COMP20003 Workshop Week 12

1	Floyd-Warshall Algorithm
2	Sample Exam Papers?
3	Assignment 2

Floyd-Warshall Algorithm

Purpose= ?

Compared with Dijkstra's ?

Floyd-Warshall Algorithm - APSP

Given a weighted DAG G=(V,E,w(E))

Find shortest path (path with min weight) between all pairs of vertices.

Idea= ?

Floyd-Warshall Algorithm - APSP

Given a weighted DAG `G=(V,E,w(E))`

Find shortest path (path with min weight) between all pairs of vertices.

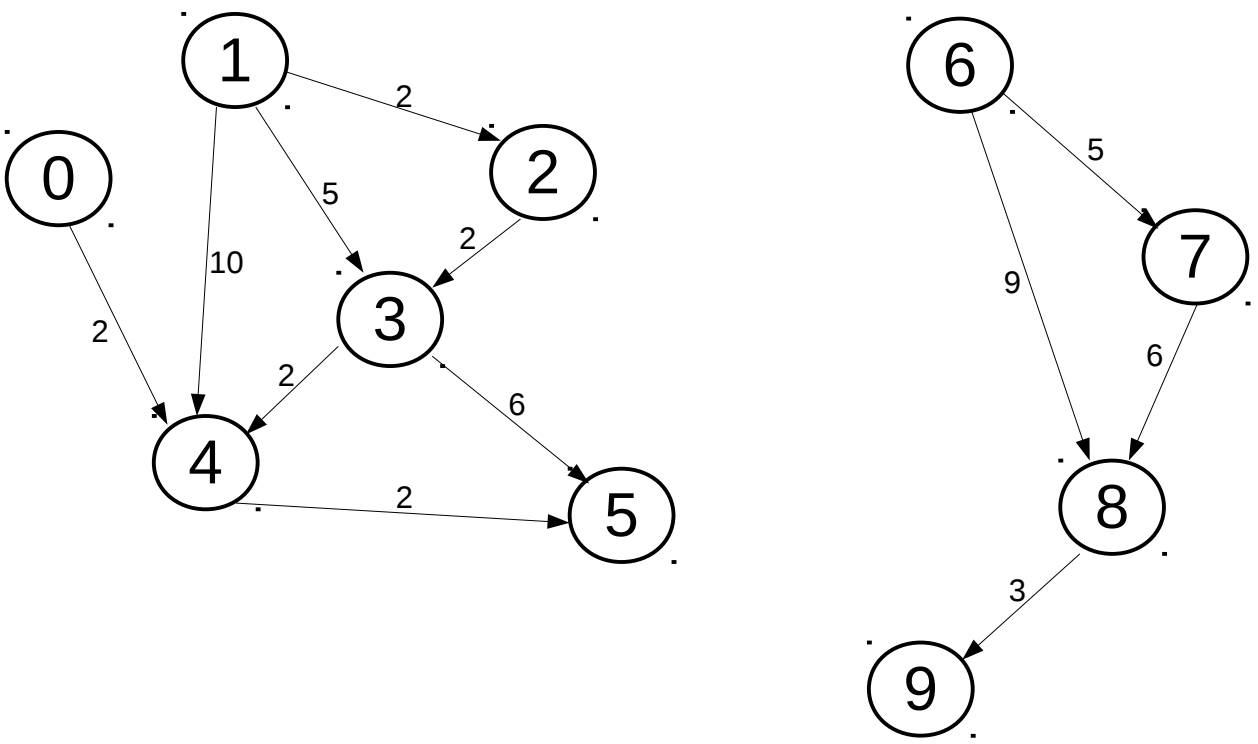
Idea:

- suppose `dist[s][t] = w(s,t)`
- how about take a node `i` and use `i` as an interim step, ie. examine the potential path `s→i→t ?`
- how about apply the above step for all possible nodes `i`?

What about retrieving the shortest paths?

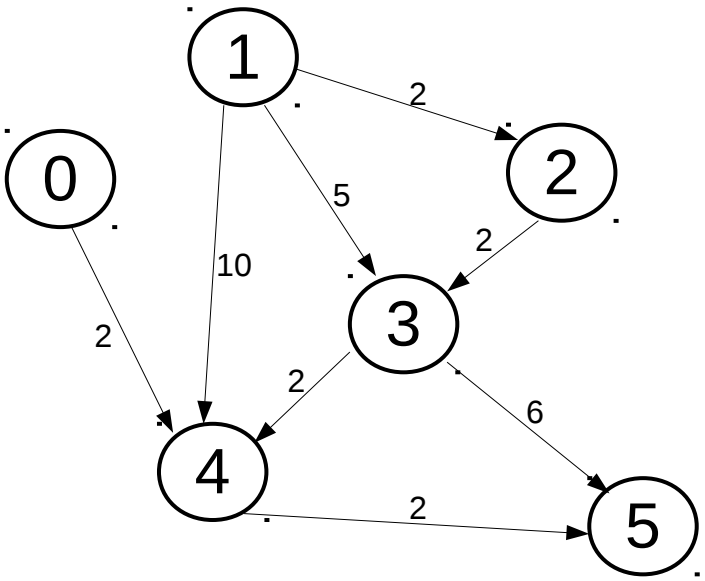
Floyd-Warshall Algorithm: write C code

Q11.1 (from JupyterHub)



a) Draw matrix representation of the graph
b) Trace the operation of Floyd-Warshall algorithm, also find:
c) *What's a shortest path from 1 to 5

Q 11.1 [simplified]



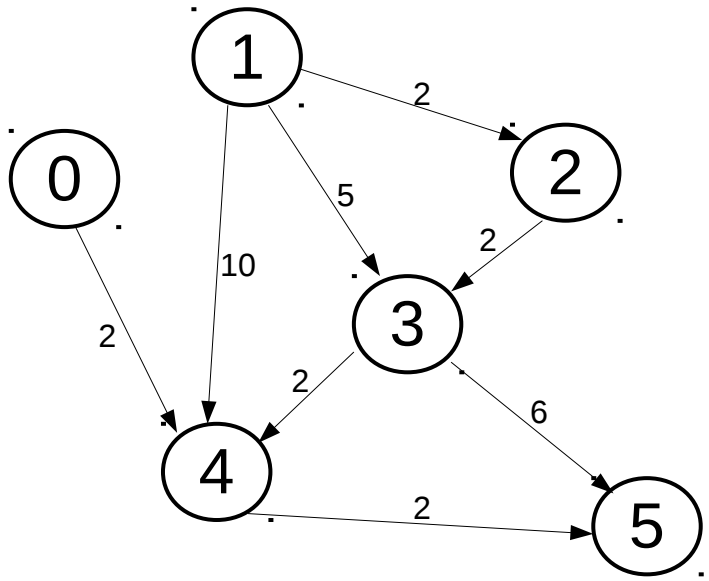
Draw the matrix representation.
Trace the Floyd-Warshall algorithm.

TO

FROM

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Q 11.1 [simplified]



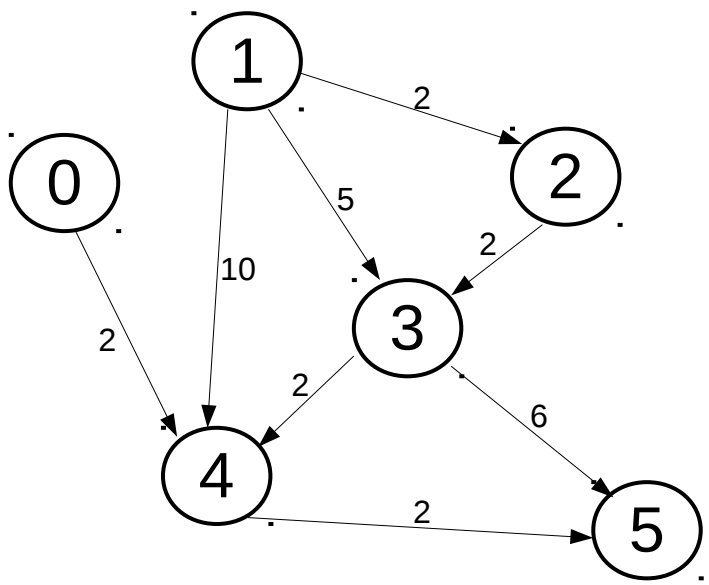
Trace the Floyd-Warshall algorithm.
Step i= 0, 1, 2, 3, 4, 5
TO

FROM

	0	1	2	3	4	5
0					2	
1			2	5	10	
2				2		
3					2	6
4						2
5						

empty cell for ∞
(note A[i][i] could be zero if we want)

Q 11.1 [simplified]



Trace the Floyd-Warshall algorithm (empty means ∞).
Step i= 0, 1, 2, 3, 4, 5

TO

FROM

	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Q11.1 cnt

d) Does the Floyd-Warshall algorithm work on graphs where there are negative weights? Justify your answer.

e) Given a sparse graph represented as an adjacency list, how would you approach the all pairs shortest paths problem? Does this differ from the approach you would take for a dense graph represented as a matrix? Compare the computational complexity of the two approaches.

Q11.1: APSP – comparing Dijkstra & Floyd-Warshall

Big-O complexity
(supposing to apply Dijkstra for the APSP task)

	Dijkstra	Floyd-Warshall
General		
Sparse		
Dense		

Q11.1: APSP – comparing Dijkstra & Floyd-Warshall

Big-O complexity
(supposing to apply Dijkstra for the APSP task)

	Dijkstra	Floyd-Warshall
General Sparse	$V\ (V+E)\ \log V$	V^3
Dense	$V^2\ \log V$	V^3
	$V^3\ \log V$	V^3

5 min break: qoct

Do quality of casual teaching.

To navigate to the website, just google **qoct**

Lab's choices

Working on Sample Exam paper 2016, 2015

Assignment 2

Using JupyterHub implement the Floyd-Warshall algorithm

Sample Exam Paper 2016 & 2015

In pair: pickup questions, discuss how to solve…

Workshop Week 12: additional materials

1	MST & Greedy Algorithm for building MST
2	Prim's Algorithm
3	Kruskal's Algorithm
4	A* search

MST & Greedy Algorithm

Task: give a connected, weighted graph `G= (V, E ,w)`, find a MST of it.

MST & Greedy Algorithm

Greedy algorithm:

A myopic policy of always taking the "best" bite in each step.

In many cases it's the best policy!

Dijkstra's algorithm is greedy.

Greedy algorithm can be used for the MST task, for example:

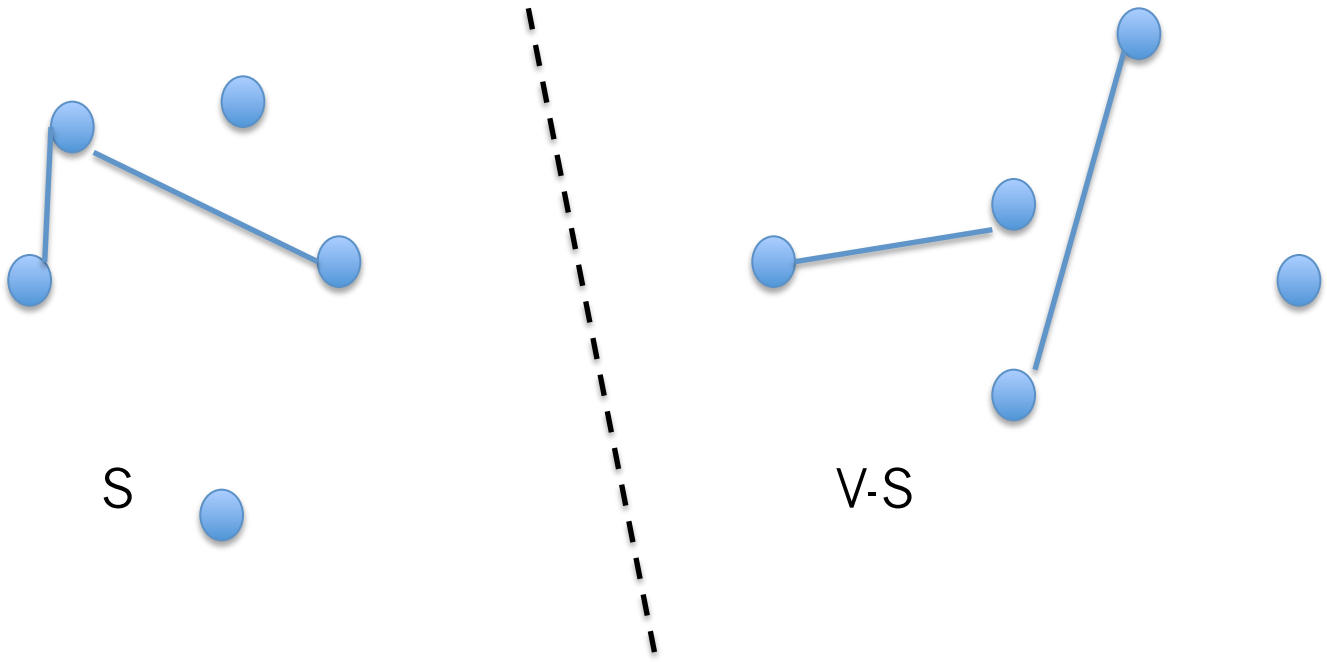
```
while ( |MST| < |V|-1 ):
    add to MST the lightest edge
    that doesn't make  cycle in MST
```

Cuts

Problem: suppose that we are in the process of building MST by adding edge-to-edge into a built-so-far MST.

At some step, we are considering a certain edge `(u,v)` to the built-so-far MST (which, of course, doesn't contain any cycle). How do we detect that adding `(u,v)` would/would not result in a cycle in MST?

Cuts



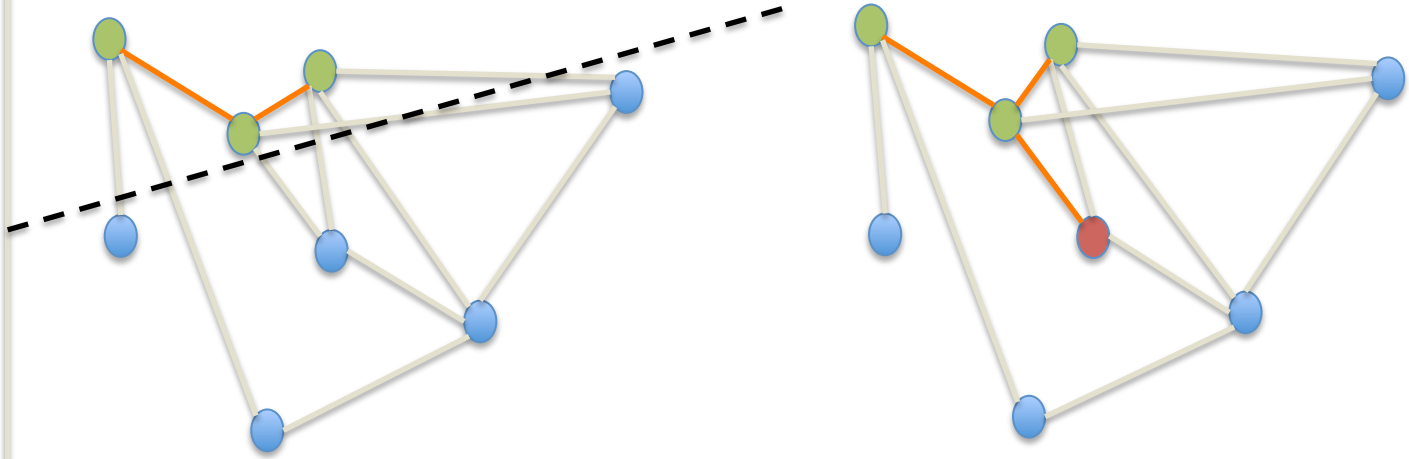
Suppose that the above 4 edges form part of a MST, and *other edges of the graph not shown*. The dash line is a cut that respects the MST: none of the edges of the MST crosses the line.

if e is the lightest edge across the cut, it's safe to add e to MST.

Prim's Algorithm

- Consider a randomly-chosen vertex as the so-far-MST.
- At each stage, expand the so-far-MST by adding a vertex to that tree (*which one? the one that is closest to the* so-far-MST).

Seems familiar?



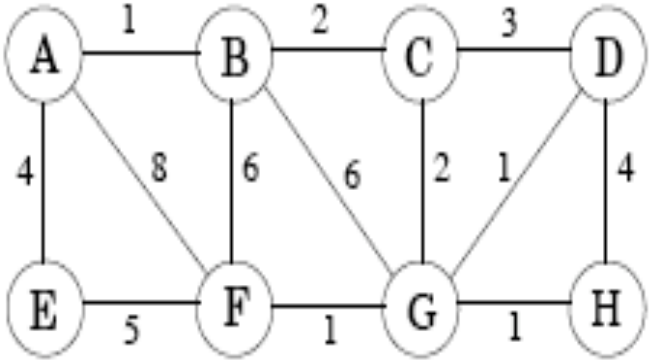
Prim's algorithm

Purpose: Find MST of **(G= (V,E), w)**, starting from **s**

```
for all u:
   set dist[u]=∞, inMST[u]= FALSE    /* MST= empty set */
dist[s]=0, pred[s]= nil
PQ= makePQ(V)       /* using dist[] as PQ weight*/
while (PQ != ∅):
   u= deleteMin(PQ)
   for each v that (u,v)∈E:
      if ( inMST[v]==FALSE && w(u,v)<dist[v]):
         dist[v]= w(u,v)
         pred[v]= u
         update dist[v] in PQ
   inMST[u]= TRUE;    /* add u to MST */
```

Example

Suppose we want to find the minimum spanning tree of the following graph.



(a) Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node *A*). Draw a table showing the intermediate values of the `cost` array.

Kruskal's algorithm (conceptual)

Purpose: Find MST of G= (V,E,w)

Prim's algorithm: processing node-by-node, ie adding a new node to MST at each step.

Kruskal's algorithm: operates edge-by-edge, at each step add an edge to MST

```
MST= emptyset of edges
for each (u,v), in increasing order of weight:
   if  ( (u,v) does not form a cycle in MST):
       add edge (u,v) to MST
```

How do we implement?

Kruskal's algorithm implementation

Implement using disjoint set:

Make $|V|$ disjoint subsets, each contains a single node of V.

If we decide to add (u,v) into the MST, then we join the respective subsets together.

If u and v belong to a same subset, we can no more add (u,v) to the MST.

The algorithm stops when only one single subset left.

We an implement subsets as **trees**. Operations:

- `makeset(u)` - return a tree that contains single u
- `find(u)` – return the root (means, ID) of the tree that contains u;
- `union(u,v)` – joins trees containing u and v.

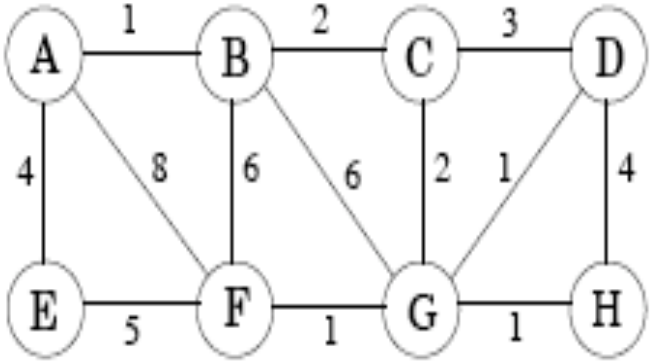
Kruskal's algorithm (conceptual)

Purpose: Find MST of (G= (V,E), w)

```
for each u: makeset(u)
set MST= empty
sort E in increasing order of weight
for each (u,v) in E:
    if  find(u) != find(v):
        add edge (u,v) to MST
        union (u,v)
```

Example: Kruskal's algorithm

Suppose we want to find the minimum spanning tree of the following graph.



Run Kruskal's algorithm on this graph. When on ties, choose edge in alphabetic order of the first vertex.

Justify the Complexity

	Prim	Kruskal
General	(E+V) log V	E log E
Dense Graph	E log V	E log E
V<<E, Prim's is faster		
Sparse Graph	V log V	V log V
Kruskal's is faster because of the data structures		

Ass2: Q&A