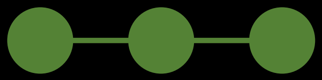
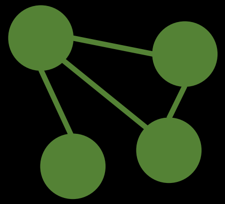
Exam 2 Review A

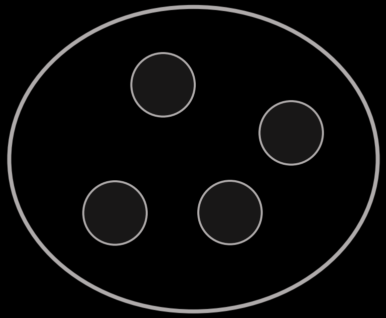
Data Structures & Algorithm

Categories of Data Structures

Linear Ordered	Non-linear Ordered	Not Ordered
Lists	Trees	Sets
Stacks	Graphs	Tables/Maps
Queues		







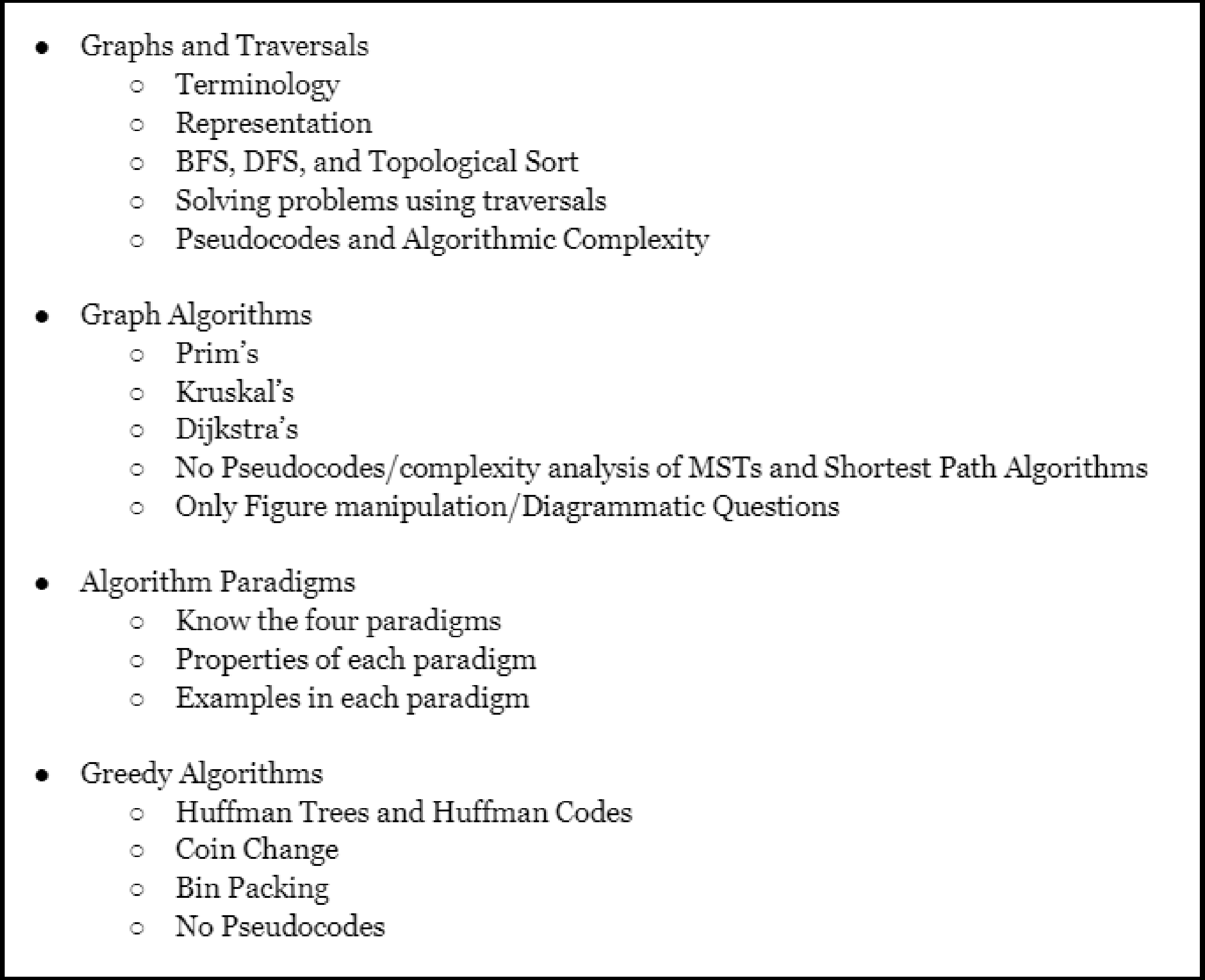
Announcements

- Exam 2 will be on Dec 2, 1 pm – 5:30 pm EST
- Must start the exam by 3:30 pm or you will lose time.
- Study Guide is now up!
- 4 sheets of blank paper. Show on camera.

Agenda

- Review
    - Graph and Graph Algorithms
    - Algorithm Paradigms
    - Greedy Algorithms: Huffman Trees

Final Exam Topics Guide



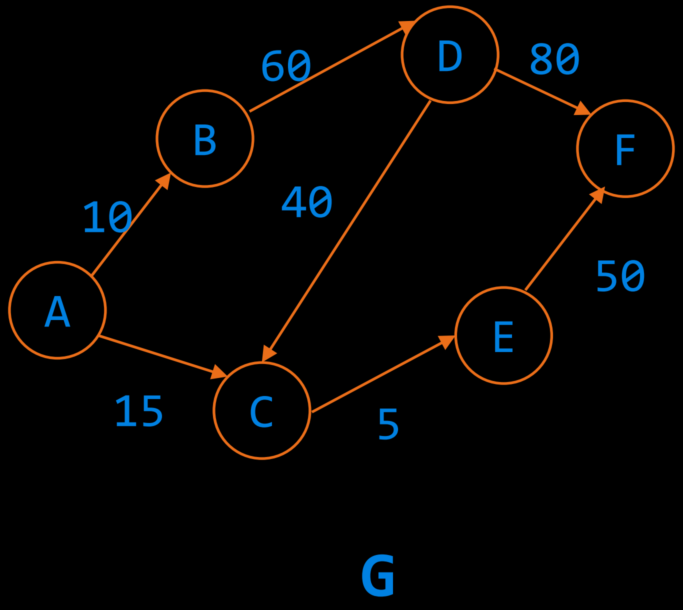
- Graphs and Traversals
    - Terminology
    - Representation
    - BFS, DFS, and Topological Sort
    - Solving problems using traversals
    - Pseudocodes and Algorithmic Complexity

- Graph Algorithms
    - Prim's
    - Kruskal's
    - Dijkstra's
    - No Pseudocodes/complexity analysis of MSTs and Shortest Path Algorithms
    - Only Figure manipulation/Diagrammatic Questions

- Algorithm Paradigms
    - Know the four paradigms
    - Properties of each paradigm
    - Examples in each paradigm

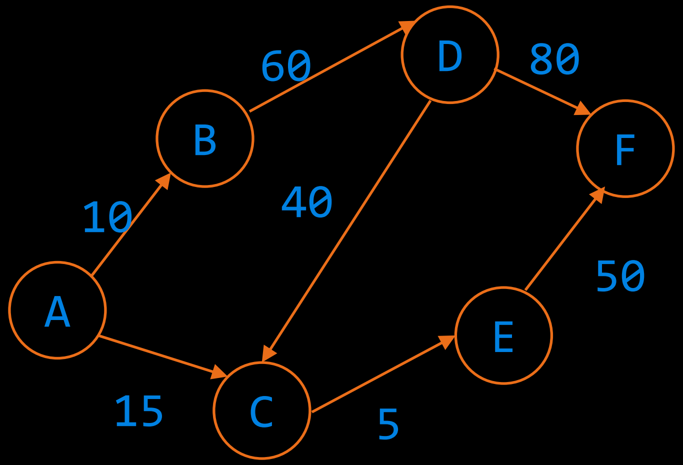
- Greedy Algorithms
    - Huffman Trees and Huffman Codes
    - Coin Change
    - Bin Packing
    - No Pseudocodes

Common Representations



- Edge List
- Adjacency Matrix
- Adjacency List

Edge List

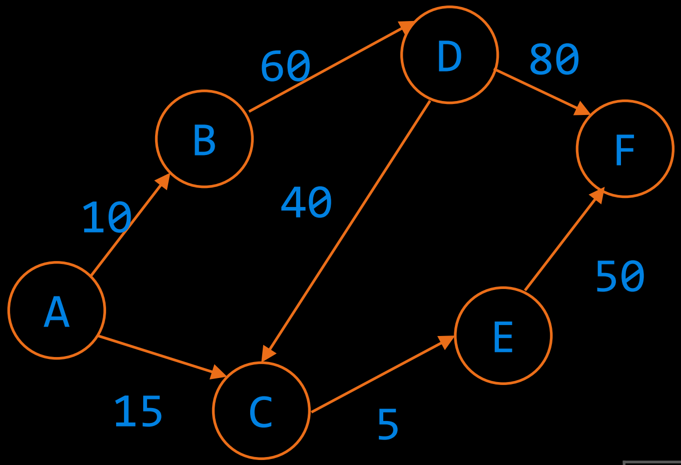


G

A	B	10
A	C	15
B	D	60
D	C	40
D	F	80
E	F	50
C	E	5

G = {(A,B), (A,C), (B,D), (D,C), (D,F), (E,F), (C,E)}

Edge List



G = {(A,B), (A,C), (B,D), (D,C), (D,F), (E,F), (C,E)}

G

A	B	10
A	C	15
B	D	60
D	C	40
D	F	80
E	F	50
C	E	5

Common Operations:

1. Connectedness

   Is A connected to B?

   ~ O(E)

2. Adjacency

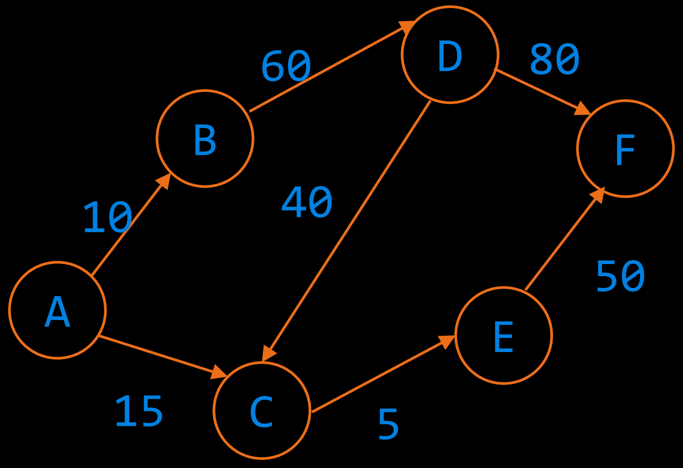
   What are A's adjacent nodes?

   ~ O(E)

O(|E|) ~ O(|V| * |V|)

Space: O(E)

Adjacency Matrix



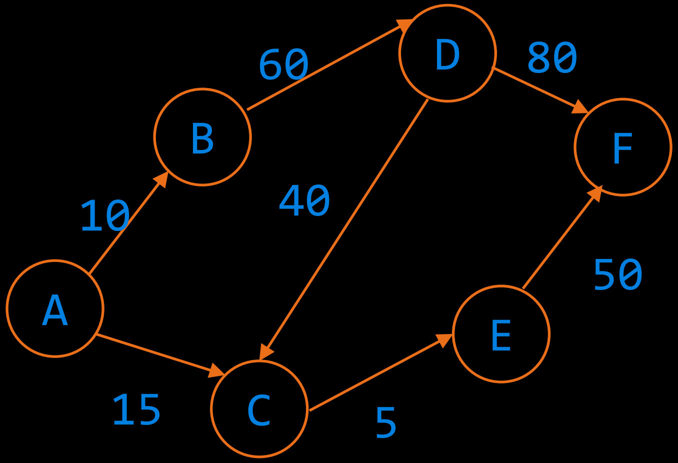
G

	A	B	C	D	E	F
A	0	10	15	0	0	0
B	0	0	0	60	0	0
C	0	0	0	0	5	0
D	0	0	40	0	0	80
E	0	0	0	0	0	50
F	0	0	0	0	0	0

Insertion:

```
G[from][to] = weight; (if there is an edge, “from” -> “to”)
G[from][to] = 0;           (otherwise)
```

Adjacency Matrix Implementation



Insertion:

```
G[from][to] = weight; (if there is an edge, "from" -> "to")
G[from][to] = 0;          (otherwise)
```

Input

```
7
A B 10
A C 15
B D 60
D C 40
C E 5
D F 80
E F 50
```

G

Map

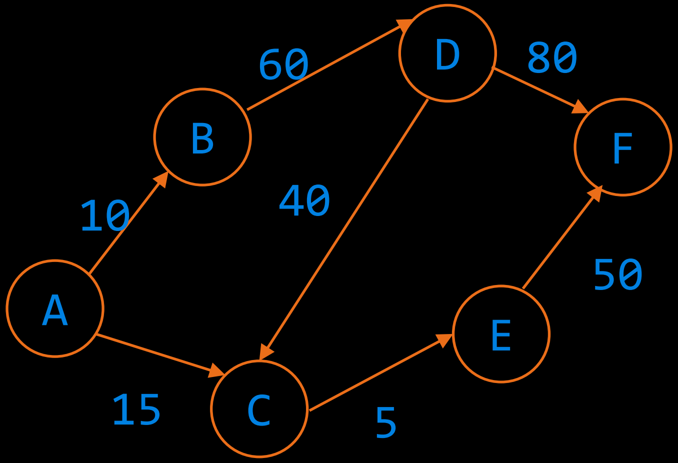
A	0
B	1
C	2
D	3
E	4
F	5

	0	1	2	3	4	5
0	0	10	15	0	0	0
1	0	0	0	60	0	0
2	0	0	0	0	5	0
3	0	0	40	0	0	80
4	0	0	0	0	0	50
5	0	0	0	0	0	0

```
#include <iostream>
#include<map>
#define VERTICES 6
using namespace std;
int main()
{
        int no_lines, wt, j=0;
        string from, to;
        int graph [VERTICES][VERTICES] = {0};
        map<string, int> mapper;
        cin >> no_lines;
        for(int i = 0; i < no_lines; i++)
        {
                cin >> from >> to >> wt;
                if (mapper.find(from) == mapper.end())
                        mapper[from] = j++;
                if (mapper.find(to) == mapper.end())
                        mapper[to] = j++;
                graph[mapper[from]][mapper[to]] = wt;
        }
        return 0;
}
```

https://www.onlinegdb.com/Hy8M0CnsS

Adjacency Matrix



Map

A	0
B	1
C	2
D	3
E	4
F	5

G

	0	1	2	3	4	5
0	0	10	15	0	0	0
1	0	0	0	60	0	0
2	0	0	0	0	5	0
3	0	0	40	0	0	80
4	0	0	0	0	0	50
5	0	0	0	0	0	0

Common Operations:

1. Connectedness

Is A connected to B?

```
G[“A”][“B”] ~ O(1)
```

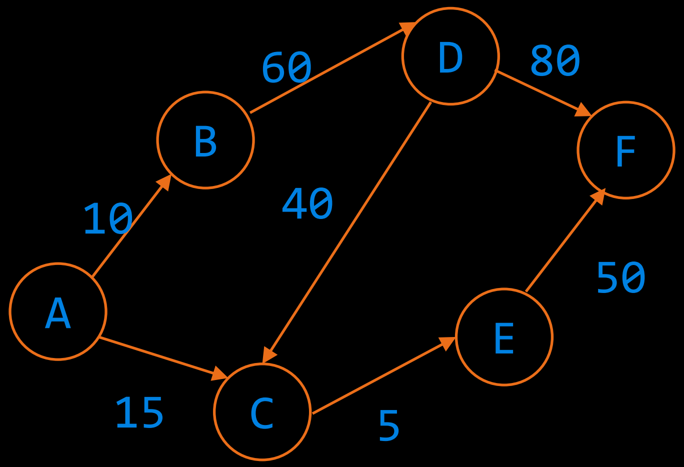
2. Adjacency

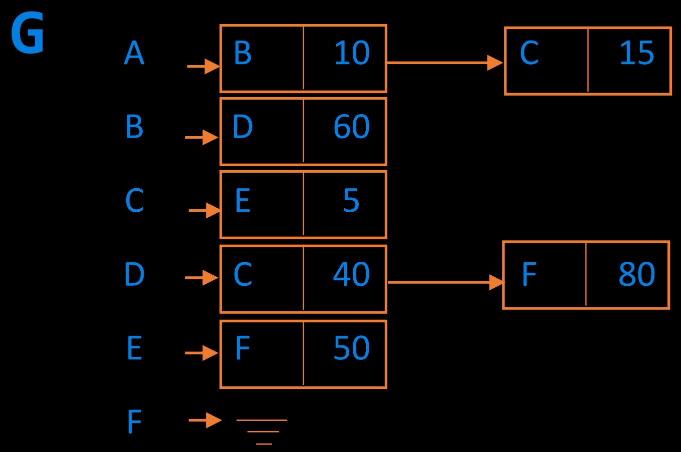
What are A’s adjacent nodes?

```
for each element x in G[“A”]
    if x ! = 0
                              ~ O(|V|)
```

Space: O(|V| * |V|)

Adjacency List





Sparse Graph:
Edges ~ Vertices

Common Operations:

1. Connectedness

   Is A connected to B?

   ```
   for each element x in G["A"]
       if x ! = 'B'
           ~ O(outdegree|V|)
   ```

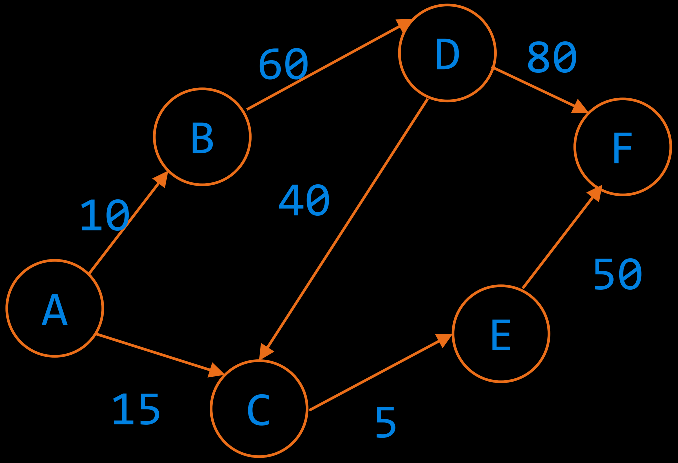
2. Adjacency

   What are A's adjacent nodes?

   ```
   G["A"] ~ O(outdegree|V|)
   ```

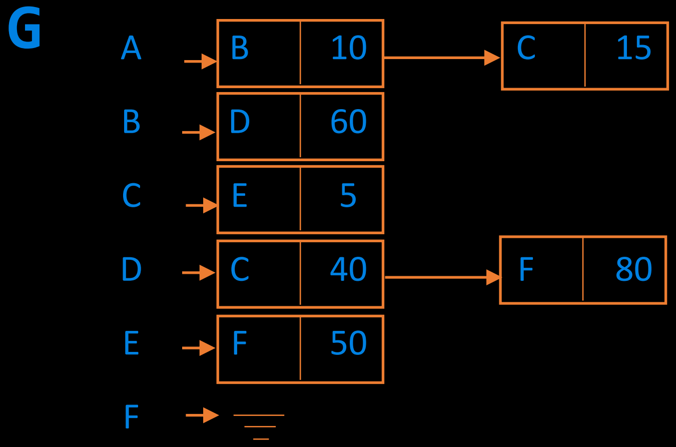
Space: O(|V| + |E|)

Adjacency List Implementation



Input

```
7
A B 10
A C 15
B D 60
D C 40
C E 5
D F 80
E F 50
```



Insertion:

If to or from vertex not present add vertex

Otherwise add edge at the end of the list

```
#include <iostream>
#include<map>
#include<vector>
#include<iterator>
using namespace std;

int main()
{
        int no_lines;
        string from, to, wt;
        map<string, vector<pair<string,int>>> graph;
        cin >> no_lines;
        for(int i = 0; i < no_lines; i++)
        {
                cin >> from >> to >> wt;
                graph[from].push_back(make_pair(to, stoi(wt)));
                if (graph.find(to)==graph.end())
                        graph[to] = {};
        }
}
```

https://onlinegdb.com/HkJq9iFaI

Graph Implementation

	Edge List	Adjacency Matrix	Adjacency List
Time Complexity: Connectedness	O(E)	O(1)	O(outdegree(V))
Time Complexity: Adjacency	O(E)	O(V)	O(outdegree(V))
Space Complexity	O(E)	O(V*V)	O(V+E)

Graph - BFS

- Which of the following are valid breadth first search traversals for this graph?

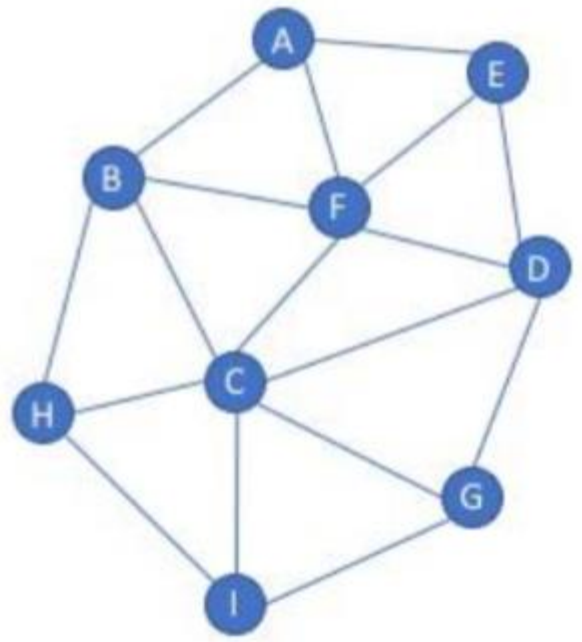
a) A F B E D C H G I

b) I C H G B F D A E

c) D C F E G H I B A

d) E A F D B H C I G

e) F A E D C B G I H



Graph - BFS

- Which of the following are valid breadth first search traversals for this graph?

a) A F B E D C H G I

b) I C H G B F D A E

c) D C F E G H I B A

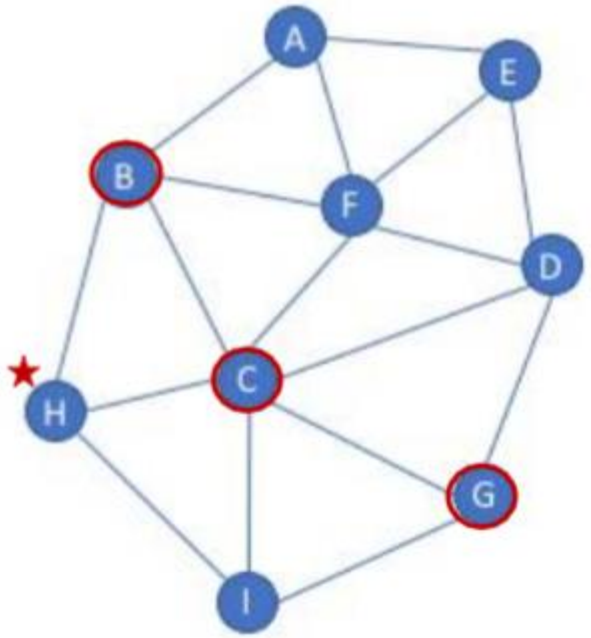
d) E A F D B H C I G

e) F A E D C B G I H

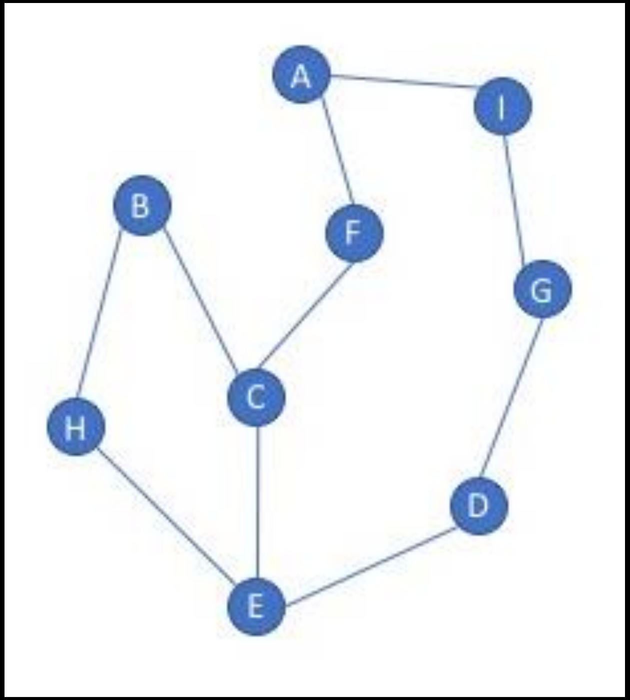
All the options except for d

Why not d?

** H is visited before C and G

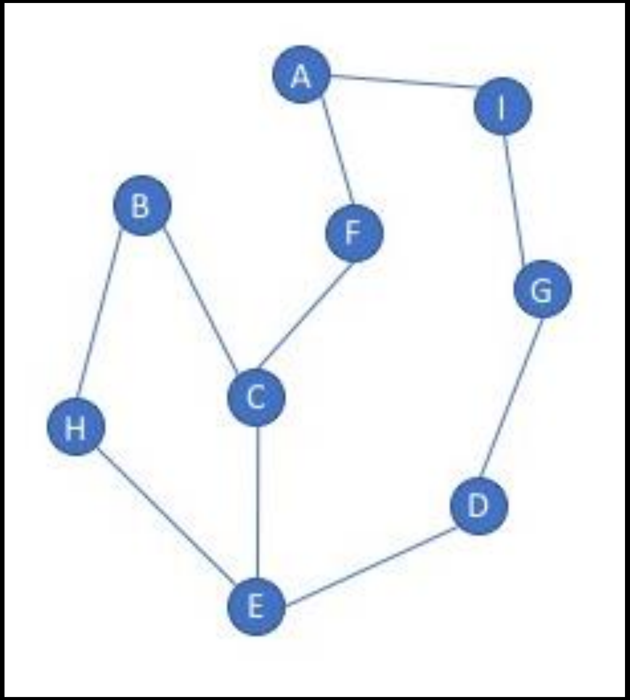


Valid DFS: Which DFS are valid?



- H E C B D G I A F
- C E H B D G I A F
- A F C E H B I G D
- D E C B H F A I G

Valid DFS: Which DFS are valid?



- H E C B D G I A F
- C E H B D G I A F
- A F C E H B I G D
- D E C B H F A I G

BFS Pseudocode

- Write pseudocode/code for implementing the **Breadth First Search Algorithm** of a graph, G that takes a source vertex S as input. (8).
- Also, state the Big O complexity of the traversal in the worst case (2).

BFS vs DFS

```
string source = "A";
std::set<string> visited;
std::queue<string> q;

visited.insert(source);
q.push(source);
cout<<"BFS: ";

while(!q.empty())
{
      string u = q.front();
      cout << u;
      q.pop();
      vector<string> neighbors = graph[u];
      for(string v: neighbors)
      {
            if(visited.count(v)==0)
            {
                  visited.insert(v);
                  q.push(v);
            }
      }
}
```

```
string source = "A";
std::set<string> visited;
std::stack<string> s;

visited.insert(source);
s.push(source);
cout<<“DFS: ";

while(!q.empty())
{
      string u = s.top();
      cout << u;
      s.pop();
      vector<string> neighbors = graph[u];
      for(string v: neighbors)
      {
            if(visited.count(v)==0)
            {
                  visited.insert(v);
                  s.push(v);
            }
      }
}
```

Theoretical Complexity: O(V+E)

Graph Algorithm Mix n Match

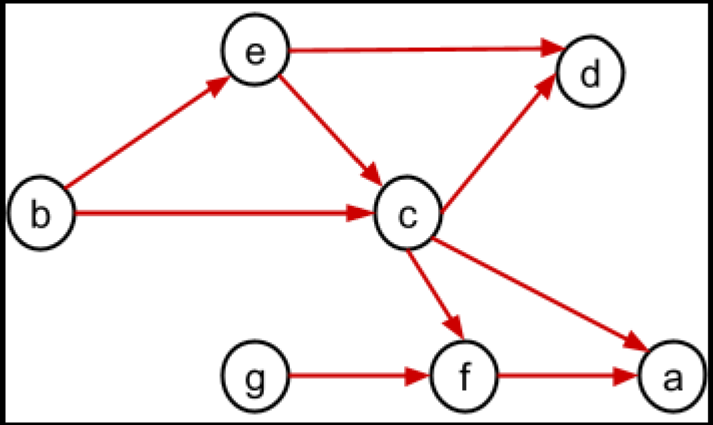
- Finds the shortest paths in a weighted graph
- Find the minimum cost connected network
- Scheduling algorithm, list steps in a process
- Finds the shortest path in an unweighted graph

Prim’s or Kruskals
BFS
DFS
Topological Sort
Dijkstra’s Algorithm

Graph Algorithm Mix n Match

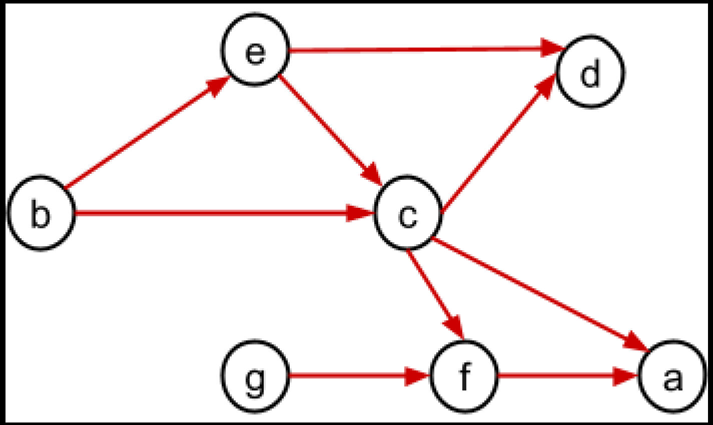
$$\begin{tikzcd}
\text{▪ Finds the shortest paths in a weighted graph} \arrow[ddddr] & \text{Prim's or Kruskals} \\
\text{▪ Find the minimum cost connected network} \arrow[ur] & \text{BFS} \\
\text{▪ Scheduling algorithm, list steps in a process} \arrow[dr] & \text{DFS} \\
\text{▪ Finds the shortest path in an unweighted graph} \arrow[uur] & \text{Topological Sort} \\
 & \text{Dijkstra's Algorithm}
\end{tikzcd}$$

Which of the choices below represent a valid topological sort ordering of this graph?



- b, e, c, g, f, a, d
- b, a, c, g, f, e, d
- b, g, f, c, e, a, d
- b, e, c, g, a, f, d
- b, g, e, c, d, f, a
- b, f, c, g, a, e, d

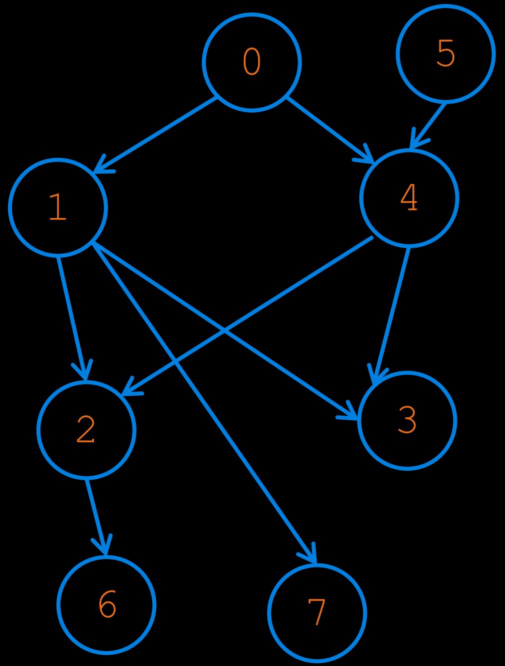
Which of the choices below represent a valid topological sort ordering of this graph?



- b, e, c, g, f, a, d
- b, a, c, g, f, e, d
- b, g, f, c, e, a, d
- b, e, c, g, a, f, d
- b, g, e, c, d, f, a
- b, f, c, g, a, e, d

Topological Sort Pseudocode

```
void Graph::topsort( )
{
    Queue<Vertex> q;
    int counter = 0;
    q.makeEmpty( );
    for each Vertex v
         if( v.indegree == 0 )
              q.enqueue( v );
    while( !q.isEmpty( ) )
    {
            Vertex v = q.dequeue( );
            for each Vertex w adjacent to v
                  if( --w.indegree == 0 )
                          q.enqueue( w );
    }
}
```



What does this code do?

```cpp
#include <set>
#include <stack>
using namespace std;

bool doSomething(const Graph& graph, int src, int dest)
{
    set<int> visited;
    stack<int> s;
    visited.insert(src);
    s.push(src);
    while(!s.empty())
    {
        int u = s.top();
        s.pop();
        for(auto v: graph.adjList[u])
        {
            if(v == dest)
                return true;
            if ((visited.find(v) == visited.end())) {
                visited.insert(v);
                s.push(v);
            }
        }
    }
    return false;
}
```

What does this code do?

```
#include <set>
#include <stack>
using namespace std;

bool doSomething(const Graph& graph, int src, int dest)
{
    set<int> visited;
    stack<int> s;
    visited.insert(src);
    s.push(src);
    while(!s.empty())
    {
        int u = s.top();
        s.pop();
        for(auto v: graph.adjList[u])
        {
            if(v == dest)
                return true;
            if ((visited.find(v) == visited.end())) {
                visited.insert(v);
                s.push(v);
            }
        }
    }
    return false;
}
```

Returns whether a given vertex is reachable from another vertex using DFS

Scenario

A county government maintains a network of roads. The county government has tabulated the cost of maintaining each road. They need to minimize the cost of road maintenance but ensure that all places in the county are accessible.

Which graph algorithm that we discussed in class could they use to solve this problem? What are the vertices, what are the edges, what are the edge values?

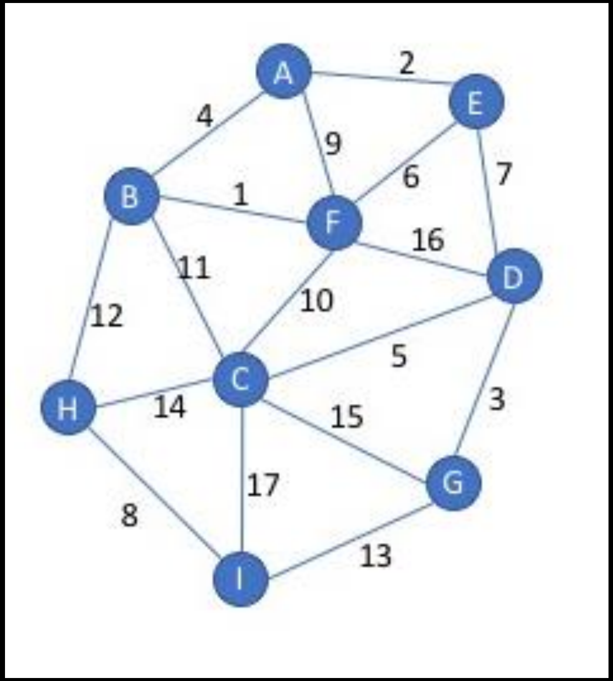
Scenario

A county government maintains a network of roads. The county government has tabulated the cost of maintaining each road. They need to minimize the cost of road maintenance but ensure that all places in the county are accessible.

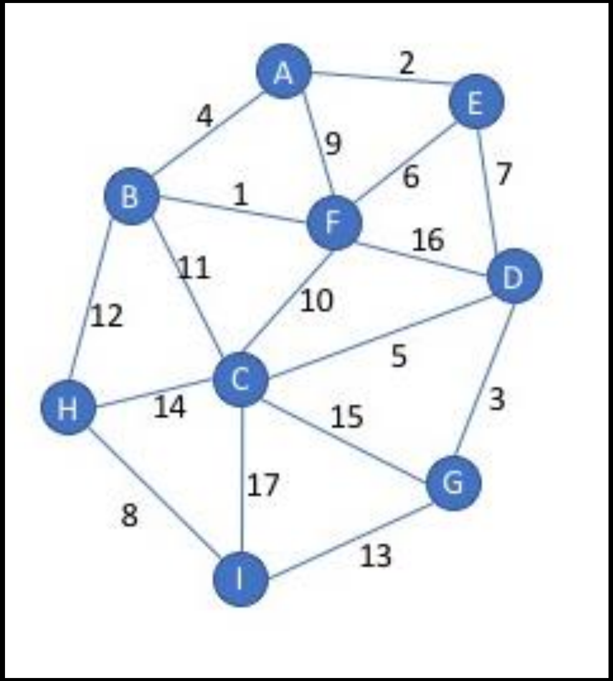
Which graph algorithm that we discussed in class could they use to solve this problem? What are the vertices, what are the edges, what are the edge values?

- Prim's or Kruskals algorithm for minimum spanning tree.
- Roads are edges.
- Ends of roads are vertices.
- Edge weights are cost for maintaining roads.

MST using Prims starting from “I”



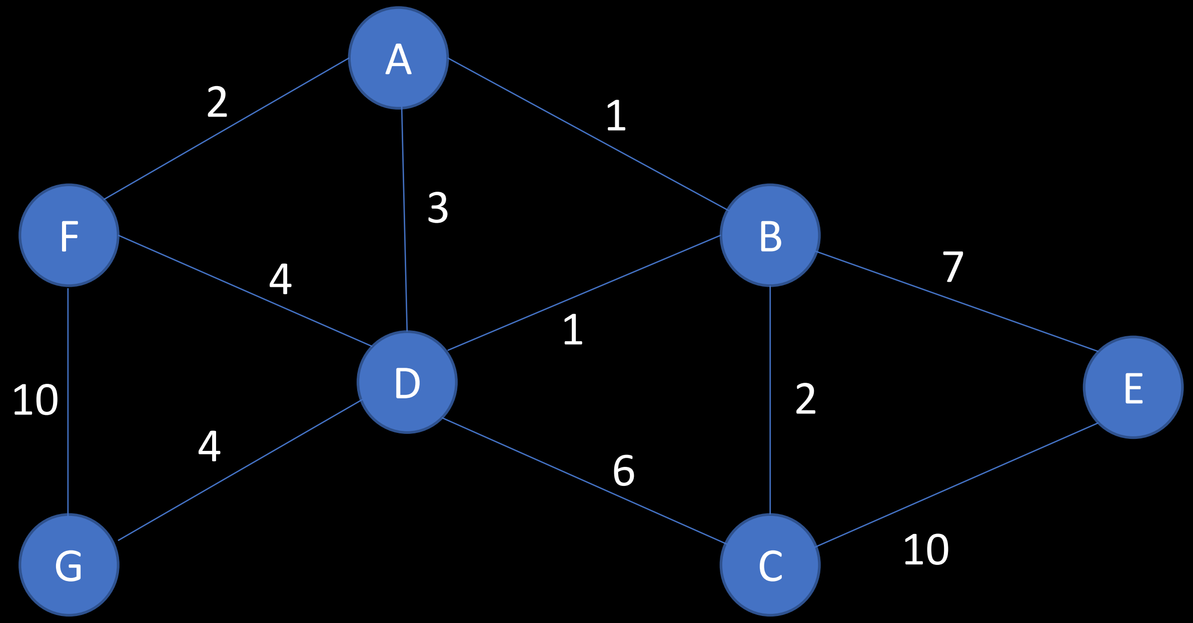
MST using Prims starting from “I”



I H B F A E D G C

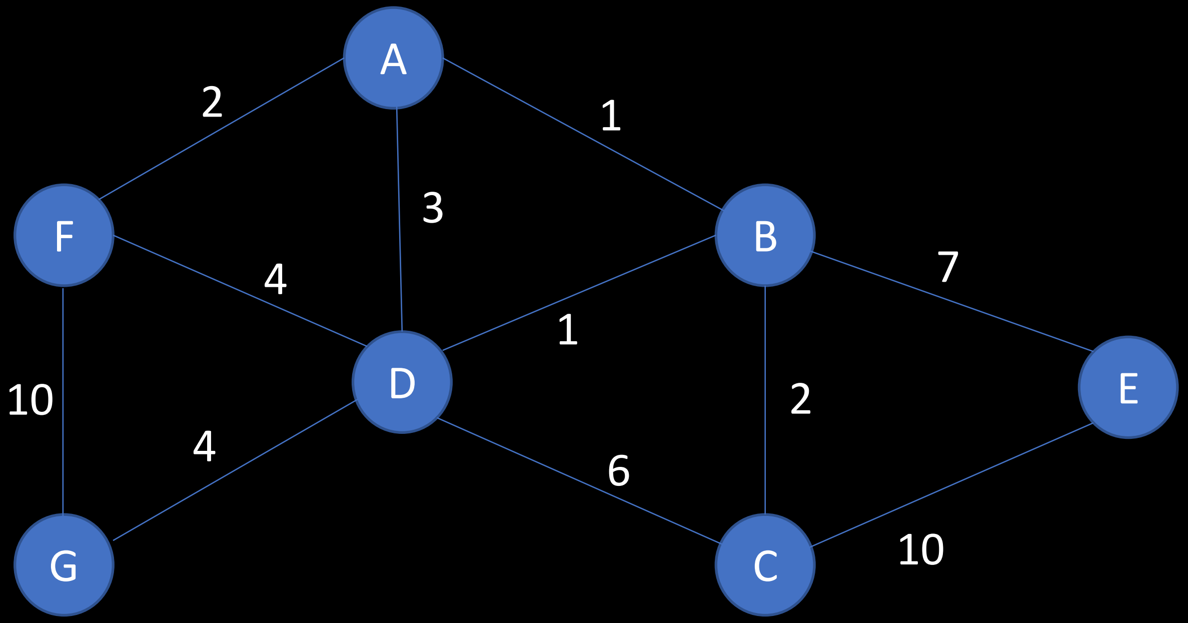
Dijkstra with A as source

v	D(v)	P(v)
A		
B		
C		
D		
E		
F		
G		

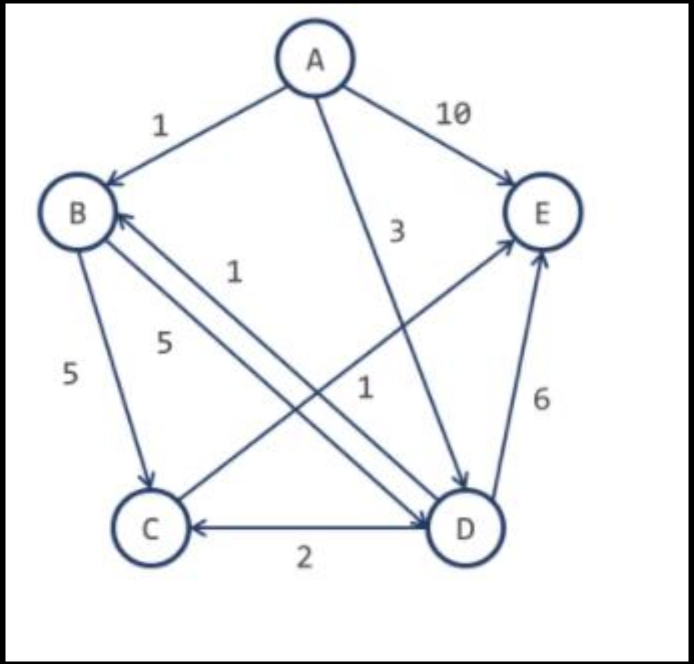


Dijkstra with A as source

v	D(v)	P(v)
A	0	NA
B	1	A
C	3	B
D	2	B
E	8	B
F	2	A
G	6	D

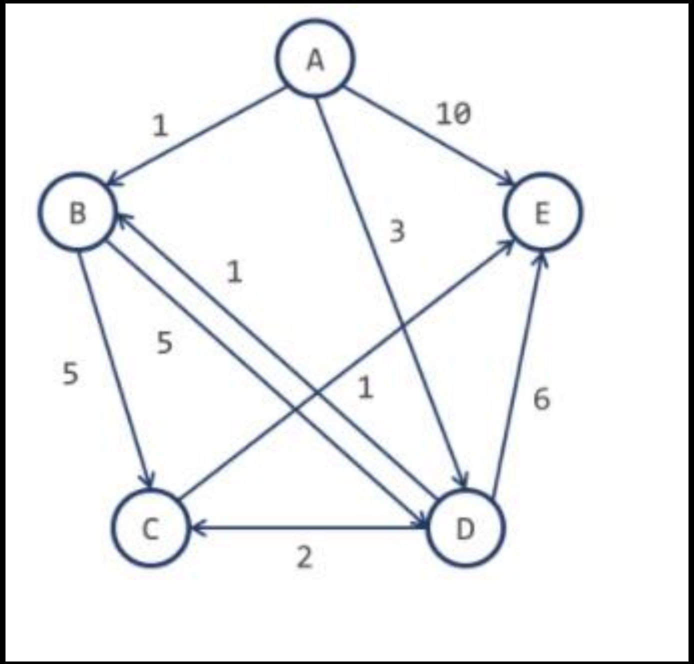


Dijkstra with A as source



V	d(V)	p(V)
B		
C		
D		
E		

Dijkstra with A as source



V	d(V)	p(V)
B	1	A
C	5	D
D	3	A
E	6	C

Algorithmic Paradigms

Algorithmic Paradigms

	Properties	Examples
Brute Force	▪ Generate and Test an Exhaustive Set of all possible combinations ▪ Can be computationally very expensive ▪ Guarantees optimal solution	▪ Finding divisors of a number, n by checking if all numbers from 1..n divides n without remainder ▪ Finding duplicates using all combinations ▪ Bubble/Selection Sort
Divide and Conquer	▪ Break the problem into subcomponents typically using recursion ▪ Solve the basic component ▪ Combine the solutions to sub-problems	▪ Quick Sort ▪ Merge Sort ▪ Binary Search ▪ Peak Finding
Dynamic Programming	▪ Optimal substructure: solution to a large problem can be obtained by solution to a smaller optimal problems ▪ Overlapping sub-problems: space of sub-problems must be small, that is, any recursive algorithm solving the problem should solve the same sub-problems over and over, rather than generating new sub-problems. ▪ Guarantees optimal solution	▪ Fibonacci Sequence ▪ Assembly Scheduling ▪ Knapsack
Greedy Algorithms	▪ Local optimal solutions at each stage ▪ Does not guarantee optimal solution	▪ Prim's Algorithm ▪ Dijkstra's Algorithm ▪ Kruskal's Algorithm

Bin Packing

Bin Packing

If we have packets that each require 7 units, 8 units, 2 units and 3 units of space, how many minimum bins are required to store all the four packets if each bin can take at most 10 units of space using the following Greedy strategies

- First Fit: scan the bins and place the new item in the first bin that is large enough.

- Best Fit: scan the bins and place the new item in the bin that finds the spot that creates the smallest empty space

Algorithm for Huffman Encoding

1. Create a table with symbols and their frequencies

Algorithm for Huffman Encoding

2. Construct a set of trees with root nodes that contain each of the individual symbols and their weight (frequency).
3. Place the set of trees into a priority queue.

Algorithm for Huffman Encoding

```
4. while the priority queue has more than one item
        Remove the two trees with the smallest weights.
        Combine them into a new binary tree in which the weight of the tree
        root is the sum of the weights of its children.
        Insert the newly created tree back into the priority queue.
```

Algorithm for Huffman Encoding

```
5. Traverse the resulting tree to obtain binary codes for characters
```

Questions