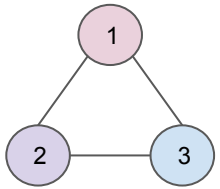
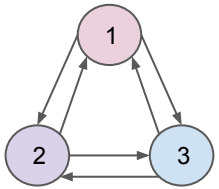
GRAPH DEFINITIONS



The maximum number of arcs in an undirected graph is (n * (n - 1)) / 2
Graph of 3 nodes
3 * (3-1) = 6 / 2 = 3



The maximum number of arcs in a digraph or order n is n * (n - 1)
Digraph of 3 nodes
3 * (3-1) = 6

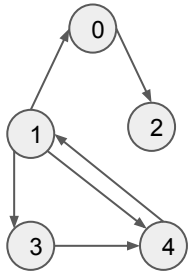
The minimum number of arcs in any graph is 0

Walk: A sequence of nodes where each node is connected to the next one by an edge.
Path: A walk with no repeated nodes.
Cycle: A walk where the only repeated nodes are the first and last, and the walk has at least 3 different nodes involved.

Order: The number of nodes
Size: The number of arcs (minimum size = 0, maximum size = n(n-1))

Sparse: Smaller size O(n)
Dense: Bigger size $O(n^2)$

Space Requirements
Matrix = $\Theta(n^2)$
List = $\Theta(n+m \log n)$



0	0	**1**	0	0
1	0	0	**1**	**1**
0	0	0	0	0
0	0	0	0	**1**
0	**1**	0	0	0

0: 2
1: 0,3,4
2:
3: 4
4: 1

	Matrix	Details	List	Details
Checking if an arc exists	$\Theta(1)$	Directly access the entry in the matrix at row u and column v to check if 1	$\Theta(d)$	Search through the list of neighbors of node u to see if v is present.
Outdegree of node	$\Theta(n)$	Count the number of 1s in the row corresponding to node u.	$\Theta(1)$	Directly return the length of the list of neighbors of node u.
Indegree of node	$\Theta(n)$	Count the number of 1s in the column corresponding to node v.	$\Theta(n+m)$	Search through all lists to count occurrences of node v. With a maintained count, you can return it directly.
adding an arc	$\Theta(1)$	Set the entry in the matrix at row u and column v to 1.	$\Theta(1)$	Append node v to the list of neighbors of node u
deleting an arc	$\Theta(1)$	Go to the column i, turn all 1's into 0,	$\Theta(d)$	Locate source node, scan through this list to find the target node and remove it.
adding a node	$\Theta(n)$	Requires creating a new matrix with one additional row and column and copying the existing entries	$\Theta(1)$	Add a new list (initially empty) for the new node.
deleting a node	$\Theta(n^2)$	Requires creating a new matrix without the row and column of the deleted node and copying the remaining entries.	$\Theta(n+m)$	Remove the list of the node and remove all occurrences of the node from other lists. In the worst case, this involves scanning through all lists.

The running time of the operation of determining if an arc (i, j) is present in a digraph of order n and size m using an adjacency list is

A. $\Theta(d_j)$ where d_j is the in-degree of node j
B. $\Theta(d_i)$ where d_i is the out-degree of node i
C. $\Theta(n + m)$
D. $\Theta(n)$

We need to search for j in list i. The complexity then depends on the length of list i. List i is length d where d is the outdegree of node i so searching for j is in Θ(d).

Hence, Answer is B.

Consider the following digraph given as adjacency lists. What is the first **column** (corresponding to node 0) in the adjacency matrix representation of the digraph?

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

0	1	1	1	0	0
0	1	0	0	0	0
1	1	0	0	1	0
0	1	1	0	0	5
1	1	0	1	0	0
1	0	0	0	0	0

Therefore, node 0's list is
0
0
1
0
1
1

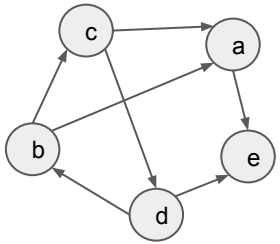
ARC DEFINITIONS

Simple Traverse:

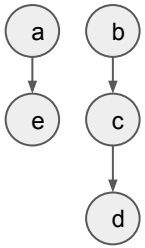
- All nodes are white to begin with
- A staring white node is chosen and turned grey
- A grey node is chosen and its out-neighbours explored
- If any out-neighbours is white, it is visited and turned grey. If no out-neighbours are white, the gray node is turned black.
- The process of choosing grey nodes and exploring neighbours is continued until all nodes reachable from the initial node are black.
- If any white nodes remain in the digraph, a new starting node is chosen and the process continues until all nodes are black.

Predecessor Array:

a	b	c	d	e
-1	-1	b	c	a



Search Forrest:



1. tree arc: All traversed edge **{a,e}, {b,c}, {c,d}**
2. forward arc: X appears before Y, direct path from x-y (is not tree arc) **{b,d}**
3. back arc: Y appears before X, direct path from y-x (is not tree arc) **{d,b}**
4. cross arc: X/Y appears before Y/X, direct path from y-x/x-y (is not tree arc) **{c,a}**

An edge (u,v) is a tree edge if v is a direct descendant of u
An edge (u,v) is a forward edge if v is a descendant of u but v was already discovered when u explores it.
- **Distinction from Tree Edge: v is not discovered directly by u but rather through another descendant of u.**

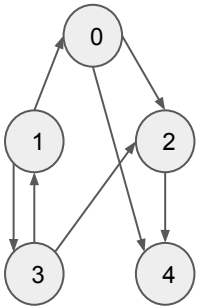
An edge (u,v) is a back edge if v is an ancestor of u
An edge (u,v) is a cross edge if v is neither an ancestor nor a descendant of u.

DEPTH FIRST SEARCH

DFS

- Visit the starting node and mark it as visited.
- From the current node, go to an unvisited neighbor node, mark it as visited, and continue this process.
- If you reach a node with no unvisited neighbors, backtrack to the previous node and continue the process from there.

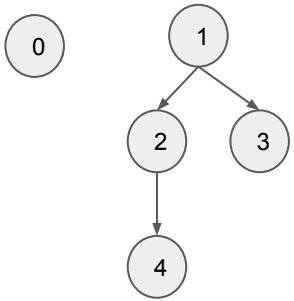
Θ(n+m) Assuming we are using an adjacency list representation.



(v,w) is a tree arc OR a forward arc
seen[v] < seen[w] < done[w] < done[v]
and v is an ancestor of w

(v,w) is a back arc
seen[v] < done[v] < seen[w] < done[w]
and v is NOT an ancestor of w

(v,w is a cross arc)
seen[w] < done[w] < seen[v] < done[v]



seen	0	6	1	7	2
done	5	9	4	8	3
pred	-1	-1	0	1	2

The following are seen and done arrays from a DFS of a digraph with node labels 0,1,2,3,4,5:

$$seen = [0, 1, 2, 9, 3, 6] \text{ and } done = [11, 8, 5, 10, 4, 7].$$

Is the arc $(1, 4)$ a

A. tree arc
B. cross arc
C. back arc
D. forward arc

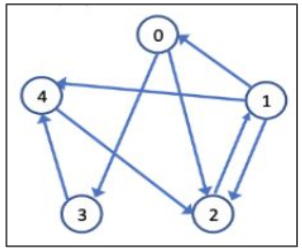
node	**0**	**1**	**2**	**3**	**4**	**5**
seen	0	1	2	9	3	6
done	11	8	5	10	4	7

For (1,4)
Seen node 1 at time 1, Done node 1 and time 8
Seen node 4 at time 3, Done node 4 at time 4

seen[1] < seen[4] < done[4] < done[1]
Since we saw 1 first, then saw 2, and then 4, 4 is not discovered directly by 1 but rather through another descendant of 1. Hence, (1,4) is a forward arc.

Perform a DFS search on the digraph shown starting at node 0 and picking the node with the lowest index when there is a choice. The arc (3,4) is a

A. back arc
B. tree arc
C. cross arc
D. forward arc

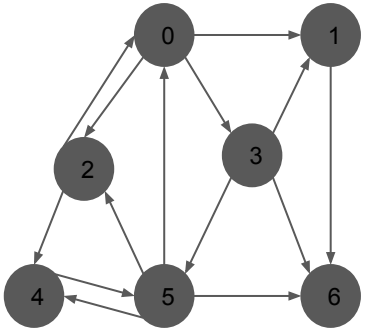


node	**0**	**1**	**2**	**3**	**4**
seen	0	2	1	7	3
done	9	5	6	8	4

For (3,4)
Seen node 3 at time 7, Done node 1 and time 8
Seen node 4 at time 3, Done node 4 at time 4

seen[4] < done[4] < seen[3] < done[3]
Hence, (3,4) is a back arc

BREADTH FIRST SEARCH



Queue: ~~0 1 2 3 6 4 5~~

BFS runs by processing nodes at distance 1, then all nodes at distance 2, etc from the root.

node	0	1	2	3	4	5	6
depth	0	1	1	1	2	2	2
pred	-1	0	0	0	2	3	1

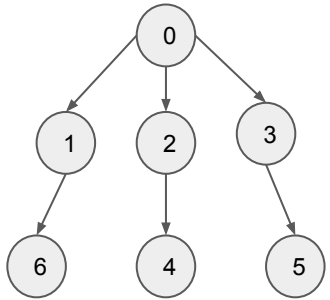
BFS does not produce forward arcs.

(v,w) is a tree arc
colour[w] = White and d[w] = d[v] + 1
The levels of w and v differ by 1

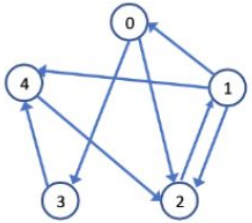
(v,w) is a back arc
colour[w] = Black and d[w] <= d[v] - 1

(v,w) is a cross arc
d[w] < d[v] - 1 and colour[w] = Black
d[w] = d[v] and colour[w] = Black or Grey
d[w] = d[v] - 1 and color[w] = Black or Grey
The levels of w and v are the same OR differ by 1

Find a tree arc, cross arc, forward arc, and back arc.
tree arc = {0,1} {0,2} {0,3} {1,6} {2,4}
cross arc = {0, 2} {5,6} {2,3} {1,2}
back arc = {2,0}
no fowad arcs exist in BFS



If BFS is run starting at node 4 in the digraph shown, what is the third node to turn grey? Where there is a choice of nodes, choose the one with the lowest index.



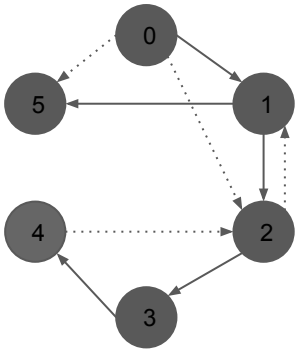
A. 1
B. 2
C. 0
D. 3

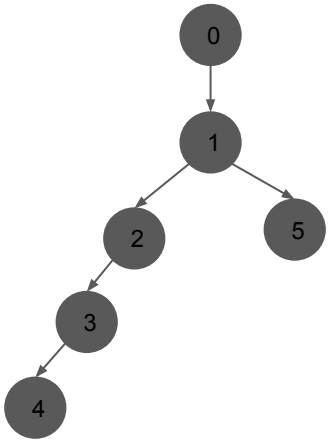
Queue: ~~4~~ ~~2~~ 1

node	0	1	2	3	4
depth		2	1		0
pred		2	4		-1

DFS vs BFS

DFS Starting at Node 0:





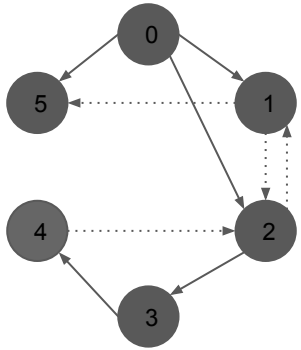
node	0	1	2	3	4	5
seen	0	1	2	3	4	8
done	11	10	7	6	5	9
pred	-1	0	1	2	3	1

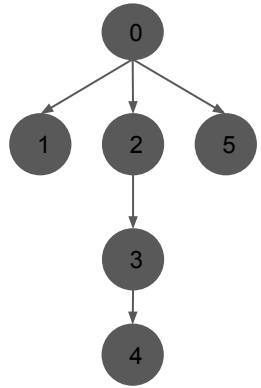
Tree arcs:
{0,1}, {1,2} {2,3} {3,4} {1,5}

Back arcs:
{4,2} {2,1}

Forward arcs:
{0,2} {0,5}

BFS Starting at Node 0:





node	0	1	2	3	4	5
depth	0	1	2	3	4	1
pred	-1	0	0	2	3	0

Tree arcs:
{0,1}, {1,2} {2,3} {3,4} {0,5}

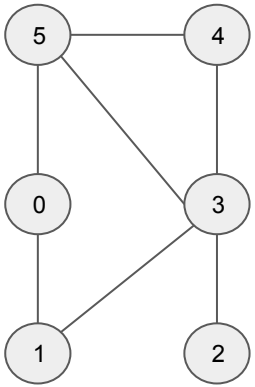
Cross arcs:
{1,5} {1,2} {2,1}

Back arcs:
{4,2}

PRIORITY FIRST SEARCH

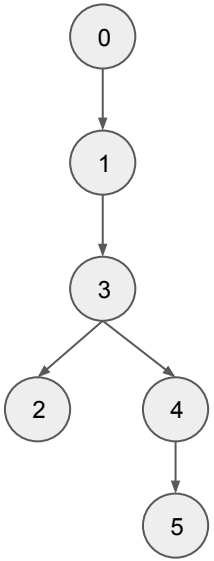
Each grey node is associated with an integer key. The key represents the priority for the node, were a lower key represents a higher priority, and a higher key represents a lower priority. Think of a key as the amount of time the node is willing to wait, a lower key means its impatient, and a higher key means its patient.

Takes $\Omega(n^2)$



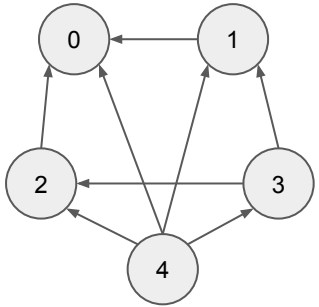
Use setkey(v) = -index(v)

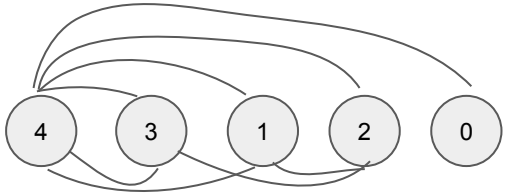
Queue	0	1	3	2	4	5
Key	0	-1	–3	-2	-4	-5



TOPOLOGICAL SORT

A Topological sort of a digraph G is a linear ordering of the nodes of G, such that if u comes before v in the ordering, (v,u) != E(G)
A graph cannot be topologically sorted if it contains a cycle.
Running time: O(n+m)





1. Find a source node

node 4 is the only source node in G

2. Find nodes that 4 points to (2, 0, 1 or 3)

Let's pick node 3

3. Finds nodes that 4 OR 3 point to (2, 0, 1)

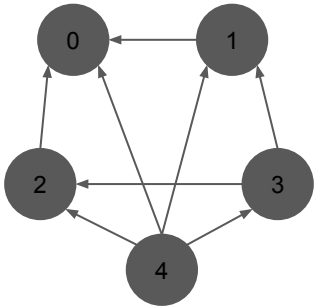
Lets pick node 1

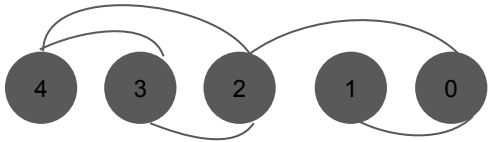
4. Finds nodes that 4, 3 OR 1 points to

Lets pick node 2

5. Only one node is left, hence add Node 0

A digraph with NO cycles is a DAG for a Directed Acyclic Graph.
A Digraph has a topological order if and only if it is a DAG.
If G is a DAG, listing nodes in reverse order of DFS finishing times is a topological sort.

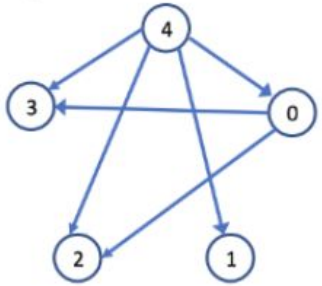




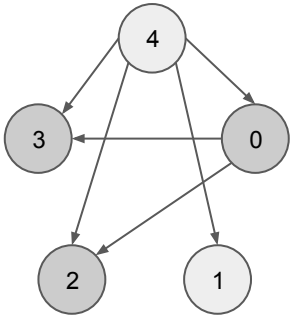
node	0	1	2	3	4
seen	2	1	5	0	7
done	3	4	5	6	8
pred	1	3	3	-1	-1

Hence, listing the 'done' times in descending order, we get 4, 3, 2, 1, 0

What is the topological order of the digraph shown found by using DFS (starting at 0, picking the lowest index where there is a choice).



A. 4,0,3,1,2
B. 4,0,3,2,1
C. 4,1,0,2,3
D. 4,1,0,3,2



node	0	1	2	3	4
seen	0	7	1	3	6
done	5	8	2	4	9
pred	-1	4	0	0	-1

Sorting the 'done' times in descending order, we get: 4, 1, 0, 3, 2

GIRTH & CONNECTIVITY

GIRTH

Girth: Length of shortest cycle in a undirected graph
Directed Girth: Length of shorted cycle in a directed graph

- If a graph is acyclic, the girth of G is undefined (0 or ∞)

We can use BFS to find the shortest cycle (directed girth) in a cyclic digraph n(n+m).

Simply run BFS, and the first time a back arc is found, we have found a cycle of length (depth +1). We run this for all the nodes and return the minimum length.

CONNECTIVITY

A graph is connected if for each pair of nodes, there is a path from u to v.
We can find the number of connected components by running BFS or DFS. Run it again for every starting node and counting how many trees we have.

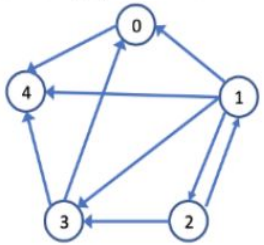
Running time = ϴ(n(n+m))

STRONG CONNECTIVITY

A digraph G is strongly connected if for each pair of nodes, there is a path from u to v AND from v to u.
Strong components:

- If there is a cycle of 2 (u→ v and v→u)
- If you start at a node and cannot get back

How many strong components does the digraph below have?



A. 2
B. 3
C. 1
D. 4

{1,2} and {2,1} are strongly connected.
Node 4: No way to get back to node 3
Node 0: No way to get back to node 3
Node 3: No way to get back to node 3

Hence, there are 4 strongly connected components in G.

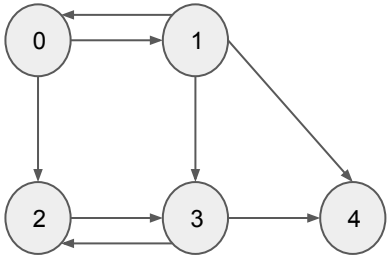
TARJAN'S ALGORITHM

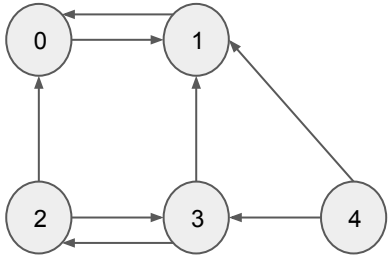
Finding the number of strongly connected components in a digraph

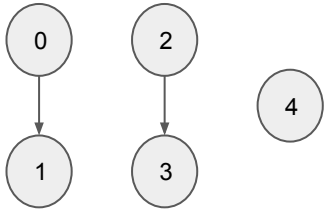
Tarjan's Algorithm

1. Run DFS on G
2. List nodes of G in reverse order of done times (last finished to first)
3. Run DFS on Gr choosing root nodes in order of list from STEP 2

This produces a forest where trees correspond to strong components of G.







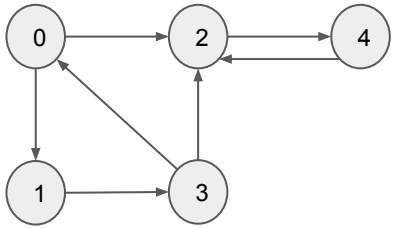
Hence there is 3 strongly connected components.

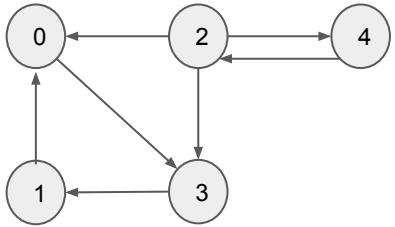
node	**0**	**1**	**2**	**3**	**4**
seen	0	1	7	2	3
done	9	6	8	5	4

node	**0**	**1**	**2**	**3**	**4**
seen	0	1	4	5	
done	3	2	7	6	

Done times = 0, 2, 1, 3, 4

Tarjan's Algorithm – Extra Example



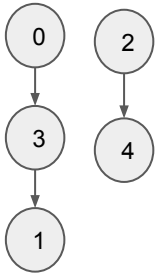


node	0	1	2	3	4
seen	0	1	3	2	4
done time	9	8	6	7	5

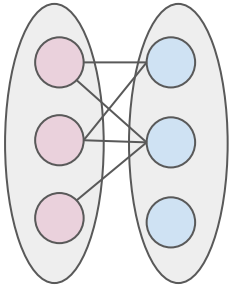
Done order: 0, 1, 3, 2, 4

node	0	1	2	3	4
seen	0	3	7	2	8
done time	6	4	10	5	9

Final number of strongly connected components = 2



BIPARTITE GRAPHS



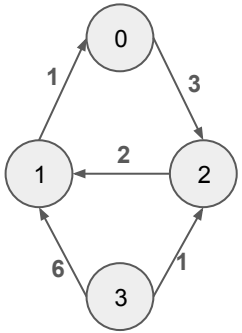
A Graph is bipartite if the nodes can be partitioned into two non-empty, disjoint sets, such that each edge has one endpoint.

Properties of BIPARTITE GRAPHS
G does NOT contain an odd length cycle

After running BFS, we get a tree-like structure with:
- Tree arcs (on the same level or differ by one)
- Cross arcs on the DIFFERENT level

WEIGHTED DIGRAPHS

Adjacency Matrix for Weighted Digraphs



Distance:

The distance from node 1 to node 2 = 1 + 3 = 4
The distance from node 2 to node 1 = 2

The distance from node 3 to node 0
- Path 1 $3 \rightarrow 1 \rightarrow 0$ = 6 + 1 = 7
- Path 2 $3 \rightarrow 2 \rightarrow 1 \rightarrow 0$ = 1 + 2 + 1 = 4

	0	1	2	3
0	0	0	0	**3**
1	**1**	0	0	0
2	0	**2**	0	0
3	0	**6**	**1**	0

Matrix Modification:
The out neighbours are now represented by their weights instead of a binary variable 1.

0: 2 3

1: 0 1

2: 1 2

3: 1 6 2 1

List Modification:
The out neighbours are now represented by the node they points too followed by the weight.

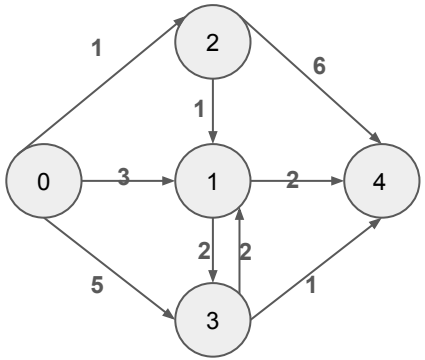
Distance Matrix

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

DIJKSTRA'S ALGORITHM

Finding the shortest path in a cyclic weighted digraph with NO negative weights, from a source node

1. **Get source node, and calculate its distance matrix row.**
2. **Find the minimum distance node, and find its distance matrix row**
3. **Repeat**



Node 0

Node	0	1	2	3	4
Dist	0	3	1	5	∞

Node 0 → 2

Node	0	1	2	3	4
Dist	0	2	1	5	7

Node 0 → 2 → 1

Node	0	1	2	3	4
Dist	0	2	1	4	4

Node 0 → 2 → 1 → 3

Node	0	1	2	3	4
Dist	0	2	1	4	4

Diameter: The diameter is the maximum length of the shortest paths between any pair of vertices in the graph.

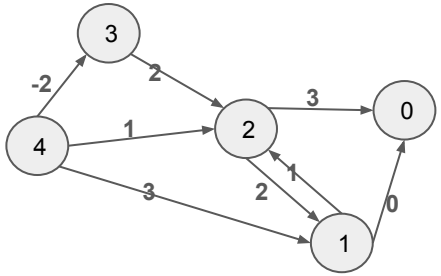
Running Time: n^2
Using a Priority Queue = Θ(n+m logn)

BELLMAN-FORD ALGORITHM

Dealing with negative weights

Running time = Θ(n*m)

Set node 4 as Source

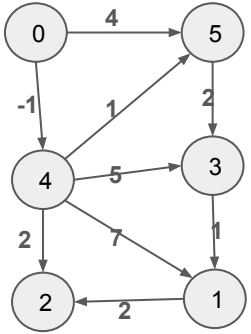


node	0	1	2	3	4
dist	∞	∞	∞	∞	0

node	0	1	2	3	4
dist	2	2	0	-2	0

If there is a negative weight cycle present, there does not exist a minimum path (Undefined).

Run bellman ford run n+1 iterations to indicate the presence or absence of a negative weight cycle. Any change in the Dist array will indicate the presence of a negative weight cycle.



node	0	1	2	3	4	5
dist	0	∞	∞	∞	∞	∞

node	0	1	2	3	4	5
dist	0	3	-1	2	-1	0

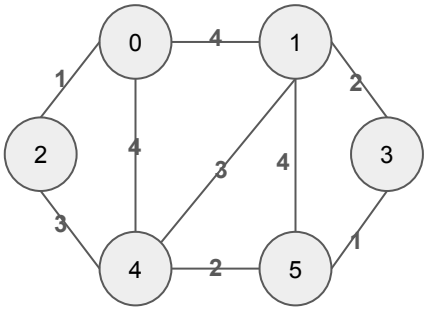
After the second interaction, nothing changed, hence, this must be the shortest possible traversal.

FLOYD'S ALGORITHM

All Pairs Shortest Paths Problem: finding the shortest path in-between every pair of vertices in a weighted graph. This means determining the minimum weight or cost required to travel from any node to any other node in the graph.

DIJKSTRA: n m log n
BELLMAN FORD: n^2m
FLOYD'S: n^3

1. Build the cost matrix
2. Compare for x=0, x=1, x=2, x=3, x=4, and x=5
   - Whether or not the BOX is greater than x + column of interest
   - For example, in first iteration, ∞ is greater than 4 + 1, hence that box is updated to 5. Then ∞ is greater than 1 + 4, hence that box is updated to 5
   - In the next iteration we use x=1 as the benchmark. (Column 1)
   - IN OTHER WORDS: Is there a path from node (row) to (column), using node x



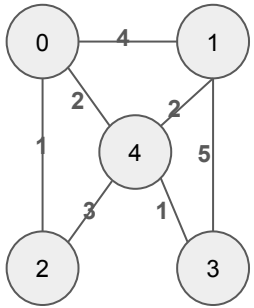
Cost Matrix

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

Distance Matrix

	0	1	2	3	4	5
0	0	4	1	6	4	~~8~~ 6
1	4	0	5	2	3	4
2	1	5	0	~~7~~ 6	3	9
3	6	2	~~7~~ 6	0	~~5~~ 3	1
4	4	3	3	~~5~~ 3	0	2
5	8	4	9	1	2	0

Extra Example for Floyd's Algorithm for the All Pairs Problem



Cost Matrix

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

After Floyd's Algorithm

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

ALGORITHM SUMMARY

	SSSP	**APSP**	**NEGATIVE WEIGHTS**
BFS	For weighted graphs no. But for unweighted graphs, yes with a time of n+m	For weighted graphs no. But for unweighted graphs, yes with a time of n(n+m)	No
DIJKSTRA	Yes, (m log n) with a primary queue utilising a heap structure.	Yes. n*m logn	No
BELLMAN FORD	Yes. slower runtime than dijkstra. n*m	Yes. n^2*m	Yes
FLOYD	Yes. n^3	Yes. n^3	Yes

Generally, unweighted digraph (dense) = Dijkstra
For SSSP or APSP, use bellman ford on a sparse digraph, but for dense digraphs use Floyd.

PRIM'S ALGORITHM
KRUSKAL ALGORITHM

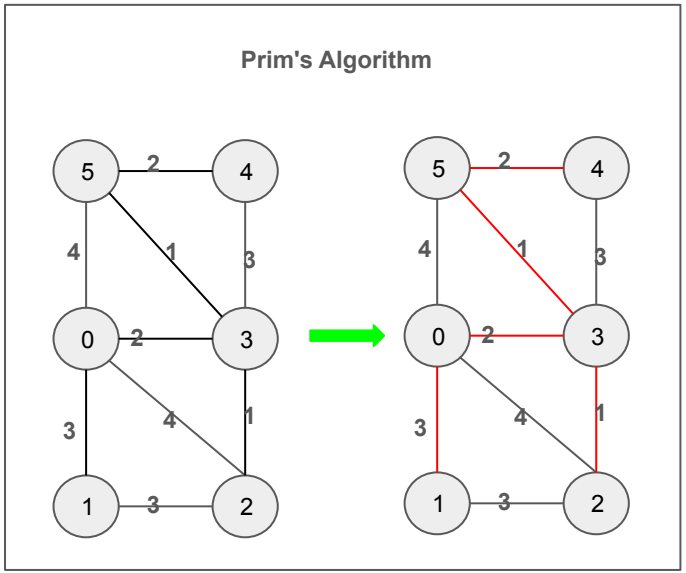
A spanning tree of a graph G is a subgraph of G that contains all does of G, and is a connected acyclic graph.
A minimum spanning tree is a spanning tree that which has the minimum total weight (sum of all edge weights)

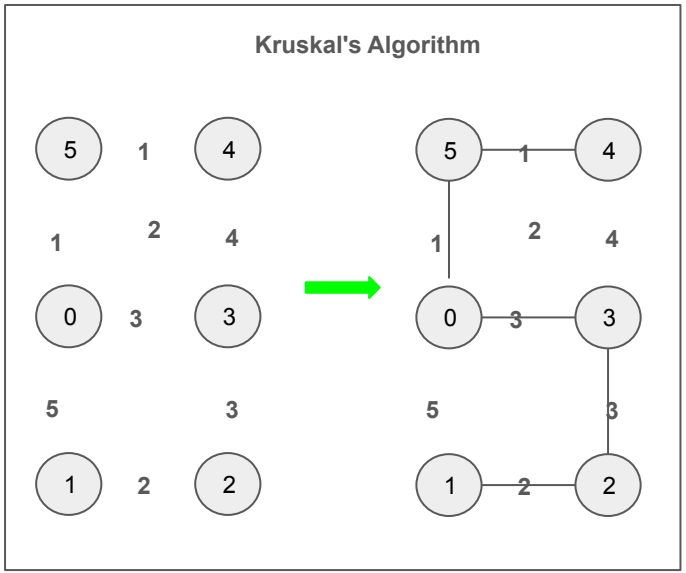
Prim's Algorithm (m log n):

- Choose a starting node
- Find it's neighbours and traverse the arc with the lowest weight as long as connecting this node does not produce a cycle
- find the neighbours and traverse the arc with the lowest weight of the last node visited and continue

KRUSKAL Algorithm (m log n):

- Start with an empty set of edges
- At each step choose an edge of minimum weight from the remaining edges ensuring that adding the edge does not create a cycle in the subgraph built for far
- stop when the subgraph is a spanning tree.





SEMESTER ONE 2023

18. Consider an algorithm takes as input a digraph $G = (V, E)$ where $|E| = m$ and $|V| = n$. The algorithm finds the in-degree of all nodes and then removes the node with the smallest in-degree. After removing that node, the remaining node with the smallest in-degree is removed, and so on until all nodes have been removed. All nodes are listed in the order of removal. If the digraph is represented using adjacency lists, and you can cache interim results, what is the most precise characterisation of the running time of the algorithm?

A. $\Theta(n(n+m))$
B. $\Theta(n^3)$
C. $\Theta(n^2)$
D. $\Theta(n+m)$

Search through all lists to count occurrences of node v. With a maintained count, you can return it directly. (n+m)

Answer = D

19. Consider the following digraph given as adjacency lists. What is the first **column** (corresponding to node 0) in the adjacency matrix representation of the digraph?

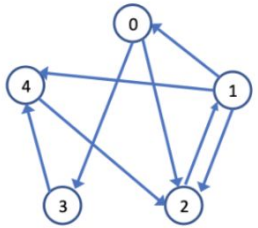
0	1 2 3
1	1
2	0 1 4
3	1 2 5
4	0 1 3
5	0

0	1	1	1	0	0
0	1	0	0	0	0
1	1	0	0	1	0
0	1	1	0	0	1
1	1	0	1	0	0
1	0	0	0	0	0

The first Column corresponding to node 0 will be 0,0,1,0,1,1

Answer = B

20. If BFS is run starting at node 4 in the digraph shown, what is the third node to turn grey? Where there is a choice of nodes, choose the one with the lowest index.



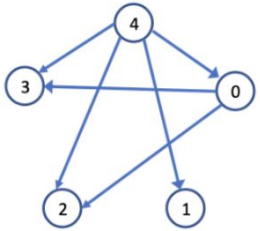
A. 1
B. 2
C. 0
D. 3

QUEUE: **4 2** 1

NODE	0	1	2	3	4
DEPTH		2	1		0

Answer = A

21. What is the topological order of the digraph shown found by using DFS (starting at 0, picking the lowest index where there is a choice).



A. 4,0,3,1,2
B. 4,0,3,2,1
C. 4,1,0,2,3
D. 4,1,0,3,2

node	0	1	2	3	4
seen	0	6	1	3	8
done	5	7	2	4	9

Topological Order = Done times in reverse = 4, 1, 0, 3, 2

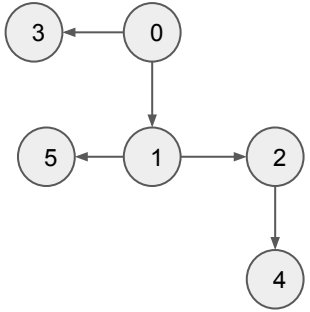
22. The following are seen and done arrays from a DFS of a digraph with node labels 0,1,2,3,4,5:

$$seen = [0, 1, 2, 9, 3, 6] \text{ and } done = [11, 8, 5, 10, 4, 7].$$

Is the arc $(1, 4)$ a

A. tree arc
B. cross arc
C. back arc
D. forward arc

node	0	1	2	3	4	5
seen	0	1	2	9	3	6
done	11	8	5	10	4	7

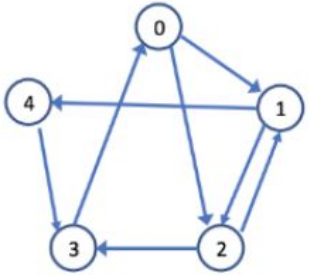


(1,4)
Node 4: Seen at 3, Done at 4
Node 1: Seen at 1, Done at 8

Seen Node 1 < Seen node 4 < Done node 4 < Done node 8
Tree arc or Foward arc.

According to the search Tree, node 4 is NOT a direct ancestor of node 1, hence this must be a forward arc.
Answer = D

23. What is the girth of the digraph shown?

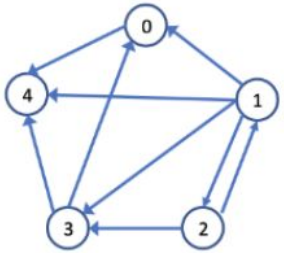


A. undefined
B. 2
C. 3
D. 4

Girth is defined by the shortest cycle in a GRAPH.
Since this is a digraph, we just ignore the directions, and view the underlying graph. Hence the shortest cycle would be of length 3.

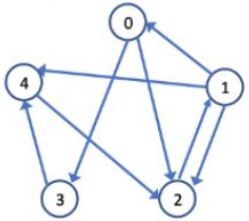
Answer = C

24. How many strong components does the digraph below have?



A. 2
B. 3
C. 1
D. 4

25. Perform a DFS search on the digraph shown starting at node 0 and picking the node with the lowest index when there is a choice. The arc (3,4) is a

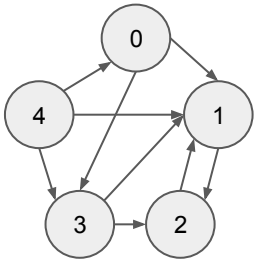


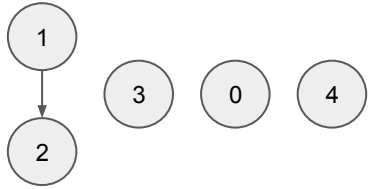
A. back arc
B. tree arc
C. cross arc
D. forward arc

node	0	1	2	3	4
seen	0	4	5	6	1
done	3	9	8	7	2
pred	-1	-1	1	2	0

node	0	1	2	3	4
seen	6	0	1	4	8
done	7	3	2	5	9
pred	-1	-1	1	-1	-1

Reverse done times = 1, 2, 3, 0, 4

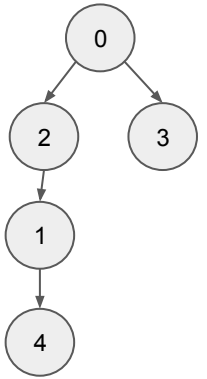




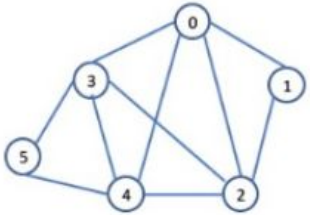
Total trees io search forrest = 4, hence there are 4 strongly connected components.

Answer = D

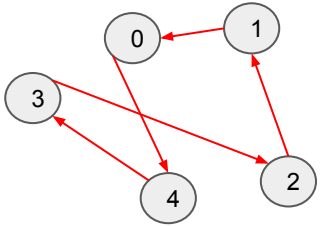
node	0	1	2	3	4
seen	0	2	1	7	3
done	9	5	6	8	4
pred	-1	2	0	0	1



26. The vertex sequence 3,2,1,0,4,3 in the graph below is most accurately described as

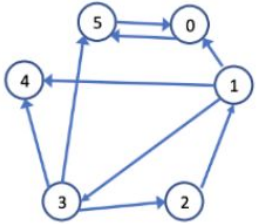


A. a path
B. a walk but not a path
C. not a walk
D. a cycle



By definition, a ccle is a pat (no nodes are repeated) with the exception that the first and last node can be repeated.
Answer = D

27. If you perform Tarjan's algorithm on the following digraph, what order are root nodes chosen in the second DFS. Assume DFS starts at 0 and picks the node with the lowest index when there is a choice.

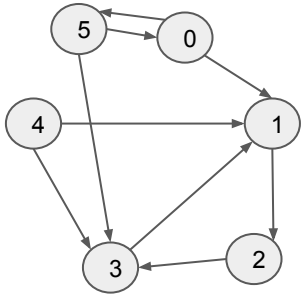


A. 3,2,5
B. 0,4,2
C. 1,4,5
D. 1,4,0

node	0	1	2	3	4	5
seen	0	4	6	5	8	1
done	3	11	7	10	9	2

node	0	1	2	3	4	5
seen	8	0	1	2	6	9
done	11	5	4	3	7	10

Reversed done time order = 1, 3, 4, 2, 0, 5



1 4 0
Answer = D

28. The running time for an algorithm that determines whether a graph of size m and order n has an odd cycle through a specified vertex v is

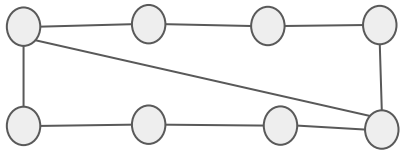
A. $\Theta(n^2)$
B. $\Theta(m+n)$
C. $\Theta(mn+n^2)$
D. $\Theta(n^3)$

Running BFS can determine whether or not an odd cycle exists,
BFS running time of (m+n), hence = (m+n)

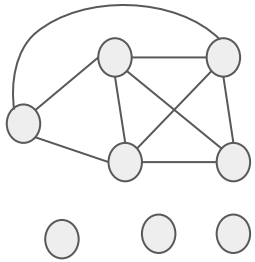
Answer = B

29. Suppose a graph has order 8 and size 9. What is the difference between the maximum number of connected components it could have and the minimum number of connected components it could have?

A. 3
B. 4
C. 2
D. 1



Minimum number of connected components = 1



Maximum number of connected components = 4

Difference = 4-1 = 3

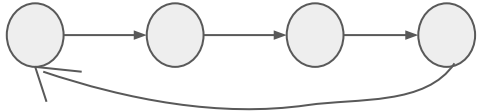
Answer = A

30. Suppose that G is a digraph with exactly 4 strongly connected components. If we create a new digraph by adding a single arc to G, what is the minimum number of connected components the new digraph could have?

A. 2
B. 1
C. 4
D. 3

If a graph has 4 strongly connected components, this means there are four nodes that cannot get back to themselves.

If we add one arc, this would create a cycle, as one node would have to point to an ancestor.



Hence, the minimum number of connected components it can have is 1.

Answer = B

31. Suppose a digraph of order n has a topological order. How many strong components does the digraph have?

A. Fewer than n
B. n
C. 1
D. There is not enough information to say

32. Suppose we wish to solve the all pairs shortest paths problem on a sparse weighted digraph and we know that the weight matrix has no negative entries. Which algorithm would be the best choice?

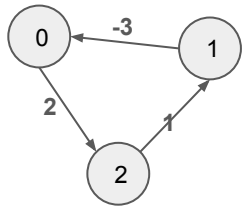
A. There is not enough information to say
B. Bellman-Ford
C. Dijkstra
D. Floyd

33. Some of the algorithms we looked at to solve the all pairs shortest paths problem are easily able to detect negative cycles. Why are negative cycles of interest for this problem?

A. When a negative cycle is present, the problem has no defined solution
B. When a negative cycle is present, the problem can be solved more swiftly because negative cycles make paths shorter.
C. When a negative cycle is present, only paths that avoid those cycles should be considered.
D. When a negative cycle is present, the Bellman-Ford algorithm should be used instead of other methods, which is slower.

34. Consider the weighted digraph with nodes $\{0, 1, 2\}$ and arcs $(0, 2)$ with weight 2, $(2, 1)$ with weight 1, and $(1, 0)$ with weight -3. If we wish to solve the SSSP problem starting at node 0

A. The problem makes no sense for this digraph
B. Dijkstra produces the incorrect result but Bellman-Ford produces the correct result
C. Both Djikstra and Bellman-Ford give the correct result
D. Both Djikstra and Bellman-Ford give the incorrect result



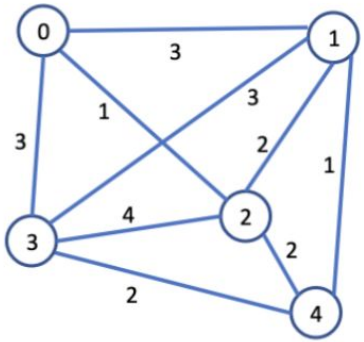
35. The running time of the operation of determining if an arc (i, j) is present in a digraph of order n and size m using an adjacency list is

A. $\Theta(d_j)$ where d_j is the in-degree of node j
B. $\Theta(d_i)$ where d_i is the out-degree of node i
C. $\Theta(n+m)$
D. $\Theta(n)$

36. Which of the following would indicate a graph, G, was not bipartite?

A. The presence of a back arc in a DFS traversal
B. A 3-colouring of G
C. The presence of a cross arc on the same level in a BFS traversal
D. A cycle in G

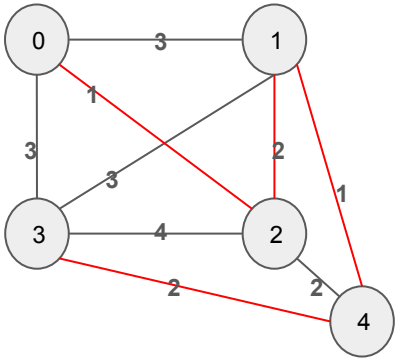
37. What is the predecessor array output by running Prims algorithm on the graph shown if you start from vertex 0 and choose the vertex with the lowest index where there is a choice?



A. $[-1, 2, 0, 4, 1]$
B. $[-1, 2, 0, 2, 3]$
C. $[-1, 4, 0, 4, 2]$
D. $[-1, 0, 0, 0, 1]$

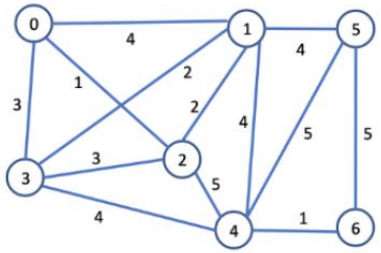
Prim's Algorithm (m log n):

- Choose a starting node
- Find it's neighbours and traverse the arc with the lowest weight as long as connecting this node does not produce a cycle
- find the neighbours and traverse the arc with the lowest weight of the last node visited and continue

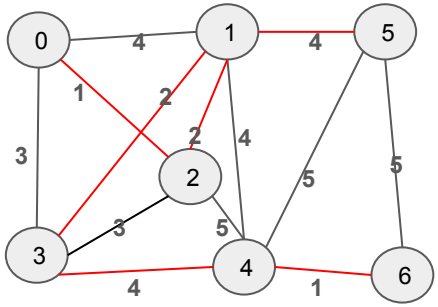


0	1	2	3	4
-1	2	0	4	1

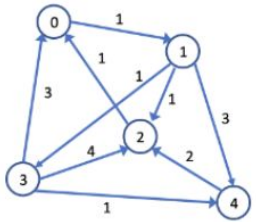
38. What is the weight of the minimum spanning tree in the weighted graph shown?



A. 14
B. 13
C. 11
D. 12



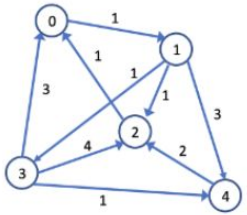
39. What is the fourth node to turn black if you run Dijkstra's algorithm on the following graph? Assume that we start from node 3 and choose the vertex with the lowest index where there is a choice?



A. 2
B. 1
C. 4
D. 3

	0	1	2	3	4
3 →	3	∞	4	0	1
3→4	3	5	3	0	0
3→4→0	3	5	3	0	0
3→4→0→2					

40. What is the second list (corresponding to index 1) of the weighted adjacency lists representation of the digraph shown?



A. 1 2 1 3 3 4
B. 0 0 1 1 3
C. 2 1 3 1 4 3
D. 2 3 4

0:
1: 2 1 3 1 4 3
2:
3:
4:

2024 S1 EXAM

Q1:
Quickselect Average runtime = Θ(n)
T(1000) = C * 1000 = 1 second
C = 1 / 1000
T(1000000) = 1 / 1000 * 1000000 = 1000000 / 1000 = 1000 seconds

Q2:
i ← 1
While i < n:
  i ← 3 * i
  if i % 2 == 1:
    {C elementary Operations}

i increases by a factor of 3 in each iteration while it is less than n, so the while loop runs $\log_3(n)$ times. In each loop, the if statement always runs as i is initially set to 1, being multiplied by 3 wil always yield a multiple of 3, which is always odd. Therefore, every IF statement is TRUE, and also runs C elementary operations.

Overall runtime = $\log_3(n)$ * C

i = 1	i = 3	i = 9	i = 81	i = 243	i = 729	i = 2187	…	$\log_3(n)$
i = 3 * 1 = 3 3 % 2 == 1	i = 3 * 3 = 9 9 % 2 == 1	i = 3 * 9 = 27 27 % 2 == 1	i = 3 * 27 = 81 81 % 2 == 1	i = 3 * 81 = 243 243 % 2 == 1	i = 3 * 243 = 729 729 % 2 == 1	i = 3 * 729= 2187 2187% 2 == 1	…	

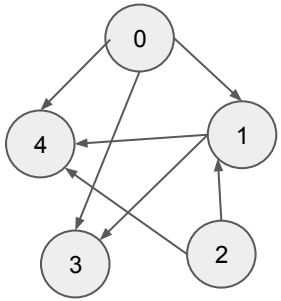
Q31
seen[v] < done[v] < seen[w] < done[w]

seen[v] < seen[w] < done[v] < done[w]
10 90 150 200

seen[v] < seen[w] < done[w] < done[v]

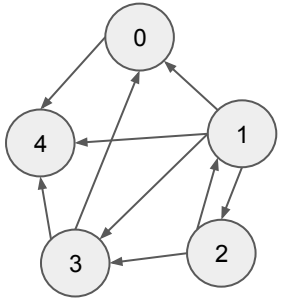
seen[v] < seen[w] < done[w] < done[v]

Q20.
Topological Order: 2,0,1,4,3



	0	1	2	3	4
seen	0	1	8	2	4
done	7	6	9	3	5

Q32.
Predecessor Array = [3, -1, 1, -1, 3]



	0	1	2	3	4
depth	1	0	1	0	1
pred	3	-1	1	-1	3

Q11.

```
Function f(n):
   if n < 1: return 0
   return g(n+1)
Function g(n):
   return f(n-2)
```

Runtime = n

1	2	3	4

Function f(1)
  if 1 < 1: return 0
  return g(2)
Function g(2)
  return f(0)
 	
Function f(0)
  if 0 < 1: return 0
0
	Function f(2)
	  if 2 < 1: return 0
	  return g(3)
	Function g(3)
	  return f(1)
	
	0
		Function f(3)
		  if 3 < 1: return 0
		  return g(4)
		Function g(4)
		  return f(2)
		
		0
			Function f(4)
			  if 4 < 1: return 0
			  return g(5)
			Function g(5)
			  return f(3)
			
			0

Q26:
Weight of spanning Tree = 13