String Distance and Dynamic Programming

Lecture 5

Life is similar

- Life is based on a repertoire of successful structural and interrelated building blocks which are passed around
- The vast majority of proteins are the result of a series of genetic duplications and subsequent modifications
- “Everything in life is so similar that the same genes that work in flies are the ones that work in humans” (Wieschaus, 1995)

Comparison and analogy

- By identifying and comparing related objects we can distinguish variable and conserved features, and thereby determine what is crucial to structure and function
- Biological universality occurs at many levels of details, so we can compare not only the sequence data, but 3D shapes, chemical pathways, morphological features etc.

Why compare biosequences

- The biological sequences encode and reflect higher-level molecular structures and mechanisms
- **In bimolecular sequences (DNA, RNA or protein), high sequence similarity <u>usually</u> implies significant structural and functional similarity**
- A tractable, though partly heuristic way to infer the structure and function of an unknown protein is to search for the similar known proteins at the sequence level

Keep in mind

- There is not a one-to-one correspondence between similar sequences and similar structures or between sequences and functions:
  - Similar structures can be obtained from completely unrelated sequences
  - Very similar sequences can produce very different structures depending on the location of a change

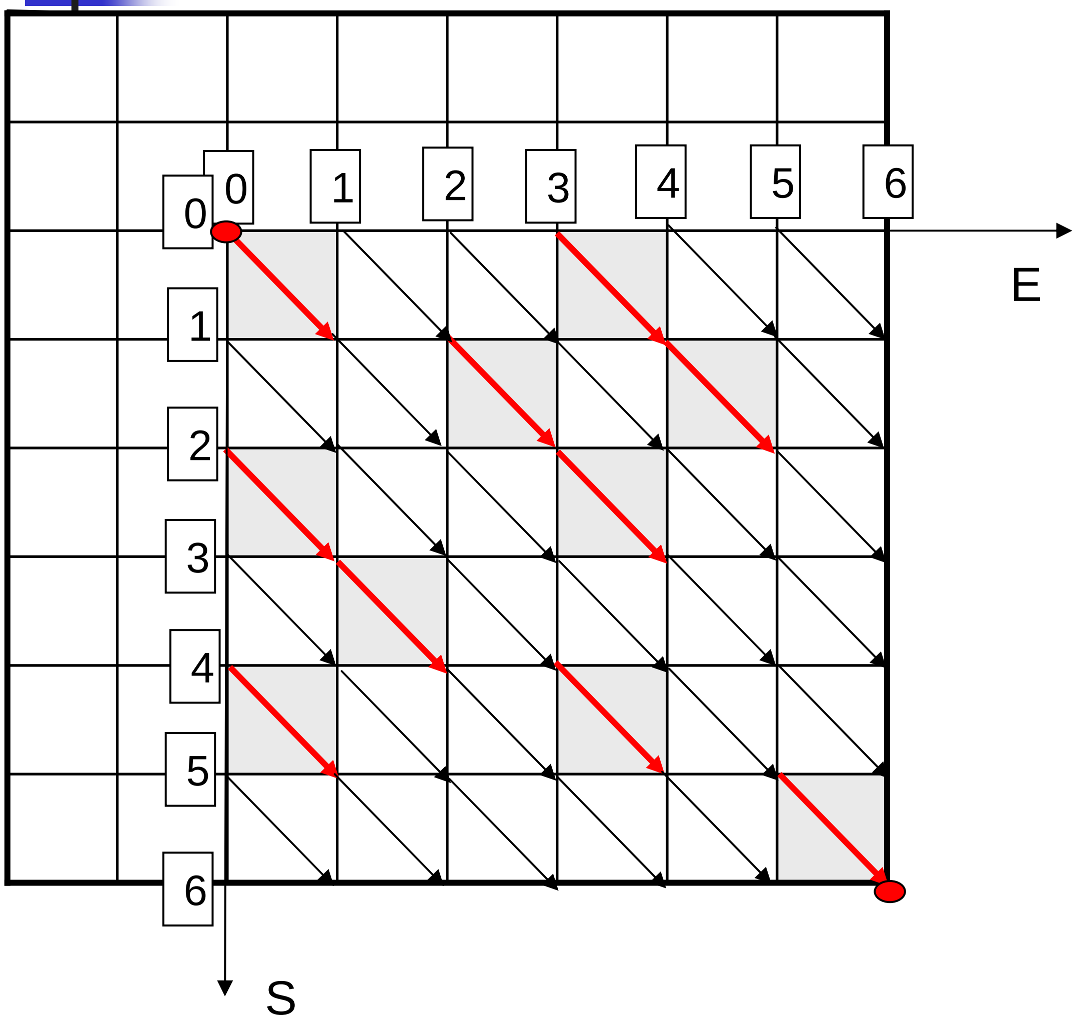
A shift to approximate pattern matching

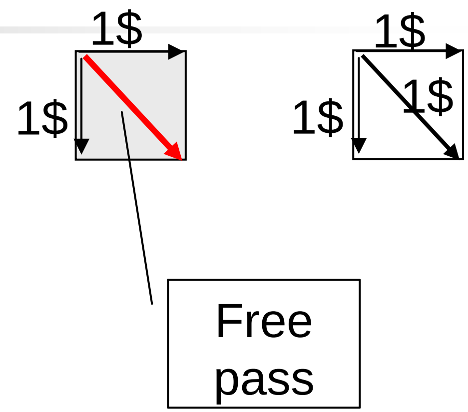
- Approximate – means some errors are allowed in valid matches
- The shift is accompanied by a shift in technique: *dynamic programming*

Dynamic programming

The main tool in approximate pattern matching

The cheapest path

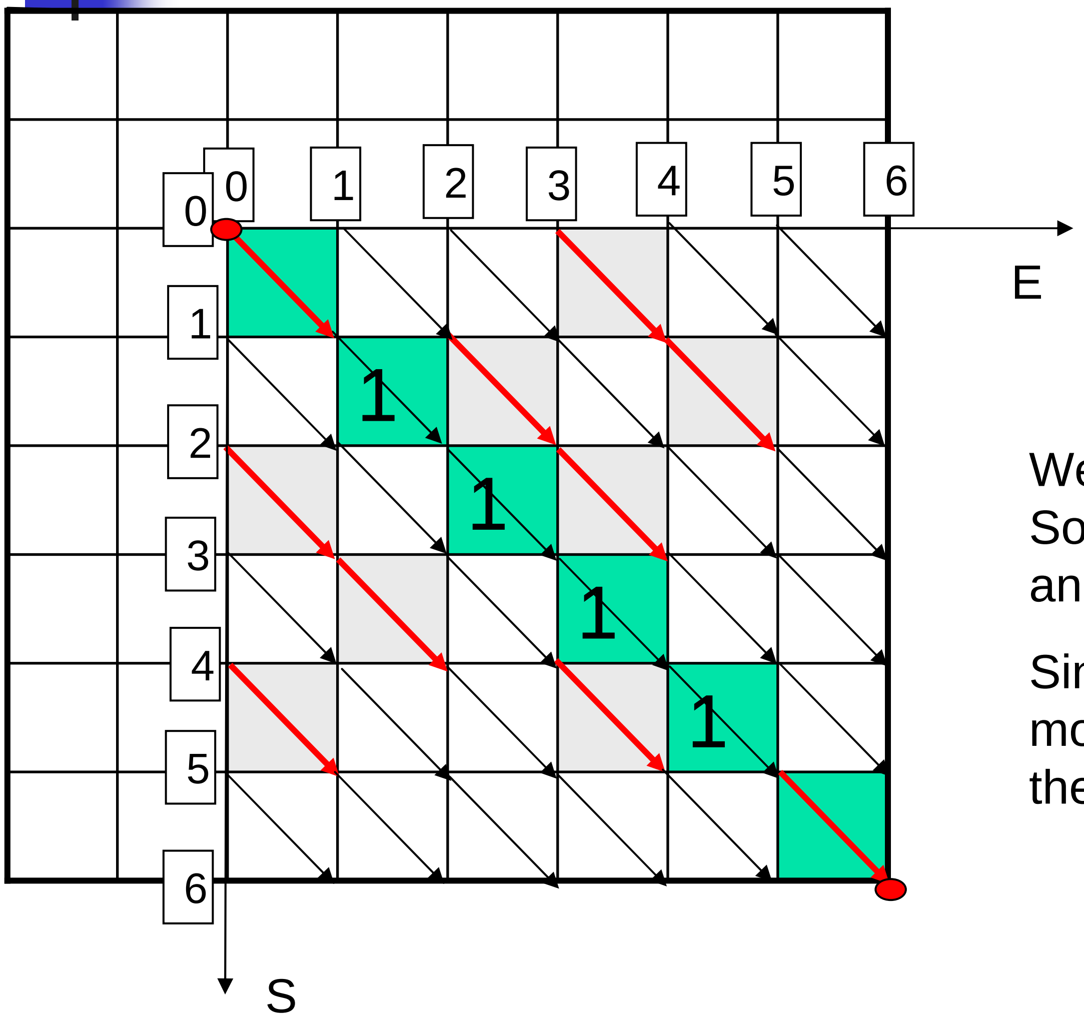


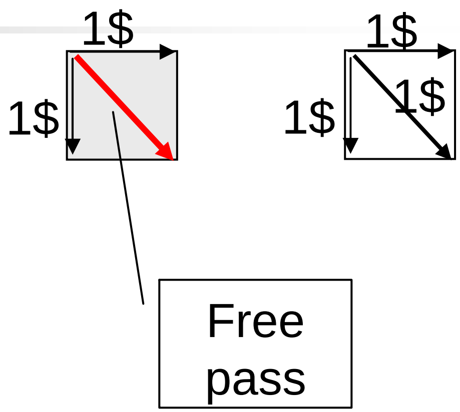


Problem:

find the cheapest path from (0,0) to (6,6)

The path without a map

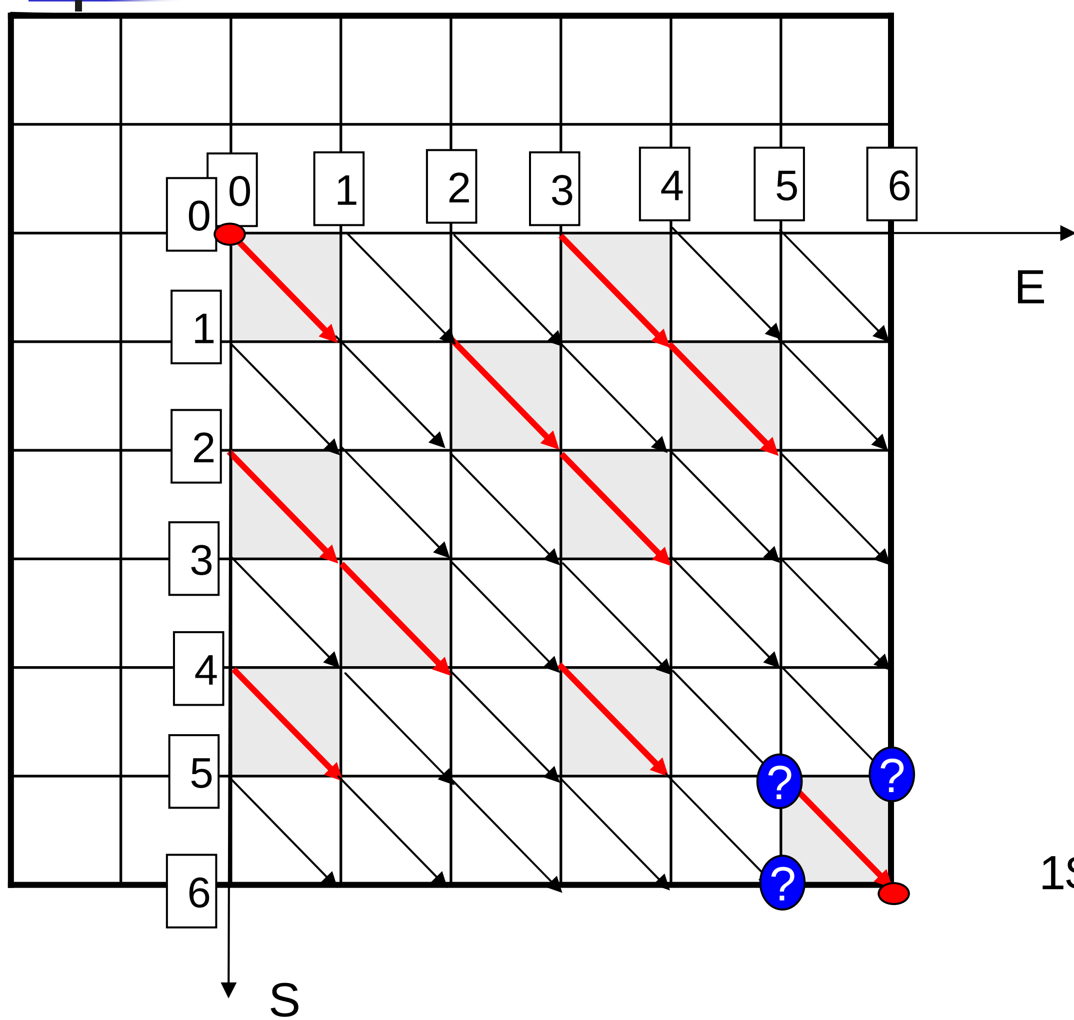


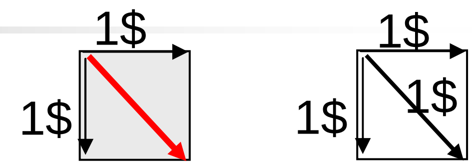


We will always choose the South-East direction (diagonal), and we will pay 4$

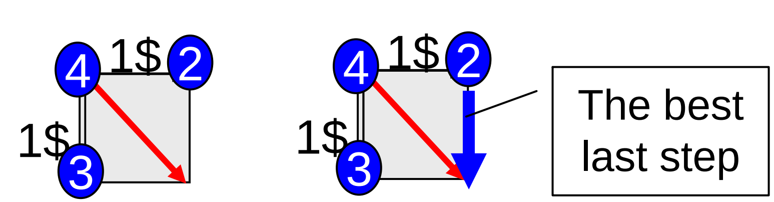
Since we don't know that if we move strictly East or South, there are more free-pass cells

Sub-problems approach

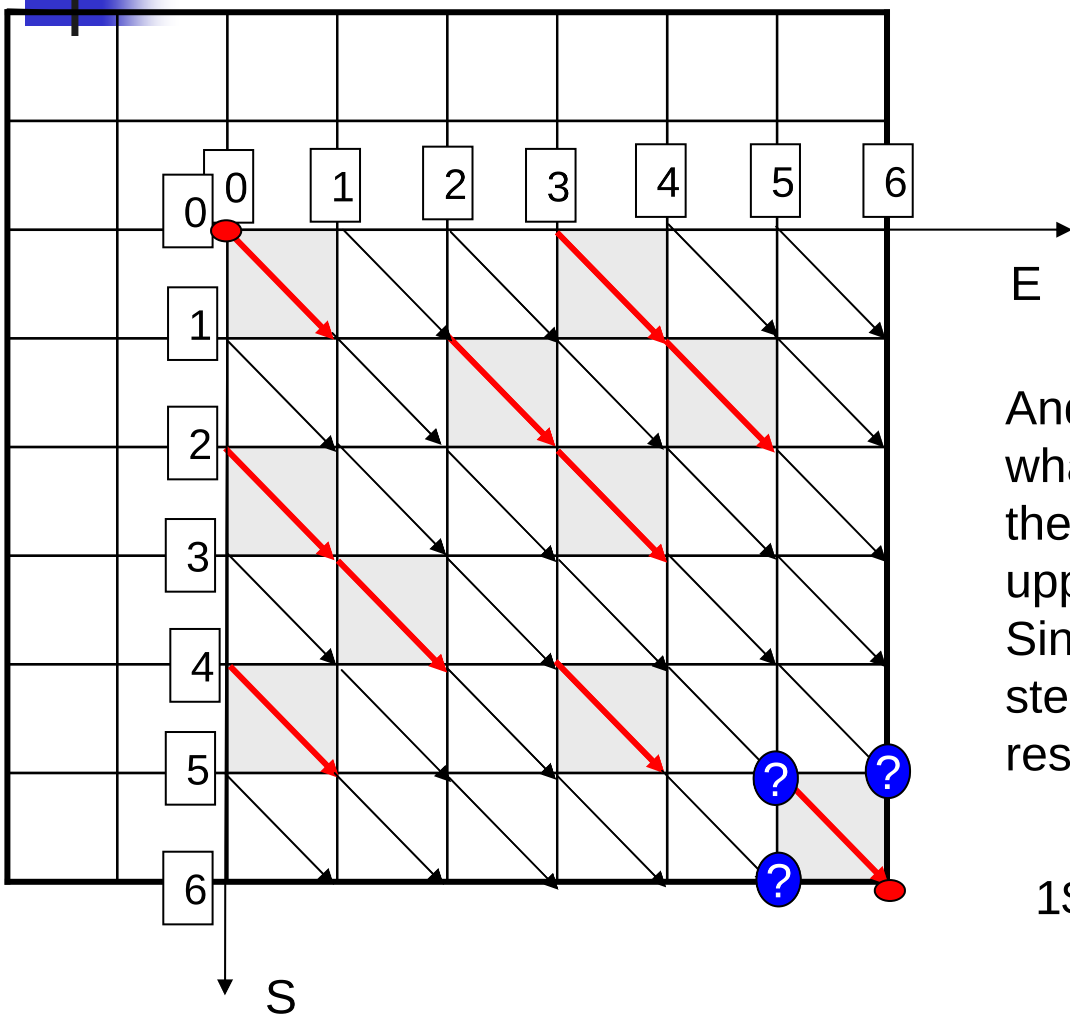


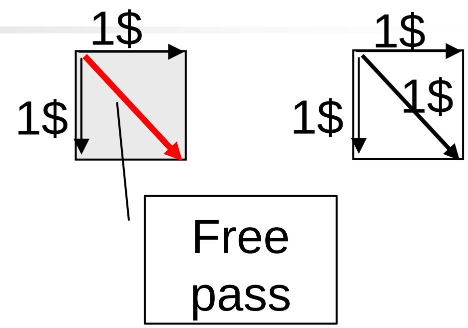


If we knew the cheapest paths
from (0,0) to (5,5)
from (0,0) to (6,5)
from (0,0) to (5,6)
we could choose the best last step to the destination:
For example, if:

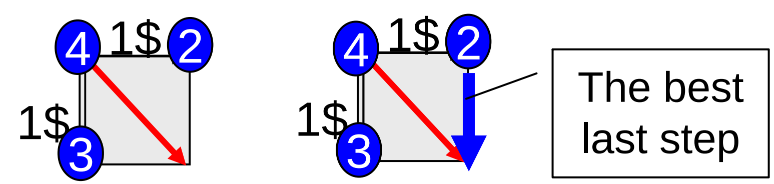


The sub-problems approach

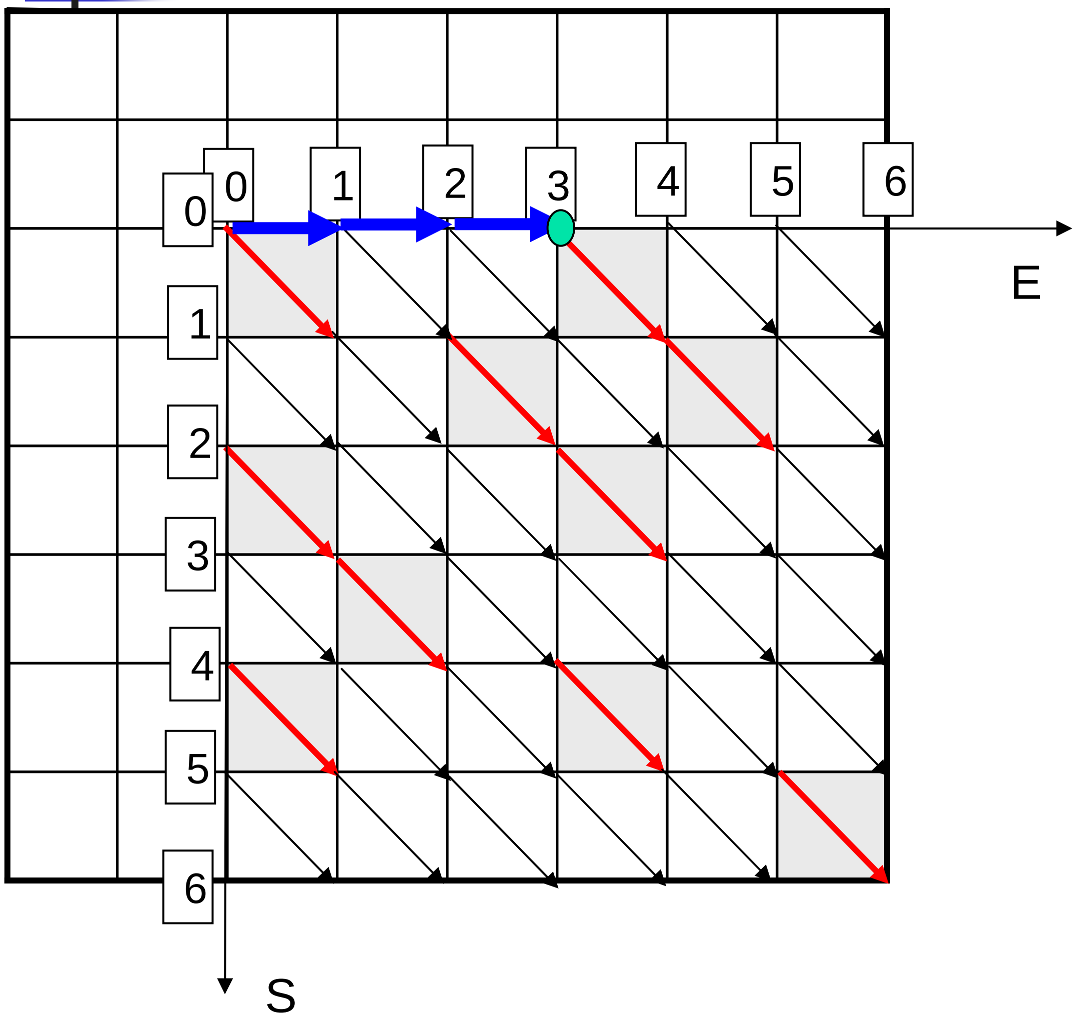




And this is true for any cell – what path to chose depends on the cheapest paths to the left, upper, and upper-left corner. Since we are choosing only 1 step, we can take the min of the result



The recurrence relation – base condition



When i=0, there is no cheaper way of going from (0,0) to (0,j) than to pay j $ - heading strictly to the right (East)

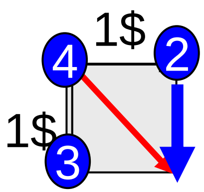
The same for j=0.

The base condition:

if i=0 then COST(i,j)=j

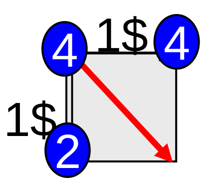
if j=0 then COST(i,j)=i

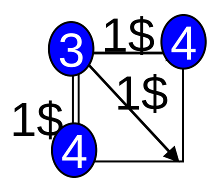
The recurrence relation (for i>0 and j>0)

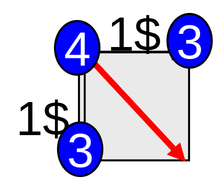


$$\text{COST}(i,j)=\min\begin{cases}\text{COST}(i-1,j)+1\\ \text{COST}(i,j-1)+1\\ \text{COST}(i-1,j-1)+\text{DIAGONAL}(i,j)\end{cases}$$

For each case, what is the best move?



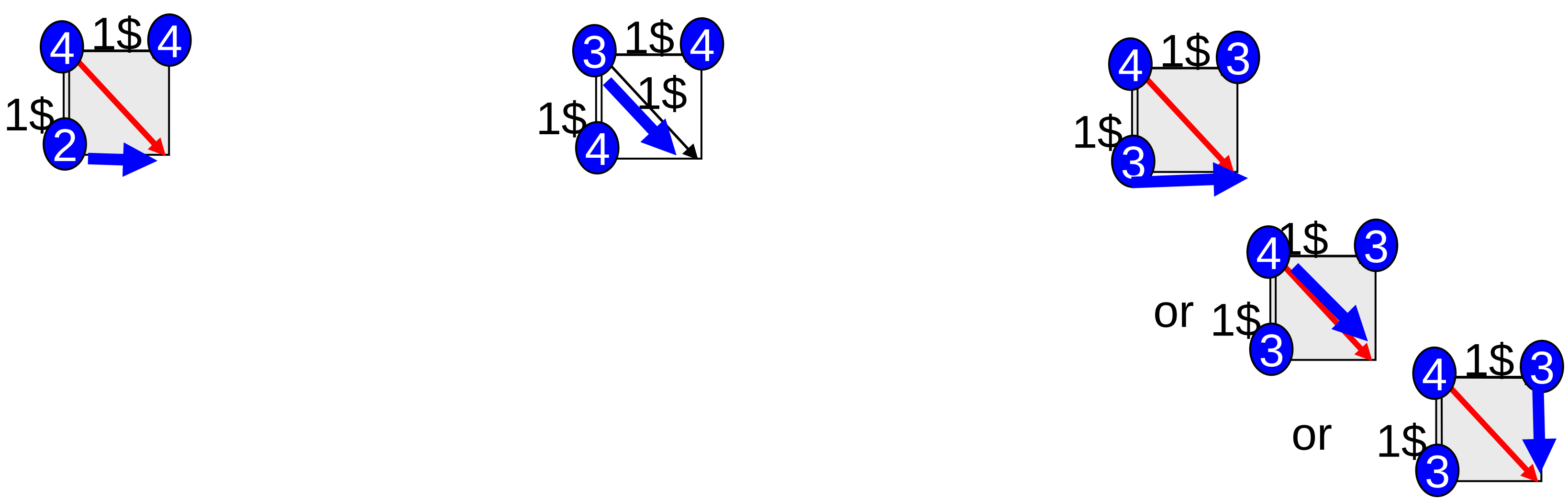




The recurrence relation

$$\text{COST}(i,j)=\min\begin{cases}\text{COST}(i-1,j)+1\\ \text{COST}(i,j-1)+1\\ \text{COST}(i-1,j-1)+\text{DIAGONAL}(i,j)\end{cases}$$

The best moves:



The top-down (usual) recursion

$$COST(i,j)=\min\begin{cases} COST(i-1,j)+1 \\ COST(i,j-1)+1 \\ COST(i-1,j-1)+DIAGONAL(i,j) \end{cases}$$

algorithm cheepestCost (**array** *diagonalCost, N*, *M*)

    return ***cost*** (*N*, *M*)

algorithm ***cost*** (*i*, *j*)

    if *i*=0 **then**

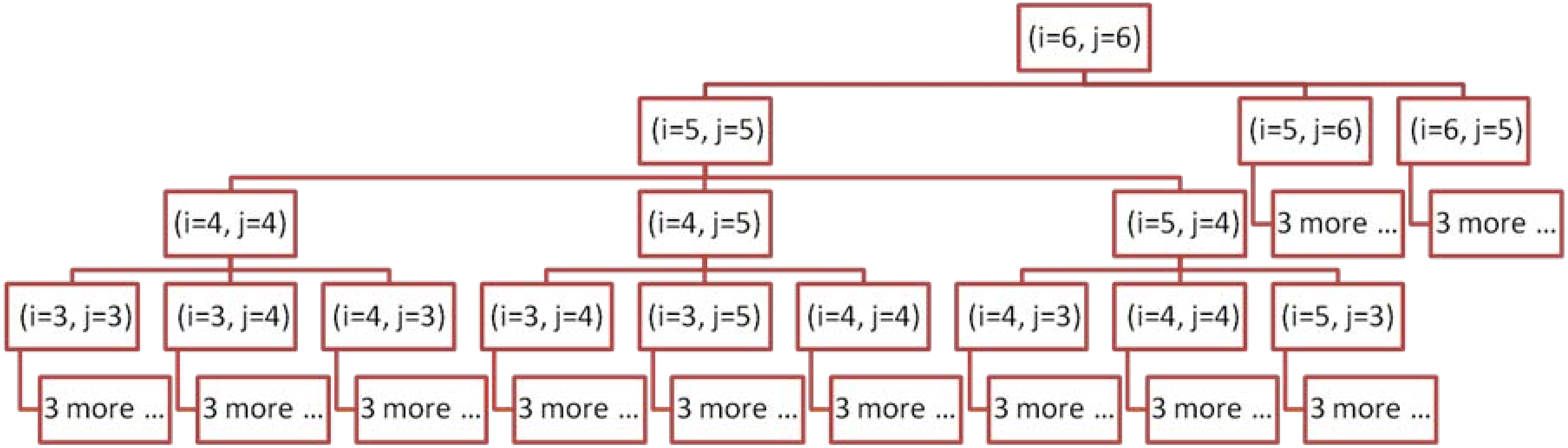
        return *j*

    if *j*=0 **then**

        return *i*

    return min (***cost*** (*i*-1, *j*) +1, ***cost*** (*i*, *j*-1)+1, ***cost*** (*i*-1, *j*-1)+*diagonalCost* [*i*] [*j*])

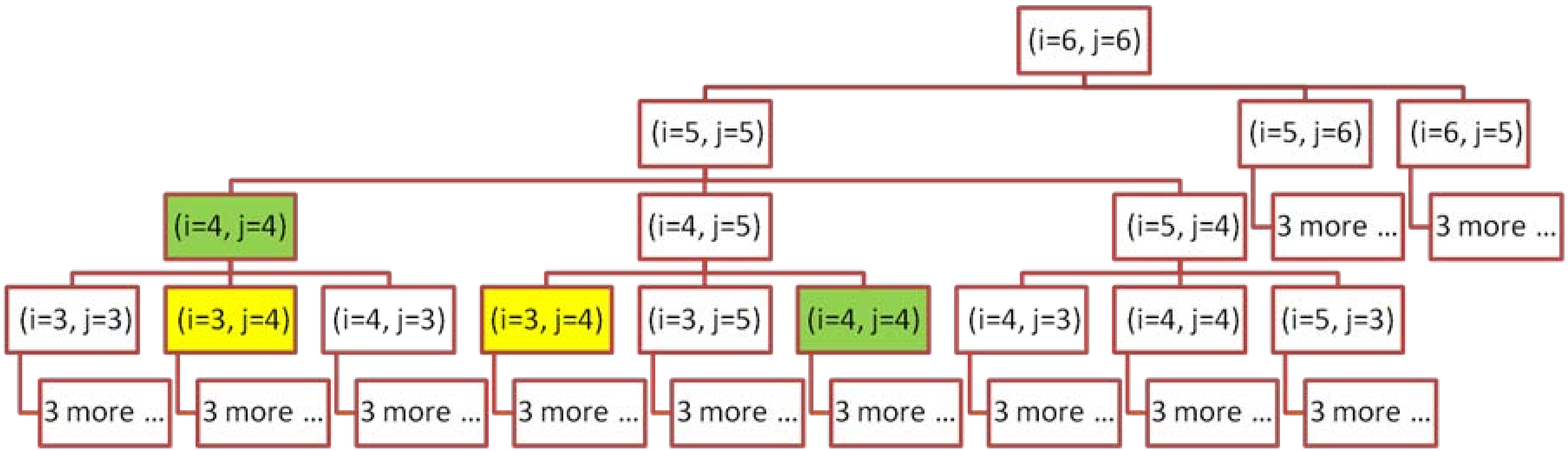
The recursion tree



$O(3^N)$?

But there are only N*M different combinations

The recursion tree



$O(3^N)$?

We call the recursive function multiple times with the same parameters

Dynamic programming steps

- The recurrence relation
- The bottom-up computation
- The traceback

Dynamic programming I

- **The recurrence relation**
- The bottom-up computation
- The traceback

The recurrence relation

The base condition:

if i=0 then COST(i,j)=j
if j=0 then COST(i,j)=i

The main relation (for i>0 and j>0)

$$COST(i,j)=\min\begin{cases} COST(i-1,j)+1 \\ COST(i,j-1)+1 \\ COST(i-1,j-1)+DIAGONAL(i,j) \end{cases}$$

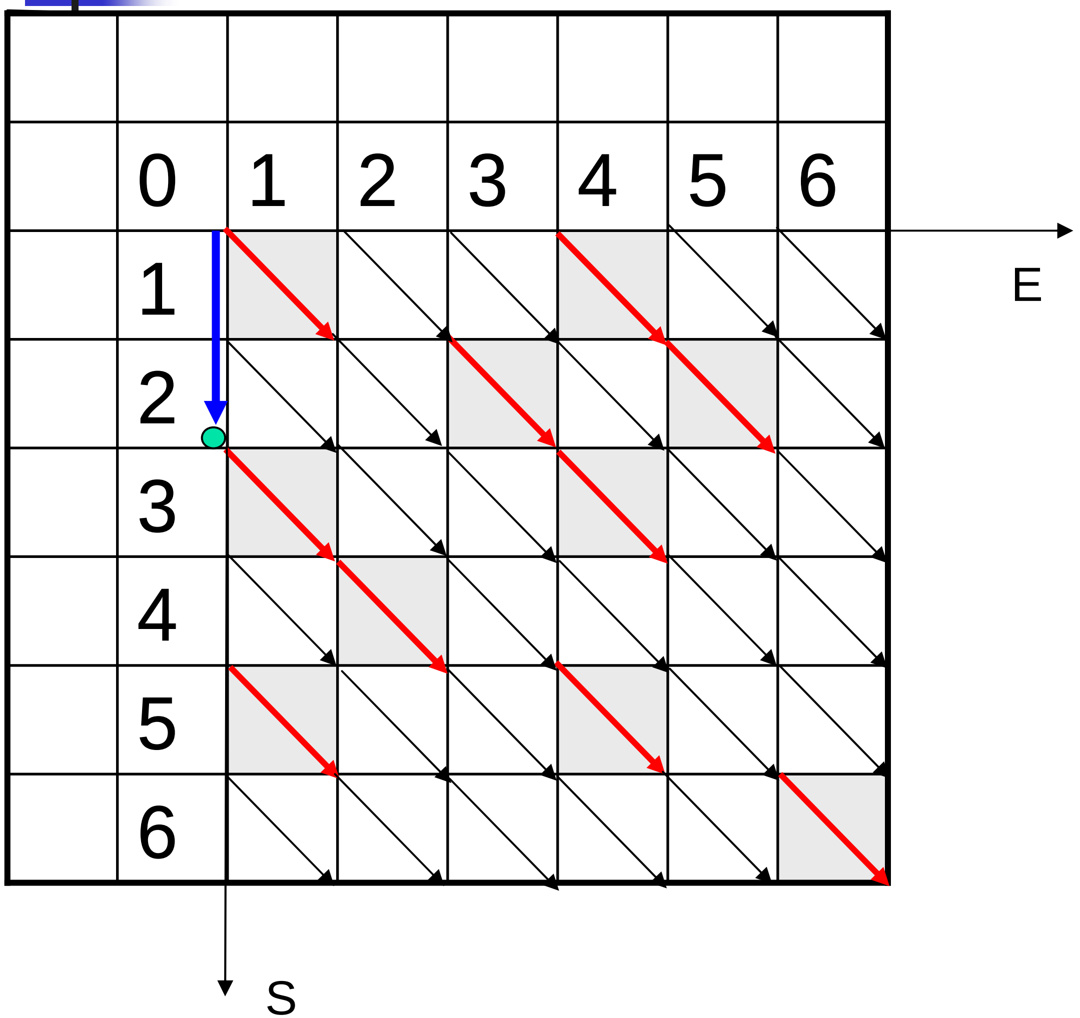
Dynamic programming II

- The recurrence relation
- **The bottom-up computation**
- The traceback

The bottom-up computation

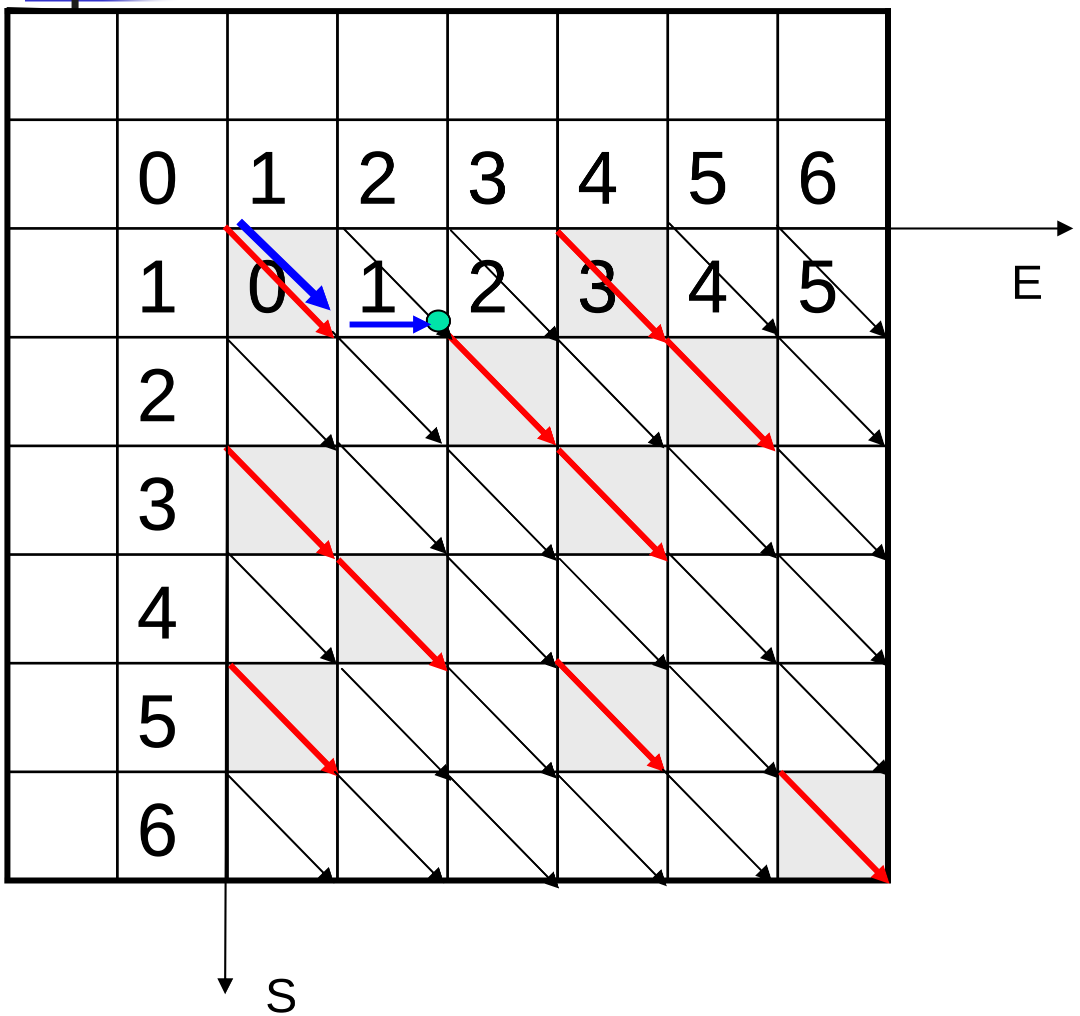
- Fill in the best values for each cell of the N*M table starting from the lowest values
- First, compute the basic values of recursion – for i=0 and for j=0
- Apply recursion relation for computing the value of each cell from the lowest numbers of i and j to the largest
- At the end, we will have the cost of the best path in the cell (N,M)

Fill values for i=0 and for j=0 (the base recursion condition)



There is no cheaper way of going to the point (2,0) than paying 2 $

Fill values for i=1 (from left to right)



Cell(1,2)=1

since the cheapest possible way is to continue the free path through the cell (1,1)

Fill in the entire table (left-to-right top-down)

	0	1	2	3	4	5	6
	1	0	1	2	3	4	5
	2	1	1	1	2	3	4
	3	2	2	2	1	2	3
	4	3	2	3	2	2	3
	5	4	3	3	3	3	3
	6	5	4	4	4	4	3

E

S

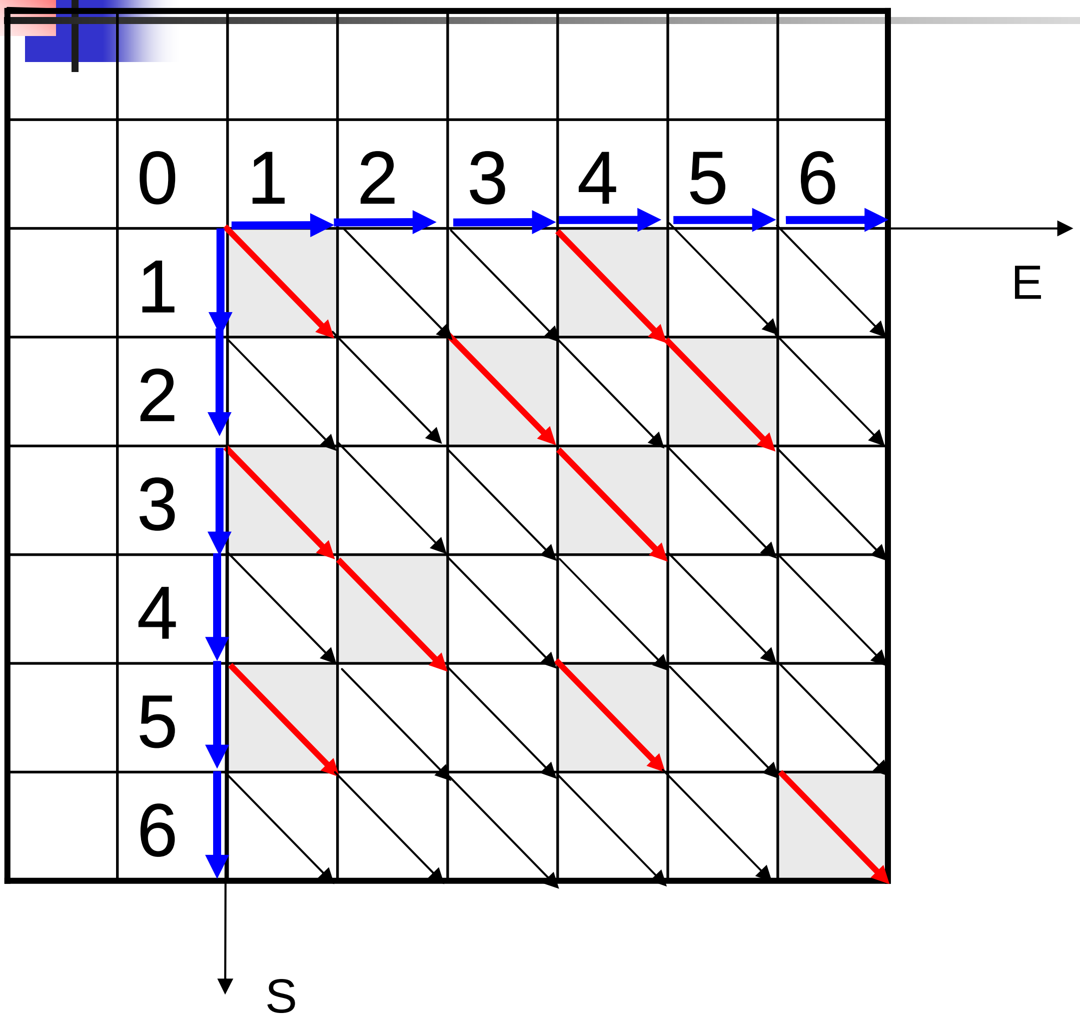
The cheapest possible path costs 3$

But what is this path?

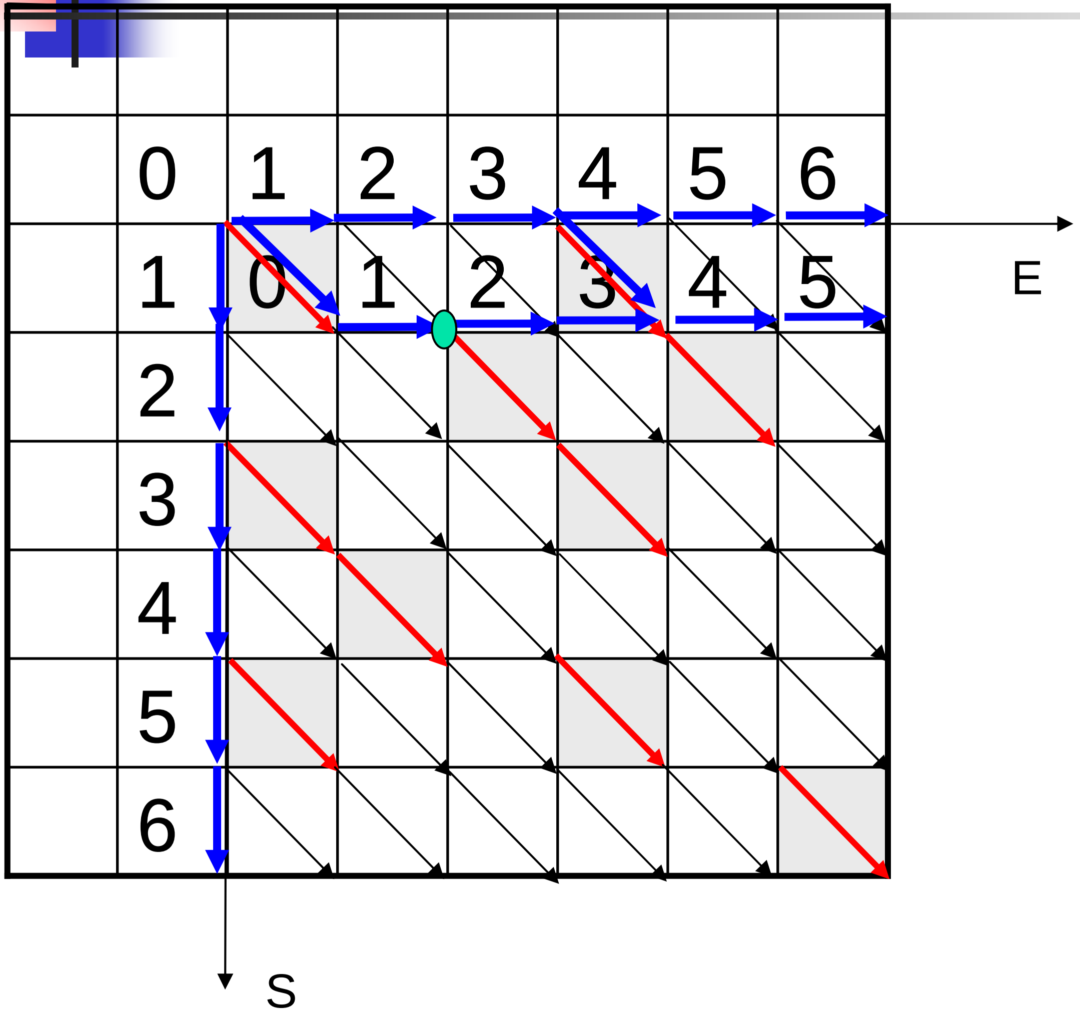
Dynamic programming III

- The recurrence relation
- The bottom-up computation
- **The traceback**

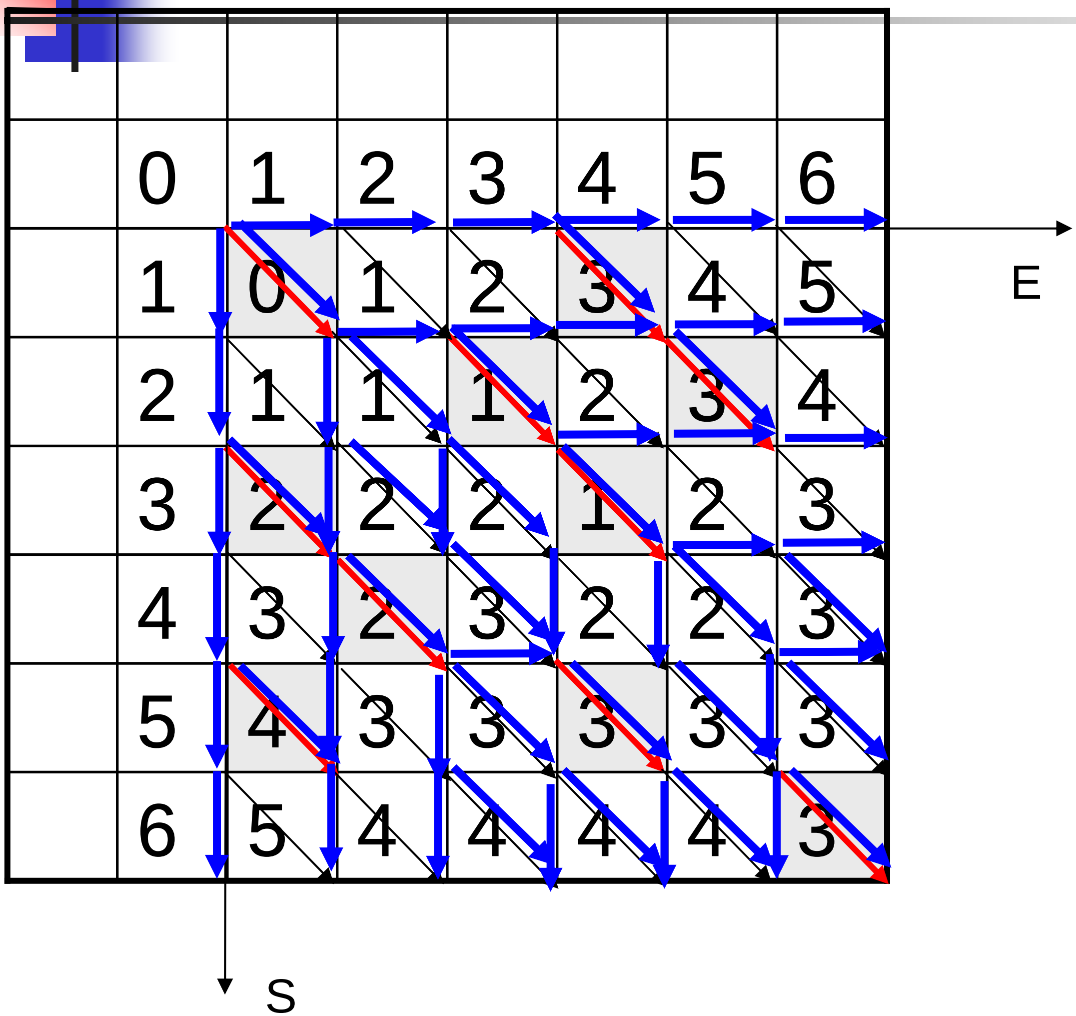
Keeping track of the source



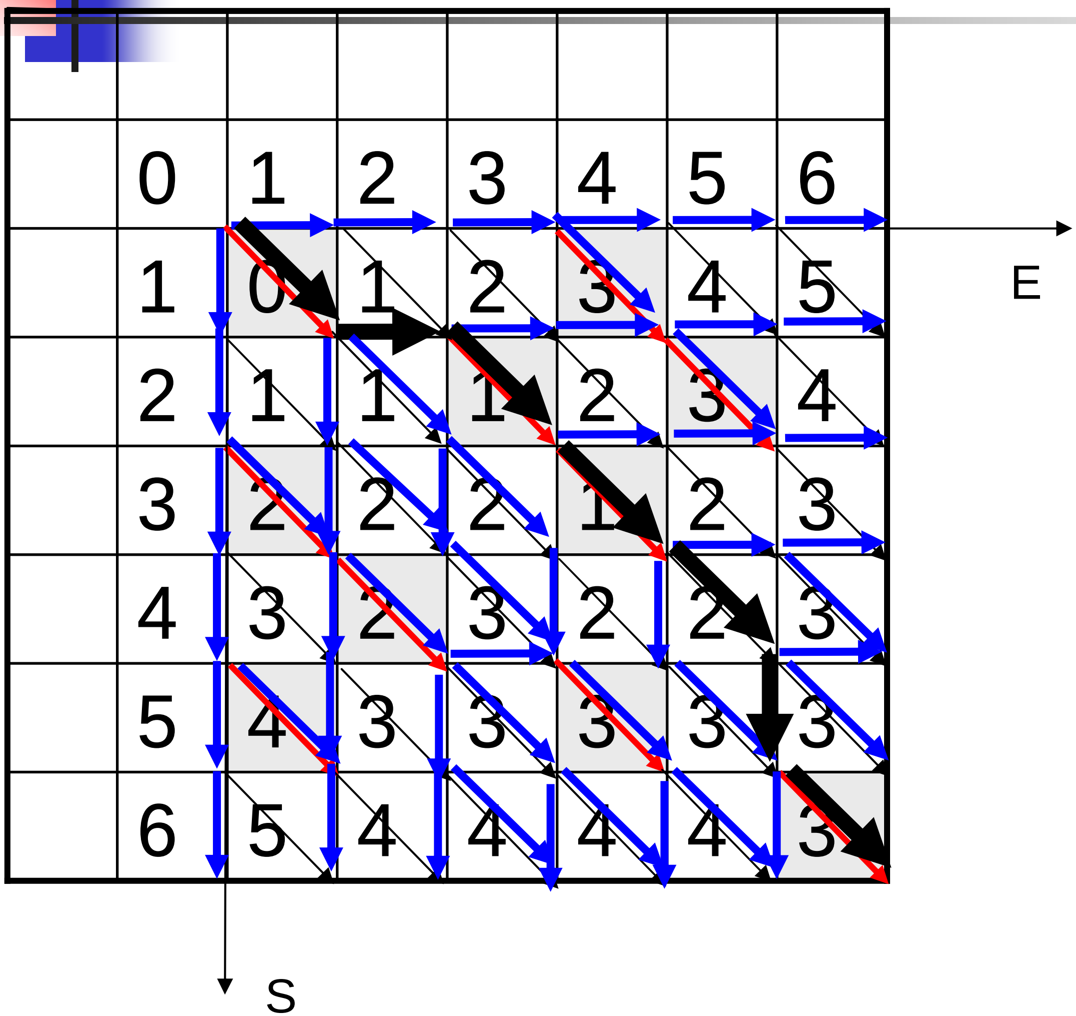
Keeping track of the source



Keeping track of the source



Trace back – how did we get the path with the cost 3



Dynamic programming with electronic tables. Cost

- Build the input table – the cost of passing through any cell by diagonal
- Create the distance table, fill the first row and the first column according to the basic recursion
- Insert the recursion formula in cell [1][1]:
  - C19= MIN(B18+C3,B19+1,C18+1)
- Spread the formula to the rest of the table by drag-and-release
- Read the cost of the cheapest path in cell [N][M] – the last cell of the cost table

Dynamic programming with electronic tables. Cost

The cost of passing through the corresponding cell of the input table

C19=MIN(B18+C3,B19+1,C18+1)

Current cell:
i=C, j=19

i-1=B,
j-1=18

i-1=B,
j=19

i=C,
j-1=18

Dynamic programming with electronic tables. Forward path

Excel code

```
C35=
IF(B18+C3<B19+1,
   IF(B18+C3<C18+1,
        "DownRight",
        "Right"),
   "Down")
```

```
IF(B18+C3<B19+1)
   IF(B18+C3<C18+1)
        C35="DownRight“
   ELSE
        C35="Right“
ELSE
   C35="Down"
```

Shows one of the possible paths to obtain the smallest cost for a path from (0,0) to the current cell

Dynamic programming with electronic tables. Backward path

Excel code

```
C49=
IF(C35="Down",
   "Up",
   IF(C35="Right",
      "Left",
   "UpLeft"))
```

```
IF(C35 ="Down“)
   C49=“Up”
ELSE
   IF(C35=“Right”)
      C49=“Left“
   ELSE
      C49=“UpLeft“
```

Replacing by the opposite direction – from the destination cell to the source cell

Dynamic programming with electronic tables. Traceback

Excel code

```
B60=
IF(AND(C61="X",C49="UpLeft"),
    "X",
IF(AND(C60="X",C48="Left"),
    "X",
IF(AND(B61="X",B49="Up"),
    "X",
    "-")))
```

```
IF( C61="X"AND C49="UpLeft")
    B60="X"
ELSE IF( C60="X" AND C48="Left")
    B60="X"
ELSE IF( B61="X" AND B49="Up")
    B60="X"
ELSE
    B60="-"
```

By placing X in the destination cell, this code reconstructs the path which gave the total minimum cost: cell is marked X if the path went through this cell, otherwise it is marked -.

Alternative: write the program
(add the traceback and the output of the path)

```
Input: array diagonalCost (NxM)
allocate array DPTable (NxM)

algorithm getCheapestCost( )
        fillDPTable( )
        return DPTable [N] [M]

algorithm fillDPTable()
        DPTable [0][0]:=0
        for i from 1 to N:
                DPTable [i][0]:=i
        for j from 1 to M:
                DPTable [0][j]:=j
        for i from 1 to N:
                for j from 1 to M:
                        DPtable [i][j]:=min (DPtable [i-1][j-1]+ diagonalCost [i][j],
                                DPtable [i-1][j]+1, DPtable [i][j-1]+ 1)
```

Complexity of the DP algorithm

- 2 nested loops: O(NM)

Edit distance

String dissimilarity

Edit Operations

- We can transform the second string S2 into the first string S1 by applying a sequence of edit operations on S2 :
  - Deleting 1 symbol
  - Inserting 1 symbol
  - Replacing 1 symbol

S1	a	c	t			a	t	g
S2	a		t	a	c	a		g

Insert c

Delete a, c

Insert t

In total, 4 edit operations

String alignment

- An *alignment* of 2 strings is obtained by first inserting spaces (gaps), either into or at the end of both strings, and then placing the 2 resulting strings one above the other, so that every character or space in either string is opposite a single character or space in the other string

alignment

S1	a	c	t	-	-	a	t	g
S2	a	-	t	a	c	a	-	g

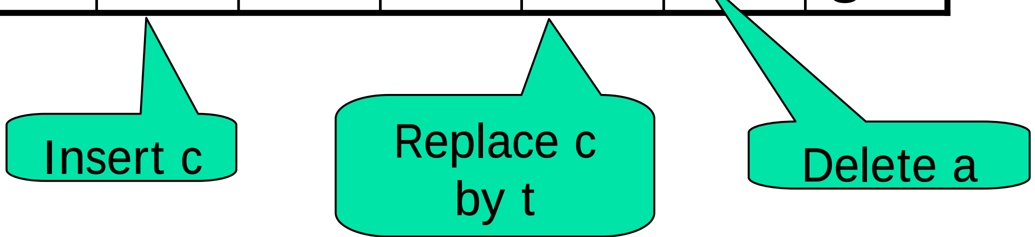
4 gaps,

no mismatches

Edit distance

- The *edit distance* between two strings is defined as the minimum number of edit operations needed to transform one string into another

S1	a	c	t	a	t		g
S2	a		t	a	c	a	g



In total, 3 edit operations

Optimal alignment

- An optimal alignment is obtained from the calculation of the edit distance

Optimal alignment

S1	a	c	t	a	t	-	g
S2	a	-	t	a	c	a	g

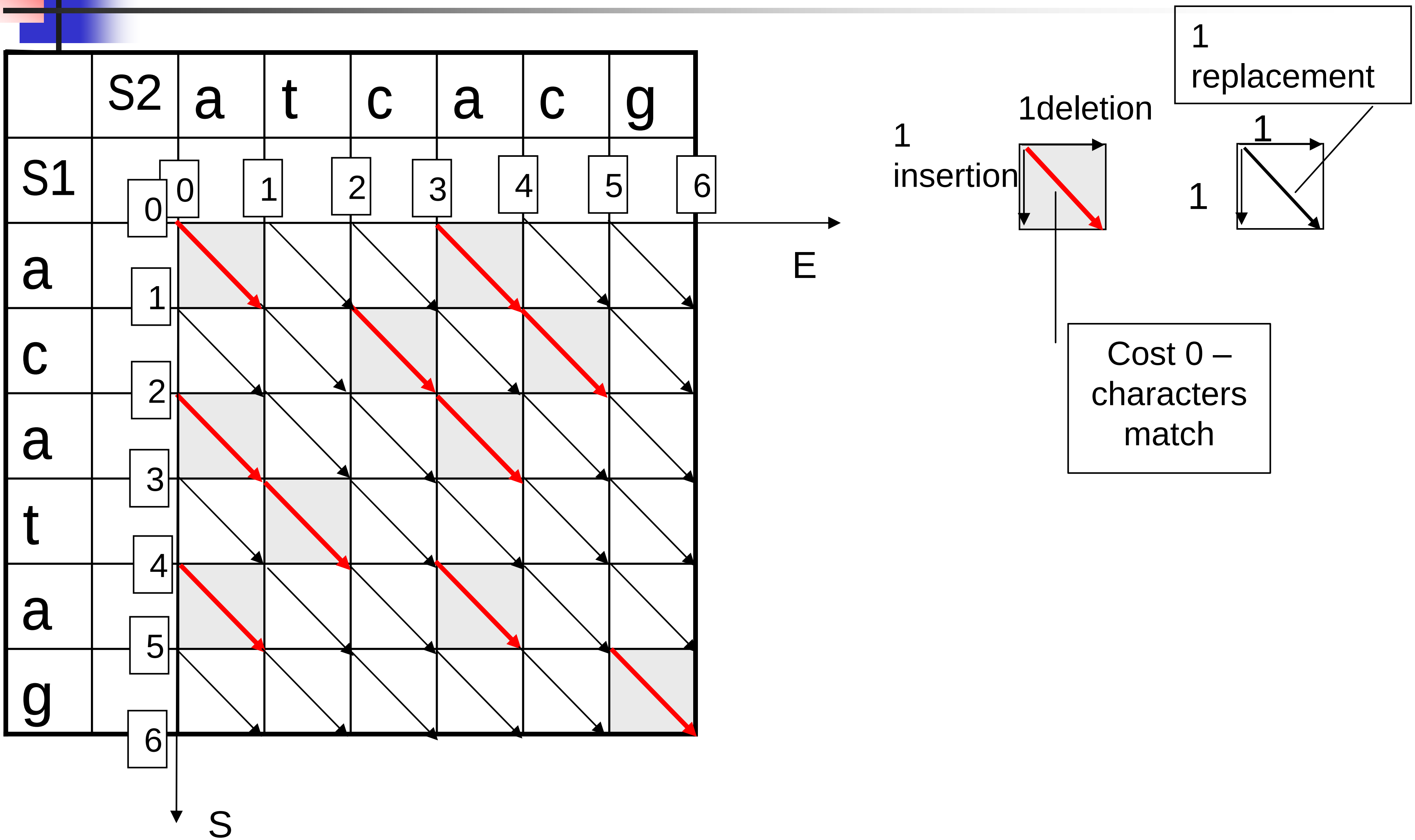
2 gaps,

1 mismatch

The edit distance problem

- Compute the edit distance between two strings along with a sequence of the operations which describe the transformation

Analogy with the cheapest path



The dynamic programming solution to the edit distance problem

- Trivially follows from the solution for the cheapest path:
  - If we moved strictly down in the grid, we inserted 1 symbol into S2
  - If we moved strictly to the right, we deleted 1 symbol from S2
  - If we moved by diagonal of cost 0, we matched the corresponding characters
  - If we moved by diagonal of cost 1, we replaced one symbol in S2 with the corresponding symbol in S1