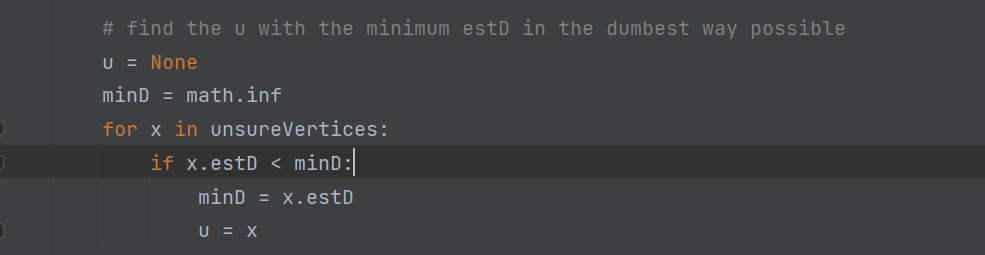
1. Summaries the main points in this module. You may include references to the learning objectives.
2. Re-visiting Dijkstra algorithm, and understanding it in detail.
3. Formulation of Bellman-Ford algorithm in form leading to Dynamic Programming.
4. Basic introduction to Dynamic Programming
5. How is this useful?
6. The shortest path between two points on the map can be found.
7. Basic introduction to Dynamic Programming. OSPF(Open Shortest Path First).
8. Detecting negative cycles using Bellman-Ford algorithm.
9. Routing Information Protocol (RIP) uses something like Bellman-Ford.
10. How do you plan to use this information?
11. Combine the knowledge learned in this class with computer networks to take your computer knowledge to the next level.
12. In the future, you can plan your route so that you don’t take detours.
13. Provide summary of your reading list — external resources, websites, book chapters, code libraries, etc.
14. <https://www.geeksforgeeks.org/bellman-ford-algorithm-simple-implementation/>
15. <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>
16. <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
17. <https://en.wikipedia.org/wiki/Dynamic_programming>
18. [CLRS] Coreman, T. and Leiserson, C. and Rivest, R. and Stein, C. Introduction to Algorithms, Chapter 24.

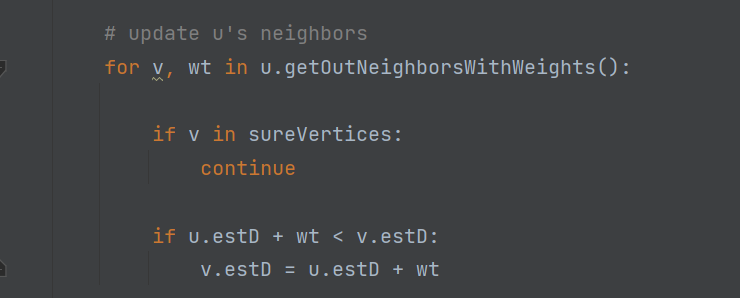
Reflecting on the content:

1. What is the most important thing you learnt in this module?
2. Definition of a weighted graph.
3. Dijkstra algorithm.
4. Bellman-Ford algorithm.
5. Dynamic Programming.
6. Floyd-Warshall algorithm.
7. How does this relate to what you already know?
8. Connected to the computer network.
9. Connect with what was said in module 6 and expand.
10. Connect with some common pathfinding algorithms in real life.
11. Activity1

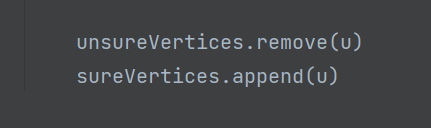
Pick the not-sure node u with the smallest estimate d[u].



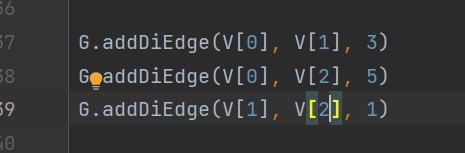
Update all u’s neighbors v: d[v] = min( d[v] , d[u] + edgeWeight(u, v))

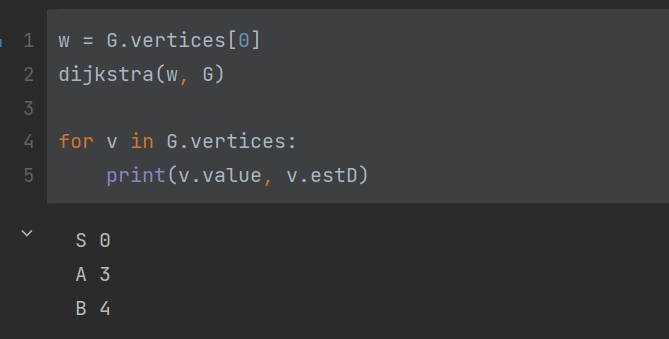


Mark u as sure.

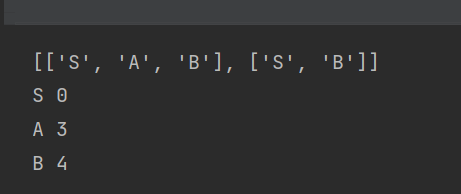


Result:

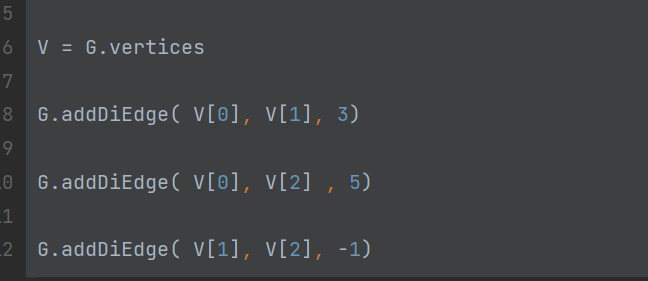




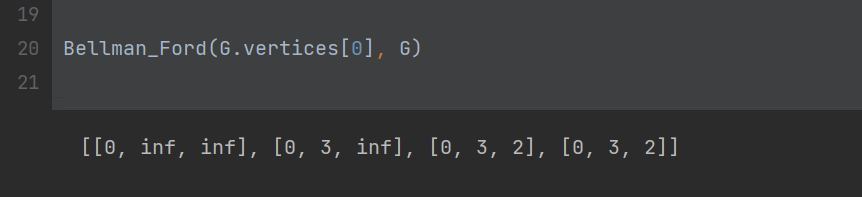
Return search path:



Negative edges:

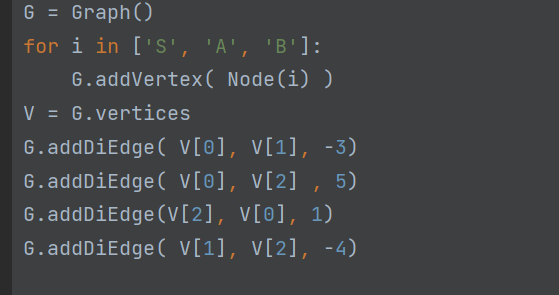


Result:

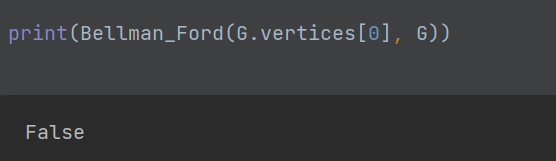


Negative cycles:

Input:

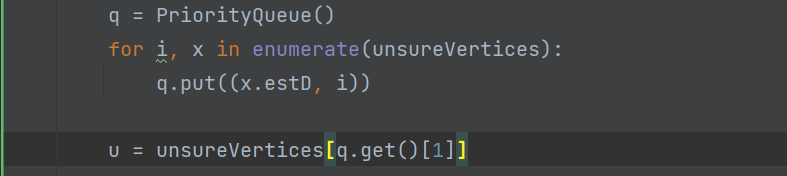


Result:



Activity 3: Have a look at the following Python’s implementation of heap (priority Queues): HeapDict, and try to convert our Dijkstra’s implementation to use this heap data structure instead.

Implementation of the heap：



Result：



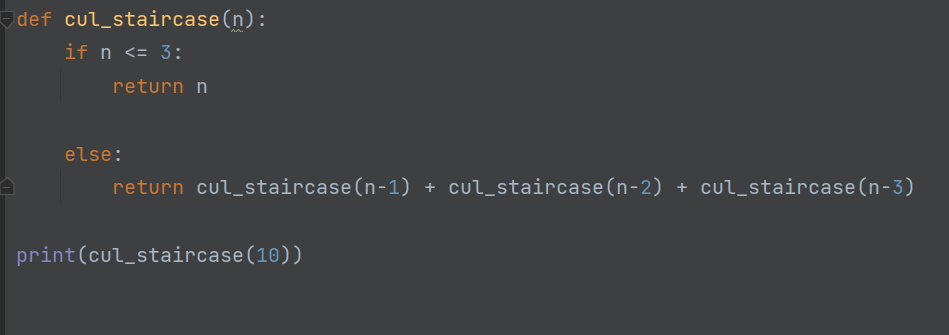
Activity 4: Write a brief report comparing ‘Dynamic Programming’ with ‘Divide-and-Conquer’ strategy? (If you are not familiar with Divide-and-Conquer strategy - revisit Quick Sort or Binary Search algorithms. You should address why we cannot use DP for Quick Sort or Binary Sort).

Dynamic programming is also a divide and conquer idea (for example, its state transition equation is a kind of divide and conquer), but unlike the divide and conquer algorithm, the divide and conquer algorithm decomposes the original problem into several sub-problems, and solves the sub-problems top-down, combine the solutions of the subproblems to get the solution of the original problem. Dynamic programming also decomposes the original problem into several sub-problems, and then solves the smallest sub-problem from the bottom up, and stores the results in a table. When solving a large sub-problem, directly query the solution of the small sub-problem from the table. Avoid double calculations, thereby improving algorithm efficiency.

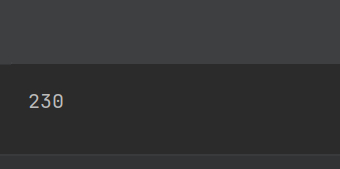
Unless there is a presence of overlapping subproblems like in the Fibonacci sequence problem, a recursion can only reach the solution using a divide and conquer approach. That is the reason why a recursive algorithm like Merge Sort cannot use Dynamic Programming, because the subproblems are not overlapping in any way.

Activity 5: You are running up a staircase with a total of n steps. You can hop either 1 step, 2 steps or 3 steps at time. Write a DP program to determine how many possible ways you can run up the stairs? (Hint: Start with a recursive solution, and then later move to top-down approach of DP).

Code：

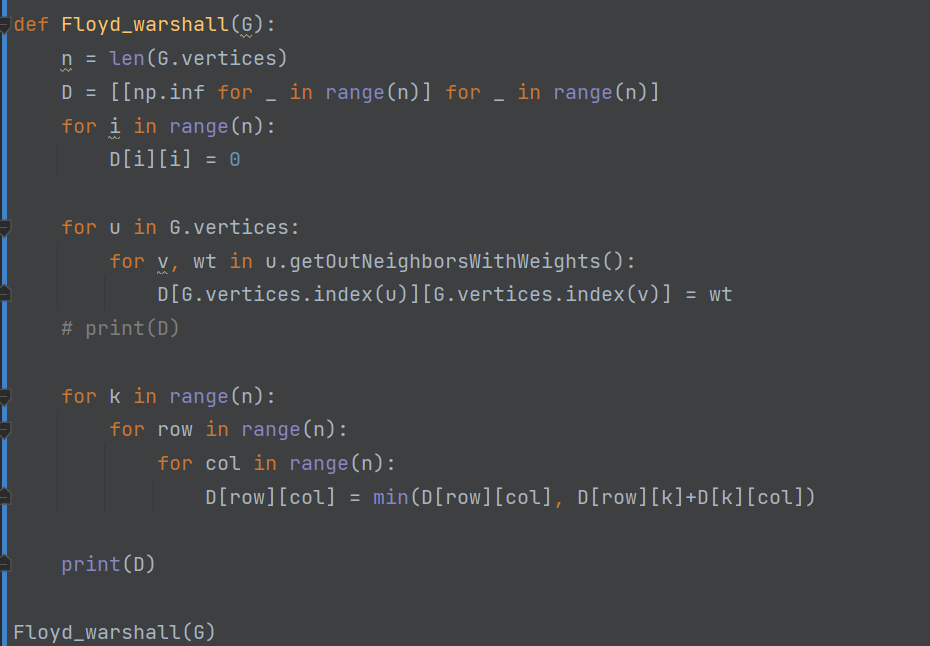


Result：



Activity 6: Implement Floyd-Warshal algorithm using above data structure (that is Node and Graph). Make sure to test all the use-cases.

Code:



Result:

