CSCI 2110 Data Structures and Algorithms

Module 9: Graphs
Part 2 – Graph Algorithms

DALHOUSIE UNIVERSITY

1

Graph Algorithms

1. Graph Traversals
2. Topological Sorting Algorithms
3. Shortest Path Algorithms

DALHOUSIE UNIVERSITY

1. Graph Traversal

- A graph traversal is a "walk" in the graph so that every vertex is visited.
- During a traversal, you are allowed to backtrack but the walk should remain connected – that is, you are not allowed to jump from one node to a distant node.
- A traversal should list each vertex in the graph exactly once.
- Traversals are also called Searches.
- There are two types of traversals/searches:

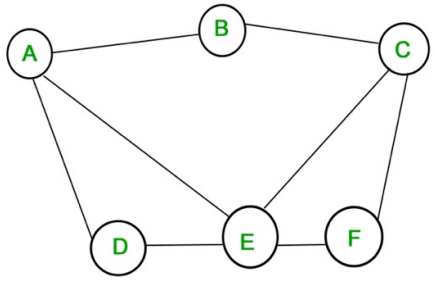
a) DEPTH FIRST SEARCH

b) BREADTH FIRST SEARCH

DALHOUSIE UNIVERSITY

a) Depth First Search (DFS)

- Start at a vertex v1.
- Mark v1 as visited.
- Pick a neighbour of v1 that is not visited, say v2. Go to v2.
- Mark v2 as visited.
- Pick a neighbour of v2 that is not visited, say v3. Go to v3.
- Continue and mark each vertex that has been visited.
- If you hit a dead-end, backtrack to the previous neighbour and pick a neighbour that has not been visited.



Example depth-first search output:
A B C F E D

DALHOUSIE UNIVERSITY

Code for Depth First Search – Recursion!

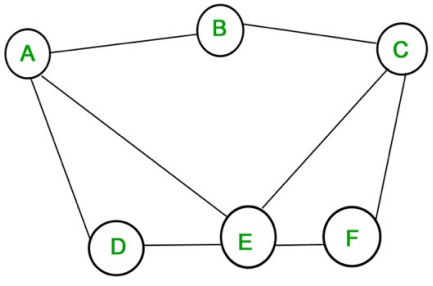
Algorithm DFS(v) where v is the starting vertex

```
visit v and mark it as read.
for each neighbour w of v
{
        if w is not visited
                DFS(w)
}
```

DALHOUSIE UNIVERSITY

b) Breadth First Search (BFS)

- Start at a vertex v1.
- Visit all the neighbours of v1.
- Then visit all the unvisited neighbours of each neighbour of v1.
- Continue until all the vertices are visited.



Example breadth-first search output:
A B E D C F

DALHOUSIE UNIVERSITY

Code for Breadth First Search (BFS) – a queue or ArrayList!

Algorithm BFS(v) where v is the starting vertex

Initialize an empty queue and a result list.

Enqueue the first vertex v.

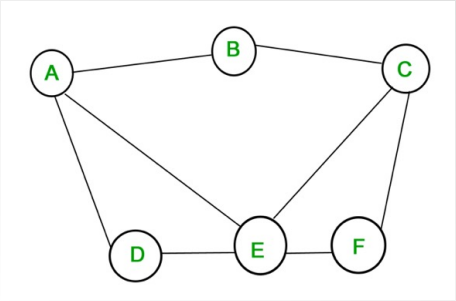
while the queue is not empty

{

Dequeue the item x and put it in the result list.

Enqueue each neighbour of x if it is not in the result list and if it is not already in the queue.

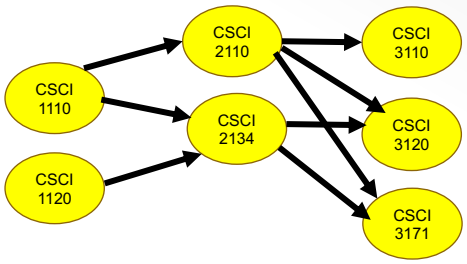
}



DALHOUSIE UNIVERSITY

2. Topological Sorting

- Topological sorting means arranging the vertices in a directed graph in a sequence so that the dependency condition is not violated.
- As an example, consider the following directed graph that shows a partial list of CS courses with prerequisites:



Disclaimer Note: This is not an accurate list of prerequisite conditions for the CS courses.
This is meant just as an example. For an accurate list, consult the academic calendar!

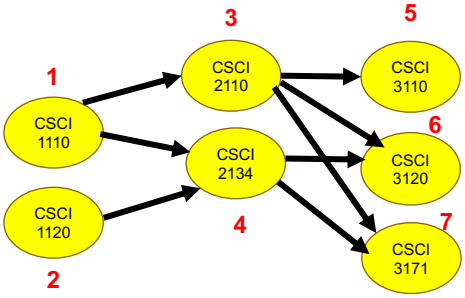
DALHOUSIE UNIVERSITY

2. Topological Sorting (cont'd.)

The sorting algorithm assigns numbers to each vertex in ascending order such that the dependency condition is not violated.

As an example, this is a valid solution:

1	2	3	4	5	6	7
1110	1120	2110	2134	3110	3120	3171



This is also a valid solution:

1	2	3	4	5	6	7
1120	1110	2110	3110	2134	3120	3171

This is **not** a valid solution:

1	2	3	4	5	6	7
1120	1110	3120	3110	2134	2110	3171

DALHOUSIE UNIVERSITY

Algorithm for Topological Sorting

```
Initialize an empty queue.
for each vertex v in the graph
        compute the predecessor count, pred(v)
for each vertex v in the graph
        if (pred(v)==0) add v to the queue.
topnum←1
while queue is not empty{
        w ←dequeue
        assign w with topnum
        topnum←topnum+1

        for each neighbour p of w{
                pred(p) ← pred(p) -1
                if (pred(p)==0) then
                        add p to queue
        }//end for
}//end while
```

All the vertices with be assigned with topnum in the topological sorting order.

DALHOUSIE UNIVERSITY

Shortest Path Algorithms

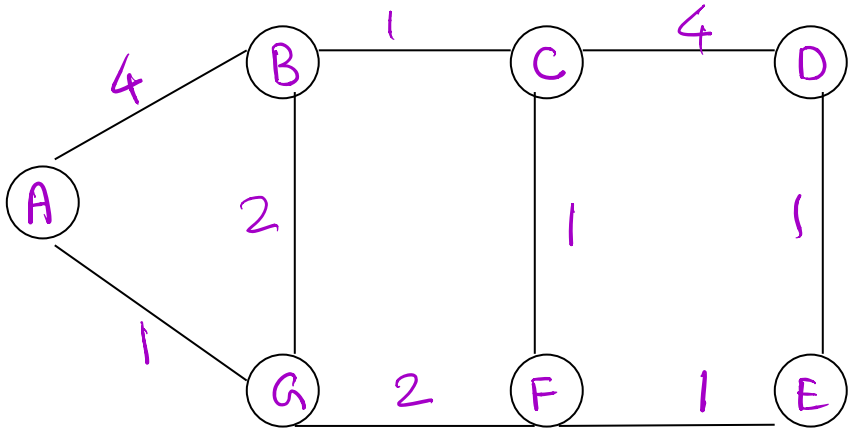
Given a weighted graph and a source node, determine the shortest paths from the source node to every other node in the graph.

DALHOUSIE UNIVERSITY

SHORTEST PATH ALGORITHMS

Problem: Given a weighted graph, find the shortest paths from a given source node to every other node.

Consider the following graph



The adjacency list for the above graph can be written as follows:

A	B	C	D	E	F	G
B, 4 G, 1	A, 4 C, 1 G, 2	B, 1 D, 4 F, 1	C, 4 E, 1	D, 1 F, 1	C, 1 E, 1 G, 2	A, 1 B, 2 F, 2

Let the source node be B.Then the expected solution is:

Dijkstra's algorithm for shortest paths

- Two lists are set up: *Confirmed (C) list* and *Tentative (T) list*.
- Each list will contain entries of the form (Destination, Cost, Next Hop).
- Cost specifies total cost from the source node to the destination
- Next Hop specifies the next hop node from the source node.

Entries are added to the T list and moved from the T list to the C list.

Final solution will be the C list.

DALHOUSIE UNIVERSITY

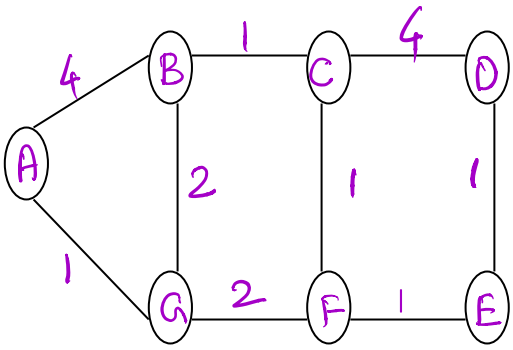
Dijkstra's algorithm in detail

Let S be the source node.

1. Place the entry for S in the C list. The entry is
   (Destination = S, Cost = 0, Next Hop = -).
2. Let NEXT be the node just added to the C list. Select its neighbors.
3. For each neighbor (NEIGHBOR) of NEXT:
   a) If NEIGHBOR is already in the C list, skip; else
   b) Calculate COST = cost from S to NEXT + cost from NEXT to NEIGHBOR
      Calculate NEXTHOP = Next Hop from S to reach NEXT
   c) If NEIGHBOR is currently not in the T list, then add ***(NEIGHBOR, COST, NEXTHOP)*** to the T list.
   d) If NEIGHBOR is currently in the T list, compare its currently listed cost. If
      COST < current cost, then replace current entry with
      (NEIGHBOR, COST, NEXTHOP).
4. If T list is empty, stop;
   Else, move the entry with the lowest cost from the T list to the C list. Go to step 2.

DALHOUSIE UNIVERSITY

Example 1:

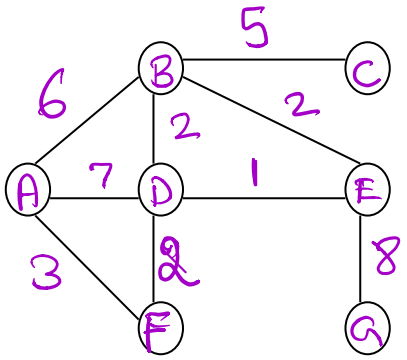


Adjacency List

A	B	C	D	E	F	G
B, 4 G, 1	A, 4 C, 1 G, 2	B, 1 D, 4 F, 1	C, 4 E, 1	D, 1 F, 1	C, 1 E, 1 G, 2	A, 1 B, 2 F, 2

Sl. No.	Confirmed (C)	Tentative (T)	Remarks

Example 2:



A	B	C	D	E	F	G
B,6 D,7 F,3	A,6 C,5 D,2 E,2	B,5	A,7 B,2 E,1 F,2	B,2 D,1 G,8	A,3 D,2	E,8

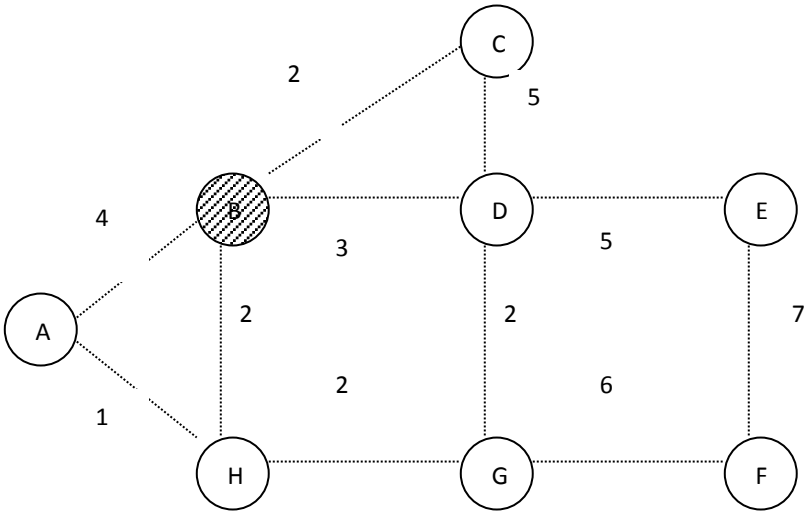
Sl. No.	Confirmed (C)	Tentative (T)	Remarks

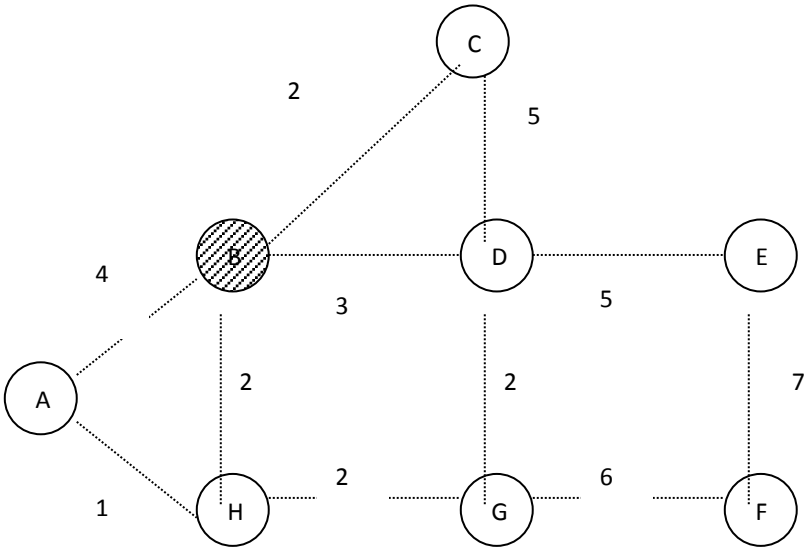
BISWAS –SAMPALLI ALGORITHM FOR SHORTEST PATHS

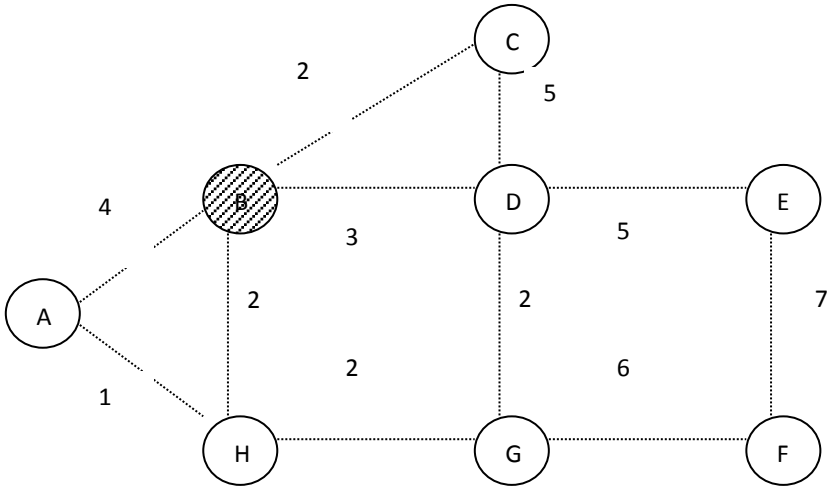
Key Idea: It is based on the presence of polygons (cycles) in the graph. Start at the source node. Find the smallest polygon connected to the source node. Find the sum of the weights of the edges. Find threshold = sum/2.

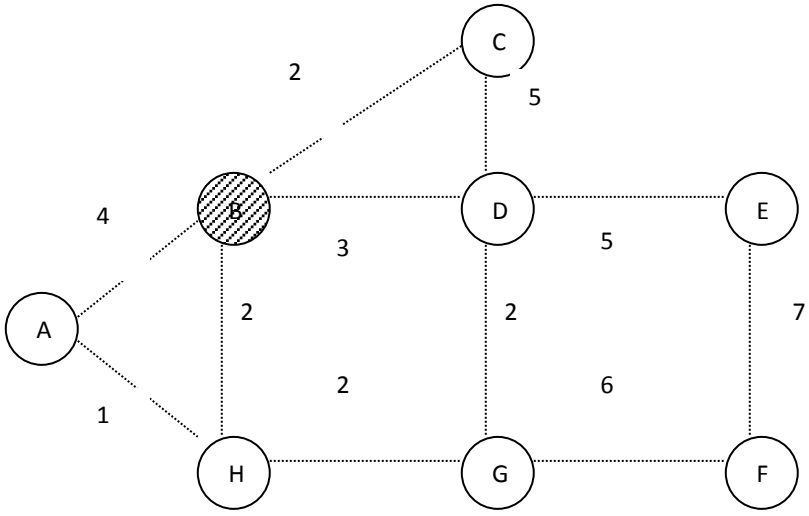
Traverse the polygon starting from the source node and keep adding up the link costs. When the cost >=threshold, eliminate the link you are on.

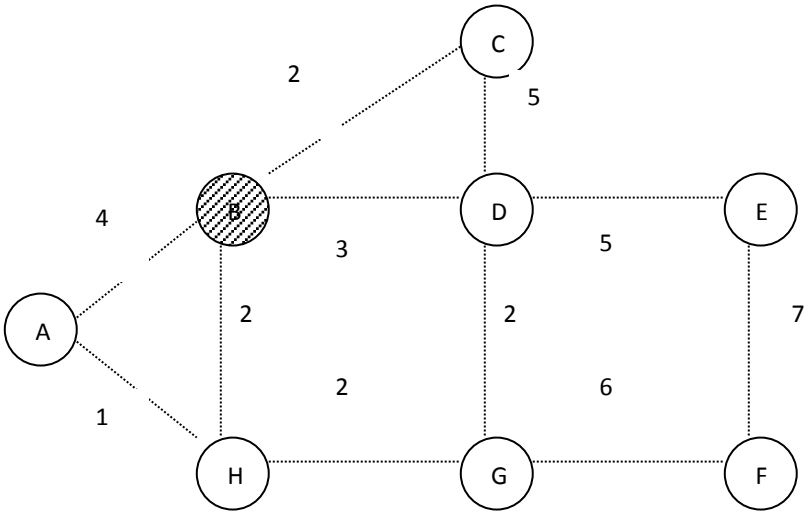
Repeat until no polygons are present in the graph.



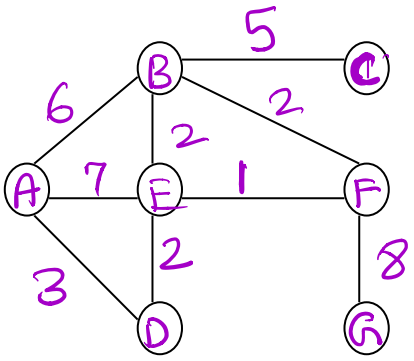



C
2
5
B
D
E
4
3
5
A
2
2
7
2
6
1
H
G
F

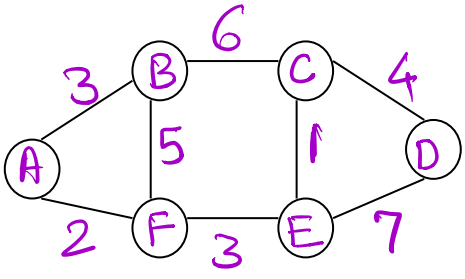

C
2
5
B
D
E
4
3
5
A
2
2
7
2
6
1
H
G
F


C
2
5
B
D
E
4
3
5
A
2
2
7
2
6
1
H
G
F

Example 2:



Example 3:



Example 4: