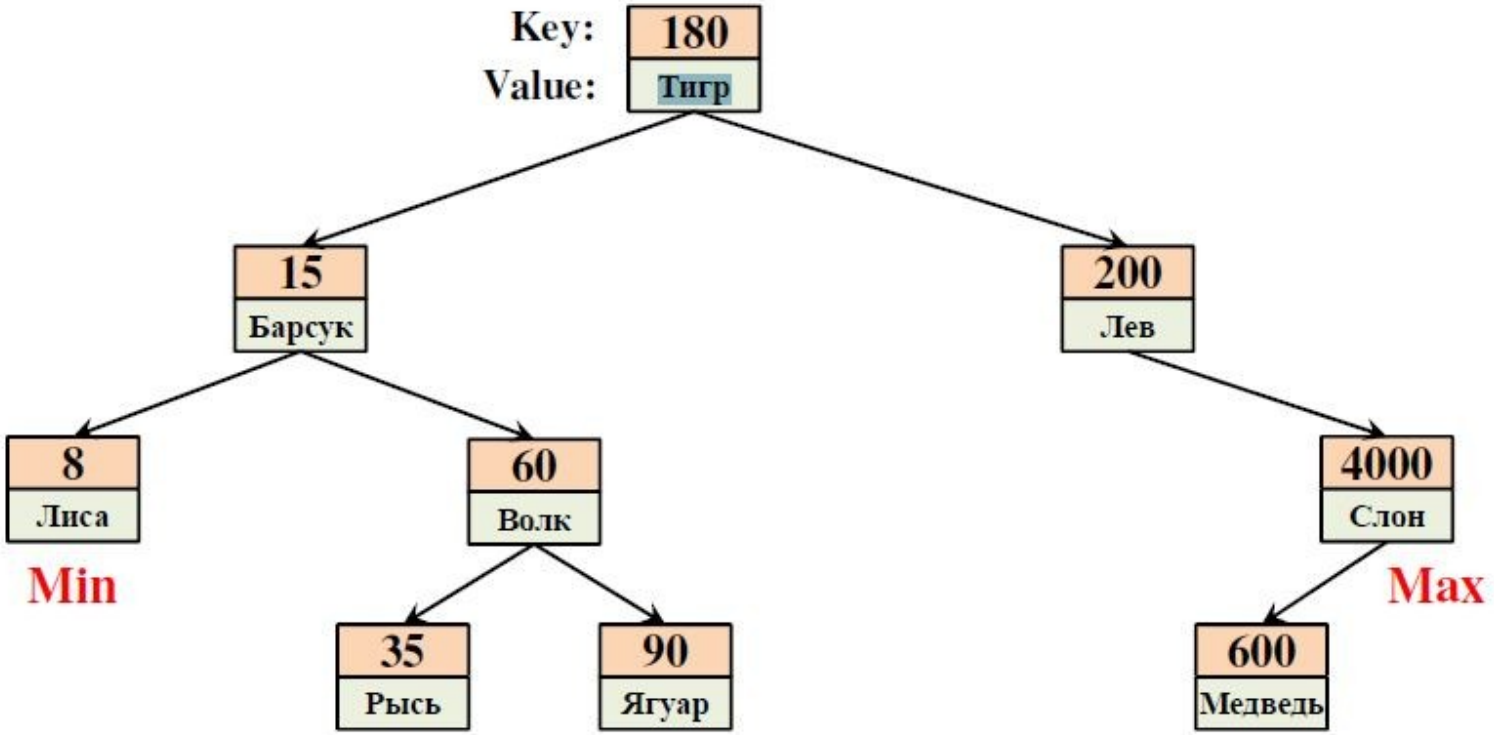
Двоичное дерево поиска (Binary Search Tree, BST) – это двоичное дерево, в котором:

1) каждый узел (node) имеет не более двух дочерних узлов (child nodes)

2) каждый узел содержит ключ (key) и значение (value) и для него выполняются следующие условия:

- ключи всех узлов левого поддерева меньше значения ключа родительского узла
- ключи всех узлов правого поддерева больше значения ключа родительского узла

Двоичные деревья поиска (Binary Search Trees)



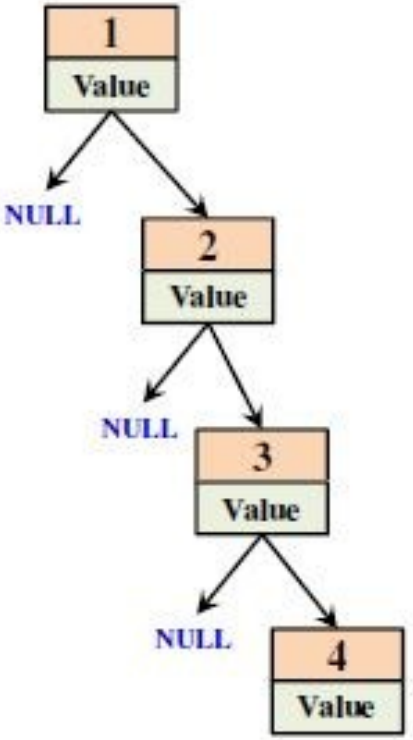
9 узлов, глубина (depth) = 3

Двоичные деревья поиска (Binary Search Trees)

Операция	Средний случай (average case)	Худший случай (worst case)
Add(*key, value*)	$O(\log n)$	$O(n)$
Lookup(*key*)	$O(\log n)$	$O(n)$
Remove(*key*)	$O(\log n)$	$O(n)$
Min	$O(\log n)$	$O(n)$
Max	$O(\log n)$	$O(n)$

Двоичные деревья поиска (Binary Search Trees)

1. Операции имеют трудоемкость пропорциональную высоте дерева
2. В среднем случае высота дерева $O(\log(n))$
3. В худшем случае элементы добавляются по возрастанию (убыванию) ключей – дерево вырождается в список длины n

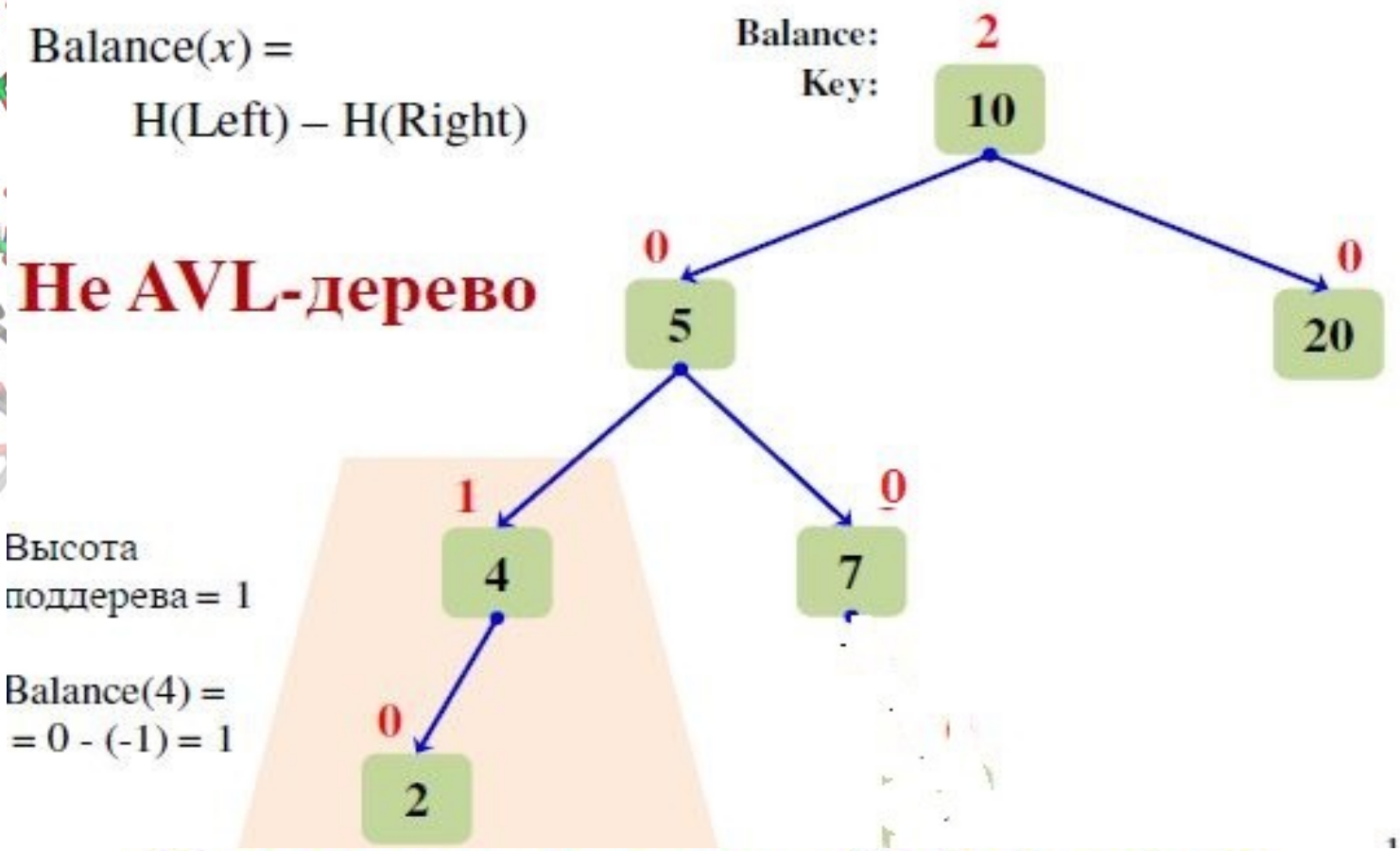



```
bstree_add(1, value)
bstree_add(2, value)
bstree_add(3, value)
bstree_add(4, value)
```

Коэффициент сбалансированности

Balance(x) =

H(Left) – H(Right)



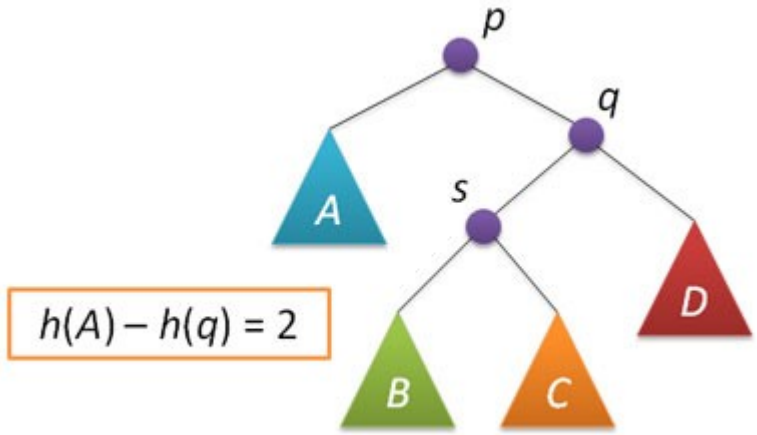
Не AVL-дерево

Высота поддерева = 1

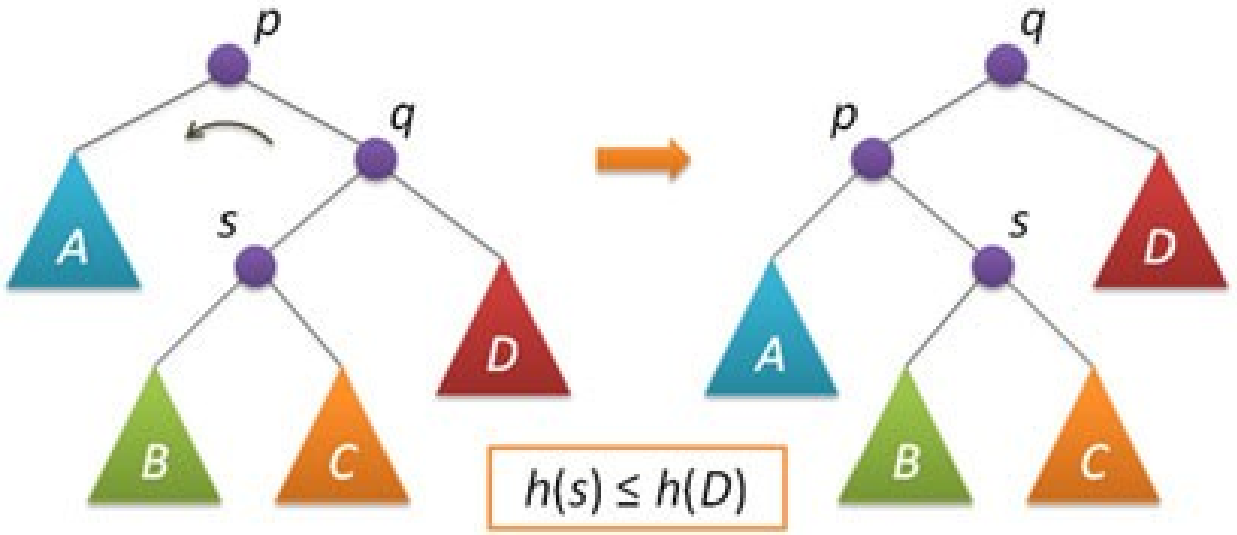
Balance(4) =
= 0 - (-1) = 1

Балансировка дерева (Rebalancing)

- **После добавления** нового элемента необходимо обновить коэффициенты сбалансированности родительских узлов
- Если любой родительский узел принял значение -2 или 2, то необходимо выполнить балансировку поддерева путем **поворота (rotation)**



$h(A) - h(q) = 2$

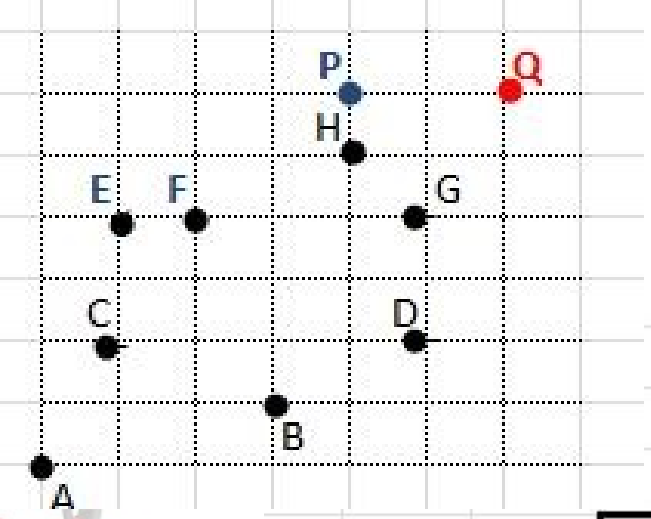


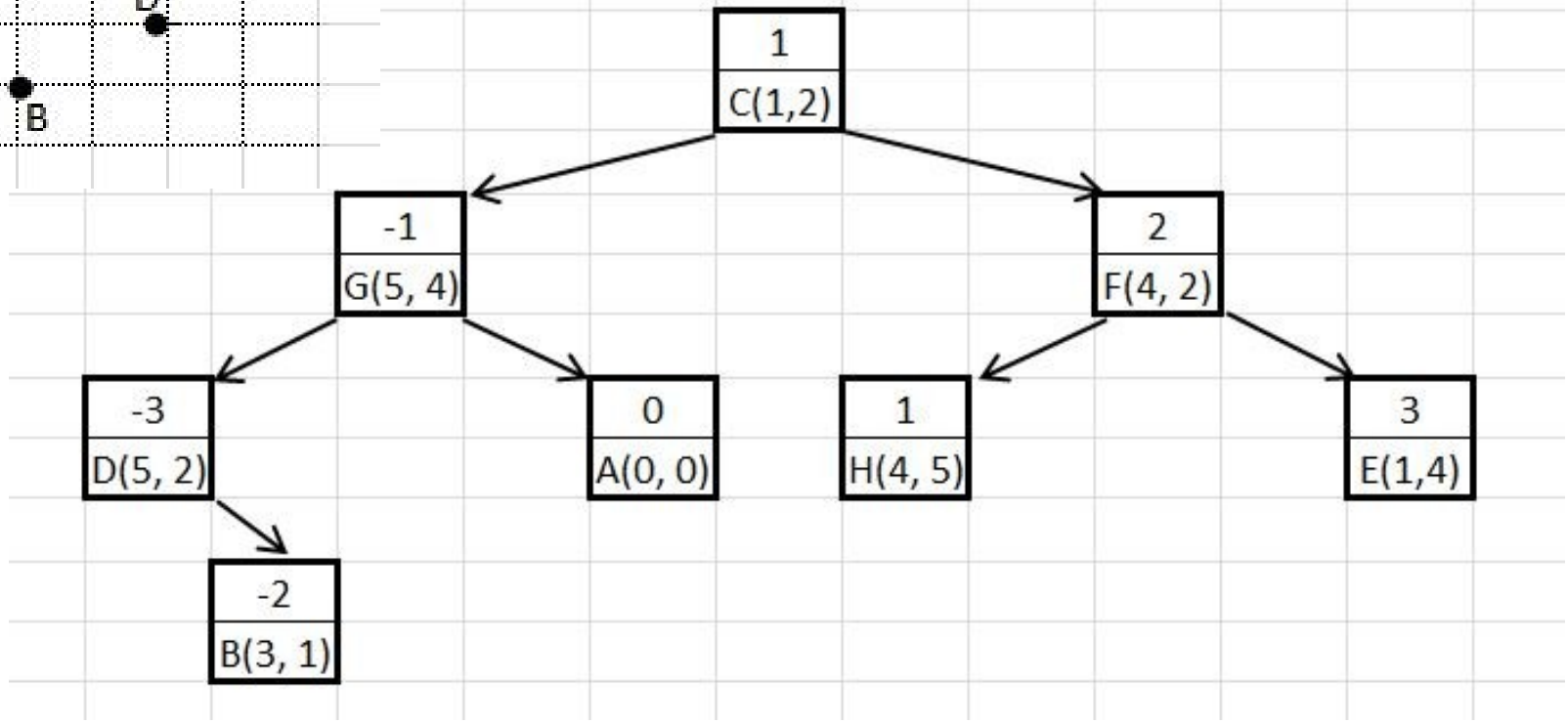
$h(s) \leq h(D)$

AVL tree

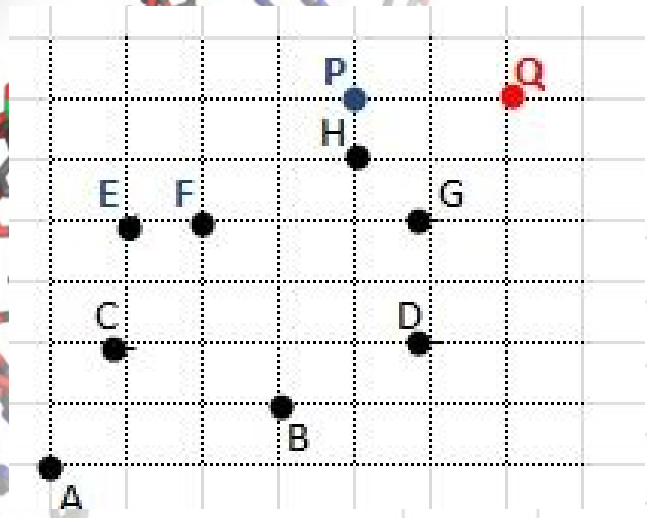
Операция	Средний случай (average case)	Худший случай (worst case)
Add(*key, value*)	$O(\log n)$	$O(\log n)$
Lookup(*key*)	$O(\log n)$	$O(\log n)$
Remove(*key*)	$O(\log n)$	$O(\log n)$
Min	$O(\log n)$	$O(\log n)$
Max	$O(\log n)$	$O(\log n)$

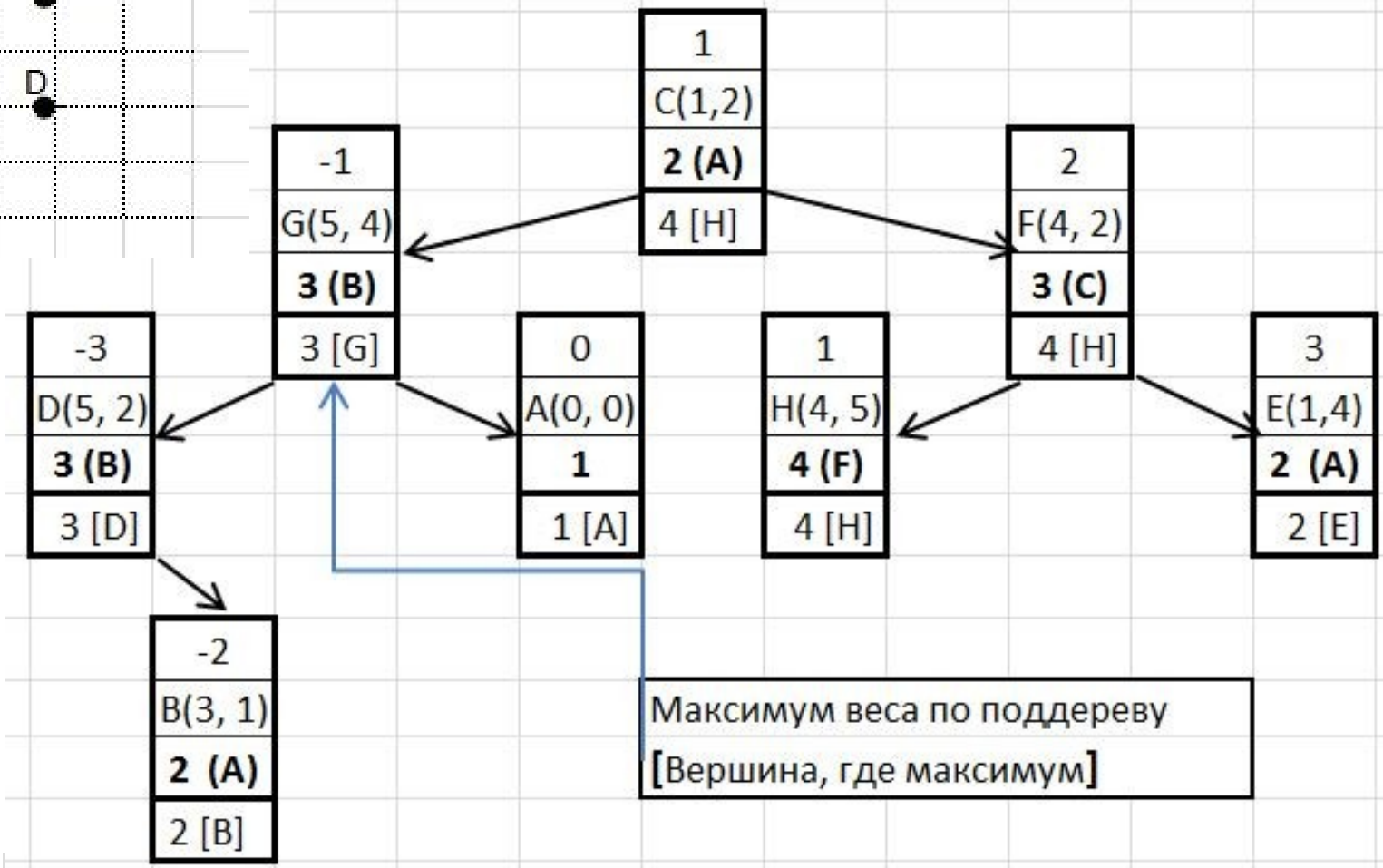
Сложность по памяти: $O(n)$



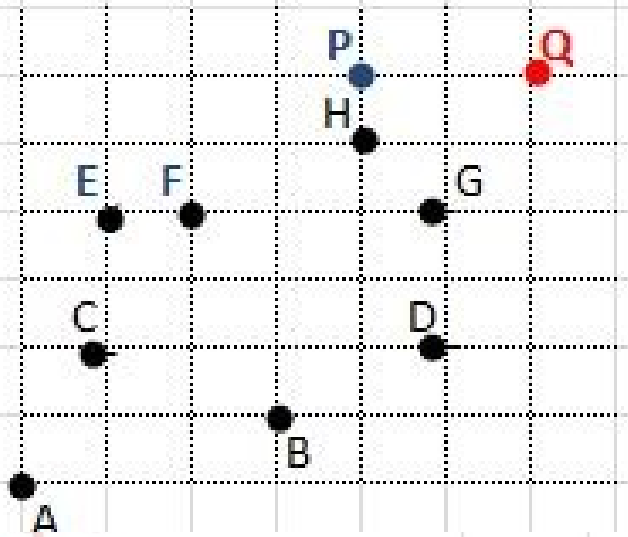


	A	B	C	D	E	F	G	H	Z
Коорд.	(0, 0)	(3, 1)	(1, 2)	(5, 2)	(1, 4)	(2, 4)	(5, 4)	(4, 5)	(6, 6)
Вес пути	1	2	2	3	2	3	3	4	





	A	B	C	D	E	F	G	H	Z
Коорд.	(0, 0)	(3, 1)	(1, 2)	(5, 2)	(1, 4)	(2, 4)	(5, 4)	(4, 5)	(6, 6)
Вес пути	**1**	**2**	**2**	**3**	**2**	**3**	**3**	**4**	



max{Score(X) | key(X) ≥ key(R)}=

: = max{Score(R),

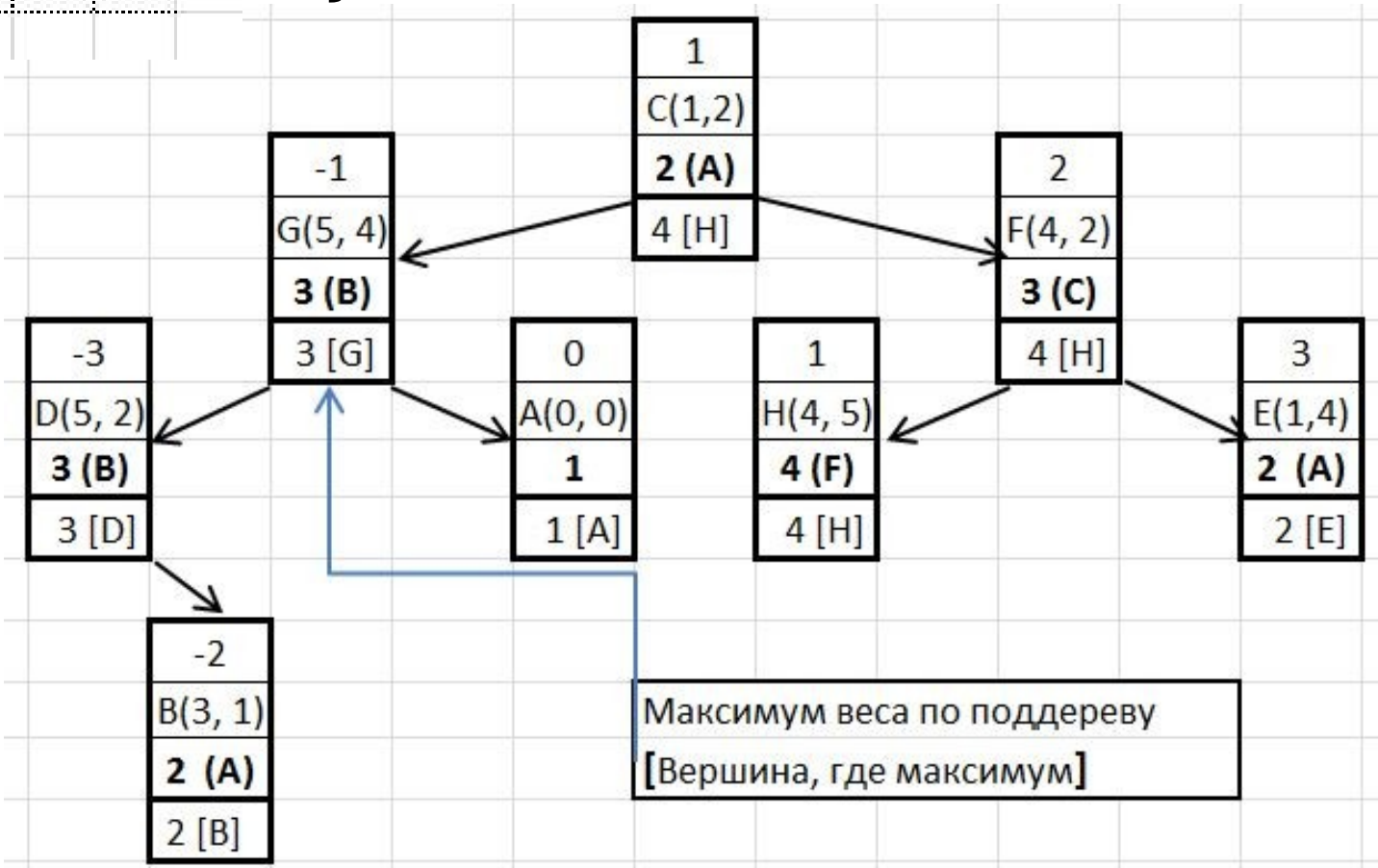
MaxSubTr(RightSon(R),

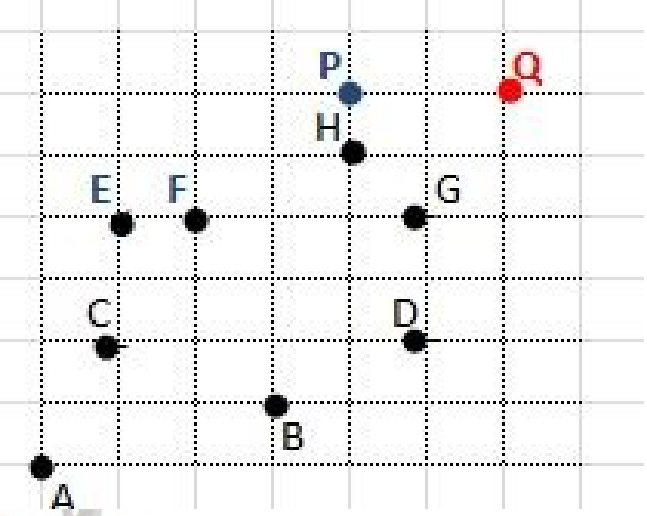
Score(“LeftParent”(R),

MaxSubTr(RSon“LeftParent”(R),

… вверх по дереву,«глядя вправо»

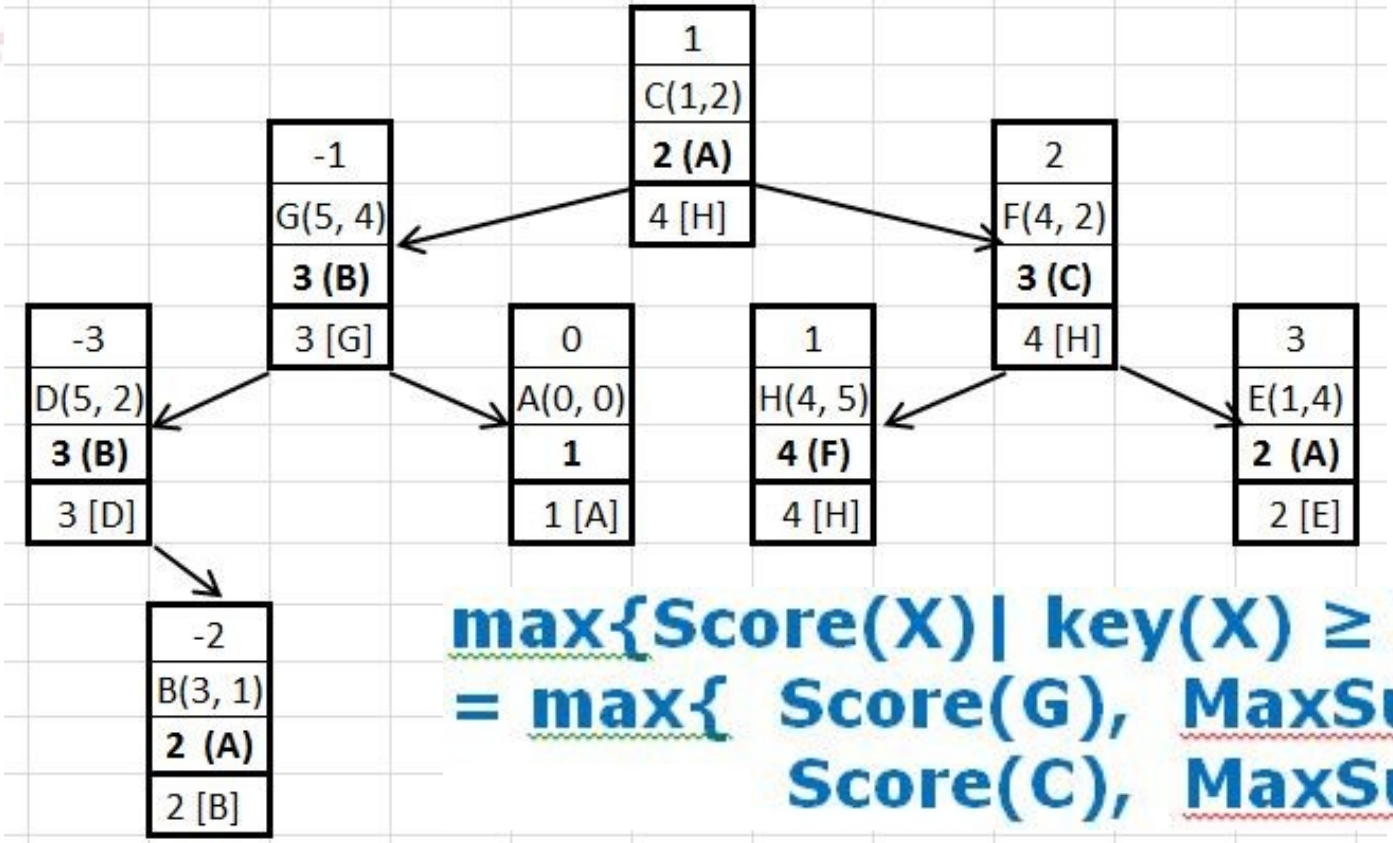
}



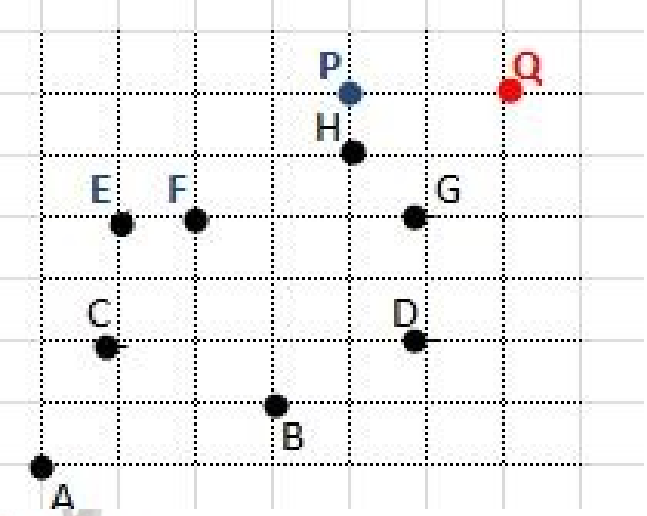


max{Score(X) | key(X) ≥ key(R)} = =
max{Score(R),
MaxSubTr(RightSon(R),
Score("LeftParent"(R),
MaxSubTr(RSon"LeftParent"(R),
и вверх по дереву, «глядя вправо»

}

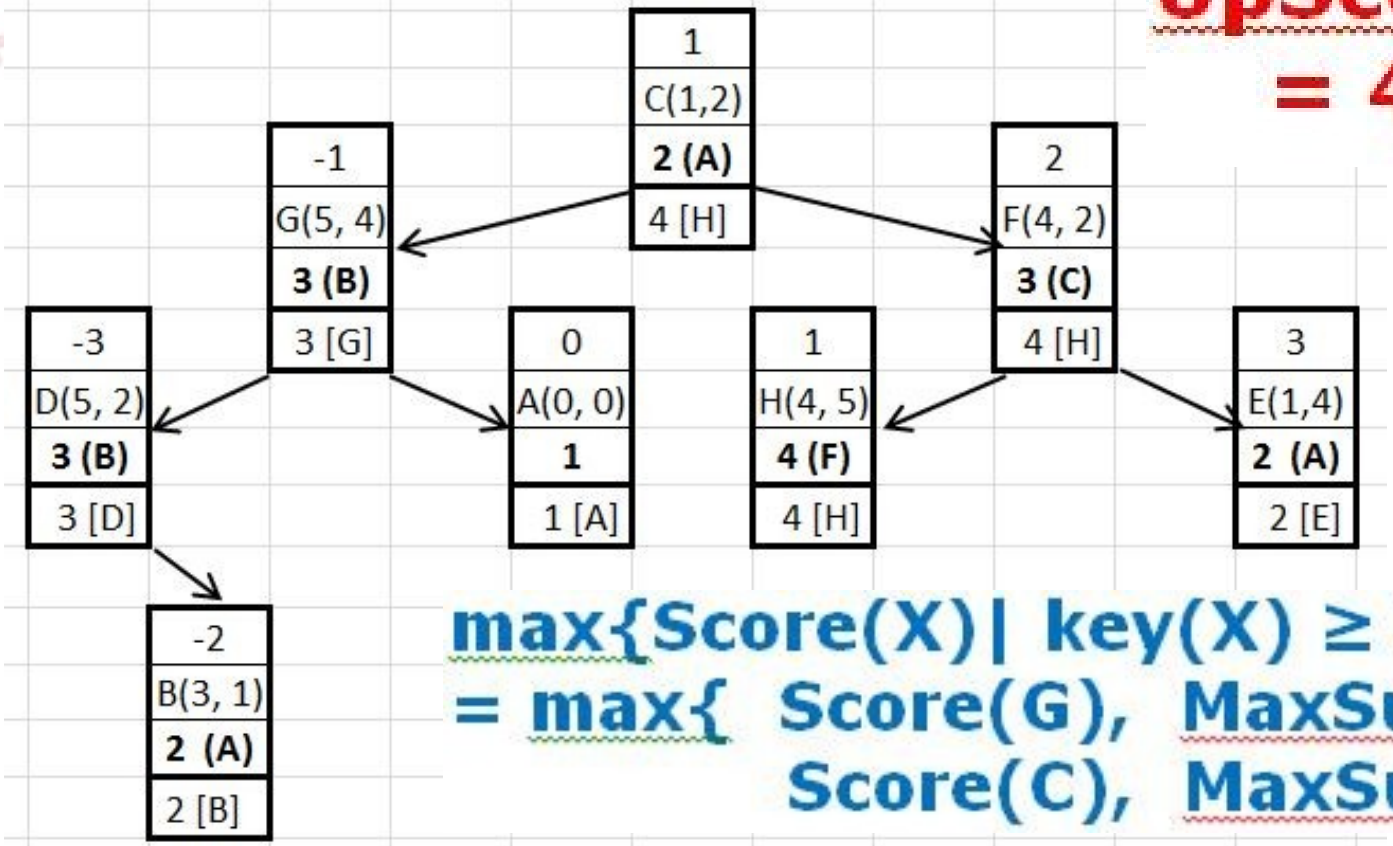


max{Score(X)| key(X) ≥ key(G}=
= max{ Score(G), MaxSubTr(A),
Score(C), MaxSubTr(F) }



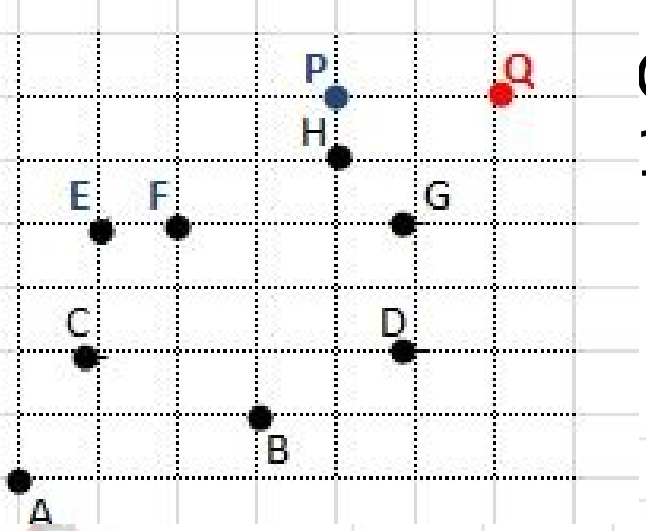
max{Score(X) | key(X) ≥ key(R)} = =
max{Score(R),
MaxSubTr(RightSon(R),
Score("LeftParent"(R),
MaxSubTr(RSon"LeftParent"(R),
и вверх по дереву, «глядя вправо»

}

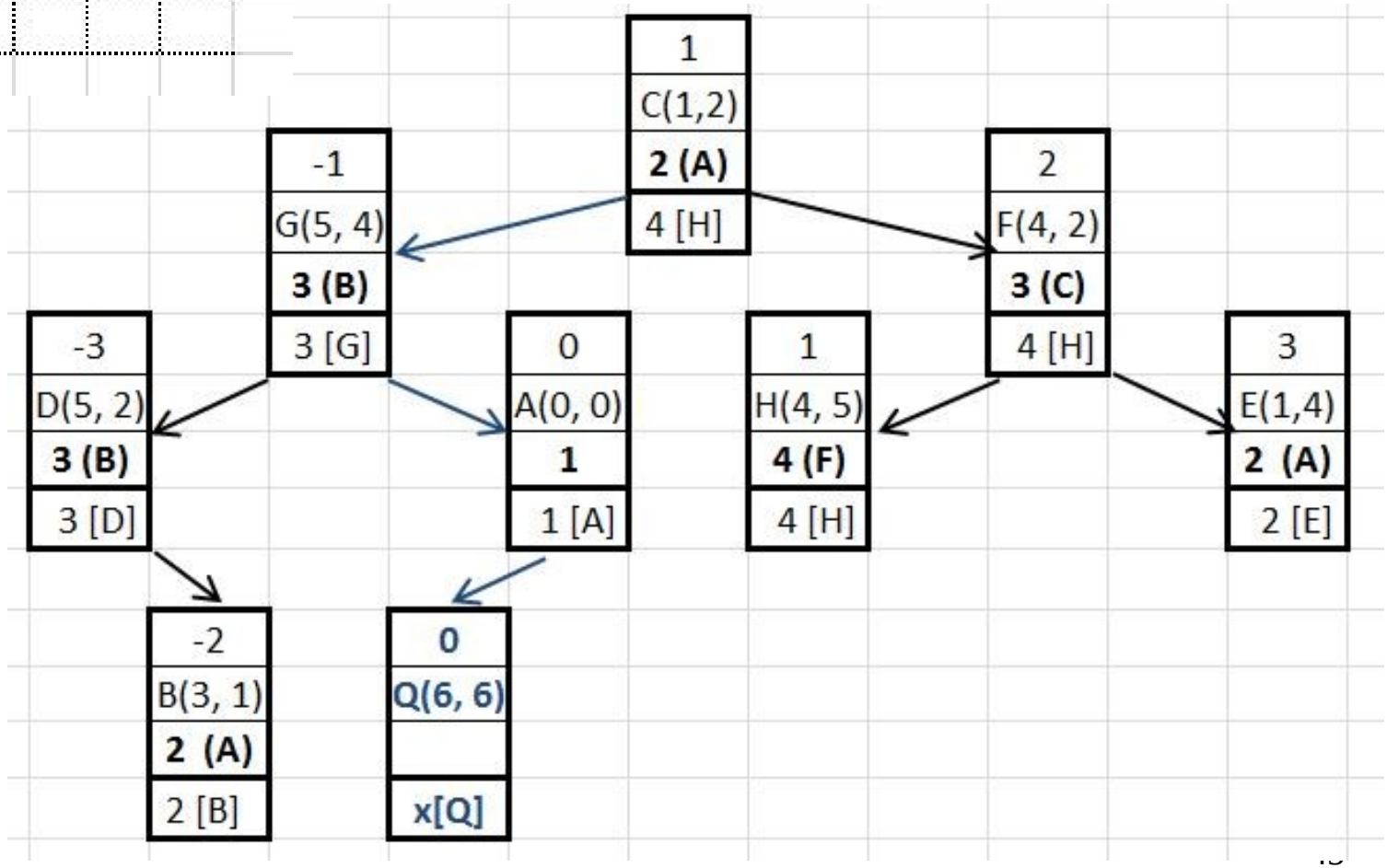


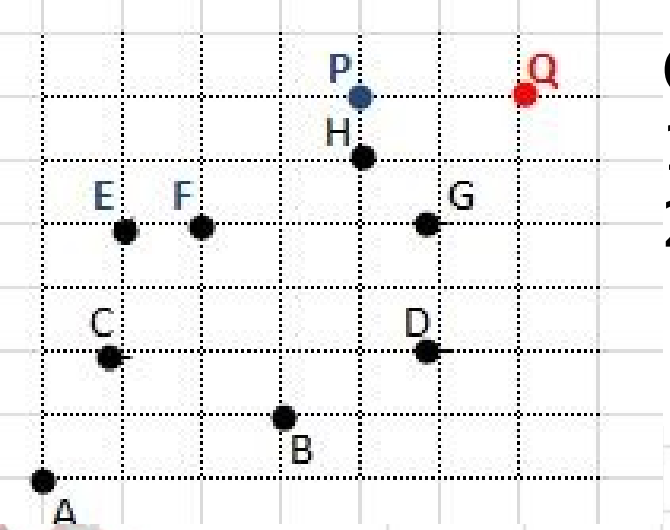
UpScore(G) =
= 4 [H]

max{Score(X)| key(X) ≥ key(G}=
= max{ Score(G), MaxSubTr(A),
Score(C), MaxSubTr(F) }



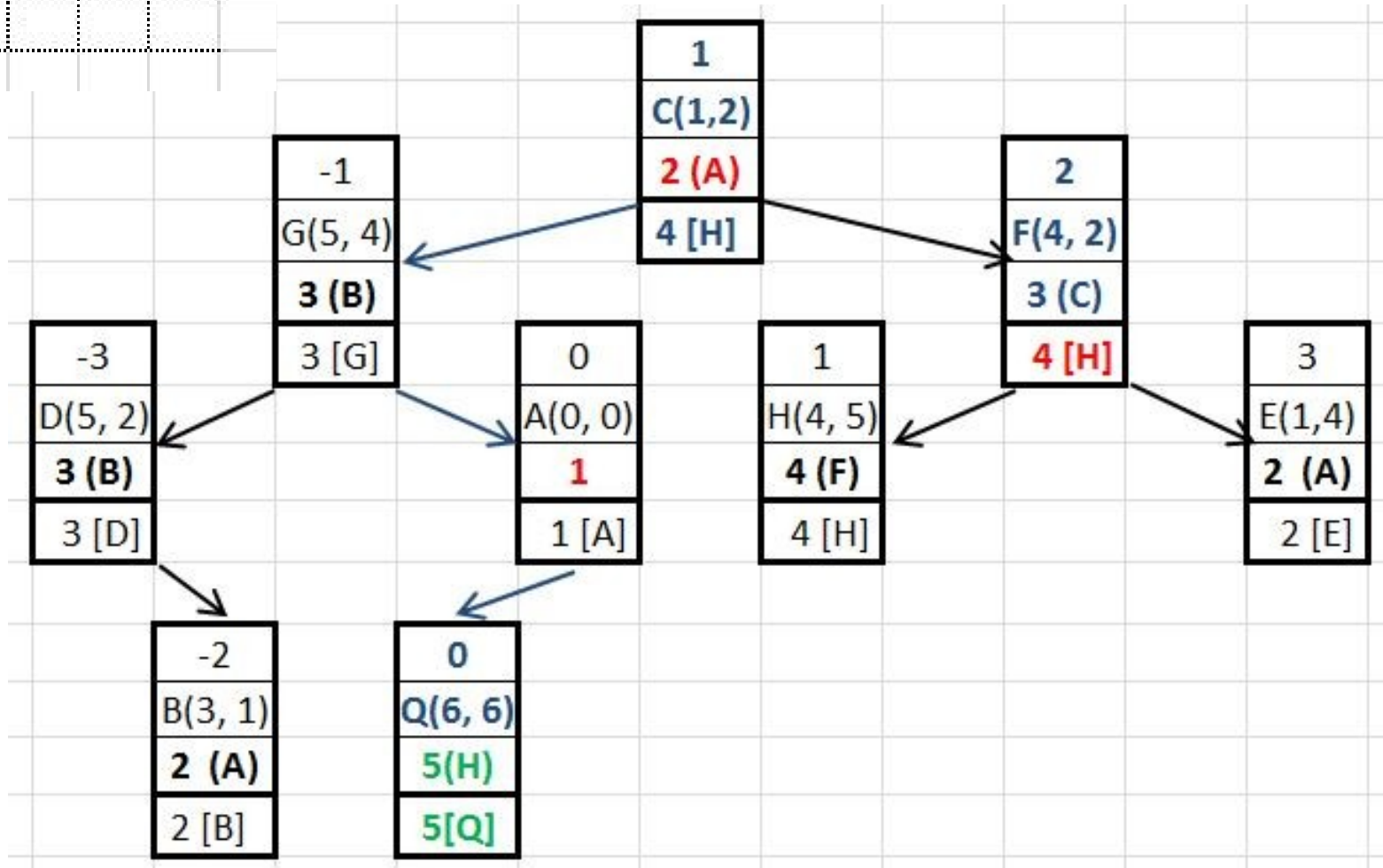
Обработка Q. key(Q) = 6 – 6 = 0
1. Вставка Q

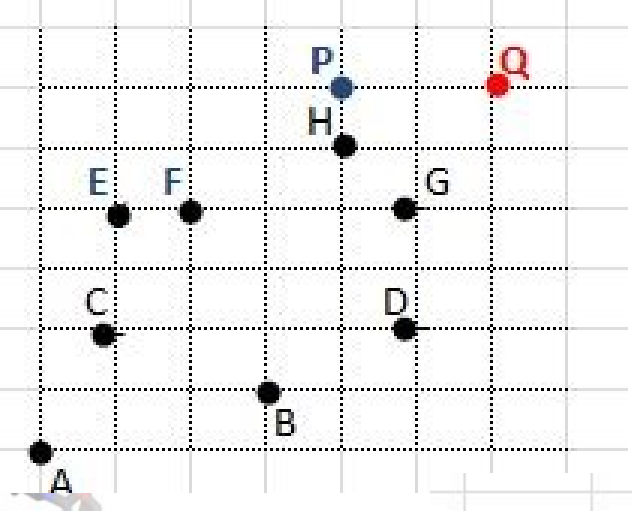




Обработка Q. key(Q) = 6 – 6 = 0

1. Вставка Q
2. Вычисление характеристик Q

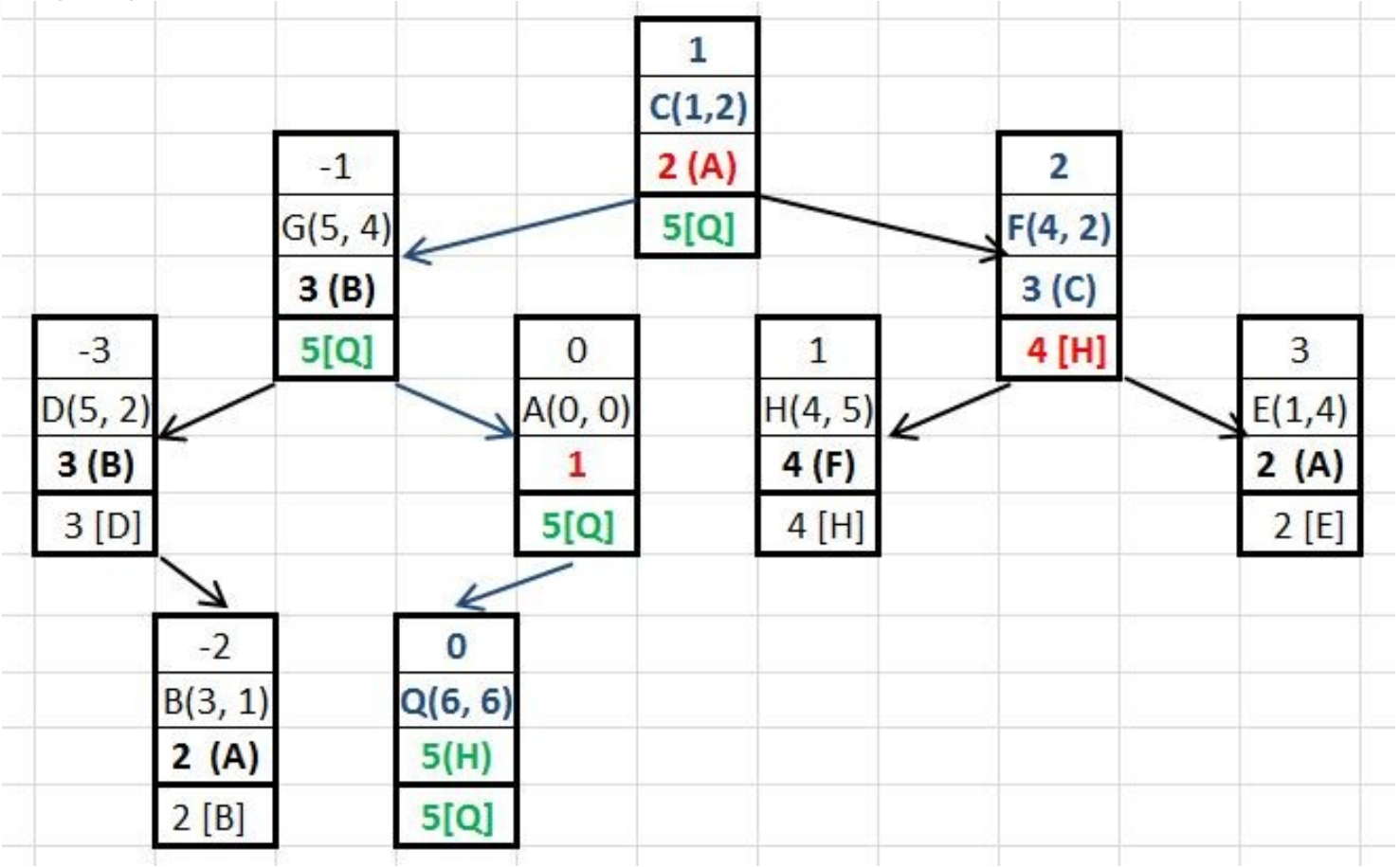


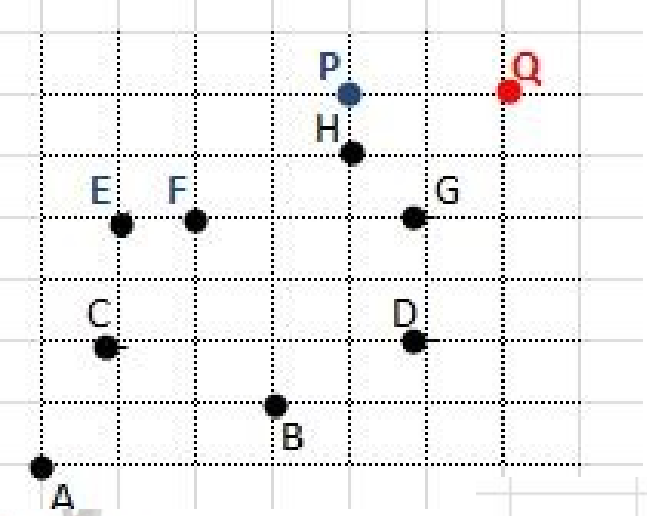


Обработка Q. key(Q) = 6 – 6 = 0

1. Вставка Q
2. Вычисление характеристик Q
3. Обновление MaxSubTr у предш.Q

Время ~ log(M)

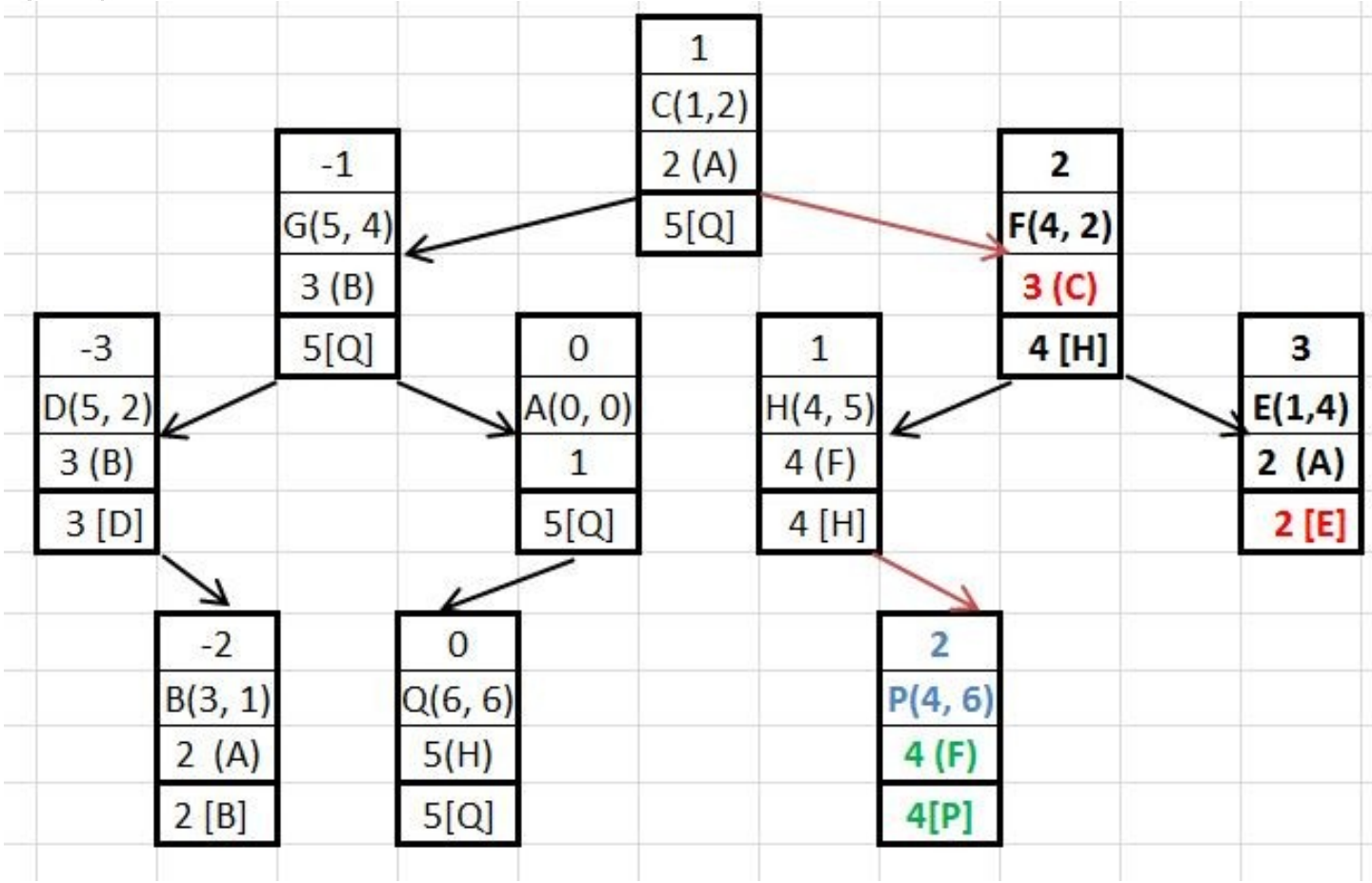


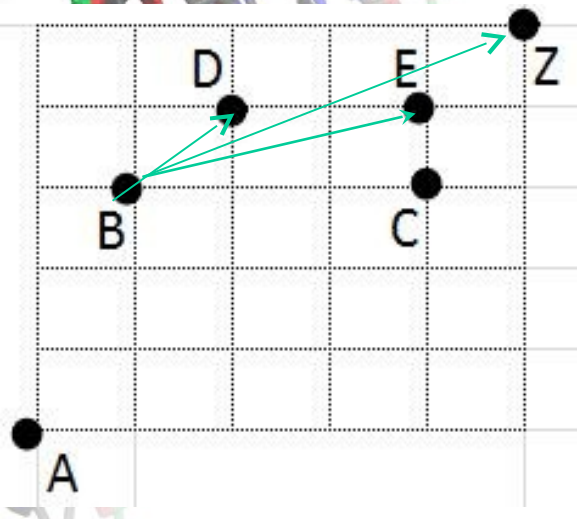


Обработка P. key(P) = 6 – 4 = 2

1. Вставка P
2. Вычисление характеристик P
3. Обновление maxSubTr

Время ~ log(M)





Постановка задачи

Решётка NxN .

Вершины в (некоторых) узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

$p(R_1, R_2)$ – штраф за переход из R_1 в R_2

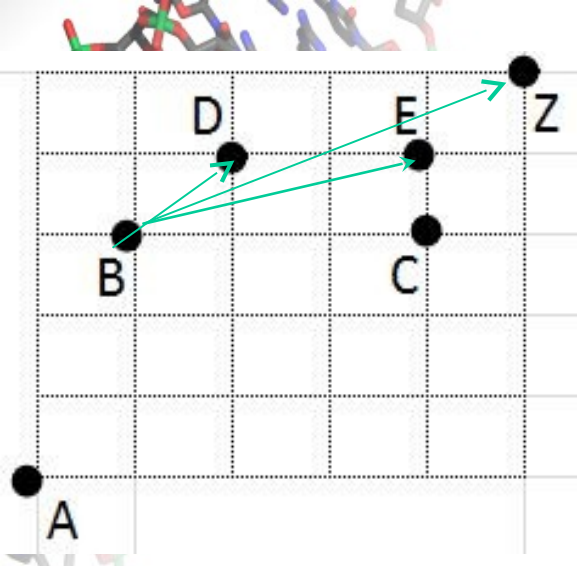
$p(R_1, R_2) = f(dx, dy)$, где $dx = x_2 - x_1$; $dy = y_2 - y_1$

Вес пути = Сумма весов – Сумма штрафов

Вес ребра $(R_1, R_2) = w(R_2) - p(R_1, R_2)$

Вес пути = Сумма весов ребер

Максимальный путь ?



Постановка задачи

Решётка NxN . **К-во вершин - M**

Вершины в (некоторых) узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

p(R1, R2) – штраф за переход из R1 в R2

p(R1, R2) = f(dx, dy), где dx = x2 - x1; dy = y2 - y1

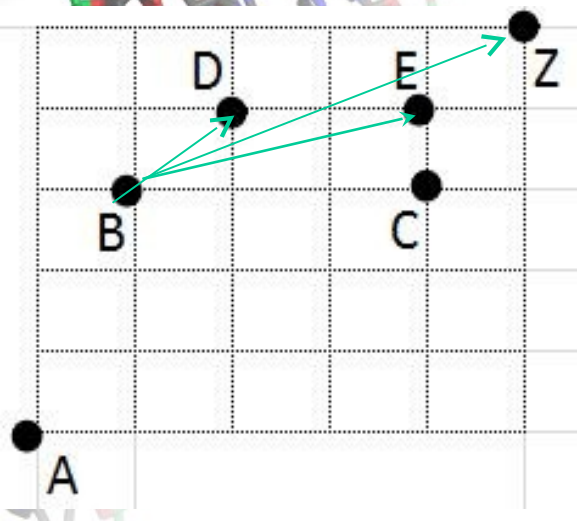
Вес пути = Сумма весов – Сумма штрафов

Вес ребра (R1, R2) = w(R2) –p(R1, R2)

Вес пути = Сумма весов ребер

Максимальный путь?

Время ~ M*log(M)



Постановка задачи

Решётка NxN . **К-во вершин - M**

Вершины в (некоторых) узлах

Ребро из $R_1(x_1, y_1)$ в $R_2(x_2, y_2)$ $\Leftrightarrow$

$\Leftrightarrow$ $x_1 < x_2$ & $y_1 < y_2$

w(R) – вес вершины R ; w(A) = w(Z) = 0

p(R1, R2) – штраф за переход из R1 в R2

p(R1, R2) = f(dx, dy), где dx = x2 - x1; dy = y2 - y1

Вес пути = Сумма весов – Сумма штрафов

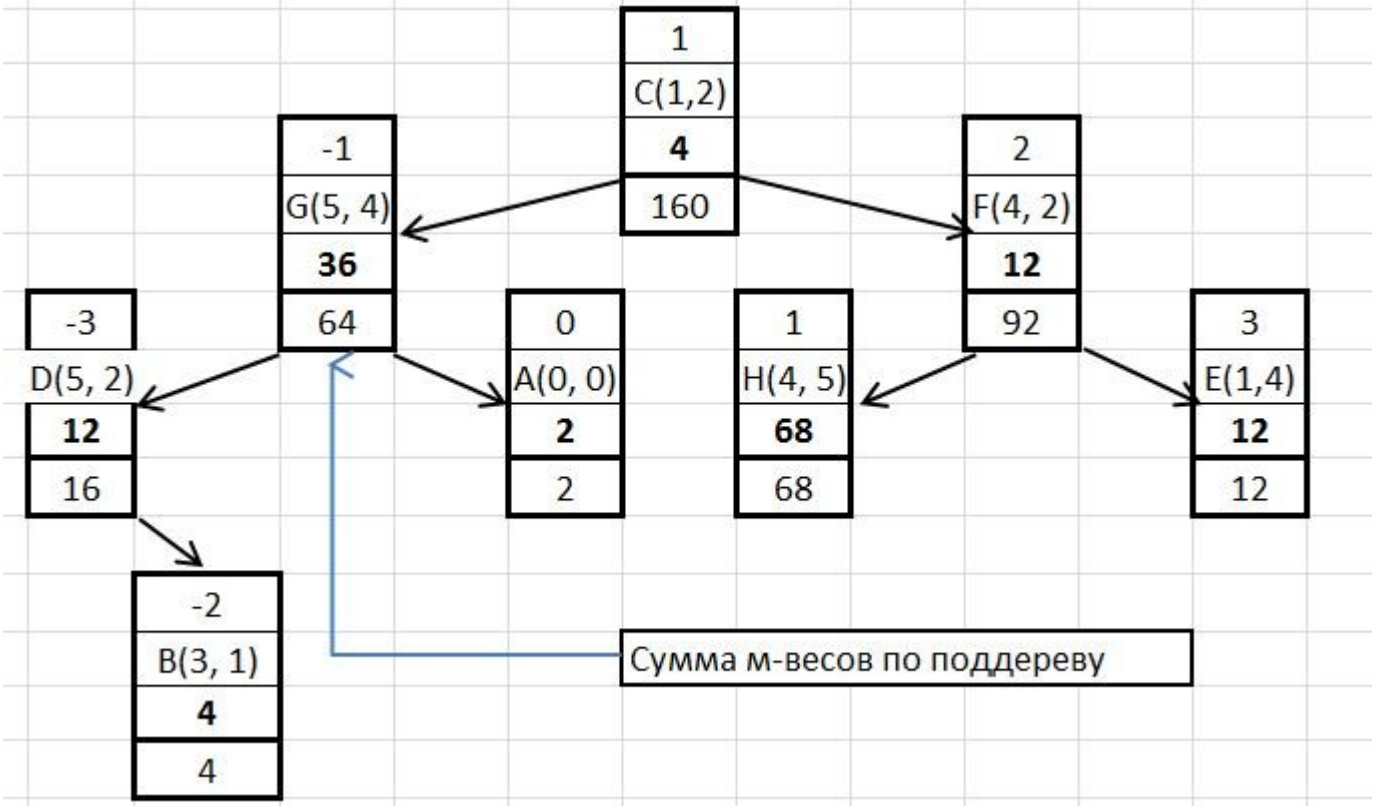
Вес ребра (R1, R2) = w(R2) –p(R1, R2)

M-вес пути = Произведение экспонент весов ребер

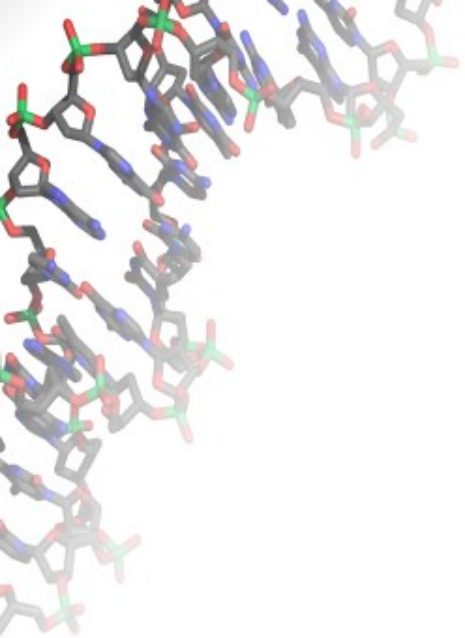
Сумма м-весов всех полных путей?

Время ~ M*log(M)

Sum{Score(X) | key(X) ≥ key(R)}=
= 2*Sum{Score(R), SumSubTr(RightSon(R),
Score("LeftParent"(R)),
SumSubTr(RSon"LeftParent"(R)),
… вверх по дереву,«глядя вправо»}



	A	B	C	D	E	F	G	H	Q	P
Коорд.	(0, 0)	(3, 1)	(1, 2)	(5, 2)	(1, 4)	(2, 4)	(5, 4)	(4, 5)	(6, 6)	(4, 6)
сумма м-весов	2	4	4	(4+2)*2 = 12	(4+2)*2 = 12	(4+2)*2 = 12	(2+4+12)*2 =36	(2+4+4+12+12)*2 = 68		



Подготовка: зоны влияния

C(k, m; i, j) =

E[k, l] - l(j - i) - (k - l)|.

Левая зона (влияния) для (k, m)

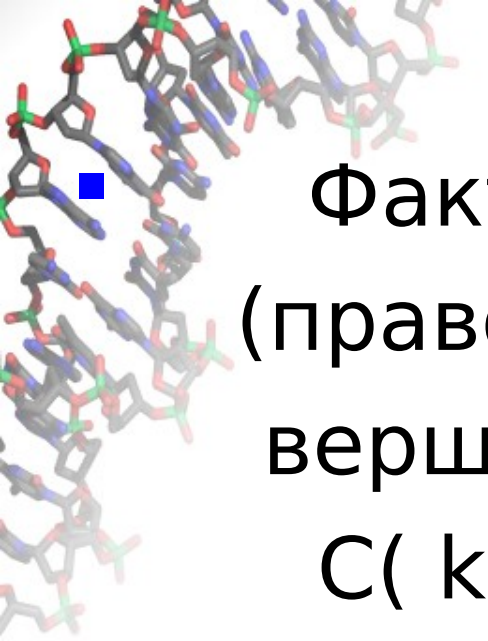
m-k ≤ j-i =>

=> C(k, m; i, j) = E[k, m] – ((j - i)-(m - k))

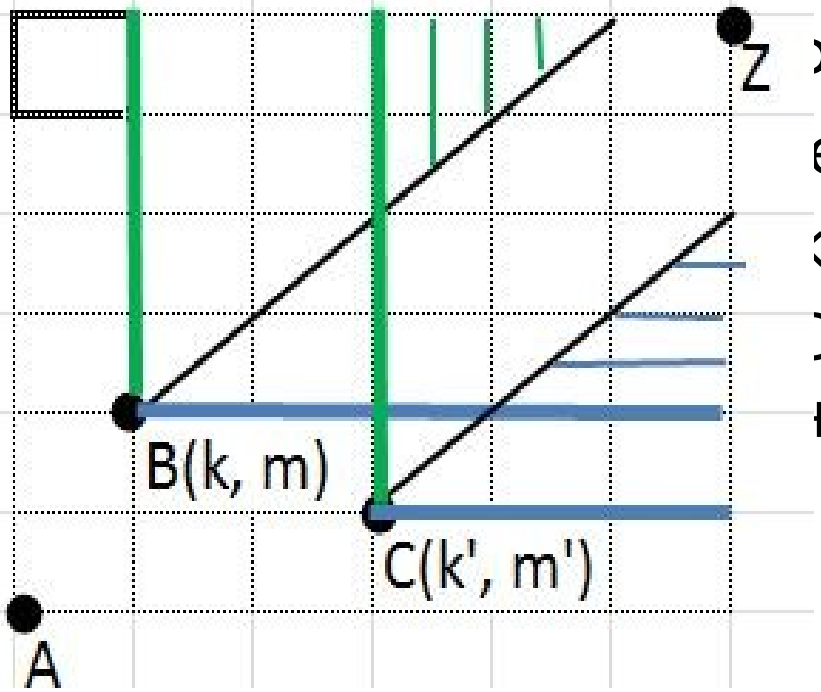
Правая зона (влияния) для (k, m)

m - k > j-i =>

=> C(k, m; i, j) = E[k, m] - ((m – k)-(j - i))



- Факт. Пусть (i, j) лежит в левой (правой) зоне одновременно для вершин (k, m) и (k’, m’) и пусть $C(k, m; i, j) \leq C(k', m'; i, j)$.
- Тогда (k, m) будет «лучше», чем (k’, m’) при вычислении C(x, y, A, B) для всех точек (A, B), которые лежат одновременно в левой (правой) зоне влияния (k, m) и (k’, m’)



Факт. Пусть (i, j) лежит в левой (правой) зоне и для (k, m), и для (k’, m’). Тогда C(k, m; i, j) - C(k’, m’; i, j) не зависит от (i, j).

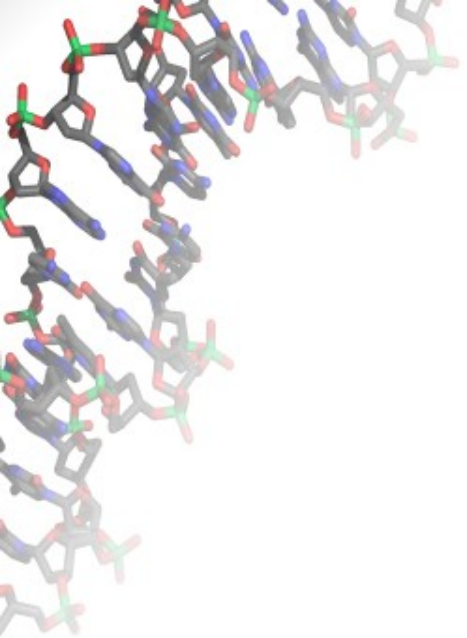
Доказательство. Пусть (i, j) лежит в левых зонах. Имеем:

$$C(k, m; i, j) = E[k, m] - ((j - i)-(m - k))$$

$$C(k', m'; i, j) = E[k', m'] - ((j - i)-(m' - k'))$$

Вычитаем. Все, что нужно – сокращается.

Для правых зон – аналогично.



ДАЛЬШЕ – НА ДОСКЕ ☺