Dynamic Programming: Sequence alignment

CS 466

Saurabh Sinha

DNA Sequence Comparison: First Success Story

- Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene’s function
- In 1984 Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene
- A normal growth gene switched on at the wrong time causes cancer !

Cystic Fibrosis

- **Cystic fibrosis** (CF) is a chronic and frequently fatal genetic disease of the body's mucus glands. CF primarily affects the respiratory systems in children.
- Search for the CF gene was narrowed to ~1 Mbp, and the region was sequenced.
- Scanned a database for matches to known genes. A segment in this region matched the gene for some ATP binding protein(s). These proteins are part of the ion transport channel, and CF involves sweat secretions with abnormal sodium content!

Role for Bioinformatics

- Gene similarities between two genes with known and unknown function alert biologists to some possibilities

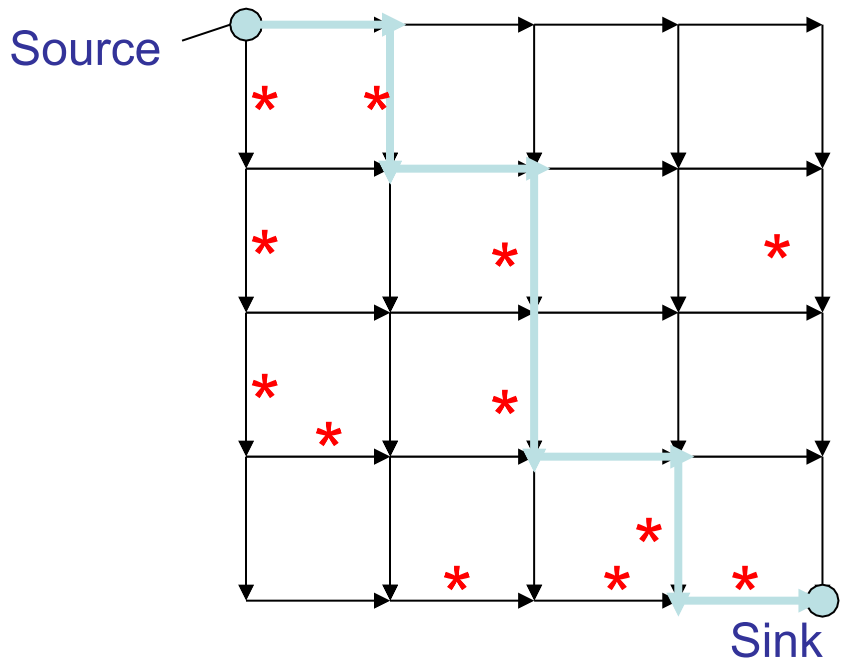
- Computing a similarity score between two genes tells how likely it is that they have similar functions

- Dynamic programming is a technique for revealing similarities between genes

Motivating Dynamic Programming

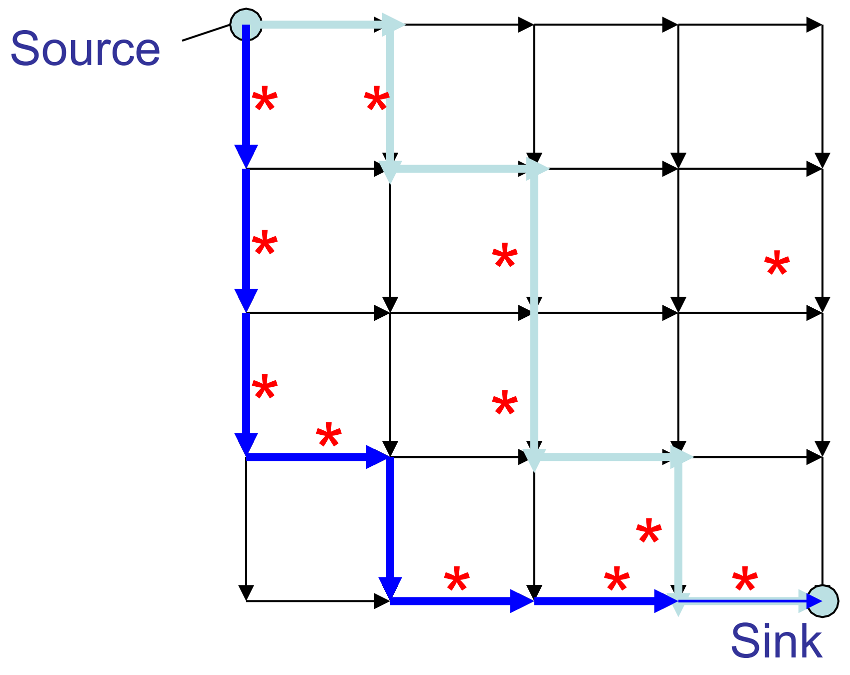
Dynamic programming example: Manhattan Tourist Problem

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



Dynamic programming example: 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



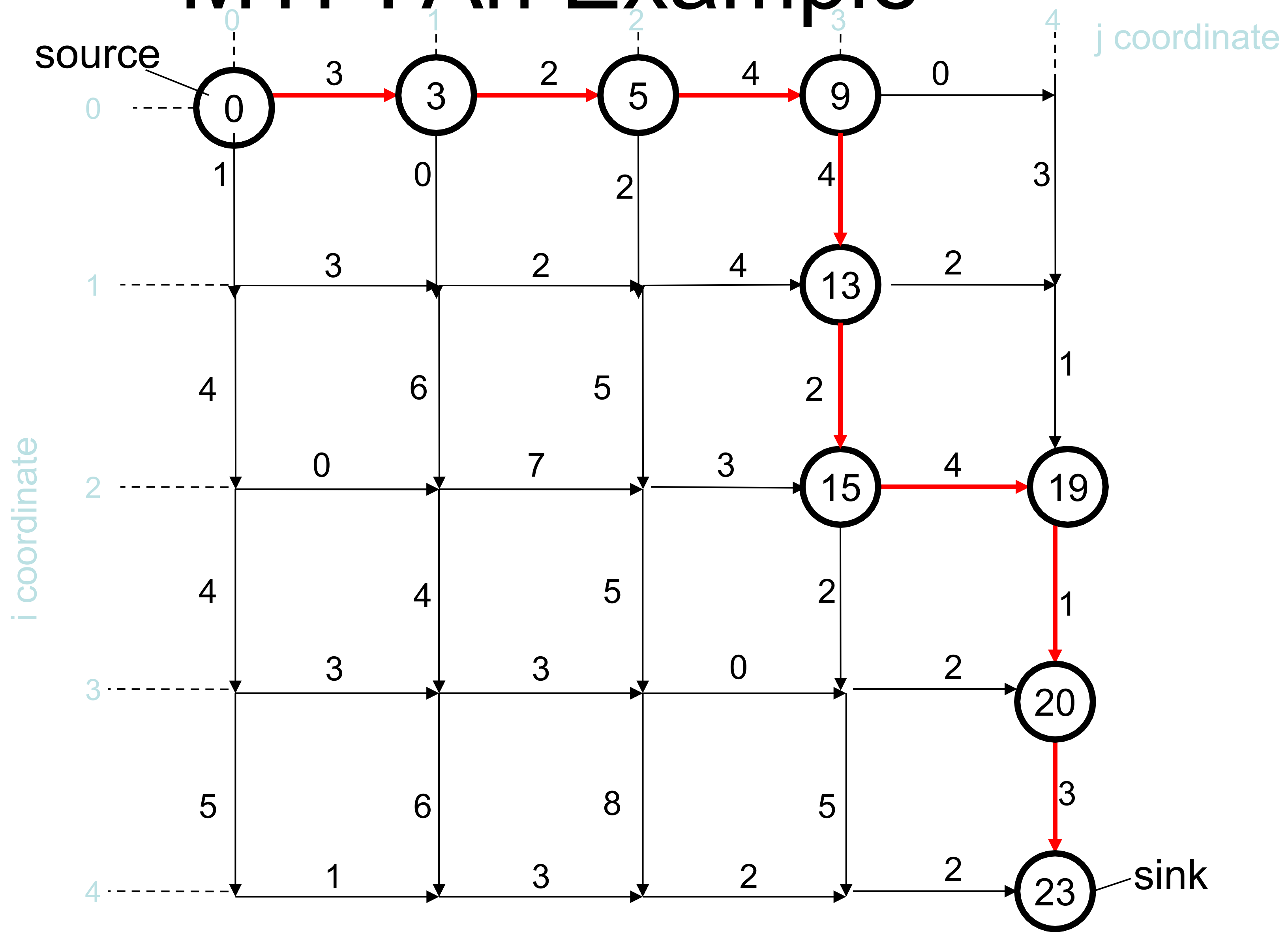
Manhattan Tourist Problem: Formulation

Goal: Find the longest path in a weighted grid.

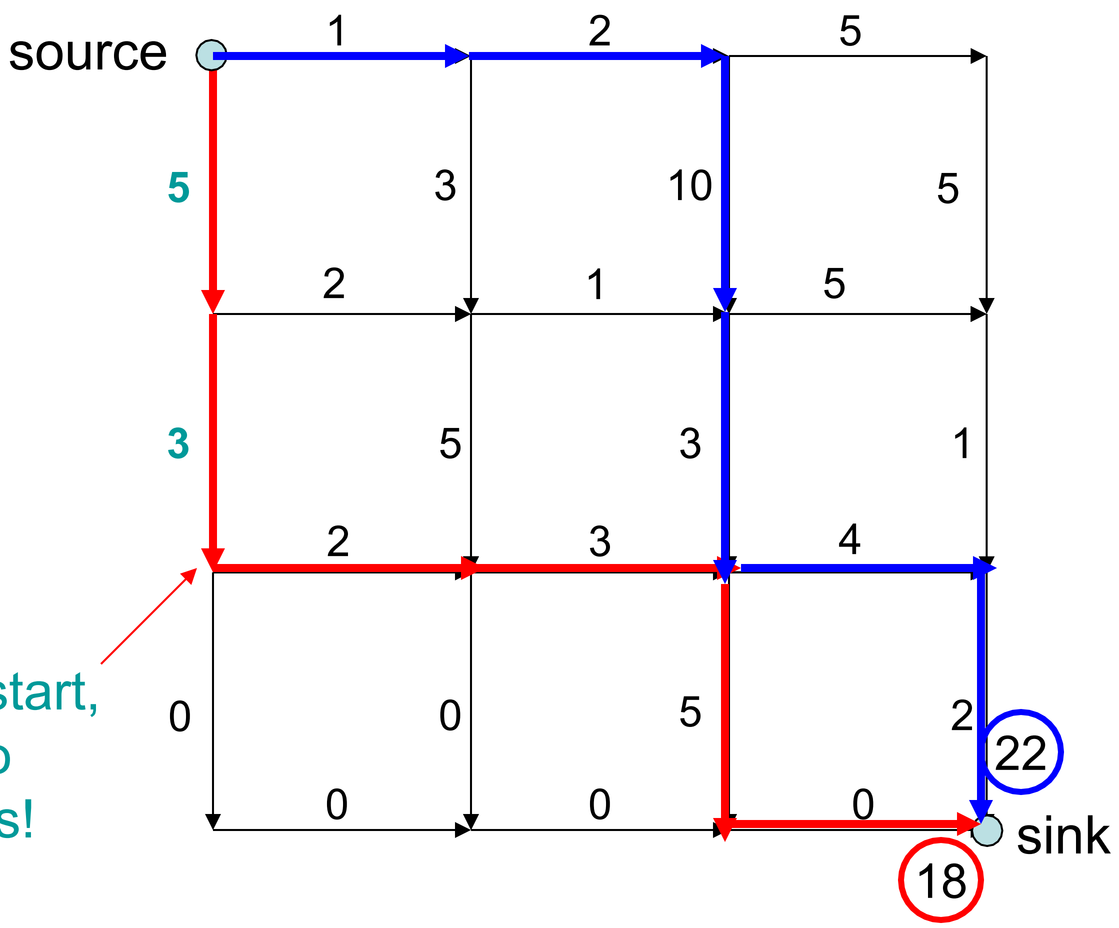
Input: A weighted grid ***G*** with two distinct vertices, one labeled “*source*” and the other labeled “*sink*”

Output: A longest path in ***G*** from “*source*” to “*sink*”

MTP: An Example



MTP: Greedy Algorithm Is Not Optimal



promising start,
but leads to
bad choices!

MTP: Simple Recursive Program

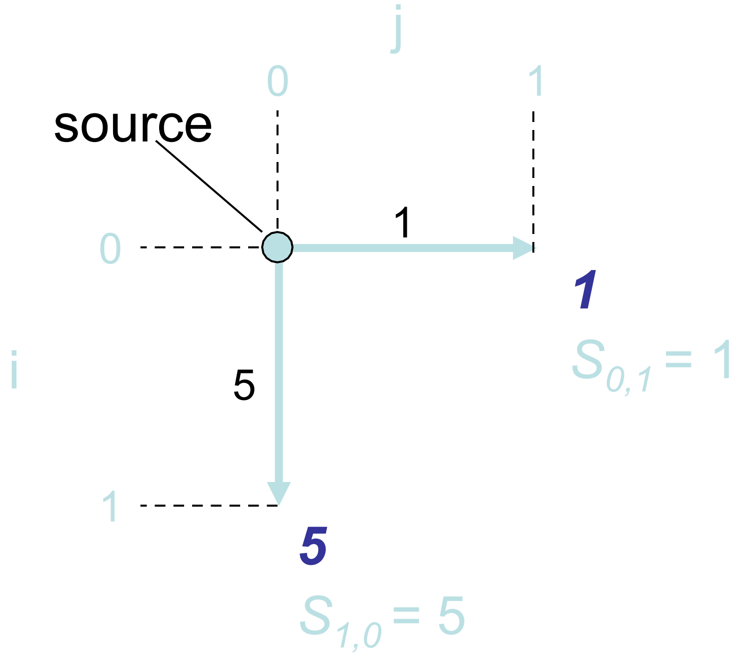
```
MT(n,m)
  if n=0 or m=0
    return MT(n,m)
  x ← MT(n-1,m)+
        length of the edge from (n- 1,m) to (n,m)
  y ← MT(n,m-1)+
        length of the edge from (n,m-1) to (n,m)
  return max(x,y)
```

What's wrong with this approach?

Here’s what’s wrong

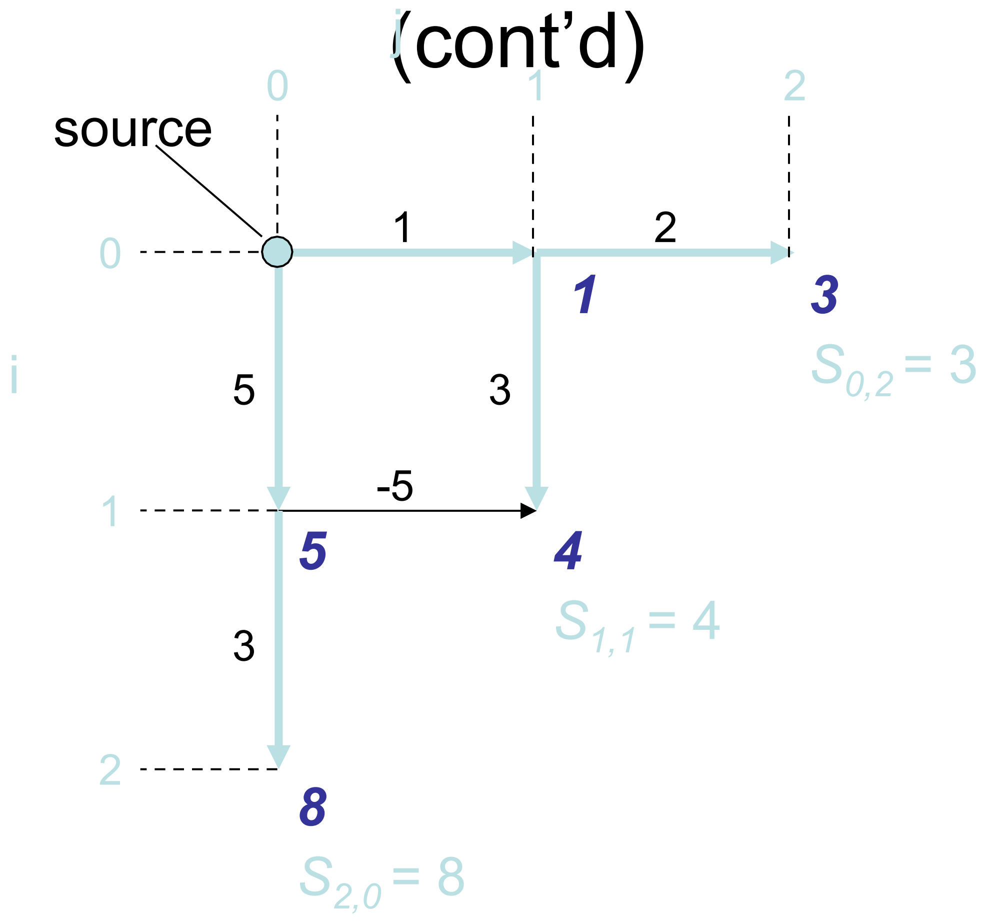
- M(n,m) needs M(n, m-1) and M(n-1, m)
- Both of these need M(n-1, m-1)
- So M(n-1, m-1) will be computed at least twice
- Dynamic programming: the same idea as this recursive algorithm, but keep all intermediate results in a table and reuse

MTP: Dynamic Programming

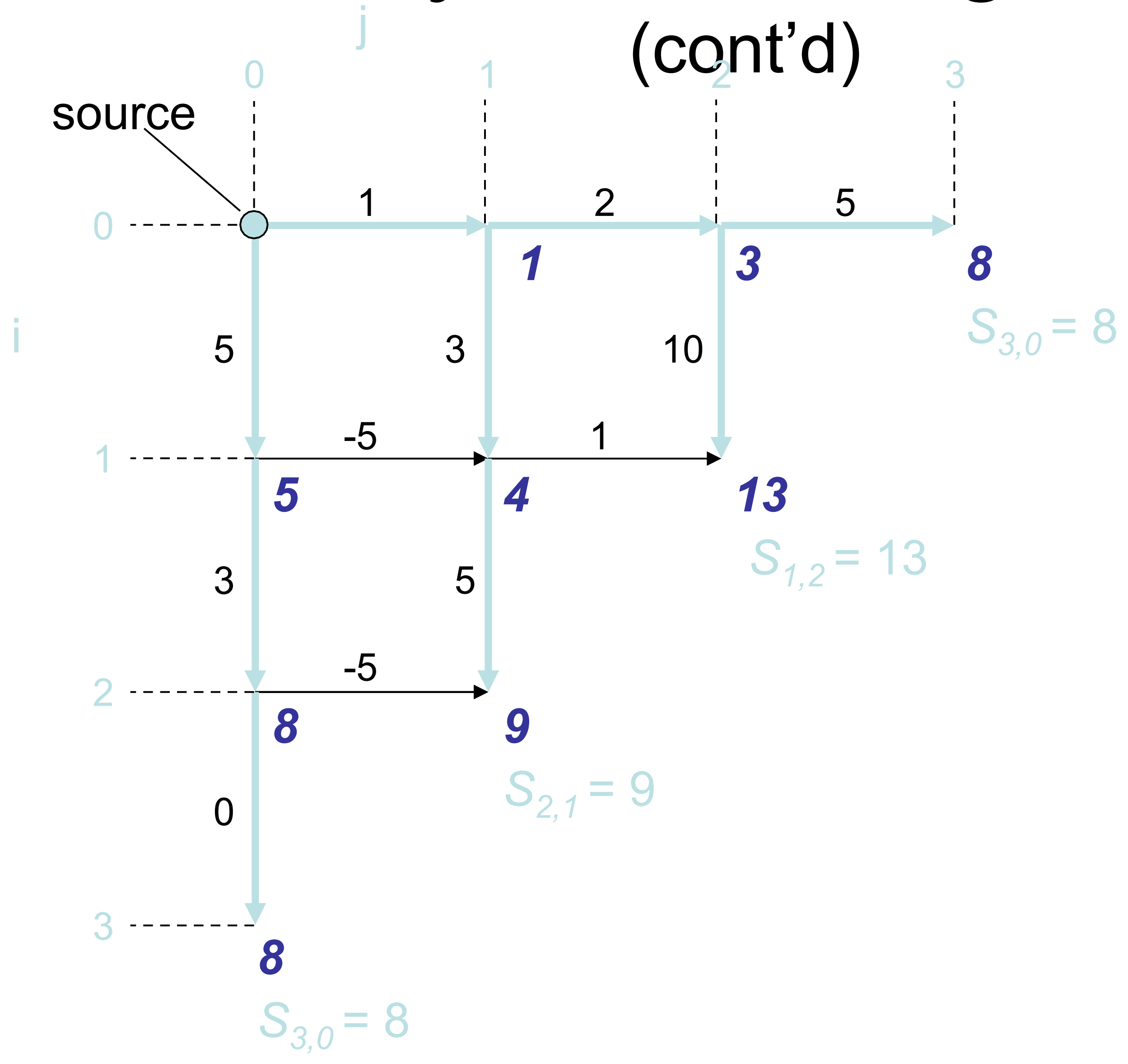


- Calculate optimal path score for each vertex in the graph
- Each vertex's score is the maximum of the prior vertices score plus the weight of the respective edge in between

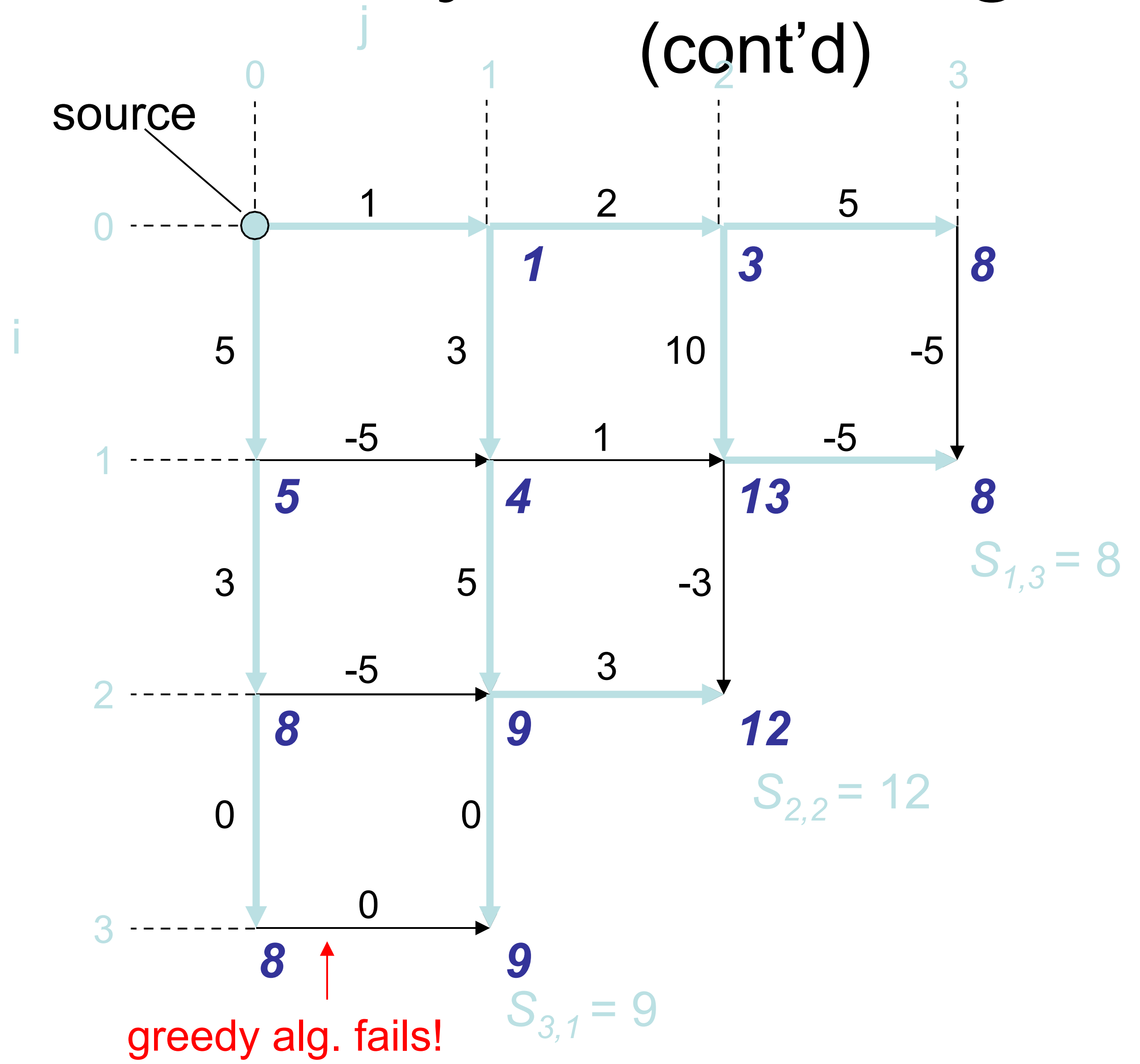
MTP: Dynamic Programming (cont'd)



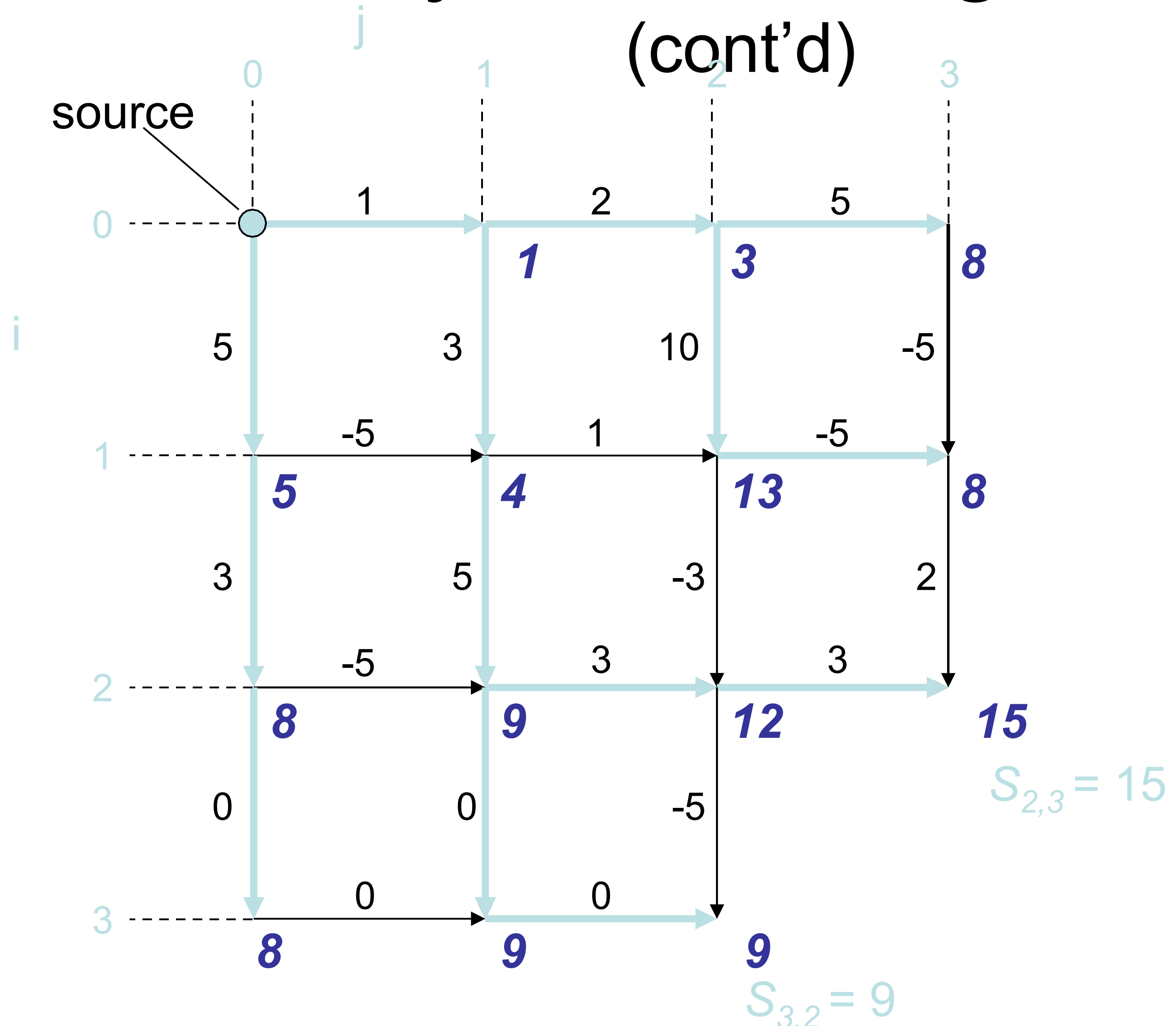
MTP: Dynamic Programming (cont'd)



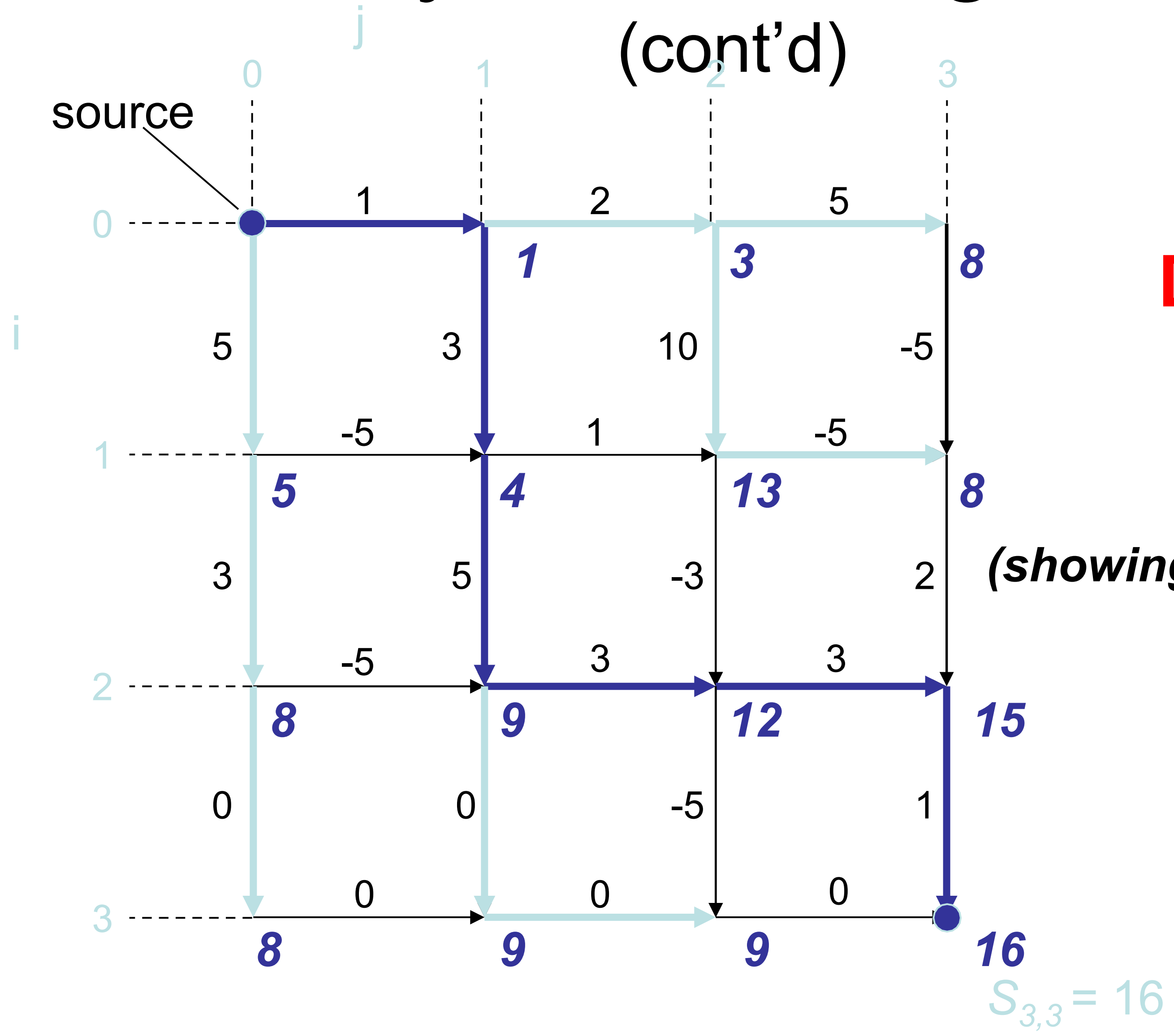
MTP: Dynamic Programming (cont'd)



MTP: Dynamic Programming (cont'd)



MTP: Dynamic Programming (cont'd)



Done!

(showing all back-traces)

MTP: Recurrence

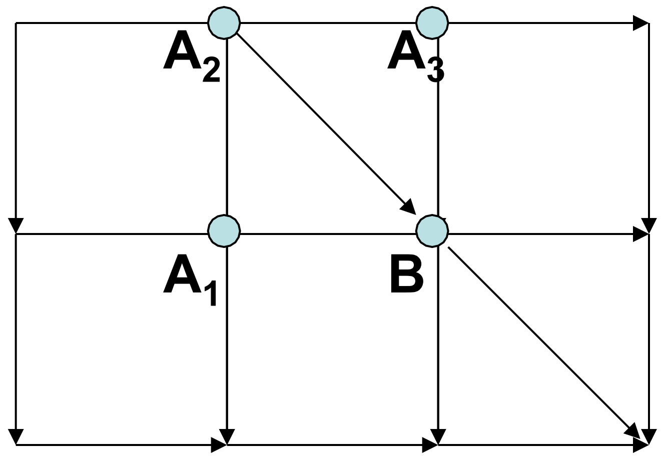
Computing the score for a point *(i,j)* by the recurrence relation:

$$s_{i,j} = \textbf{max} \begin{cases} s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\ s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \end{cases}$$

The running time is ***n* x *m*** for a ***n*** by ***m*** grid

(***n*** = # of rows, ***m*** = # of columns)

Manhattan Is Not A Perfect Grid



What about diagonals?

- The score at point B is given by:

$$s_B = \max \text{ of } \begin{cases} s_{A1} + \text{weight of the edge } (A_1, B) \\ s_{A2} + \text{weight of the edge } (A_2, B) \\ s_{A3} + \text{weight of the edge } (A_3, B) \end{cases}$$

Manhattan Is Not A Perfect Grid (cont'd)

Computing the score for point ***x*** is given by the recurrence relation:

$$s_x = \max_{\text{of}} \left\{ \begin{array}{l} s_y + \text{weight of vertex } (y, x) \text{ where} \\ \qquad\qquad\qquad y \in \text{Predecessors}(x) \end{array} \right.$$

- Predecessors (*x*) – set of vertices that have edges leading to *x*

- The running time for a graph G(***V***, ***E***) (***V*** is the set of all vertices and ***E*** is the set of all edges) is O(***E***) since each edge is evaluated once

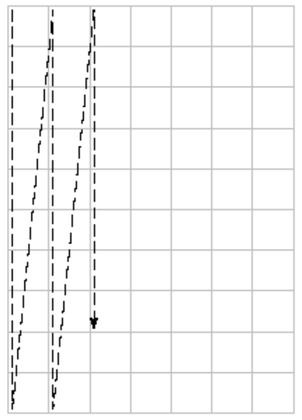
Traveling in the Grid

- 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

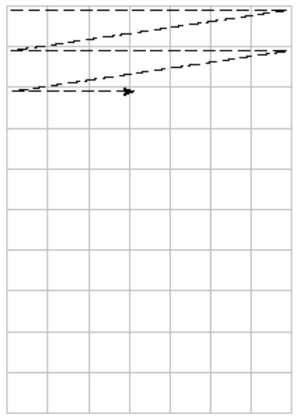
Traversing the Manhattan Grid

- 3 different strategies:
  - a) Column by column
  - b) Row by row
  - c) Along diagonals

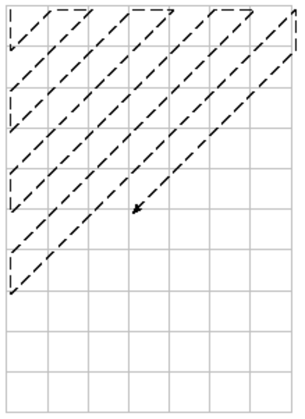
a)



b)



c)



Traversing a DAG

- A numbering of vertices of the graph is called ***topological ordering*** of the DAG if every edge of the DAG connects a vertex with a smaller label to a vertex with a larger label
- How to obtain a topological ordering ?