Sequence alignment

Correspondence between bases of two DNA sequences, or between amino acids of two protein sequences

Alignment : 2 x k matrix ($k \geq m, n$)

V = ACCTGGTAAA $n = 10$

W = ACTGCGTATA $m = 10$

8	matches
1	mismatches
1	deletions
1	insertions

V	A	C	C	T	G	—	G	T	A	A	A
W	A	C	—	T	G	C	G	T	A	T	A

“Goodness” of alignments

Given two sequences, there are many possible alignments

```
ATTTTCCC
ATTTACGC
```

distance=2

```
ATTT-TCCC
ATTTA-CGC
```

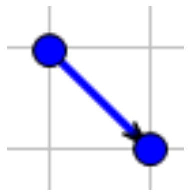
distance=3

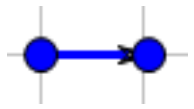
```
ATTTTCCC--------
--------ATTTACGC
```

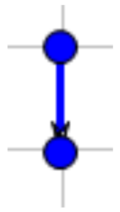
distance=16

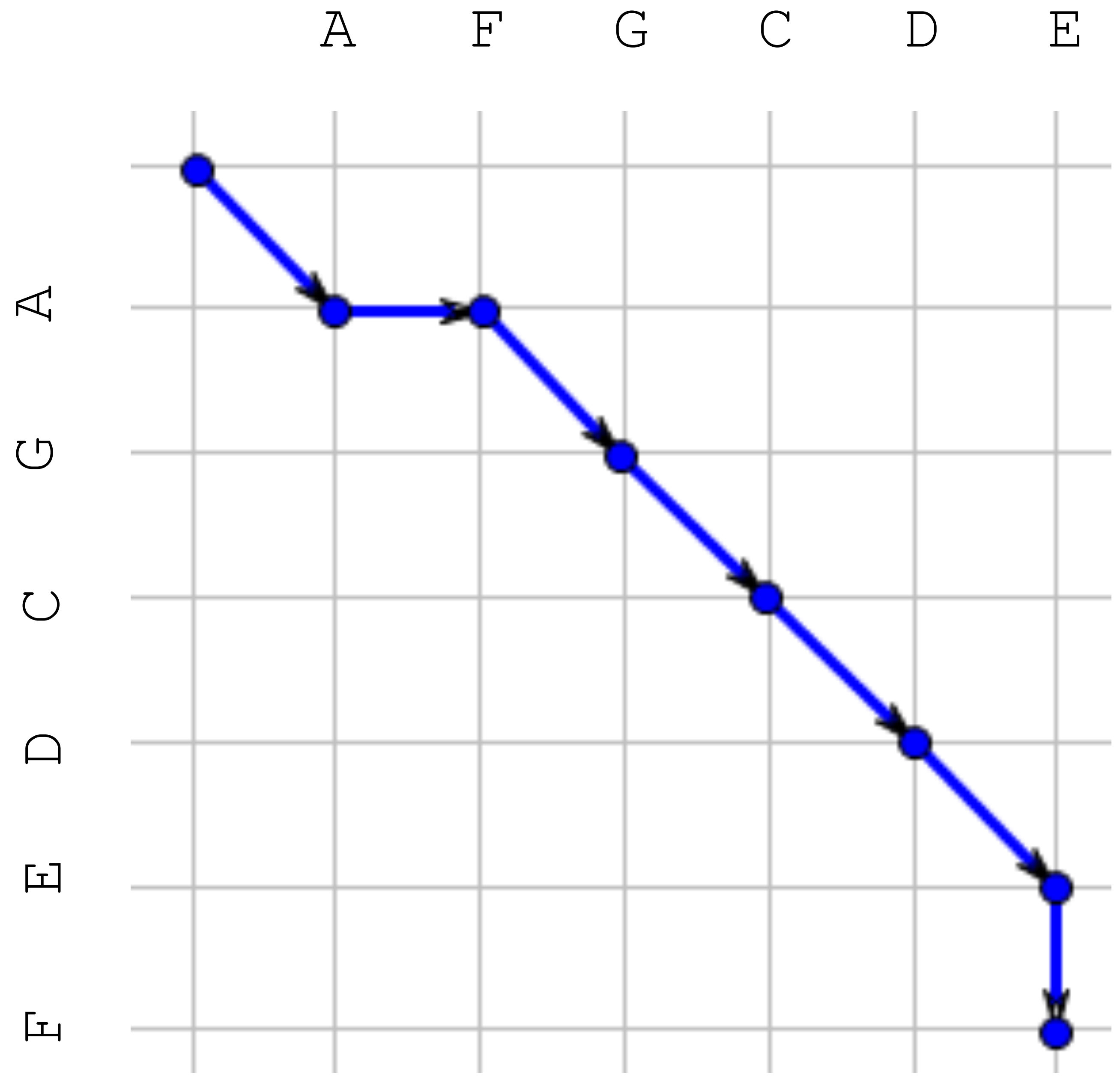
Edit distance: the total number of substitutions, insertions and deletions needed to transform one sequence to another

Enumeration of all possible alignments

Match 

Insertion_X 

Insertion_Y 

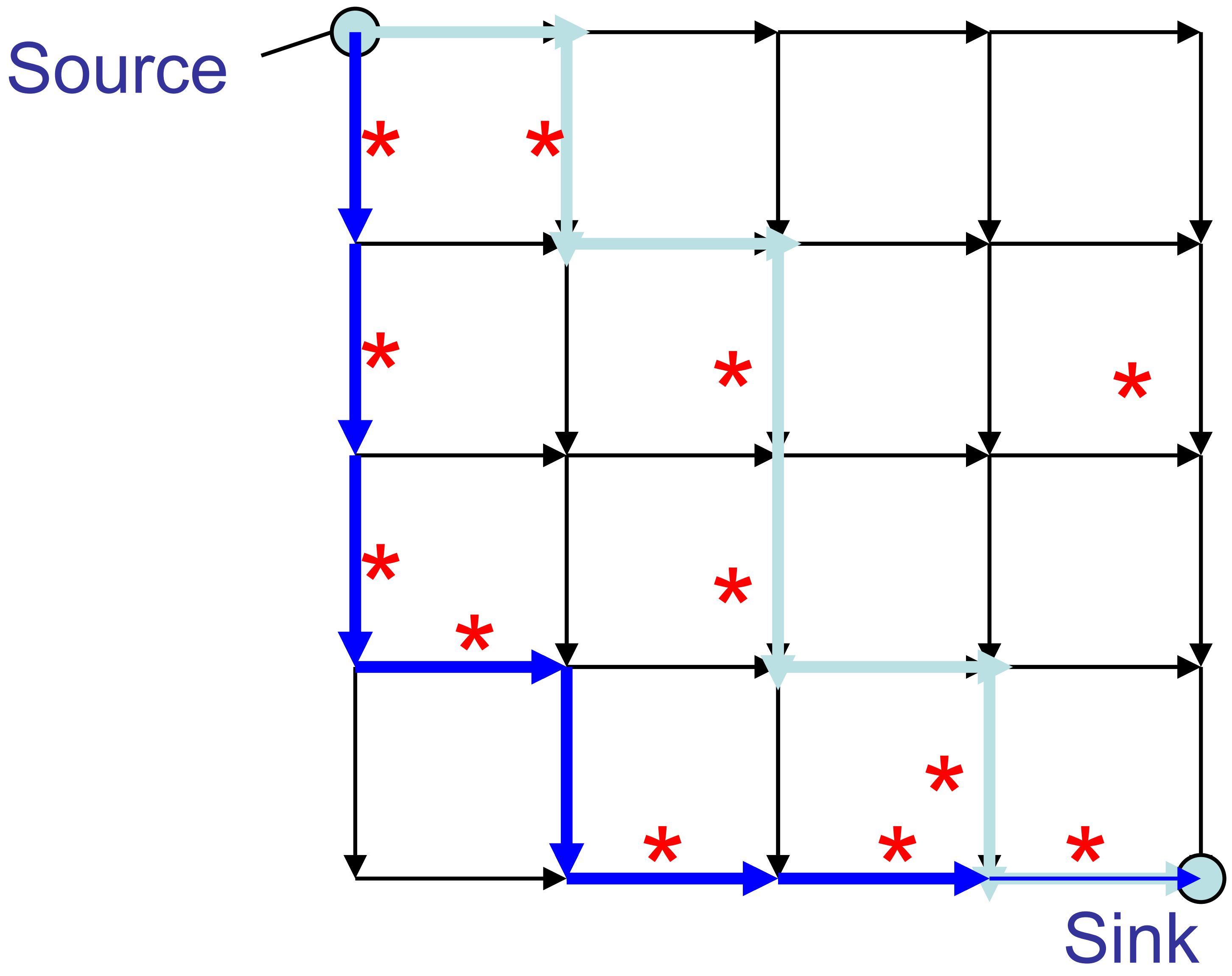


A-GCDEF
AFGCDE-

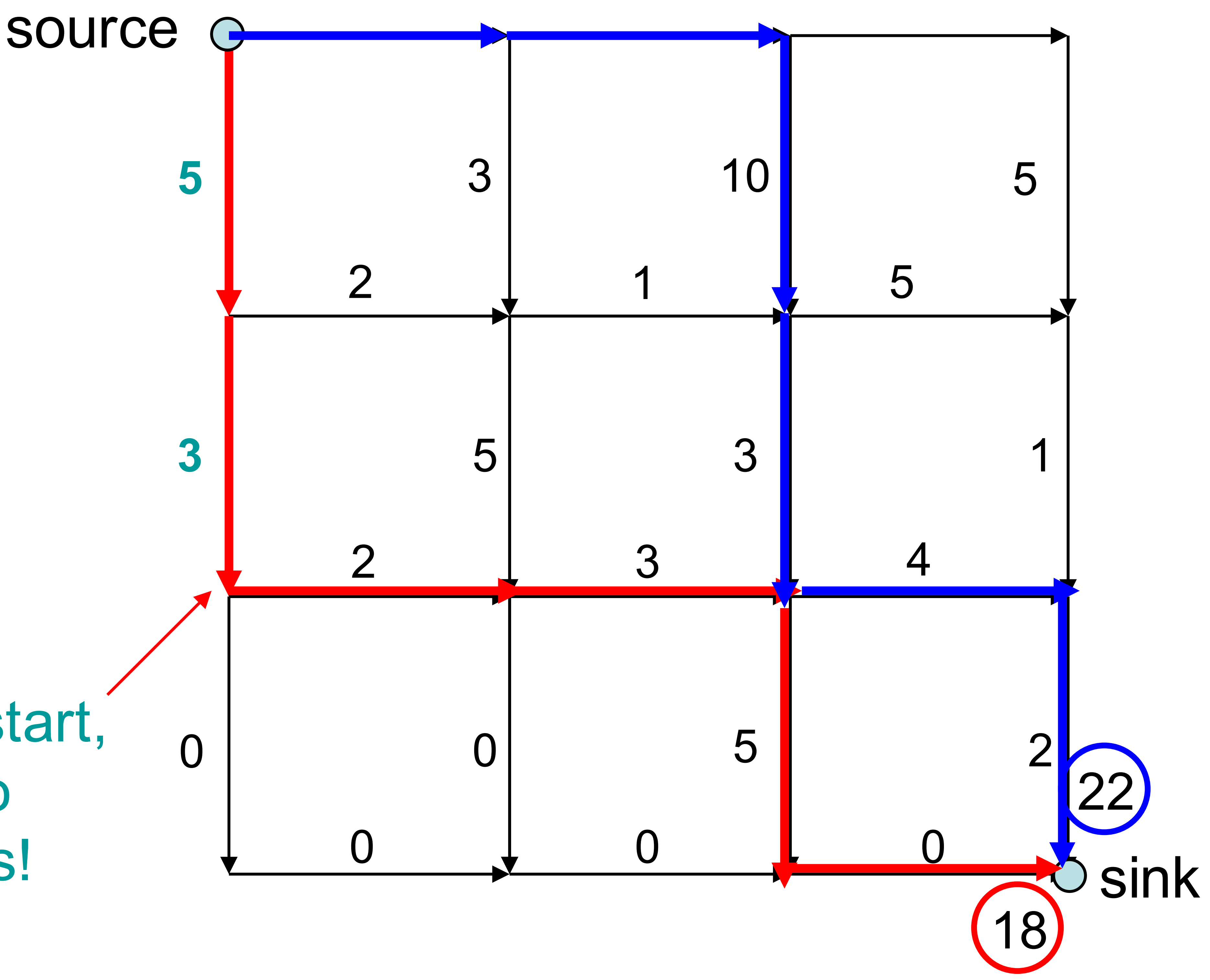
Very expensive

Manhattan tourist problem

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid



Would a greedy algorithm work?



promising start, but leads to bad choices!

How about a recursive solution?

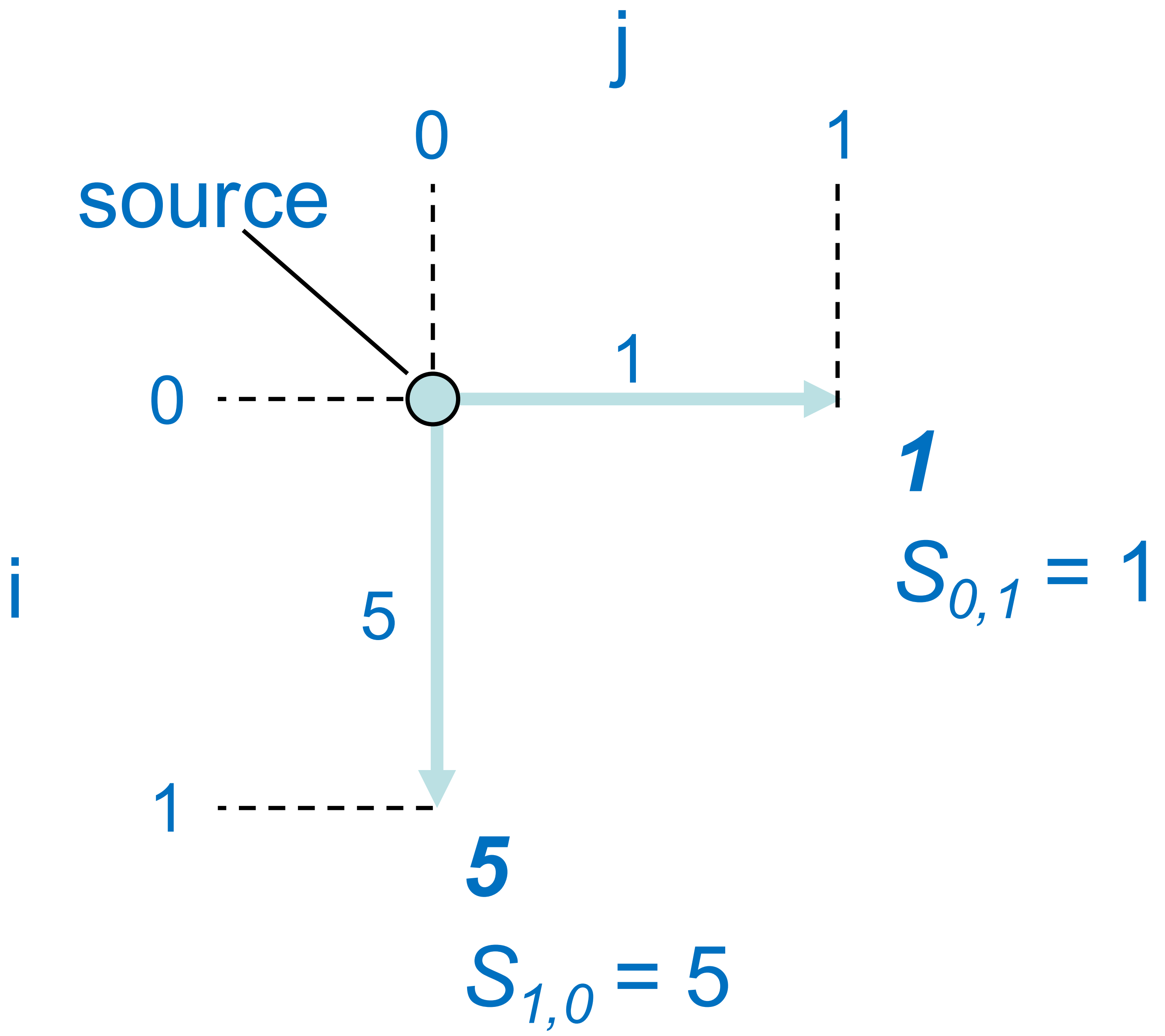
```
Function MT(n,m)

1.  x = MT(n-1,m)+
            weight of the edge from (n-1,m) to (n,m)
2.  y = MT(n,m-1)+
            weight of the edge from (n,m-1) to (n,m)
3.  return max{x,y}
```

MT(x, y) returns the “most weighted” path from point (x, y) to the “sink”.

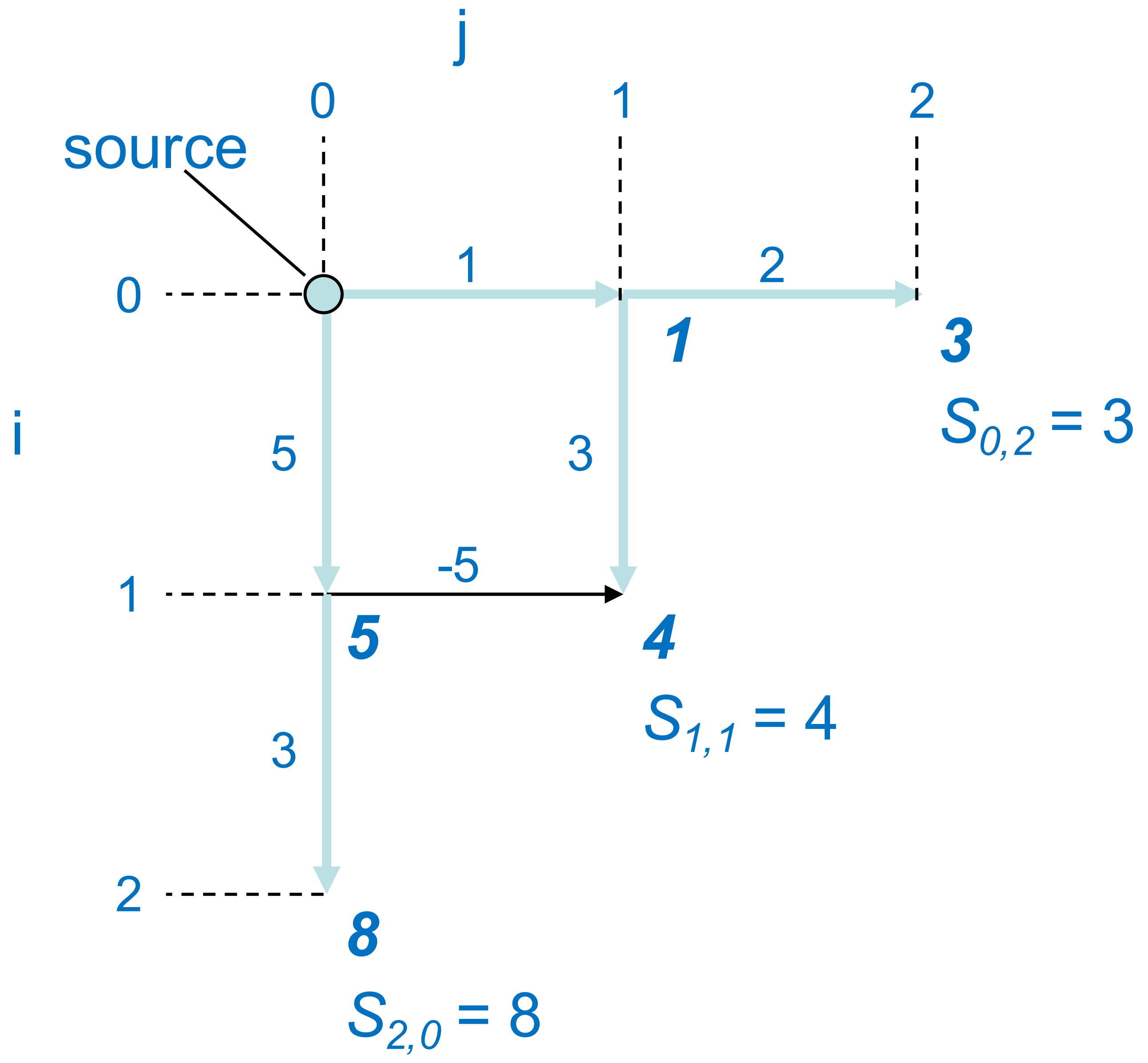
Why this is not efficient?

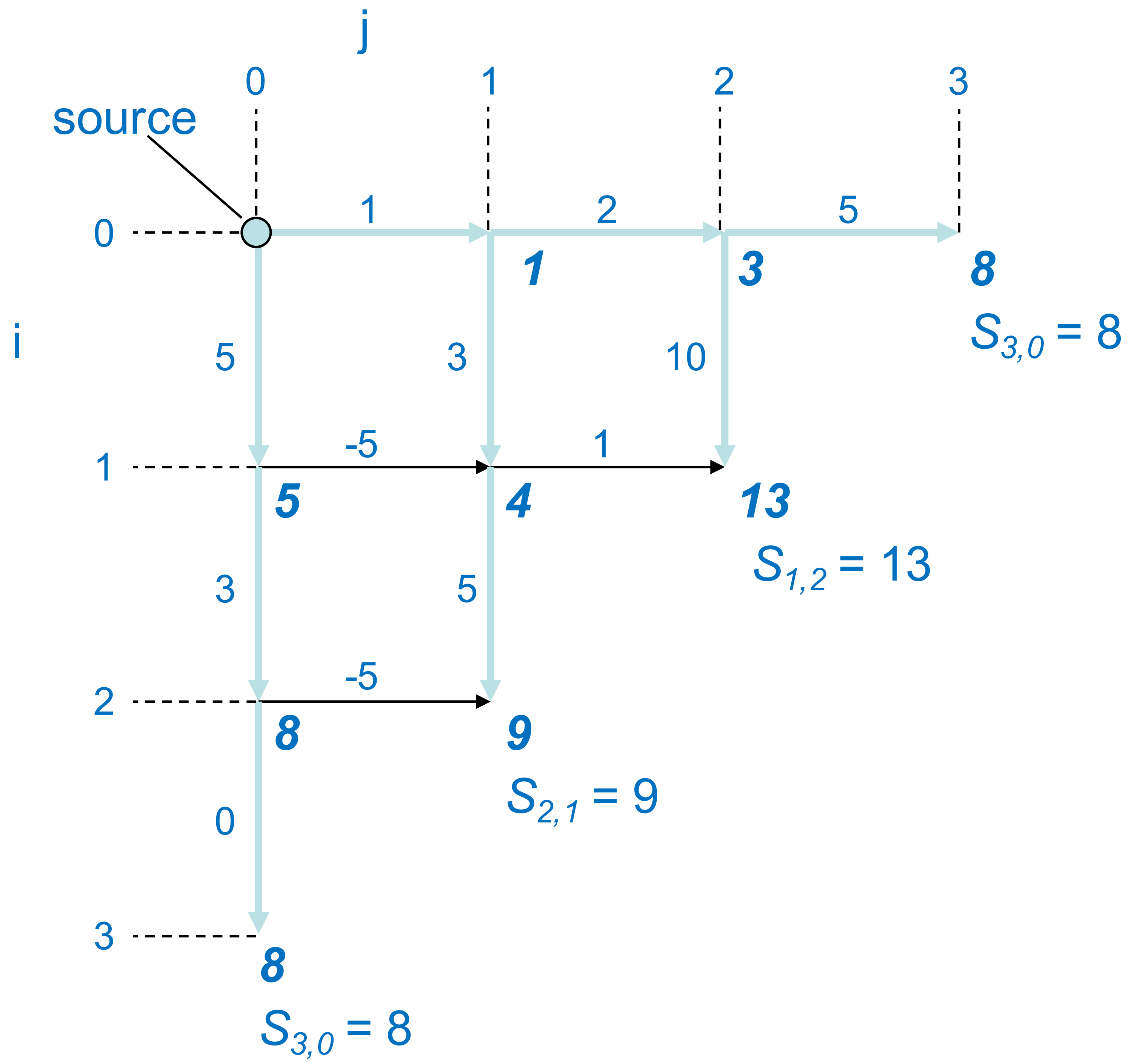
- MT(n,m) needs MT(n, m-1) and MT(n-1, m)
- Both of these need MT(n-1, m-1)
- So MT(n-1, m-1) will be computed at least twice
- Dynamic programming: the same idea as this recursive algorithm, but keep all intermediate results in a table and reuse

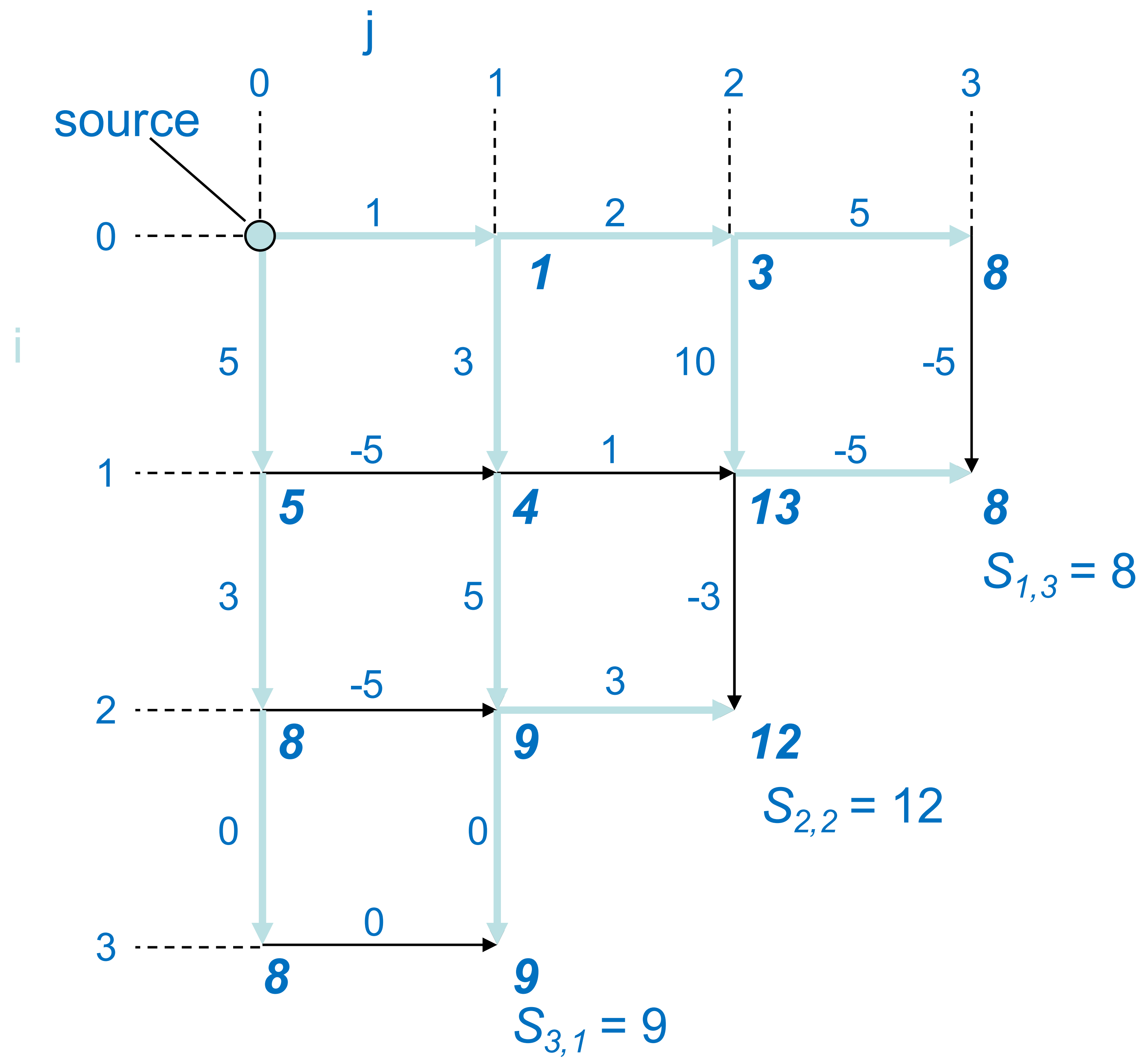
How to avoid redundant calculations

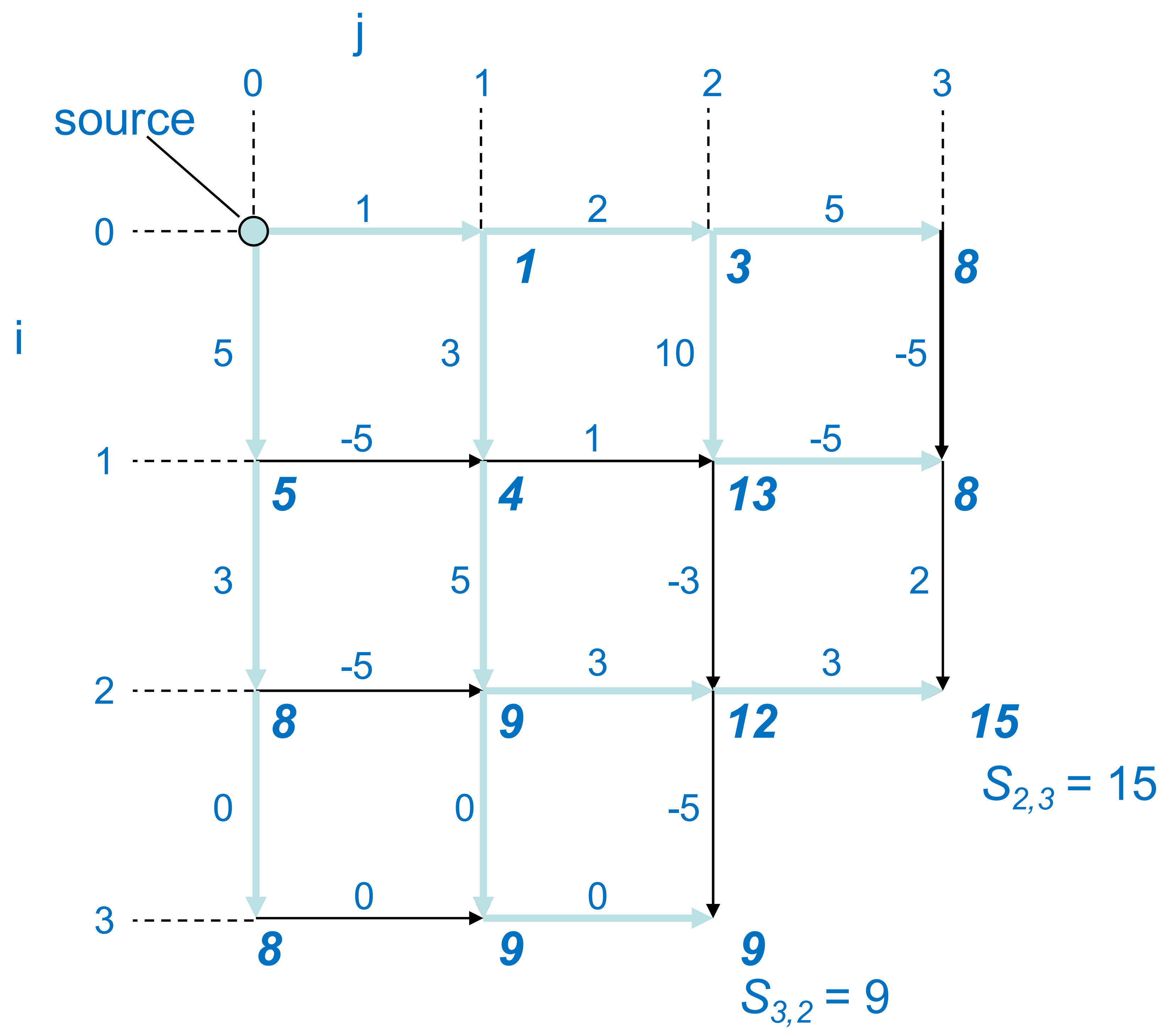


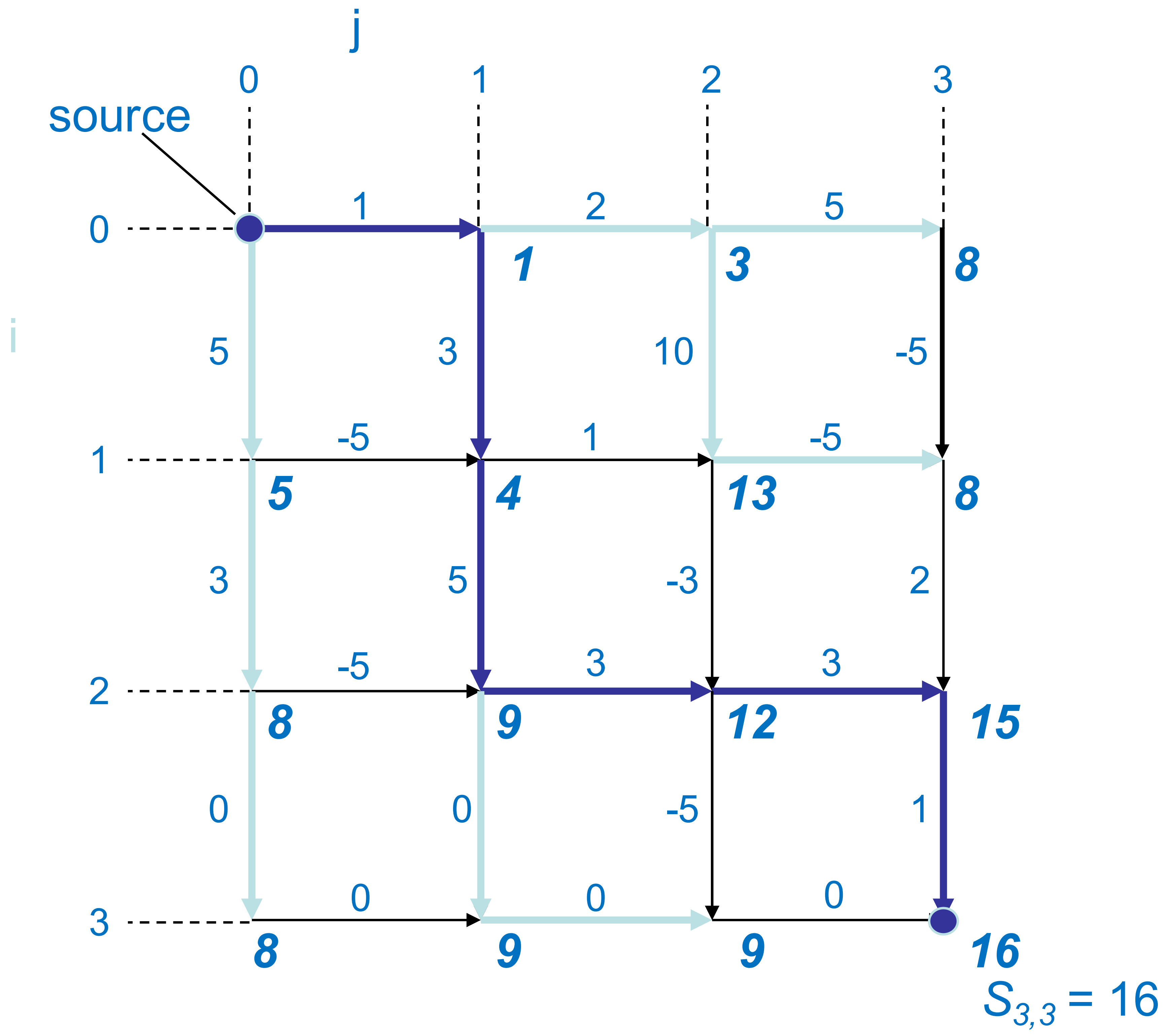
- Calculate optimal path score for each vertex in the graph
- Each vertex's score is the maximum of the prior vertices score plus the weight of the respective edge in between


j
0
1
2
source
0
1
2
i
1
2
5
3
3
-5
1
3
5
4
8
$S_{0,2} = 3$
$S_{1,1} = 4$
$S_{2,0} = 8$


j
0
1
2
3
source
i
0
1
2
3
1
2
5
5
3
10
-5
1
3
5
-5
0
1
3
8
$S_{3,0}$ = 8
5
4
13
$S_{1,2}$ = 13
8
9
$S_{2,1}$ = 9
8
$S_{3,0}$ = 8


j
0
1
2
3
source
i
0
1
2
3
1
2
5
5
3
10
-5
-5
1
-5
3
5
-3
-5
3
0
0
0
1
3
8
5
4
13
8
8
9
12
8
9
$S_{1,3} = 8$
$S_{2,2} = 12$
$S_{3,1} = 9$


j
0
1
2
3
source
i
0
1
2
3
1
2
5
5
3
10
-5
-5
1
-5
3
5
-3
2
-5
3
3
0
0
-5
0
0
1
3
8
5
4
13
8
8
9
12
15
8
9
9
$S_{2,3} = 15$
$S_{3,2} = 9$

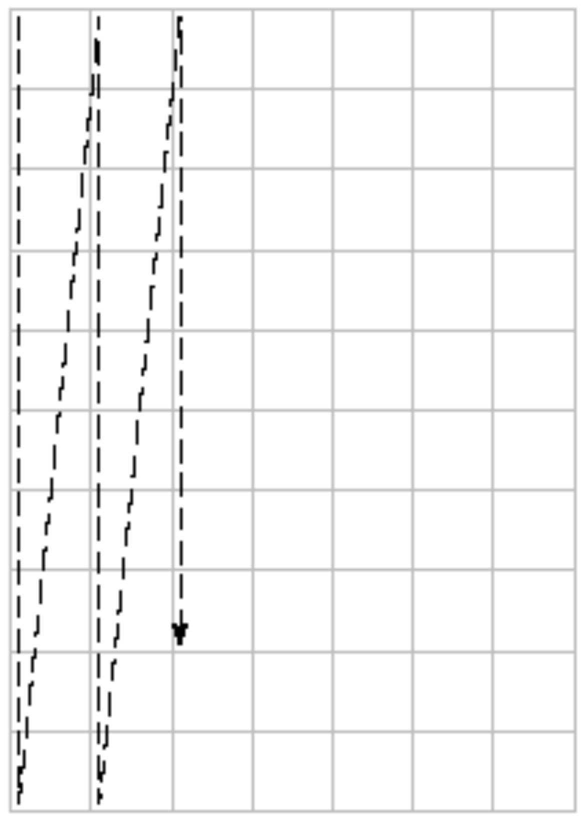
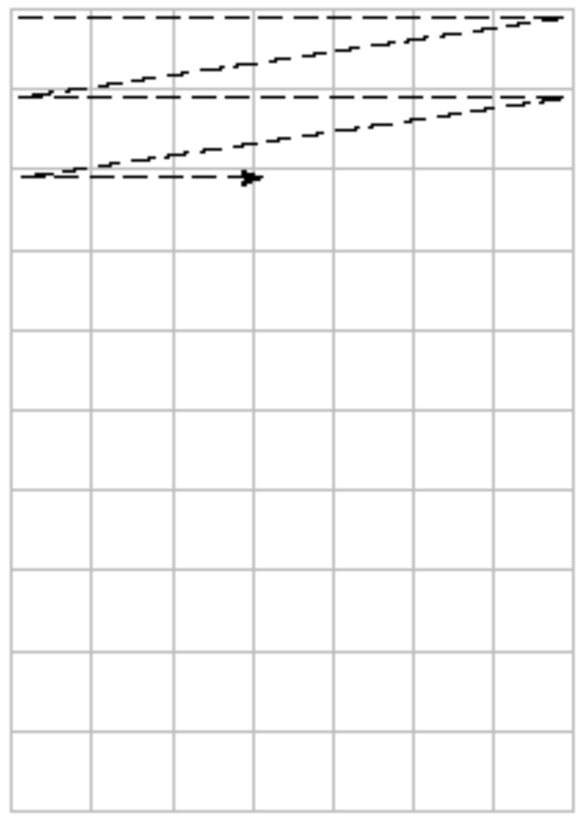
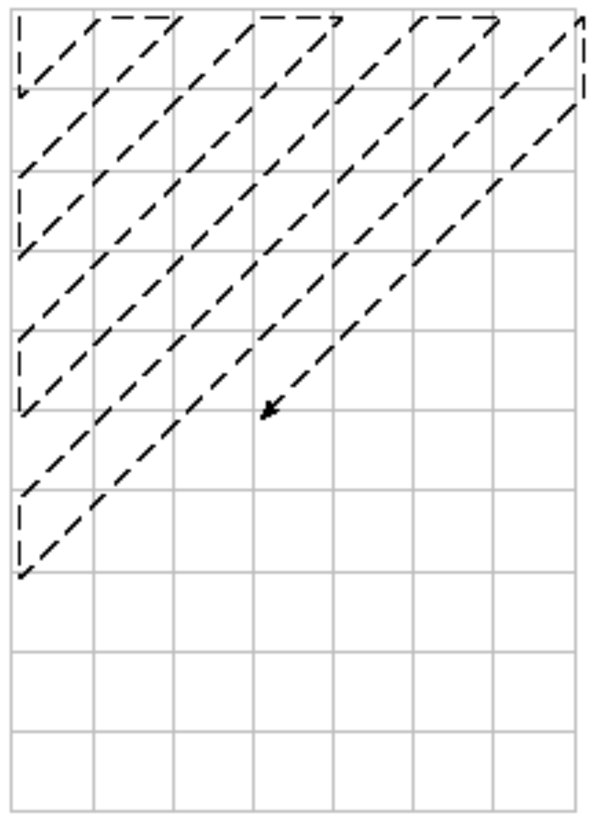

j
0
1
2
3
source
i
0
1
2
3
1
2
5
5
3
10
-5
-5
1
-5
3
5
-3
2
-5
3
3
0
0
-5
1
0
0
0
1
3
8
5
4
13
8
8
9
12
15
8
9
9
16
$S_{3,3} = 16$

To ensure the correctness

- By the time the vertex x is analyzed, the values s_y for all its predecessors y should be computed – otherwise we are in trouble.

- We need to traverse the vertices in some order

- For a grid, can traverse vertices row by row, column by column, or diagonal by diagonal

3 different strategies:

a) Column by column

b) Row by row

c) Along diagonals

Pseudocode?

Runtime?

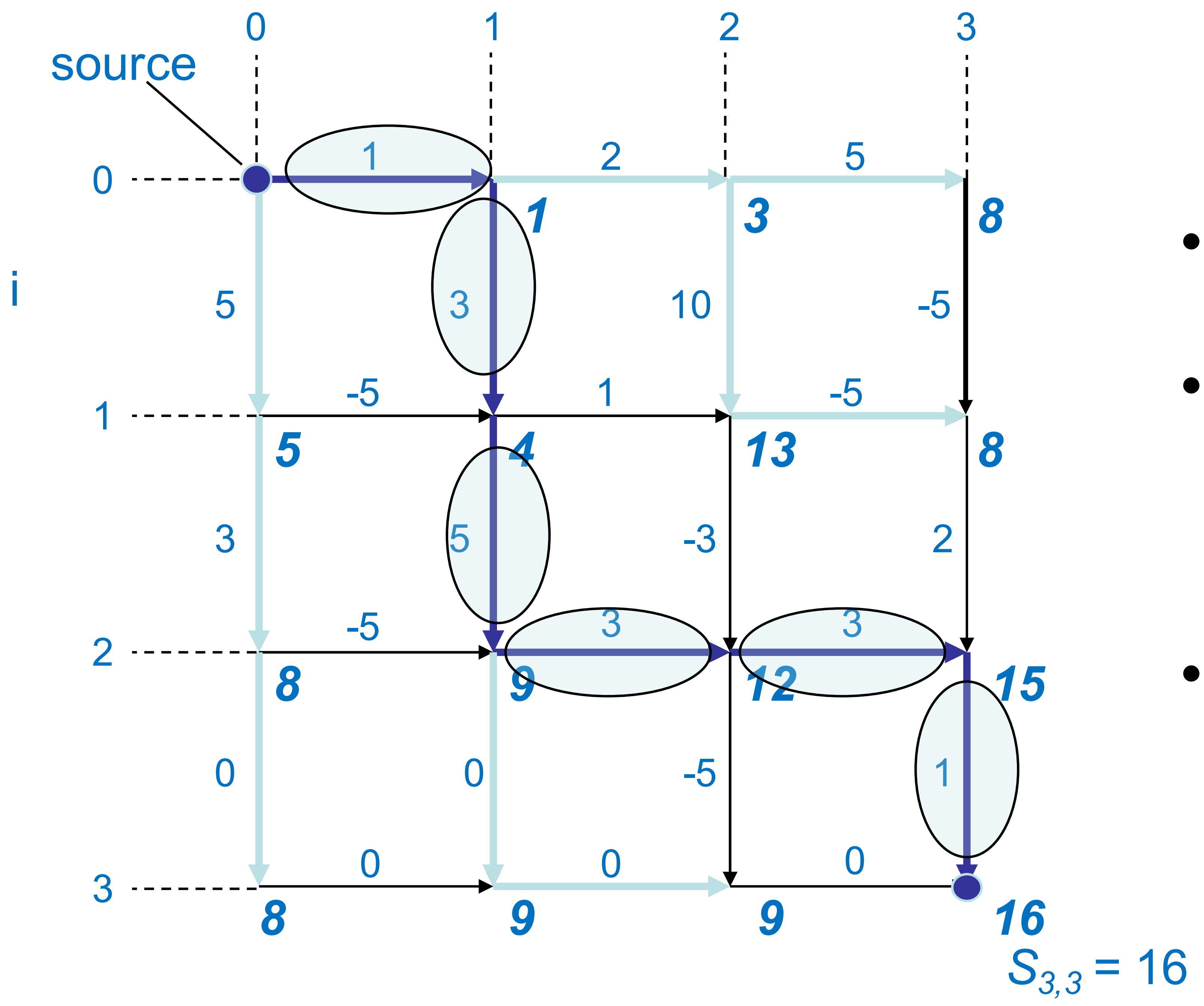
Recursive algorithm -> Dynamic programming ?

Function **MT**(*n,m*)

```
1.  x = MT(n-1,m)+
                weight of the edge from (n-1,m) to (n,m)
2.  y = MT(n,m-1)+
                weight of the edge from (n,m-1) to (n,m)
3.  return max{x,y}
```

MT(x, y) returns the “most weighted” path from point (x, y) to the “sink”.

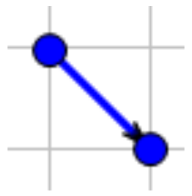
How to find the optimal path



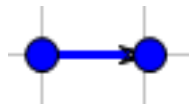
- Start from Sink.
- Find which of the two edges gave the "max". Take it.
- Repeat.

The edit distance problem

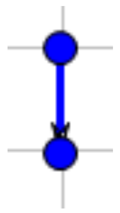
Match

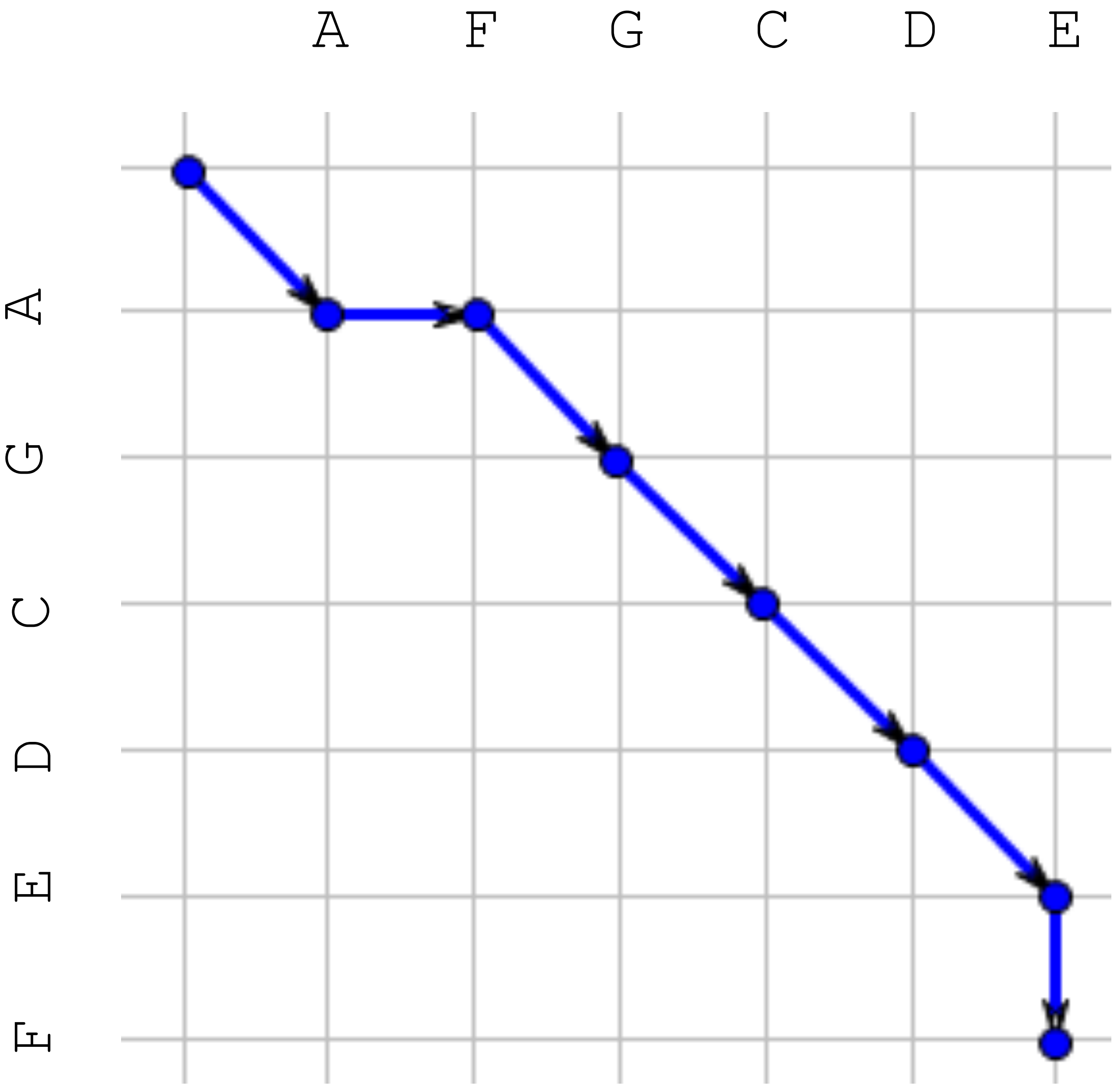


Insertion_X



Insertion_Y





A-GCDEF
AFGCDE-

Recipe

1. Identify subproblems

2. Write down recursions

3. Make it dynamic-programming!