Lab Demo 11

THE LAST ONE

goo.gl/1gfTpB

Friday, 16 November 2018

PE Debrief (1)

- Task A/B solution
  - B: store and sort all values in increasing order (or store in a set).
  - Print out each number, just include a ‘1’ in front of each
  - A: Keep a frequency counter (map<int,int>/unordered_map<int,int>)
  - After reading all values, insert them into a vector as a pair
  - Sort the pairs based on the requirements (highest frequency 1st, if equals by increasing number)
  - Can just negate frequency count and use default pair<int,int> comparator

PE Debrief (2)

- Task C/D/E
  - C: use a (ordered) map for each height to their name (insert the map[0] = “RAR” first)
  - Before inserting, get the person in front by using ‘lower_bound’, will find the person with greater height, so just decrement the iterator once (it--) to get the person in front
  - Final printing order, iterate through the map and print each name 1 by 1
  - D: The person in front will always be “RAR”, so just keep printing that
  - Final ordering: just store each name in a vector, reverse it and print them out (including the front “RAR” first
  - E: use a (ordered) map as well, but this time we map each height to a LIST of names (can also use vector/deque)
  - Before inserting, get the person in front by using ‘lower_bound’, then take the last person (may be first depending on the data structure you use) of that list.
  - When inserting, insert to the front/back of the list
  - Final printing order: iterate through the map, for each specific height print the names in the correct order (depending on how you insert), and for increasing heights

PS4 Debrief – Common Mistakes

Common mistakes:

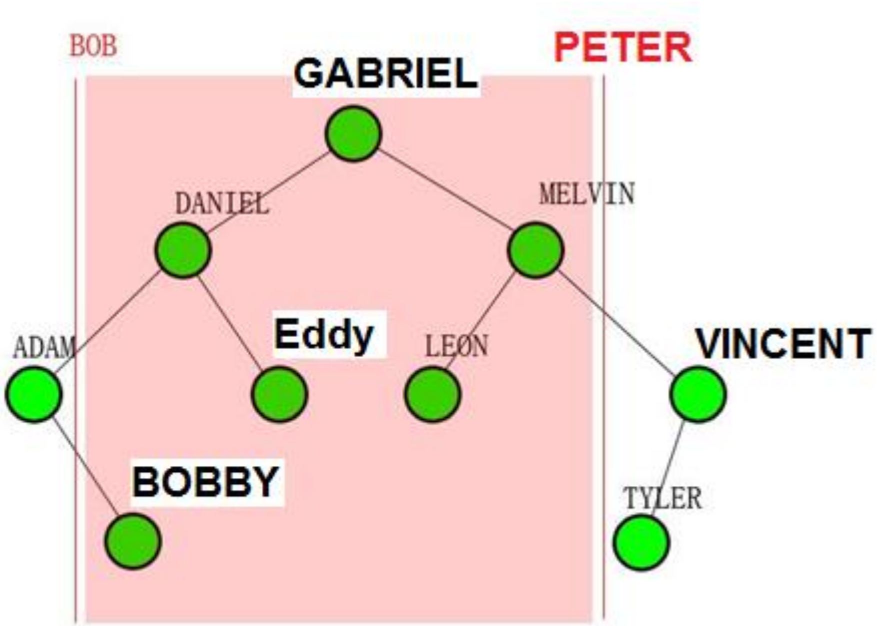
- WA/RTE in PS4 C: likely bug(s) in AVL tree insertion, reference (pointer) errors in rotateLeft/rotateRight, update height/ balance factor wrongly
- TLE in PS4 C: your 'rank' method maybe **O(n)** and not **O(log n)**, potential reasons: not splitting male vs female names, not searching upper bound and lower bound, not **using 'size'** to obtain *rank* efficiently, one quick check for **O(n)** that Lab TA have performed → see if student's solution checks BOTH left+right subtrees of bBST!
- Not AC in D (but AC in C): Your AVL tree **deletion** is incorrect

PS4 Debrief – Our Answer

The expected solution for PS4 Subtask C+D

- Write a Balanced BST routine, e.g. AVL Tree
  - Make sure the rotateLeft/rotateRight operations and all the 4 cases during insertion/deletion are handled without bug
- We split the boys and girls baby names into TWO bBSTs!
  - Important to achieve **O(log n)** per query that will be described below
- Augment this BST with “size” attribute, so that you can get a “rank” of a certain vertex in **O(log n)** as you search for it
  - This is a classic variant of BST and it has been discussed briefly in class
  - Think of “rank” as counting the number of vertices STRICTLY smaller than the value we are finding
- Then, the answer is rank(upperbound)-rank(lowerbound)
  - See the next slide for a visual explanation

Visual Explanation (Boy Names Only)



How to find boy names that contain prefix BOB (inclusive) to PETER (exclusive)?

1. Find the **<u>rank</u>** of the START name (BOB). If the START name does not exist, return the rank of a name > BOB. In this example, we return the rank of BOBBY (rank 2)
2. Find the rank of the END name (PETER). If the END name does not exist, return the rank of a name > PETER. In this example, we return the rank of TYLER (rank 8)
3. The answer is 8-2 = 6
4. PS: You can implement this idea in several other ways

PS5 Debrief

PS5

- It is about Single-Source Shortest Paths++

PS5 Debrief – Common Mistakes (1)

Typical common mistakes in PS5:

- Mostly AC in A or B, but usually with struggle (WAs/TLEs)
- This time, each Subtask requires a different code :O

PS5 Debrief – Our Answer (1)

The ultimate solution for PS5 Subtask A:

- Run SSSP on Tree (DFS or BFS), Precalculate

The ultimate solution for PS5 Subtask B:

- Just copy paste DijkstraDemo code (as shown in class on Lecture 11a), Precalculate
- Other ways exists, “SPFA” algorithm??
  - Shortest Path *Faster* Algorithm
    - The term “Faster” is really misleading though ☹, reason only discussed in CS3233
    - Read CP3.17b, google, or read past exam paper about this extra SSSP algorithm :O

PS5 Debrief – Our Answer (2)

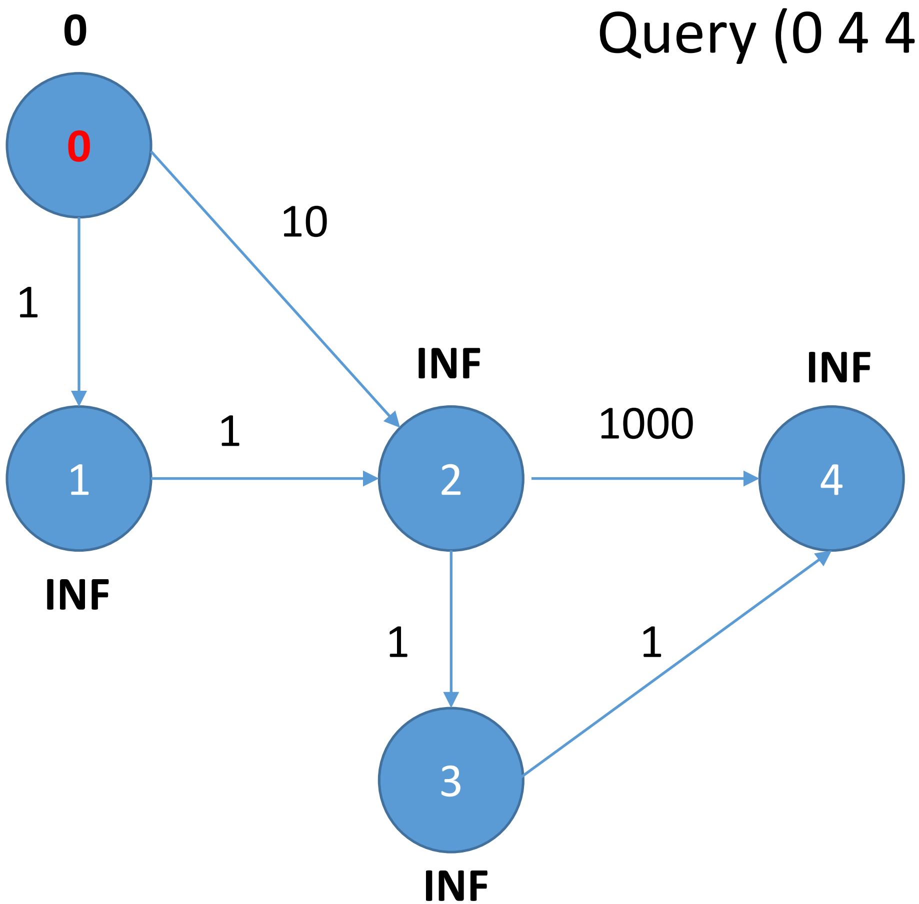
The ultimate solution for PS5 Subtask C:

- **Proper graph modeling!**
- Blow up each vertex **v** to **(v, vertices_used_so_far)**
  - This way, we can keep track of how many vertices used in the shortest path from source s to this vertex v
- Then modify your Dijkstra's implementation accordingly
  - **This is the hard part though…**
  - Wrong implementation can causes various WAs or TLEs
  - Common error: Forget to break when u == t, Break instead of continue when k_used > K, Order priority wrongly (should be distance, # vertices used, vertex number)
  - Lab TA will give closure if you fail to solve this by deadline

Graph Modelling

dist[v] = dist[u] + weight

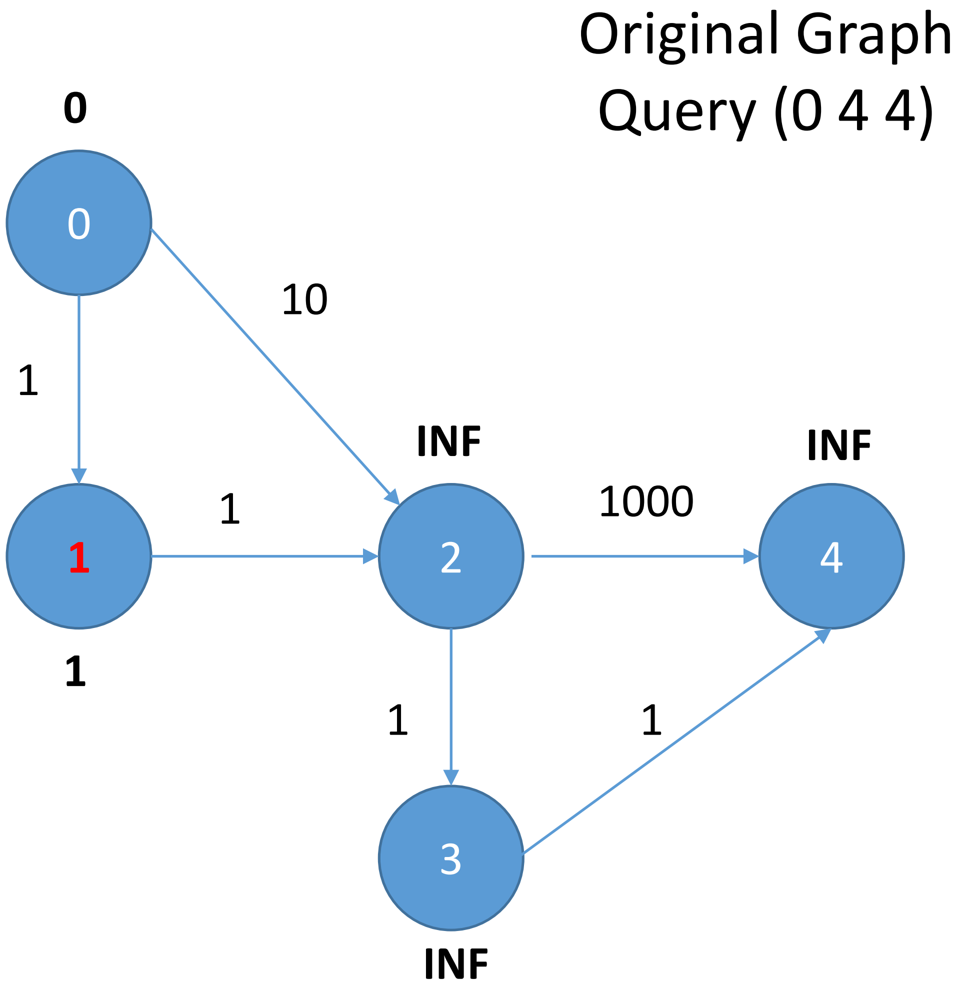
Original Graph
Query (0 4 4)



Vertex	Weight	Step
1	1	2
2	10	2

Graph Modelling

dist[v] = dist[u] + weight

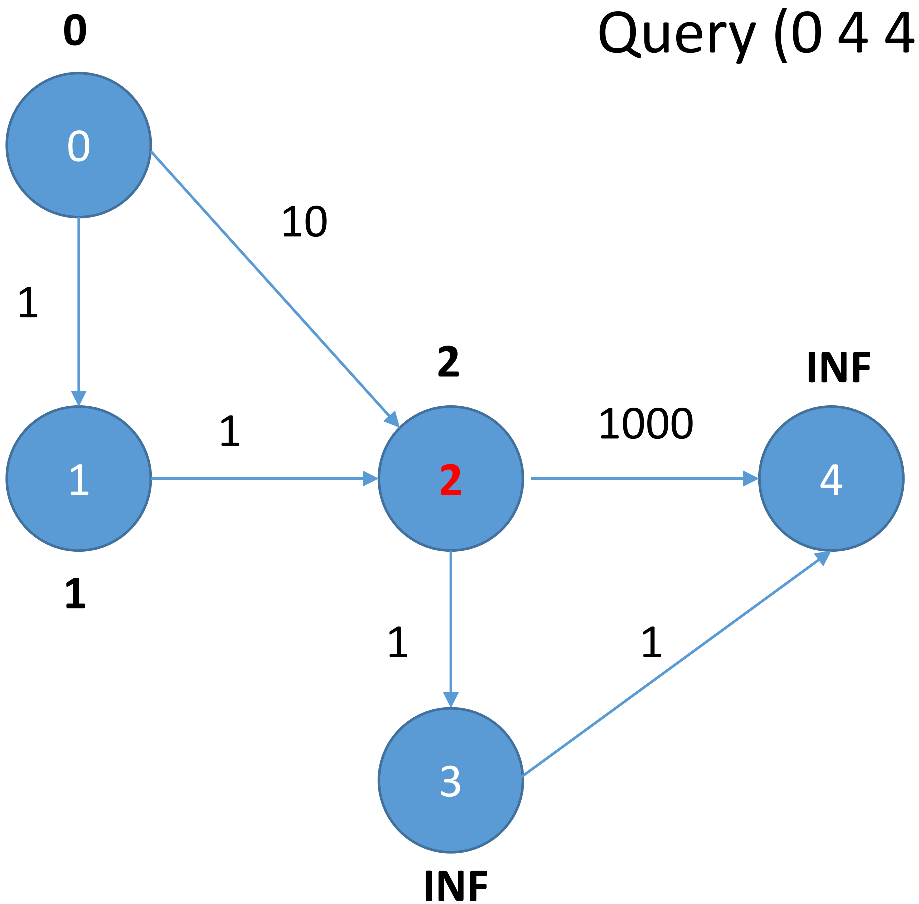


Vertex	Weight	Step
2	2	3
2	10	2

Graph Modelling

dist[v] = dist[u] + weight

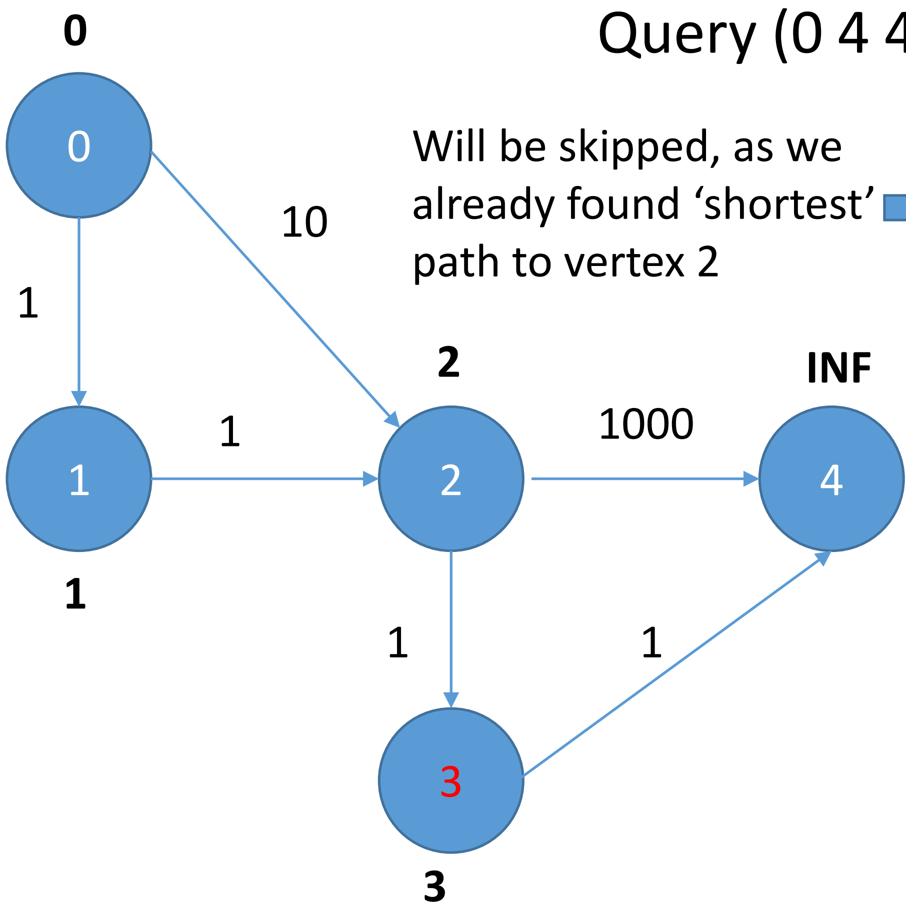
Original Graph
Query (0 4 4)



Vertex	Weight	Step
3	3	4
2	10	2
4	1002	4

Graph Modelling

dist[v] = dist[u] + weight



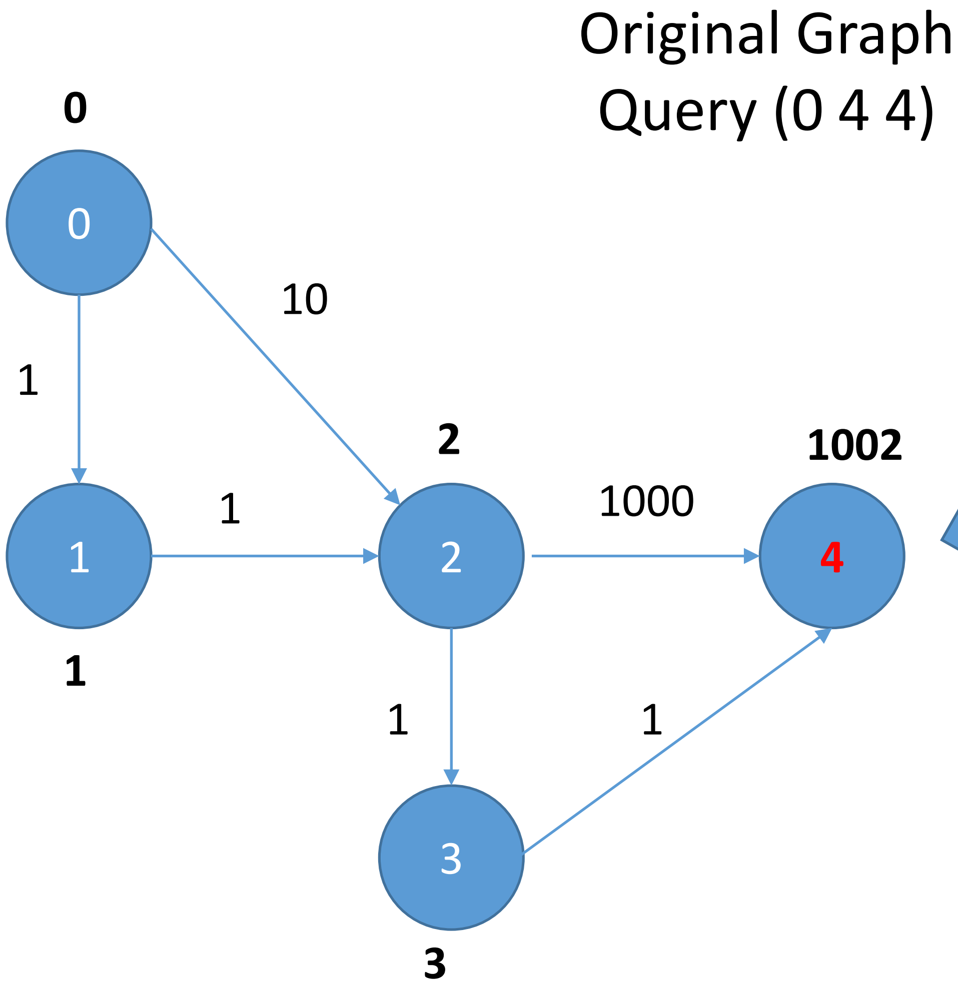
Will be skipped, as we already found ‘shortest’ path to vertex 2

Vertex	Weight	Step
2	10	2
4	1002	4

Never update vertex 4 again, as we pass the step counter!

Graph Modelling

dist[v] = dist[u] + weight



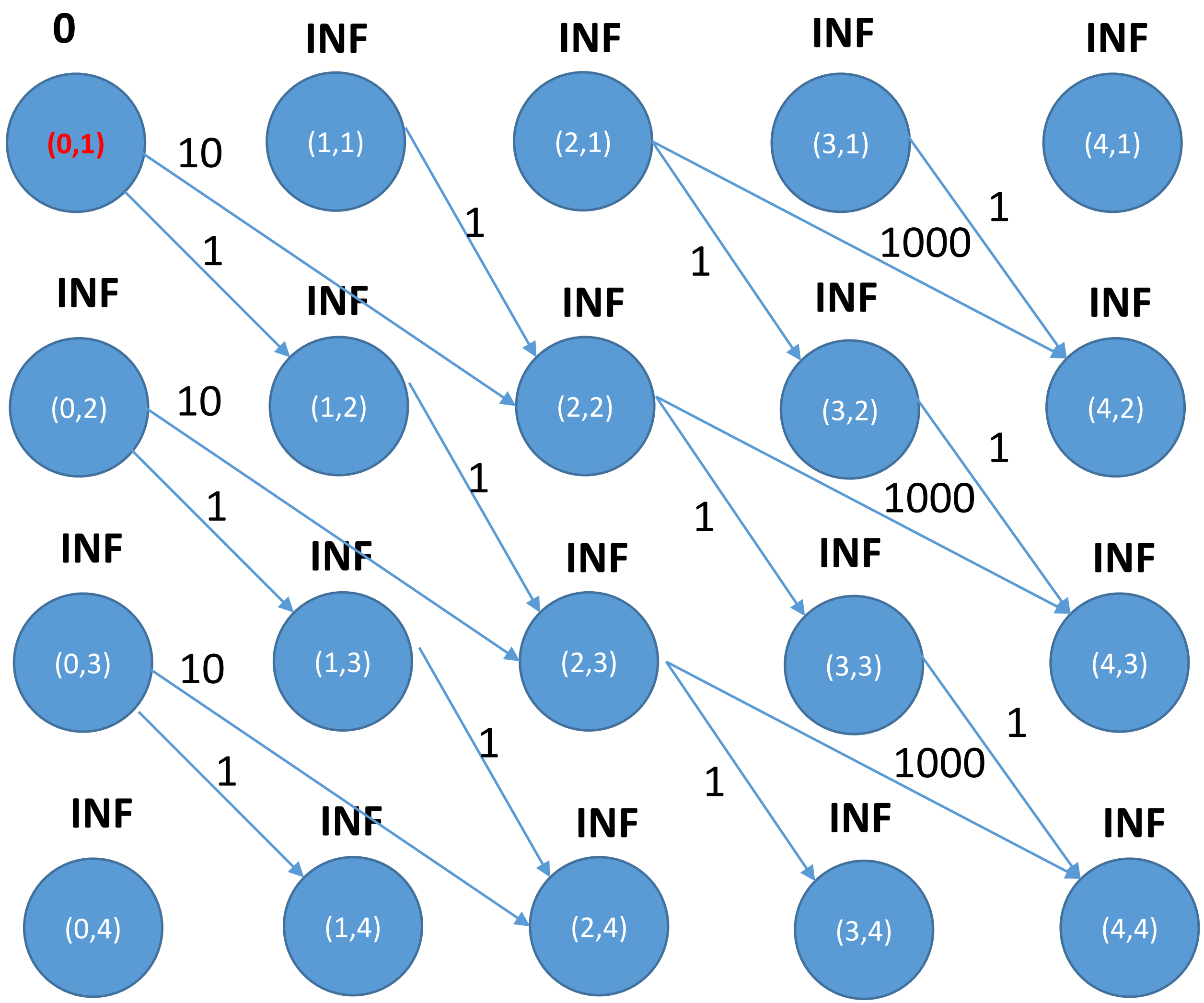
Vertex	Weight	Step

Incorrect answer! Should be 0->2->3->4, for a total weight of 12!

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)

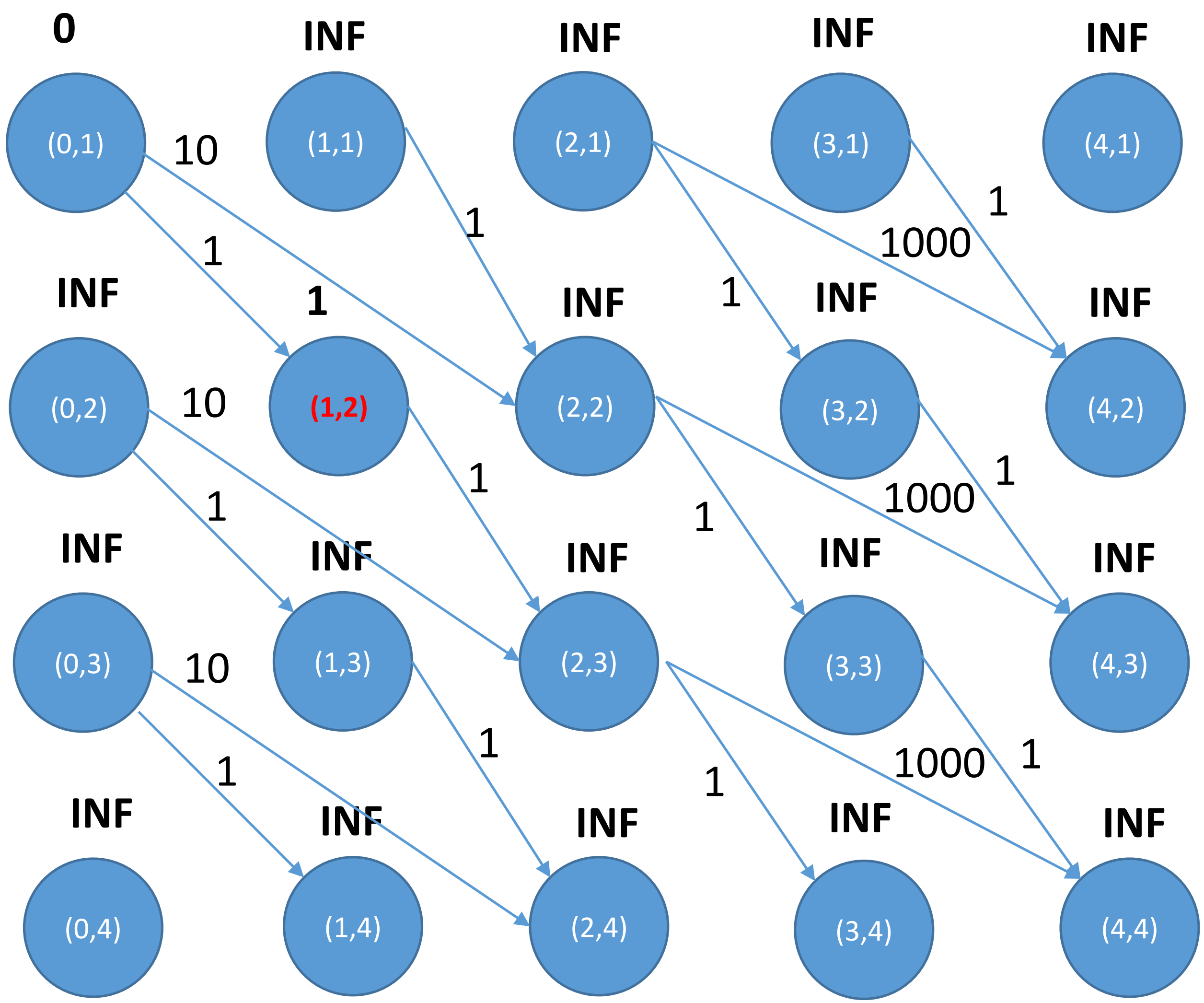


Vertex State	Weight
(1,2)	1
(2,2)	10

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)

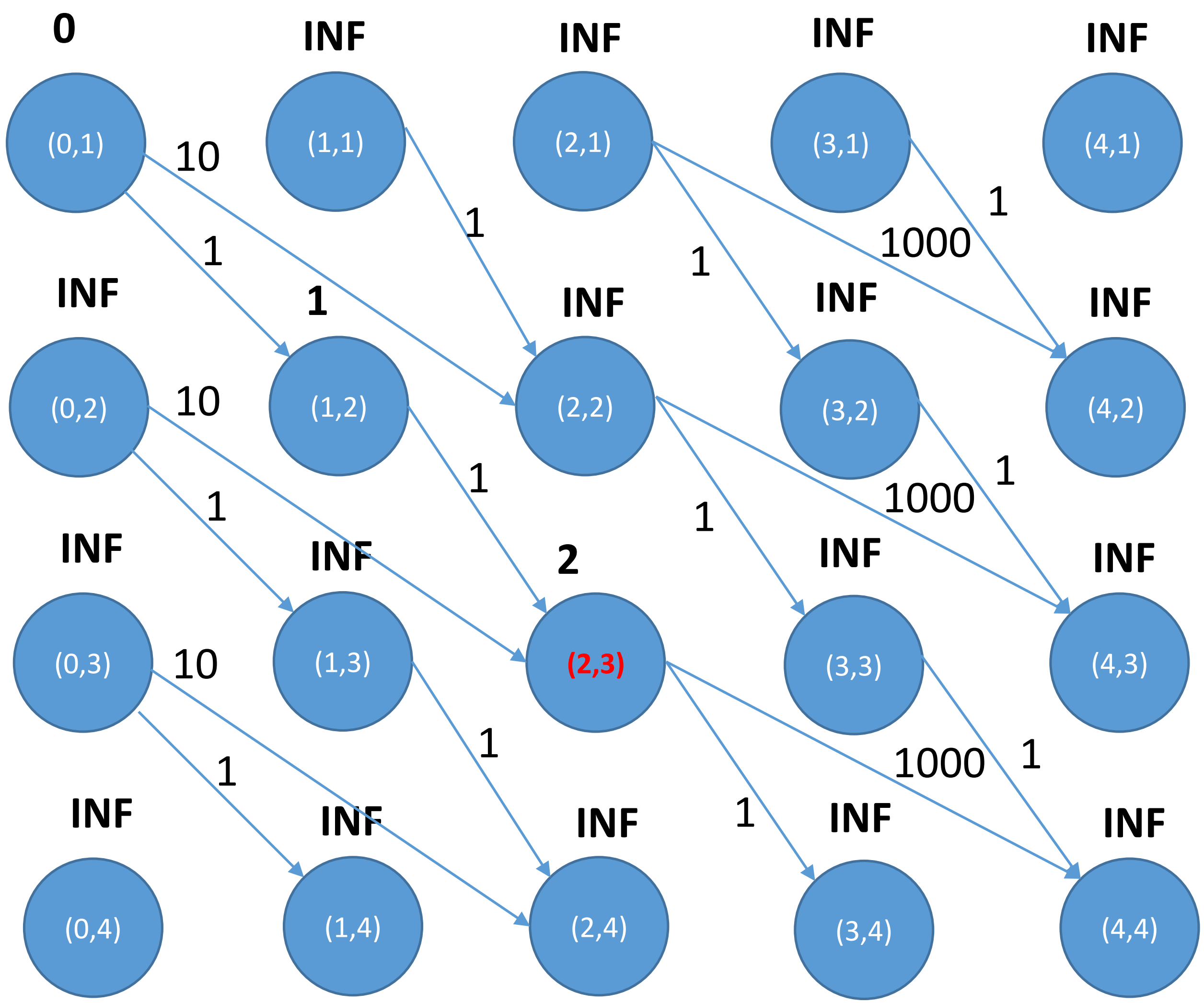


Vertex State	Weight
(2,3)	2
(2,2)	10

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)

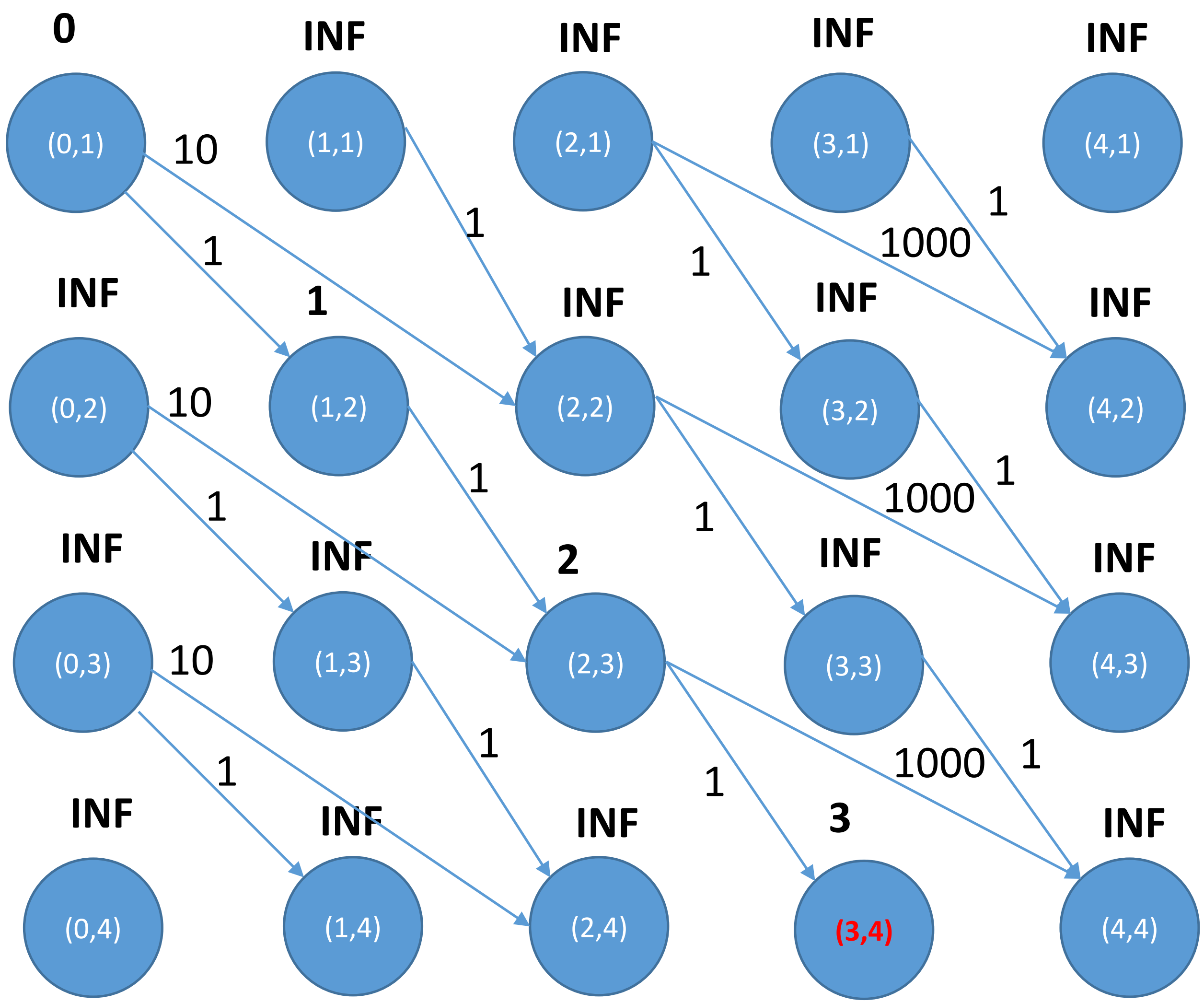


Vertex State	Weight
(3,4)	3
(2,2)	10
(4,4)	1002

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)

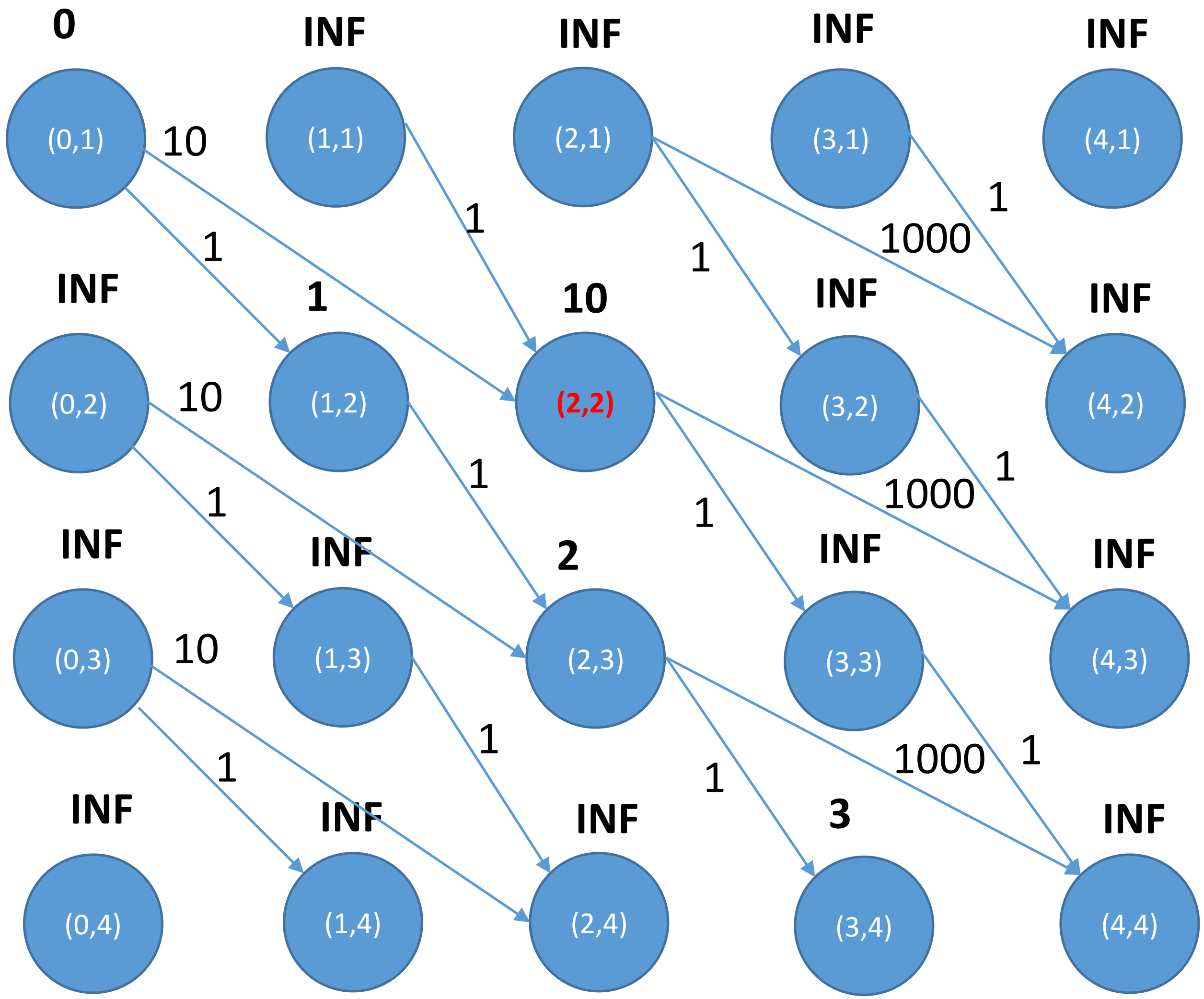


Vertex State	Weight
(2,2)	10
(4,4)	1002

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)



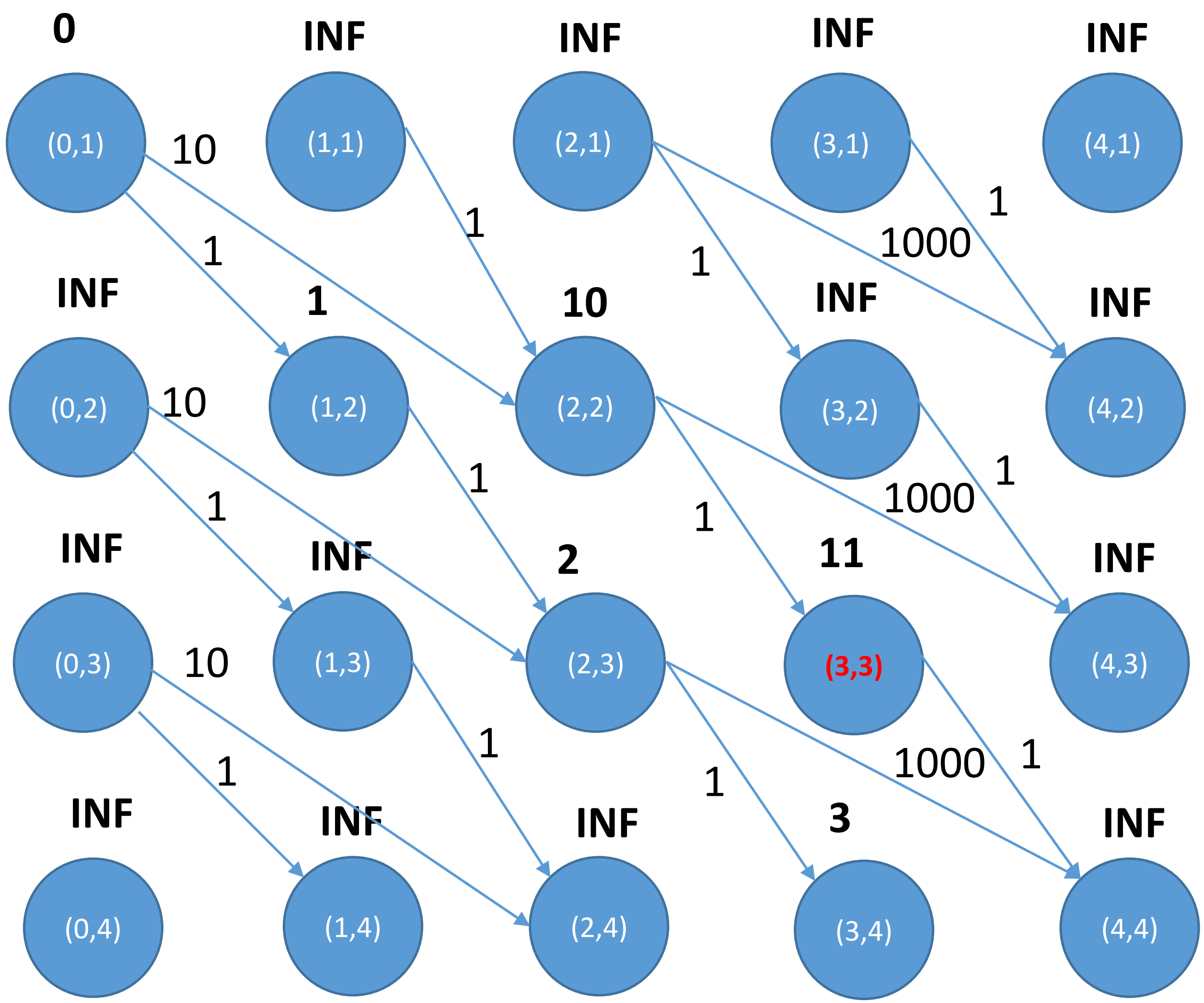
Vertex State	Weight
(3,3)	11
(4,4)	1002
(4,3)	1010

Notice that vertex state (2,2) and (2,3) has different shortest distance, even though they are technically the same vertex!

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)

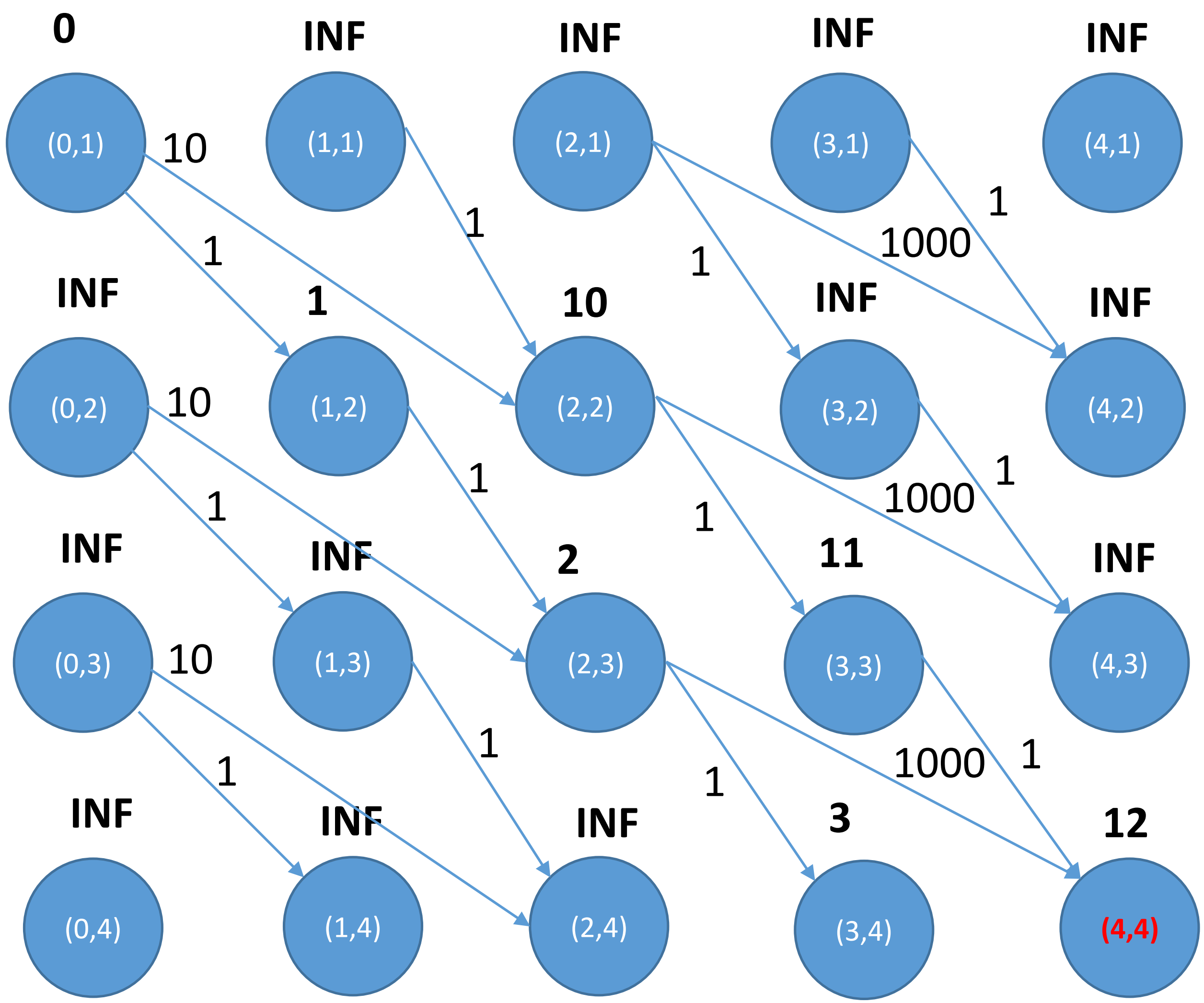


Vertex State	Weight
(4,4)	12
(4,4)	1002
(4,3)	1010

Graph Modelling

dist[step+1][v] =
dist[step][u] + weight

Modified Graph
Query (0 4 4)



Vertex State	Weight
(4,4)	1002
(4,3)	1010

Got correct answer of weight 12!

Is that a DP Problem?

Yes, we can classify PS5 Subtask C as a DP problem

This is because the transformed graph is actually a DAG

In fact, after you properly learn DP in CS3230, you may want to re-do this problem with DP technique instead of using Dijkstra’s algorithm

- But it is ‘slower’ due to the usage of recursive calls… (although the theoretical time complexity is ‘faster’ compared to Dijkstra (O(kVE) vs O(KE(logkV)))

Topological Sorting

- What if we want the lexicographically smallest topological sorting? (Kahn’s algorithm)

That’s all!

All the best for your Final Assessment in ~3 weeks time (yeah, Thursday, 04 December 2018 is still a long time from this last class)