Lecture 8: Dynamic Programming Preliminaries

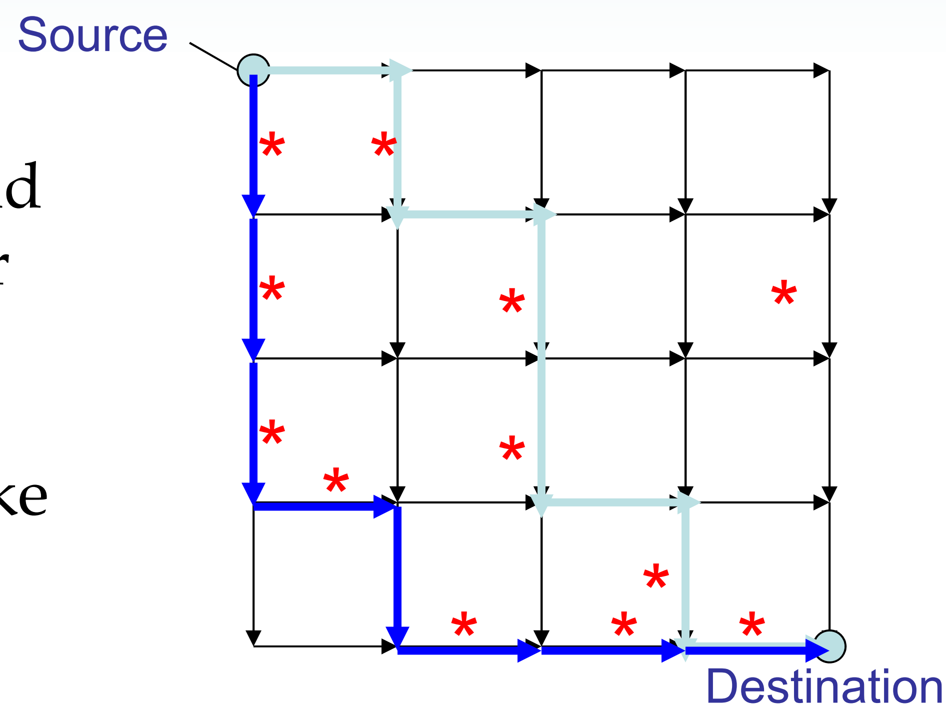
Study Chapter 6.1-6.3

Dynamic Programming

- ***Dynamic Programming*** is a technique for computing recurrence relations efficiently by storing partial or intermediate results

- Three keys to constructing a dynamic programming solution:
  1. Formulate the answer as a recurrence relation
  2. Consider all instances of the recurrence at each step
  3. Order evaluations so you will always have precomputed the needed partial results

Manhattan Tourist Problem (MTP)



Imagine seeking a path from source to destination in a Manhattan-like city grid that maximizes the number of attractions (*) passed. With the following caveat– at every step you must make progress towards the goal.

We treat the city map as a graph, with "vertices" at each corner, and weighted edges along each block. The weights are the number of attractions along each block.

Manhattan Tourist Problem: Formulation

Goal: Find the maximum-weight path in a grid.

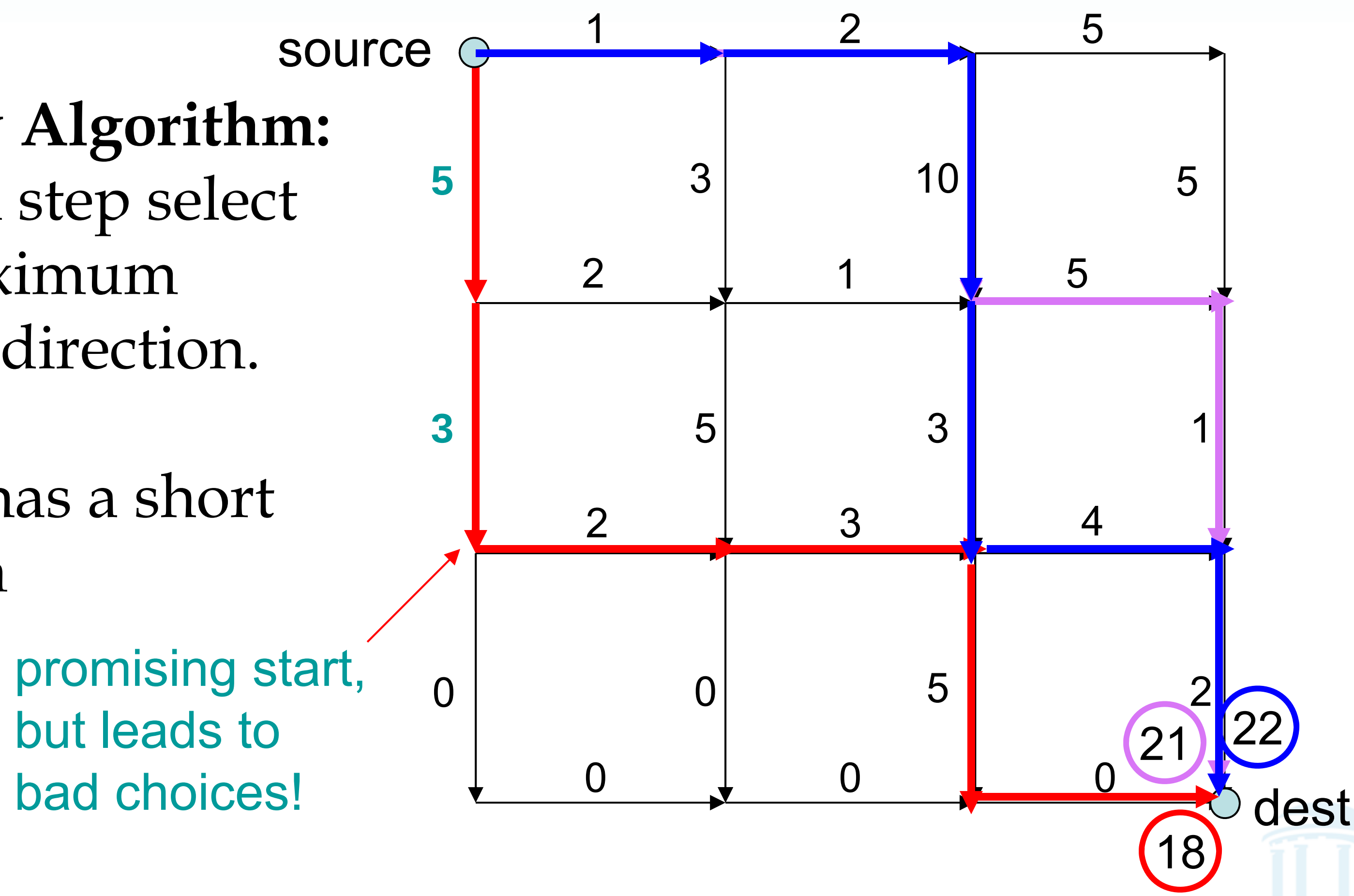
Input: A weighted grid ***G*** with two distinct vertices, one labeled "*source*" and the other labeled "*destination*"

Output: The best path (= greatest total weight) in ***G*** from "*source*" to "*destination*"

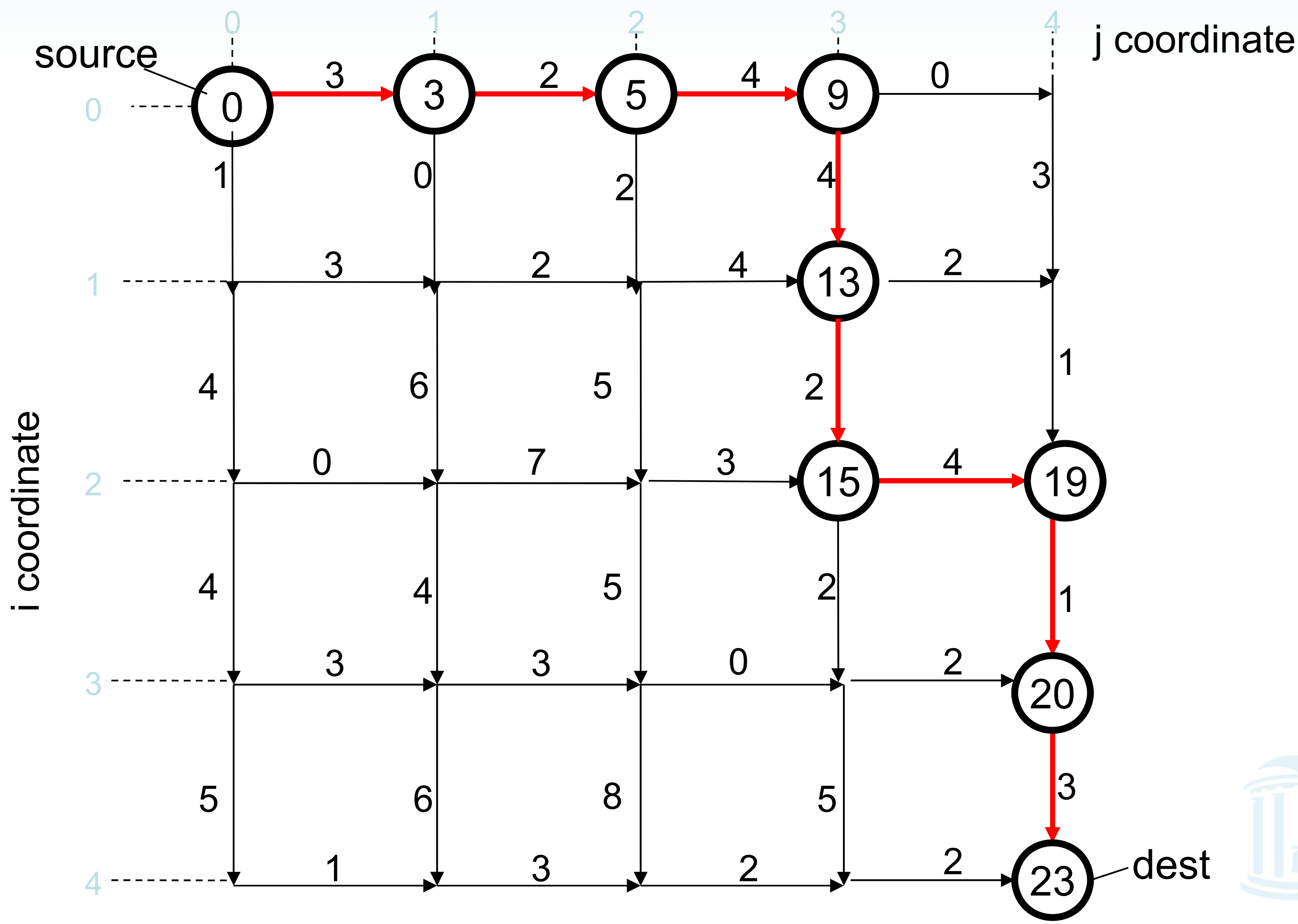
MTP: Greedy Algorithm Is Not Optimal

Greedy Algorithm:
At each step select the maximum weight direction.

Greed has a short horizon



MTP as a Dynamic Program



MTP Strategy

- Instead of solving the Manhattan Tourist problem directly, (i.e. the path from (0,0) to (n,m)) we will solve a more general problem: find the best path from (0,0) to any arbitrary vertex (i,j).
- If the best path from (0,0) to (n,m) passes through some vertex (i,j), then the path from (0,0) to (i,j) must be the best. Otherwise, you could increase your path weight by changing it.

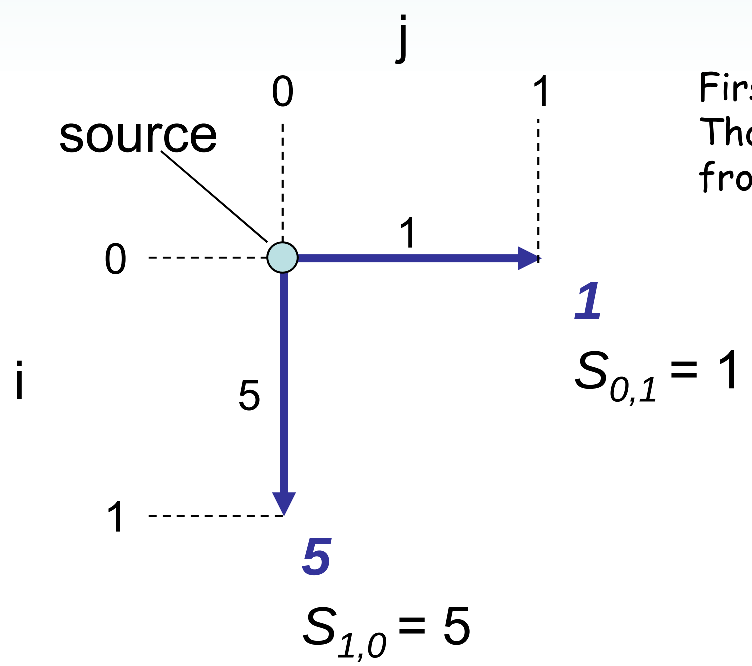
MTP: Simple Recursive Program

What's wrong with this approach?

It evaluates the same recurrence repeatedly

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

MTP: Ordering Evaluations

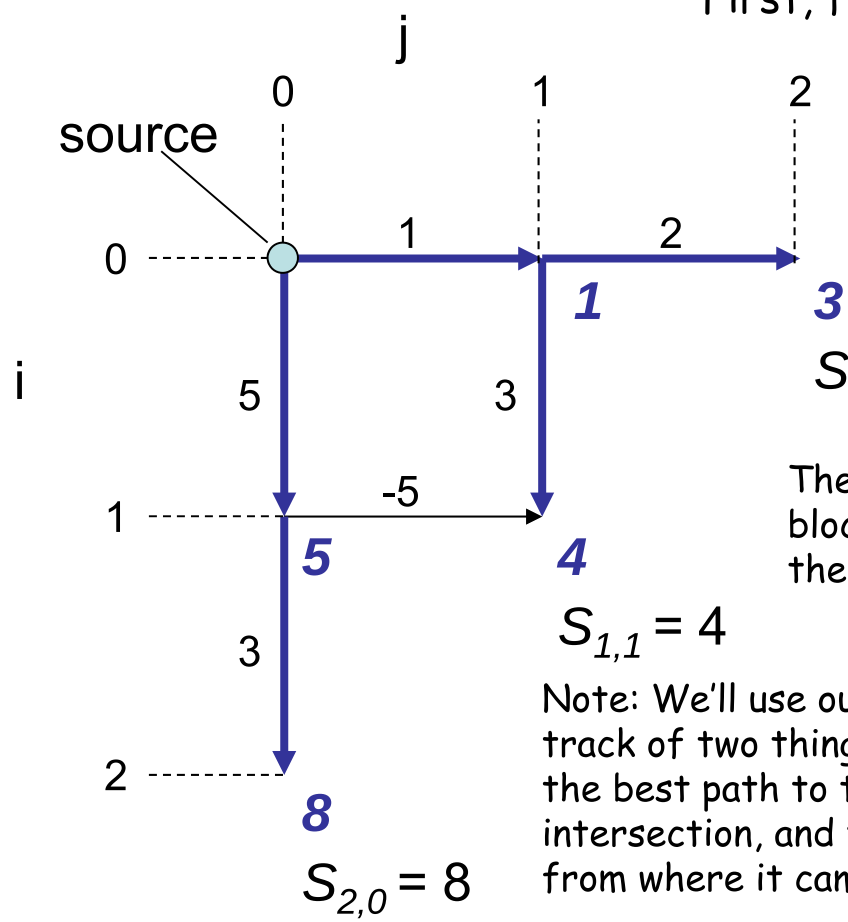


First, fill in the easy ones!
Those 1 block
from the source

- 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 connecting edge in between

MTP: Dynamic Programming (cont'd)

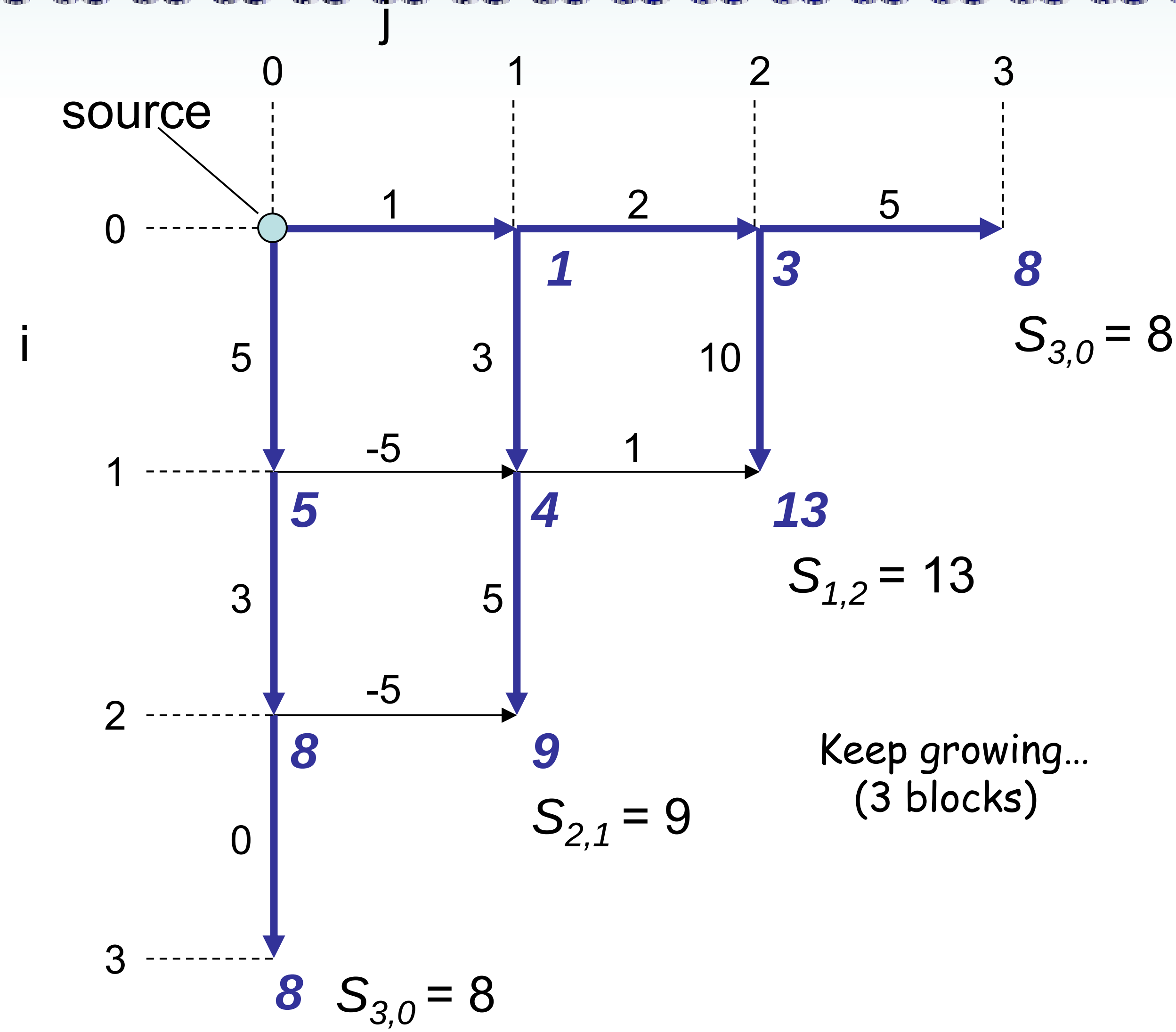
First, fill in the easy ones!



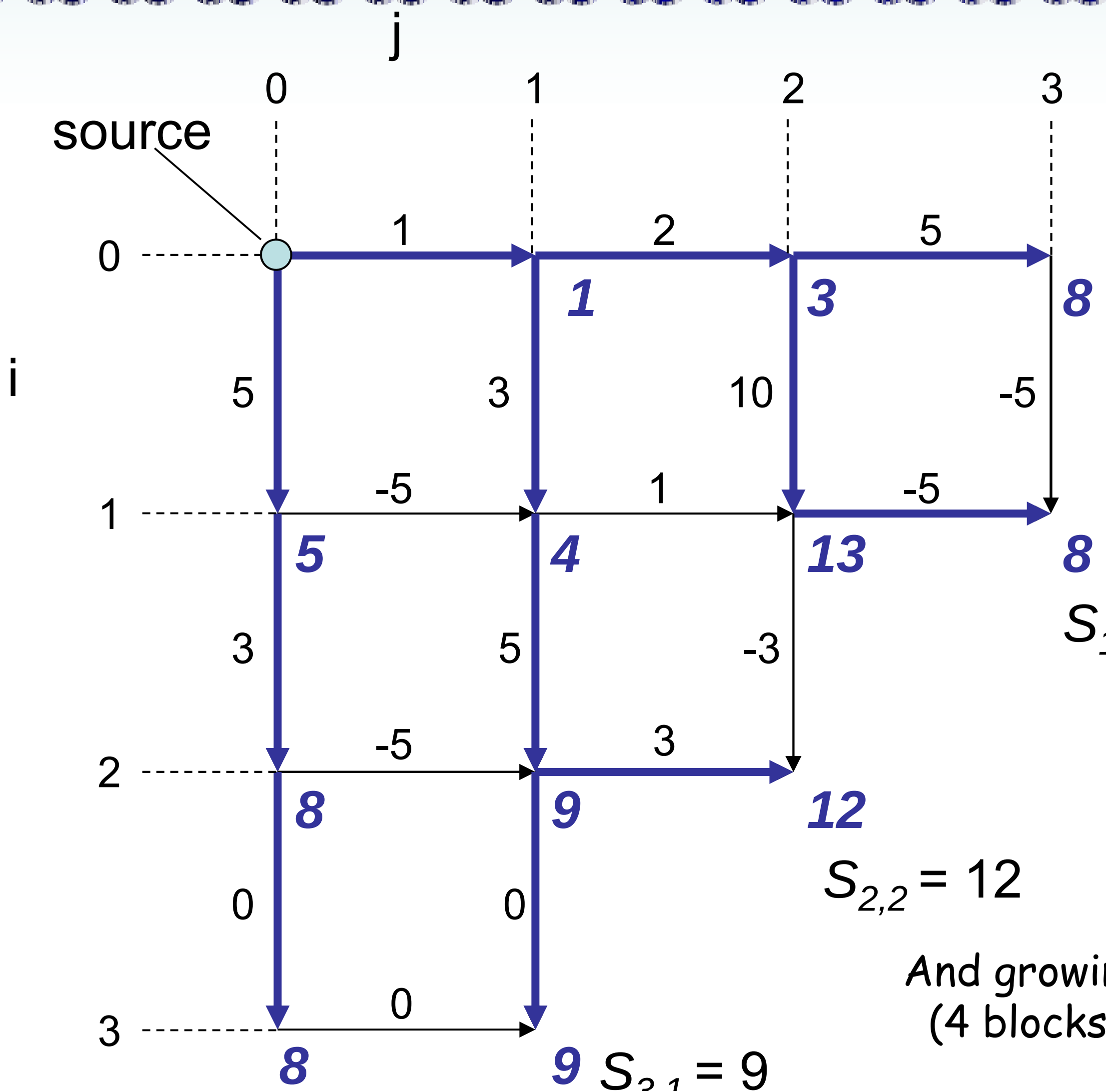
Then grow the solution a block at a time while tabulating the results for each intersection

Note: We'll use our table to keep track of two things. The value of the best path to the given intersection, and the direction from where it came

MTP: Dynamic Programming (cont'd)



MTP: Dynamic Programming (cont'd)



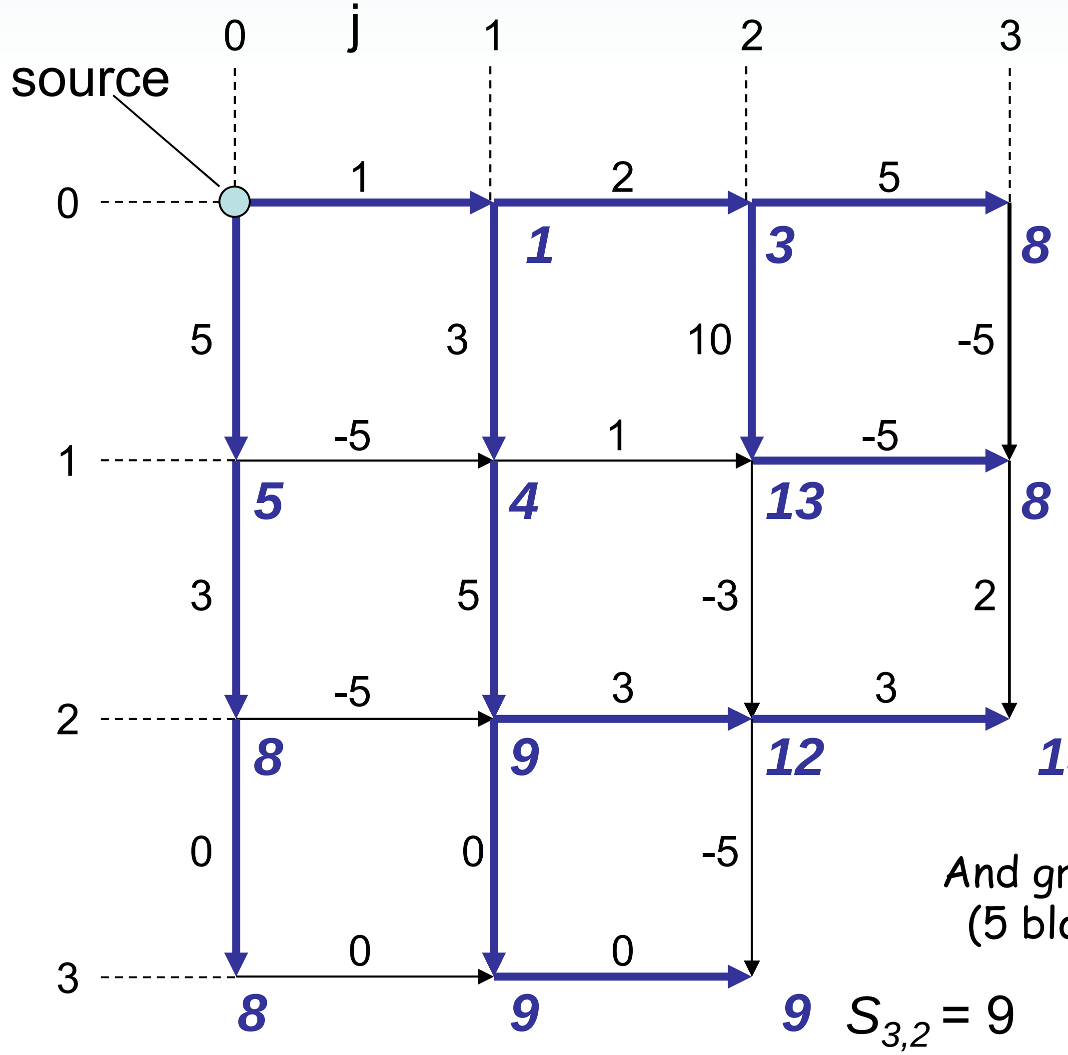
$S_{1,3} = 8$

$S_{2,2} = 12$

$S_{3,1} = 9$

And growing…
(4 blocks)

MTP: Dynamic Programming (cont'd)

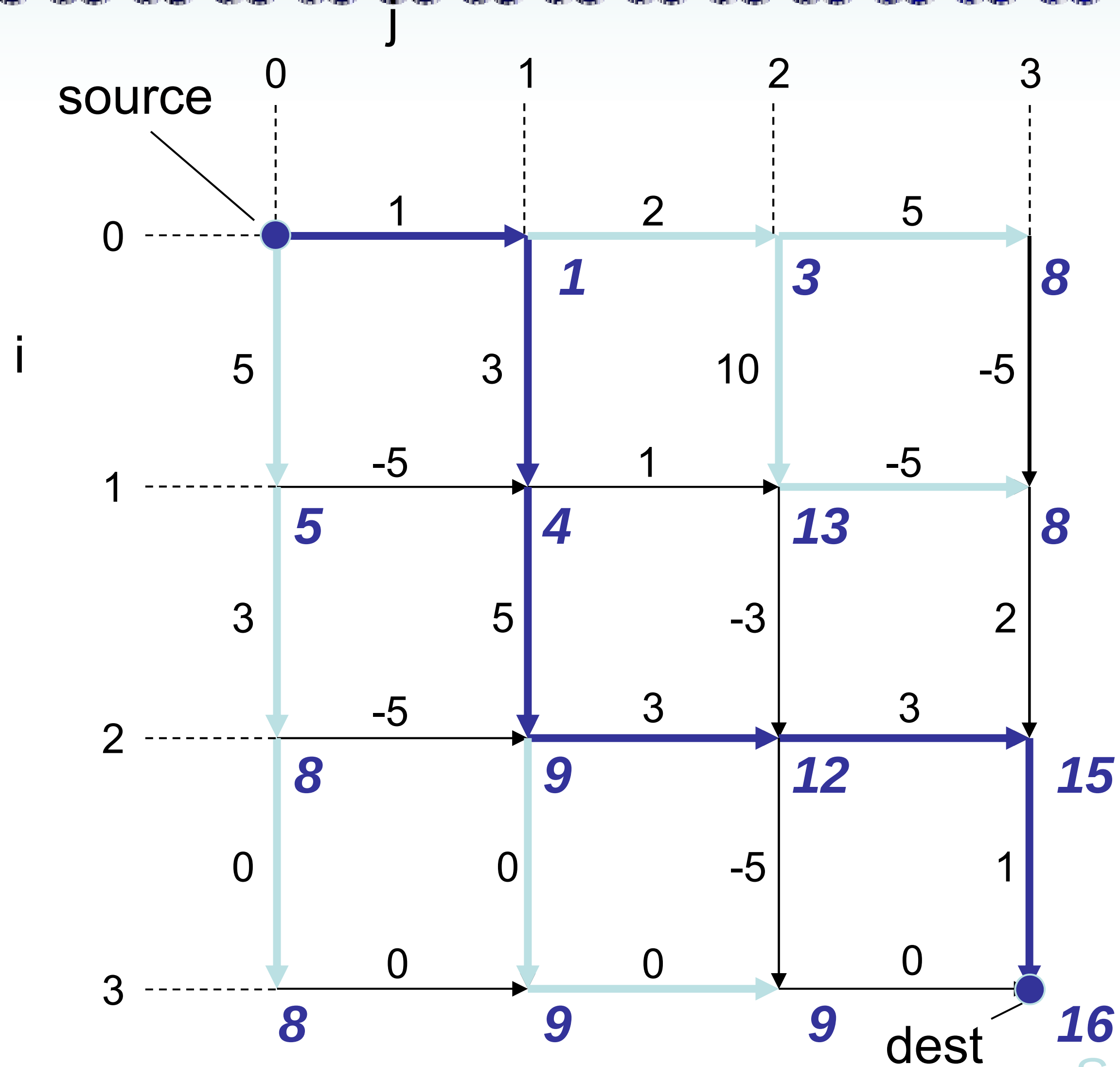


$S_{2,3} = 15$

$S_{3,2} = 9$

And growing…
(5 blocks)

MTP: Dynamic Programming (cont'd)



Once the "destination" node (intersection) is reached, we're done.

Our table will have the answer of the maximum number of attractions stored in the entry associated with the destination.

We use the "links" back in the table to recover the path. (Backtracking)

$S_{3,3} = 16$

MTP: Recurrence

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

Path to the intersection from the left

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

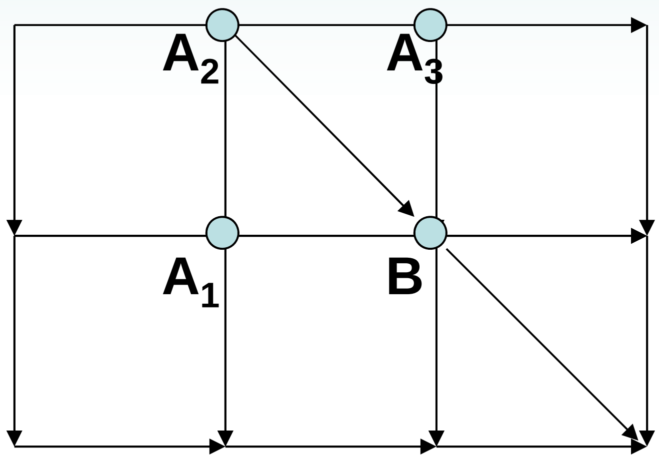
Path to the intersection from above

The running time is O(***nm***) for a ***n*** by ***m*** grid

(You visit all intersections once, and perform 2 tests)

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

Manhattan Is Not A Perfect Grid



What about diagonals?

Broadway, Greenwich, etc.

- Easy to fix. Just adds more recursion cases.
- The score at point B is given by:

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

Generalizing Manhattan to a Directed Graph

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

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

- Predecessors (*x*) – set of vertices having edges leading to *x*
- In a graph G(***V***, ***E***)
  (***V*** is the set of all vertices and ***E*** is the set of all edges)
  each edge and each vertex is considered once

Traveling in the Grid

- The only hitch is that one must decide on an *order* to visit the vertices

- We must assure that 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.

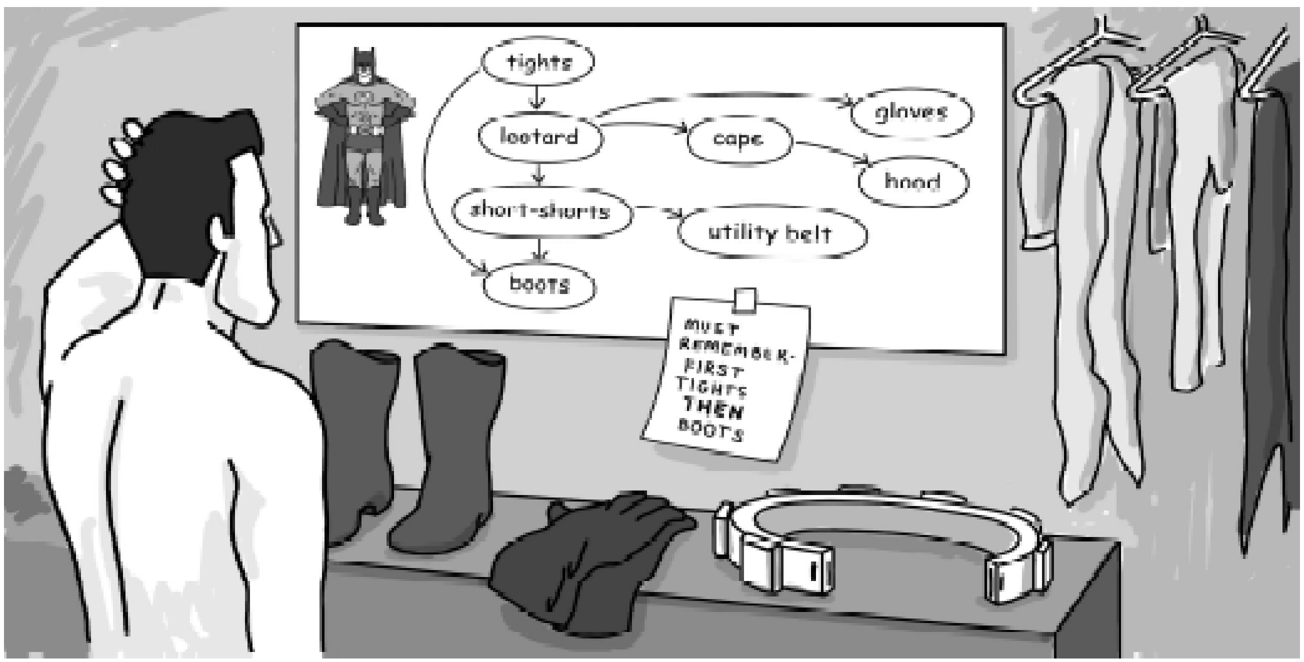
- We need to traverse the vertices in some order

- How to find such order for any directed graph?

???

DAG: Directed Acyclic Graph

- Since most cities are not perfect regular grids, we represent paths in them as a DAGs
- DAG for *Dressing in the morning* problem



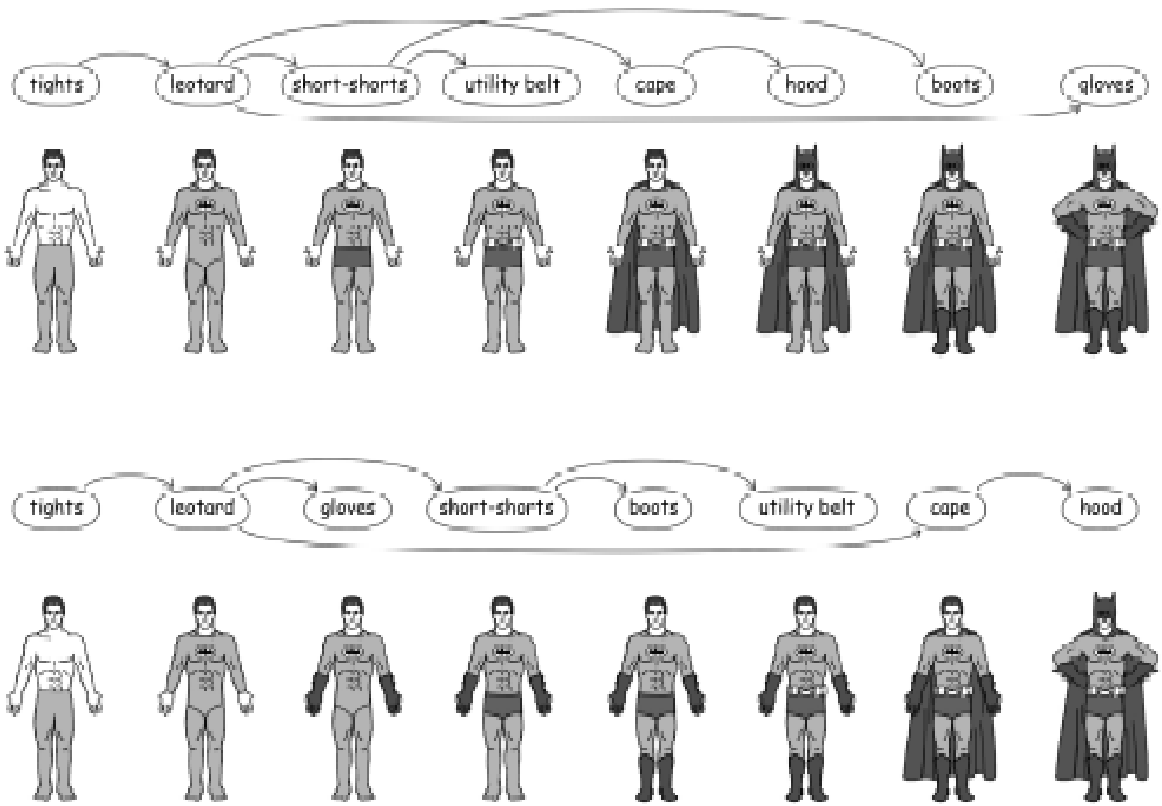
Topological Ordering

- 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

- In other words, if vertices are positioned on a line in an increasing order of labels then all edges go from left to right.

Topological Ordering

- 2 different topological orderings of the DAG



Best Path in DAG Problem

- Goal: Find highest weight path between two vertices in a weighted DAG
- Input: A weighted DAG ***G*** with source and destination vertices
- Output: A highest weight path in ***G*** from source to destination

Longest Path in DAG: Dynamic Programming

- Suppose vertex v has indegree 3 and predecessors $\{u_1, u_2, u_3\}$
- Longest path to v from source is:

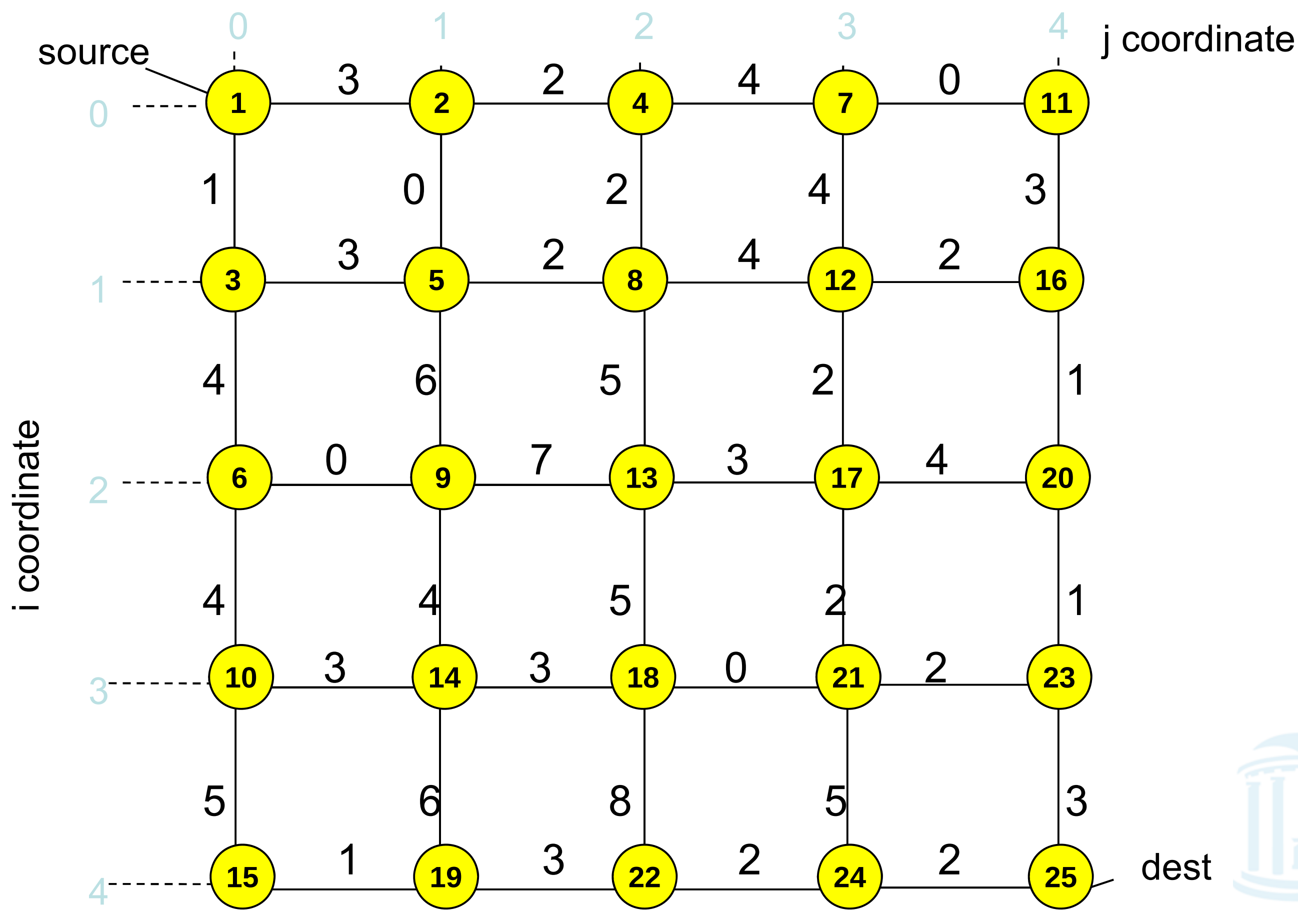
$$s_v = \max \text{ of } \begin{cases} s_{u_1} + \text{weight of edge from } u_1 \text{ to } v \\ s_{u_2} + \text{weight of edge from } u_2 \text{ to } v \\ s_{u_3} + \text{weight of edge from } u_3 \text{ to } v \end{cases}$$

<u>In General</u>:

$$s_v = \max_u \; (s_u + \text{weight of edge from } \boldsymbol{u} \text{ to } \boldsymbol{v})$$

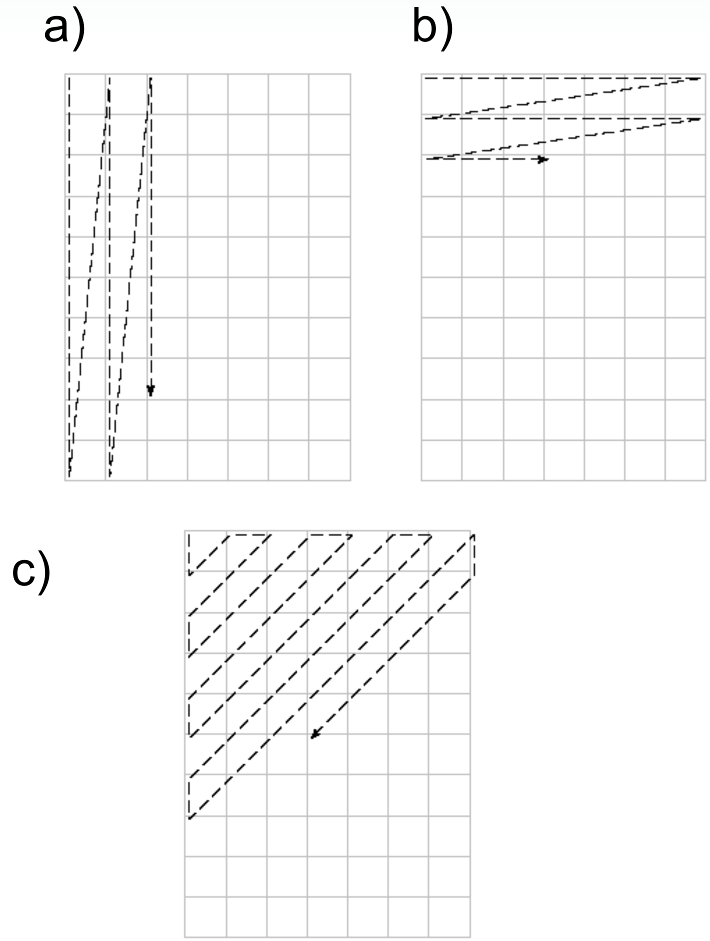
Evaluation order

- *Any* topological ordering of vertices will work!



Traversing the Manhattan Grid

- We chose to evaluate our table in a particular order. Uniform distances from the source (all points one block away, then 2 blocks, etc.)
- Other strategies:
  - a) Column by column
  - b) Row by row
  - c) Along diagonals
- This choice can have performance implications



Next Time

- Return to biology
- Solving sequence alignments using
  Dynamic Programming