CSCI-4470 HW-4

Due: November 27, Friday 11:59 PM

Homework 4

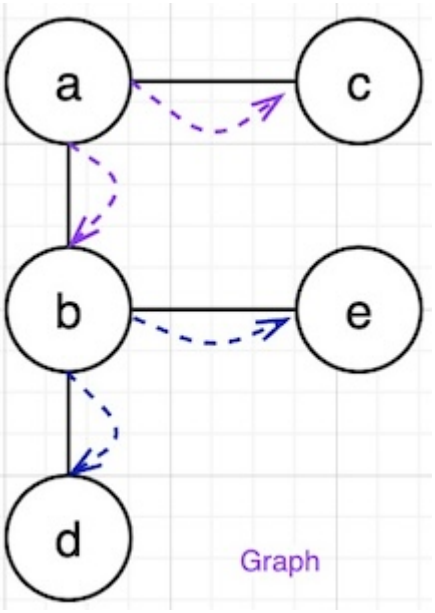
Jun Wang
ID: 811574679

Upload a soft copy of your answers **(single pdf file)** to the submission link on elc before the due date. The answers to the homework assignment should be your own individual work. Make sure to show all the work/steps in your answer to get full credit. Answers without steps or explanation will be given zero.

Extra credit: There is 5 percentage extra credit if you don't submit hand written homework including the figures. You can use latex or any other tool to write your homework. For figures you can use any drawing tool and include the figure as a jpeg or a png file in your latex file.

1. (10 points) When changing the order of the adjacency lists or vertex order in BFS, can this make $u.\pi$ change for some $u \in V$? What about $u.d$?

Justify your answer for both $u.d$ and $u.\pi$. Specifically, if the value could change, give an example of how changing the order of the adjacency list changes the value. If the value does not change, argue that the value can't change.



For Example, In BFS traversing can be in two cases

Case 1: starting at `a`, then to `b`, then to `c`, from the vertex `b` to `d`, then to `e`

the vertex order of case 1 is `a, b, c, d, e`

```
a --> b --> c, b --> d --> e
```

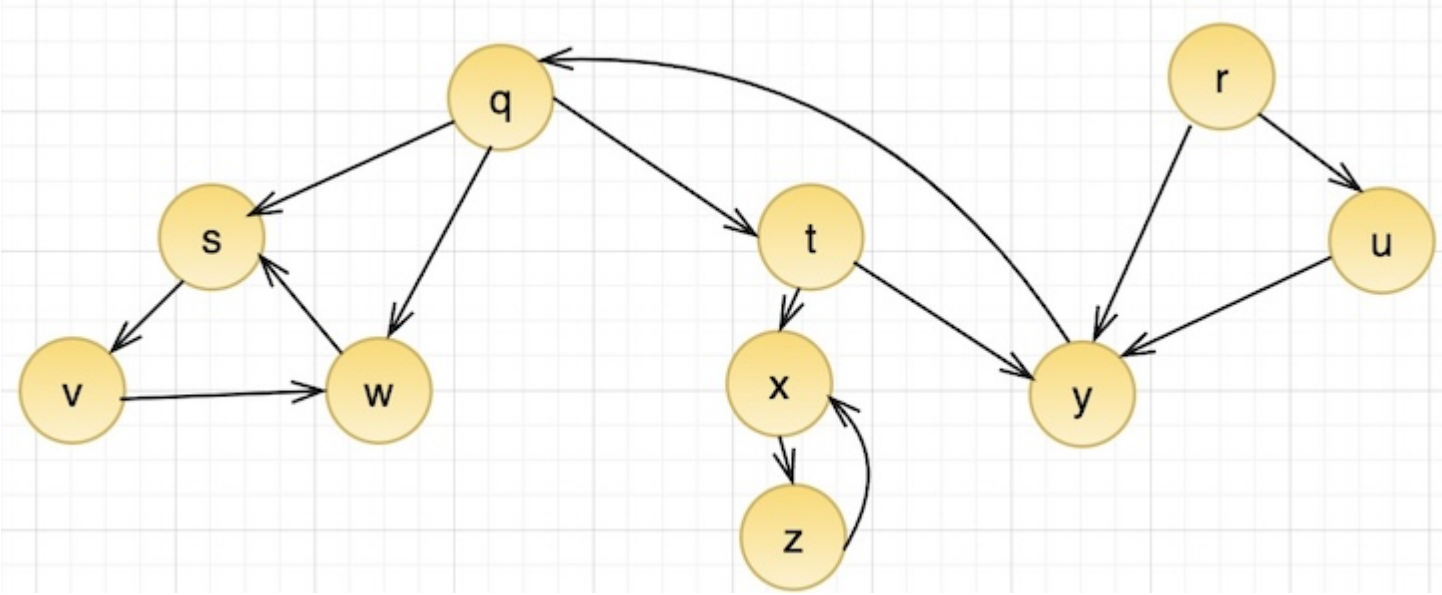
Case 2: starting at `a` to `c`, then to `b`, from `b` to `e`, then to `d`

the vertex order of case 2 is `a, c, b, e, d`

```
a --> c --> b, b --> e --> d
```

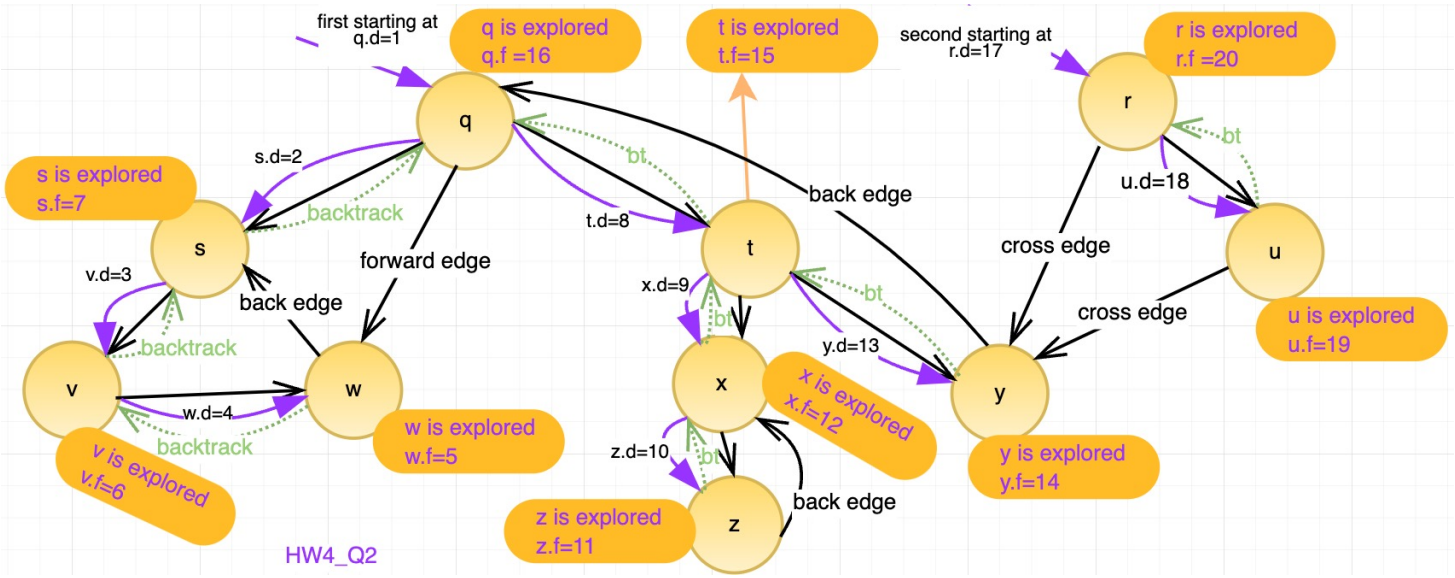
- In BFS the distance $u.d$ from the source vertex `a` to all other vertices remaining same, as BFS guarantees the shortest path in an unweighted graph.
- The $u.\pi$ of some vertices can change depending on the accessing order of the adjacency lists

2. (20 points) Show how depth-first search works on the graph of **Figure 20.6**. Assume that the for loop of ***lines 5 – 7*** of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finish times for each vertex, and show the classification of each edge.




```
5. for each vertex u ∈ G.V
6.      if u.color == WHITE
7.          DFS-VISIT(G, u)
```

- Let's use DFS traversal the graph, the first starting vertex at `q`, the second starting vertex at `r`,



Vertex	$v.d$	$v.f$
q	1	16
r	17	20
s	2	7
t	8	15
u	18	19
v	3	6
w	4	5
x	9	12
y	13	14
z	10	11

Show the classification of each edge:

- **Tree Edges =** `{(q,s), (s,v), (v,w), (q,t), (t,x), (x,z), (t,y), (r,u)}`
- **Back Edges =** `{(w,s),(z,x),(y,q)}`
- **Forward Edges =** `{(q,w)}`
- **Cross Edges =** `{(r,y), (u,y)}`

3. (15 points) The table below is related to the colors the two endpoints of an edge in a directed graph can have during the operation of DFS. For table entry (i, j) you are considering if it is possible to have an edge from a vertex with color i to a vertex with color j. For example the first entry in the table is `white/white` so can there be an edge from a white vertex to a white vertex during DFS if yes then enter the type of edge it would be **(1. tree, 2. back, 3.**

forward, or 4. cross). You can enter the appropriate number(s) in each table entry. Some entries might have multiple types of edges. If the color combination can never happen, enter **"impossible"** in that table entry.

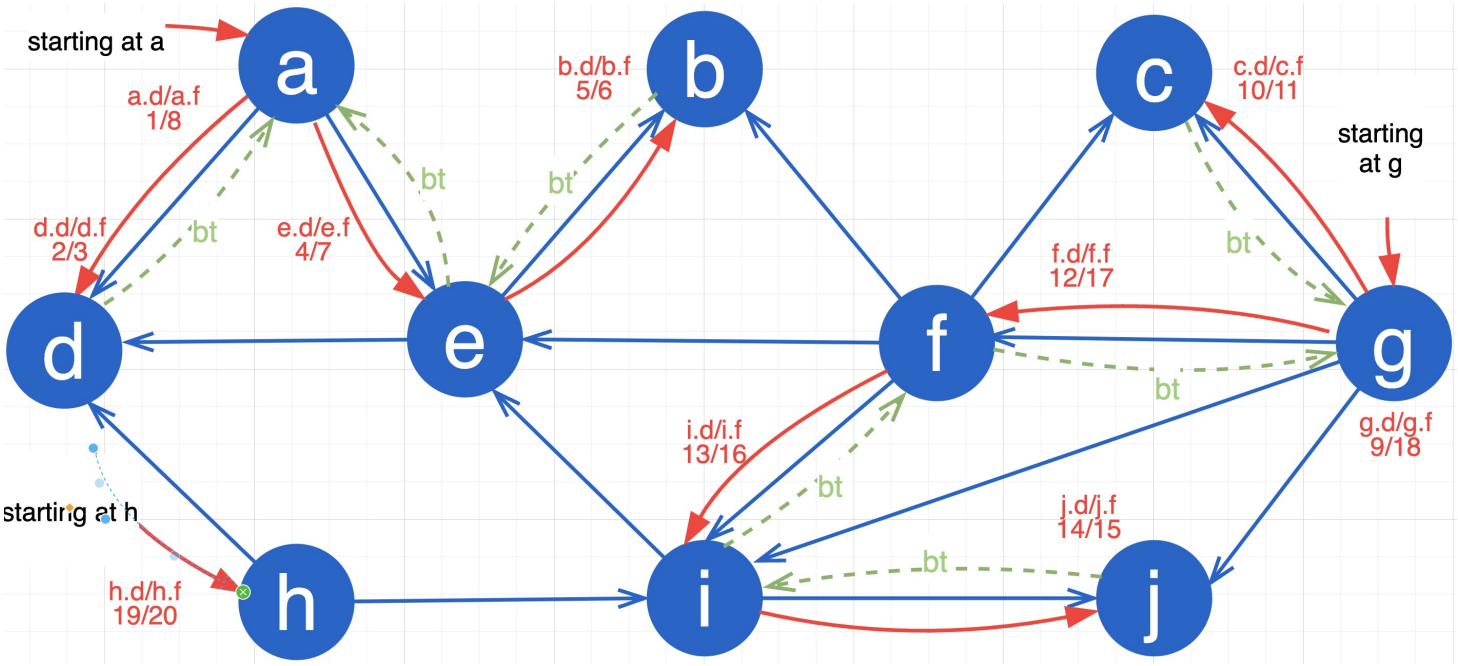
$i \setminus j$	**White**	**Gray**	**Black**
White	impossible	2	impossible
Gray	1, 3	2	3,4
Black	impossible	impossible	4

You do not need to justify your answers, but it might help with partial credit if your answer is incorrect. For justifications, I suggest you focus on discovery times and finish times

- **White to White**, Impossible. Once DFS visits a vertex, it immediately becomes gray
- **White to Gray**, Possible, it can be back edges, indicating a cycle
- **White to Black**, Impossible, In DFS Once a vertex is discovered, all reachable vertices should have already been discovered.
- **Gray to White**, Possible, can be tree edges or forward edges.
- **Gray to Gray**, Possible, can be back edges, indicating a cycle
- **Gray to Black**, Possible, can be forward edges, or cross edges.
- **Black to White**, Impossible, In DFS Once a vertex is discovered, all reachable vertices should have already been discovered.
- **Black to Gray**, Impossible, Gray vertices are currently being explored, and they should be black before any previously black vertices.
- **Black to Black**, Possible, can be cross edges.

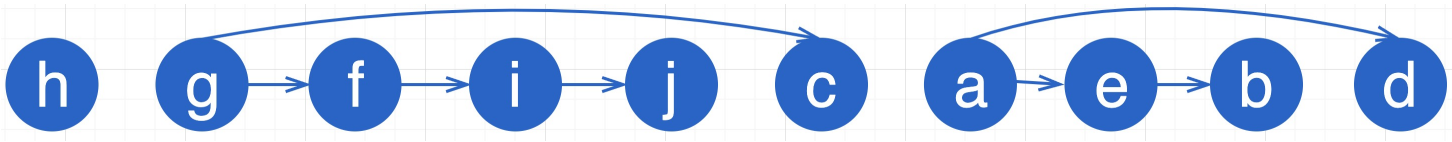
4. (15 points) Consider the following directed acyclic graph. Find a topological sorting of the vertices in this graph. When running DFS, consider all vertices and process vertices in the increasing alphabetical order from the adjacency list.

- For the Topological Sort of DAG, Consider the vertex that has no incoming edges.
  - In the graph there are 3 of them `a, g, h`



v	$v.d$	$v.f$	v	$v.d$	$v.f$
a	1	8	f	12	17
b	5	6	g	9	18
c	10	11	h	19	20
d	2	3	i	13	16
e	4	7	j	14	15

The Topological Sort: `h, g, f, i, j, c, a, e, b, d`



5. (10 points) Let's say in the algorithm for strongly connected components we use the original graph (instead of the transpose) graph in the second depth-first search and scan the vertices in order of increasing finish times.

(a) Show that algorithm can produce correct results by using a toy example. Make sure that graph is connected and has more than one component in your example.

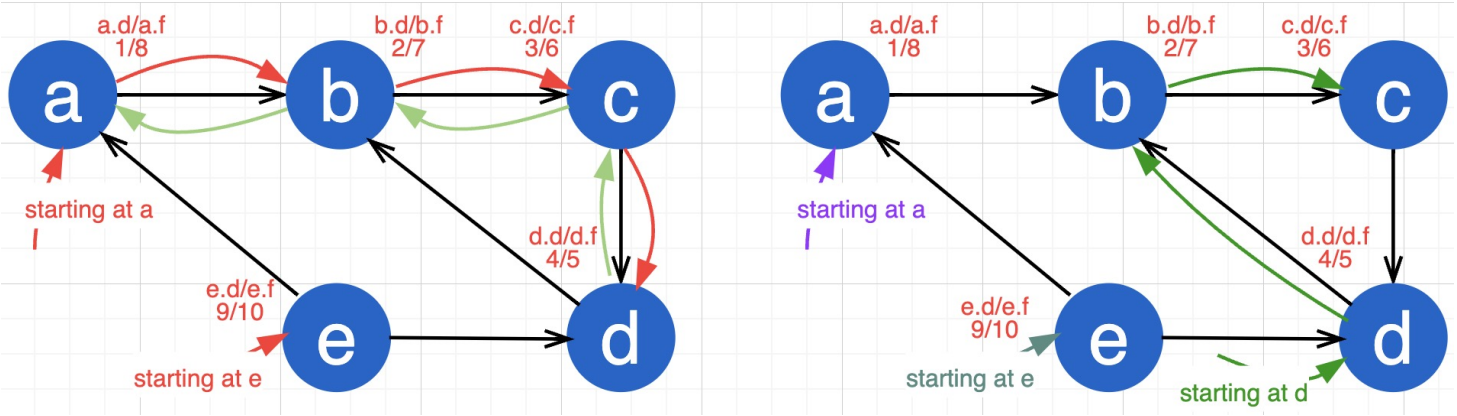
Consider a directed graph with vertices $\{a, b, c, d, e\}$ and edges $\{(a, b), (b, c), (c, d), (d, b), (e, a), (e, d)\}$.

1. First DFS(G) on Original Graph:

- Start from a, DFS order: $a \to b \to c \to d$
- e is not reachable from a, so we start a new DFS from e. (Notable, this new DFS is not Second DFS)
- DFS continues: $e \to a$ (again, but a is already visited)

- Discovery and finish times: $a(1/8), b(2/7), c(3/6), d(4/5), e(9/10)$

v	$v.d$	$v.f$	-	v	$v.d$	$v.f$
a	1	8		d	4	5
b	2	7		e	9	10
c	3	6				



2. Second DFS on Original Graph (in order of increasing finish times):

- Start from d **(finish time 5)**, then can go to vertices: b, c, and form the first SCC: $\{b, c, d\}$.
- Move to next vertex a **(finish time 8)**, No new vertices are reachable, then form the second SCC: $\{a\}$.
- Move to next vertex e **(finish time 10)**, No new vertices are reachable, then form the last SCC: $\{e\}$.

3. Result:

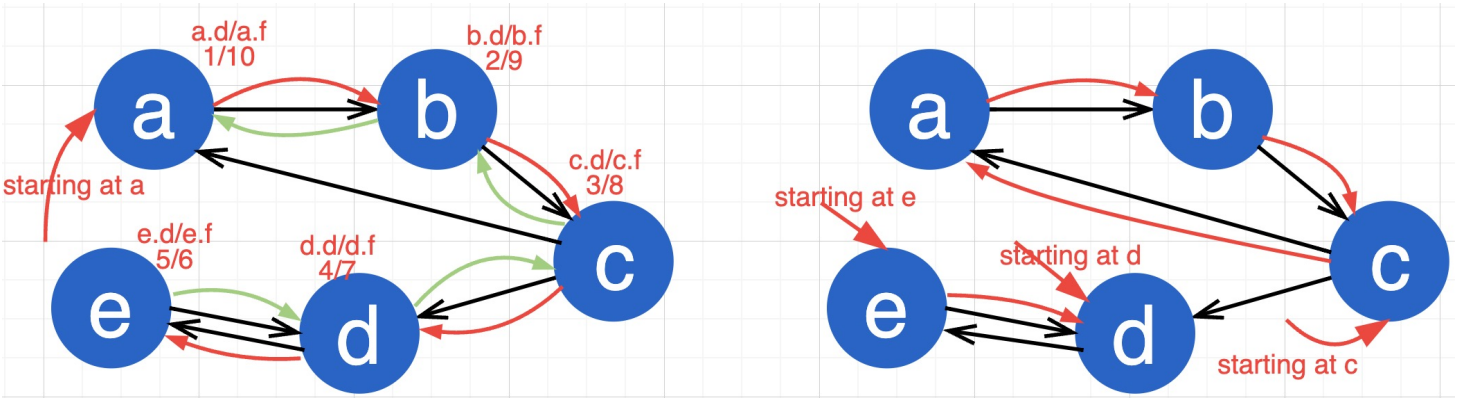
- Three SCCs are found: $\{a\}, \{b, c, d\}, \{e\}$

(b) Does it always produce the correct output? Give justification to your answer or you can again use a toy example to show that it does not work in all cases.

Consider a directed graph with vertices $\{a, b, c, d, e\}$ and edges $\{(a, b), (b, c), (c, a), (c, d), (d, e), (e, d)\}$.

First DFS on Original Graph:

1. Start from a, DFS order: $a \to b \to c \to d \to e$
   - Discovery and finish times: $a(1/10), b(2/9), c(3/8), d(4/7), e(5/6)$



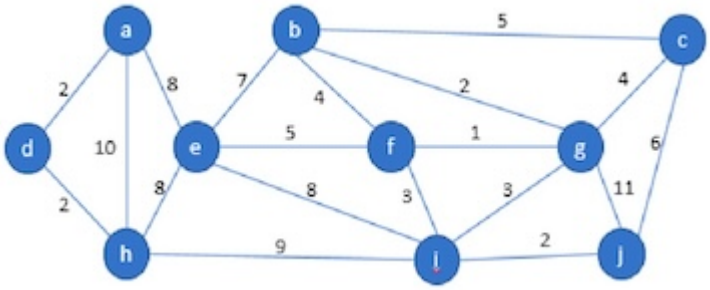
Second DFS on Original Graph (in order of increasing finish times):

2. Start from e **(finish time 6)**, can go the vertex d, then form the first SCC: $\{d, e\}$
3. Move to next vertex d **(finish time 7)**, All vertices in its SCC are already visited.
4. Move to next vertex c **(finish time 8)**, can go to vertices: a, b, d, e, then form the SCC: $\{a, b, c, d, e\}$

Result:

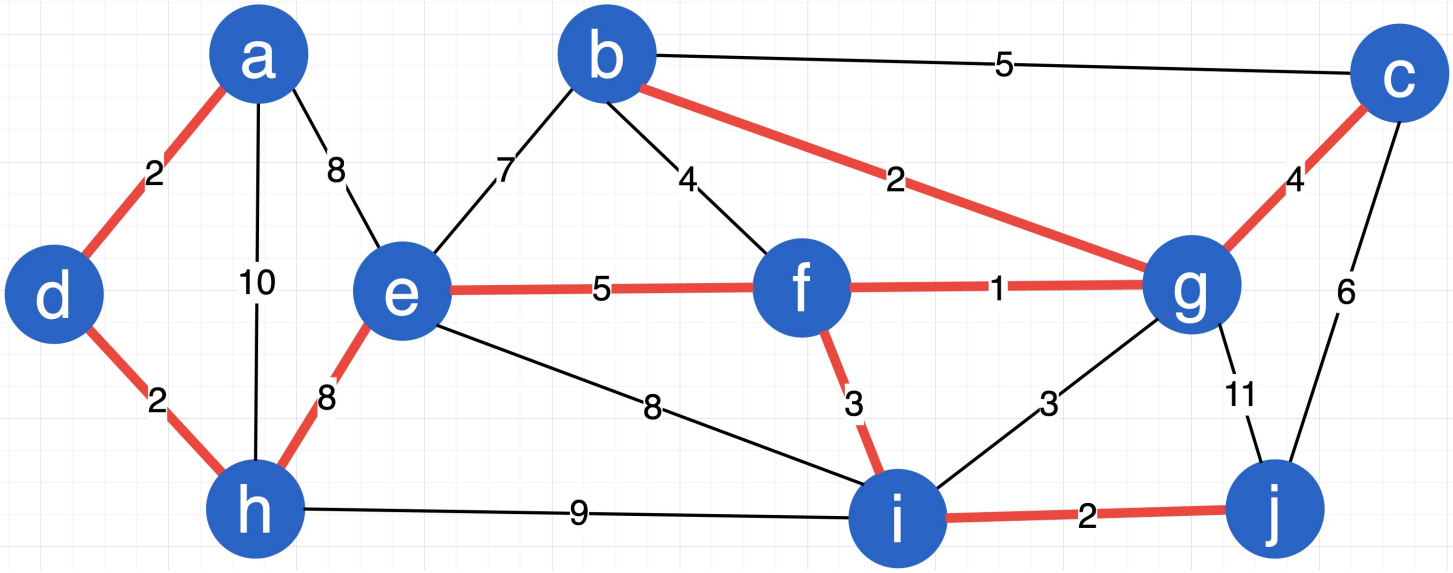
- The algorithm incorrectly identifies the entire graph as a single strongly connected component.

6. (10 points each) Consider the graph G shown below. Find 2 different minimum spanning trees for G – one using Kruskal's algorithm and the other using Prim's. Draw MST in both cases and also show list the order in which edges were added to the MST.



Finding MST By Kruskal's Algorithm:

- Select first $|V| - 1$ edges which do not generate a cycle
- Select the weight of edges from lowest to highest, and avoid the cycle, starting with the lowest weight `1`, then `2`, and go on

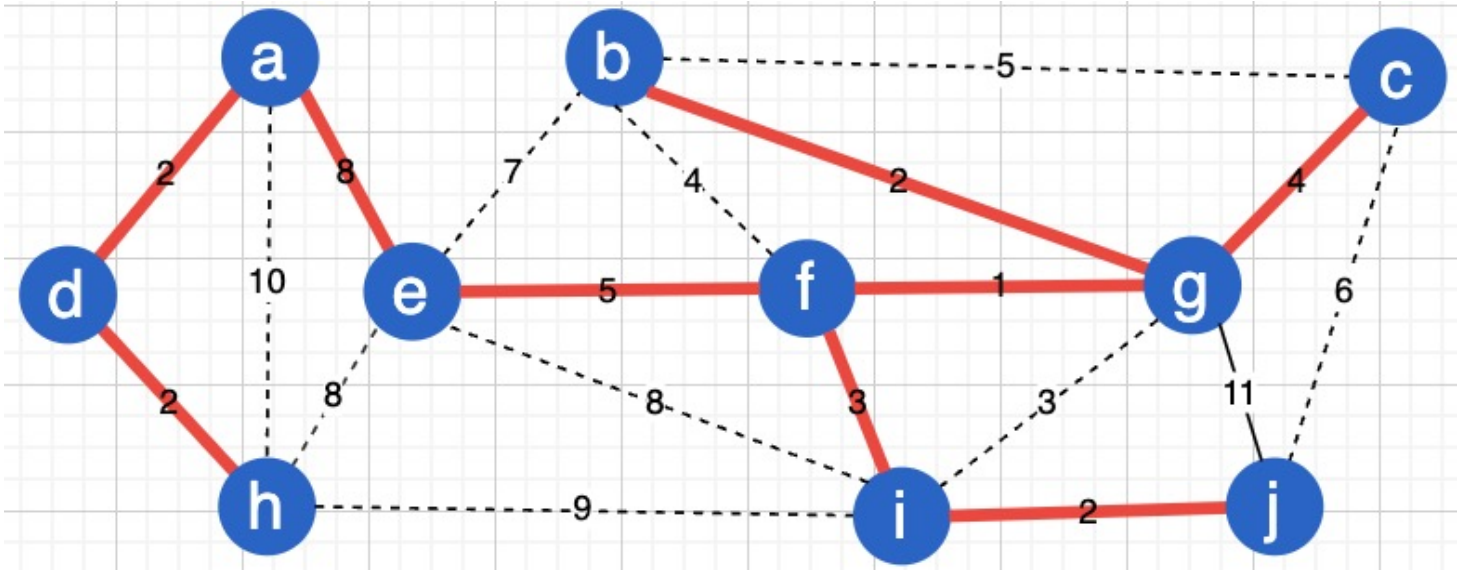


Edge	d_v	√ ✕	-	Edge	d_v	√ ✕
(f, g)	1	√		(e, f)	5	√
(a, d)	2	√		(c, j)	6	✕
(b, g)	2	√		(b, e)	7	✕
(d, h)	2	√		(a, e)	8	✕
(i, j)	2	√		(e, h)	8	√
(f, i)	3	√		(e, i)	8	✕
(g, i)	3	✕		(h, i)	9	✕
(b, f)	4	✕		(a, h)	10	✕
(c, g)	4	√		(g, j)	11	✕
(b, c)	5	✕				

- **Total Weight:** $1 + 2 + 2 + 2 + 2 + 3 + 4 + 5 + 8 = 29$

Finding MST By Prim's Algorithm:

Step 1: Pick starting vertex `a` , find the minimum between `(a,b,2)` and `(a,e,8)` select (a,d,2) which weight is 2, and go on…



v	**Key**	π
a	0	NIL
b	~~∞~~, ~~7~~, 2	~~e~~, g
c	~~∞~~, 4	g
d	~~∞~~, 2	a
e	~~∞~~, 8	a
f	~~∞~~, 5	e
g	~~∞~~, 1	f
h	~~∞~~, ~~10~~, 2	~~a~~, d
i	~~∞~~, ~~9~~, ~~8~~, 3	~~h~~, ~~e~~, f
j	~~∞~~, ~~11~~, 2	~~g~~, i

- **Total Weight:** $1 + 2 + 2 + 2 + 2 + 3 + 4 + 5 + 8 = 29$

7. (10 points) Let (u, v) be a minimum-weight edge in a connected $graph\ G$. Prove that (u, v) belongs to some minimum spanning tree of G.

Proof:

1. Assume edge (u, v) isn't in an MST, called T.
2. Add (u, v) to T. This creates a cycle since T was already spanning.
3. In the cycle, there's an edge (x, y) with weight $\geq w(u, v)$. Remove (x, y).
4. The new tree is still spanning and has weight $\leq T$, contradicting T being an MST.

Thus, (u, v) must be in some MST.