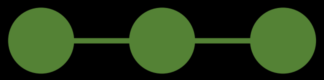
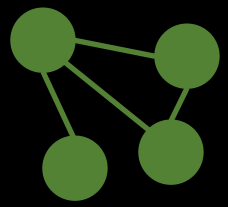
Final Exam Review

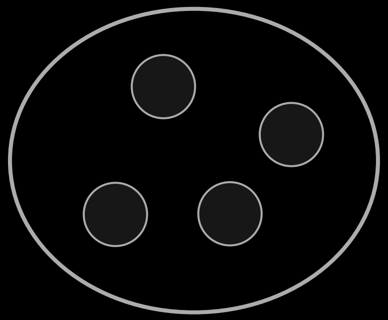
Data Structures & Algorithm

Categories of Data Structures

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







Announcements

- Final Exam will be on April 28th: 7:30 am to 9:30 am as per the date set by the registrar.
- The exam will be on Honorlock and will not contain any pseudocode questions.
- There are no makeups for the final and you must take it during the scheduled time.
- It will cover Modules 1-9 (everything except the Complexity Theory module).
- We will release the topic and expectations guide on April 22.
- You are allowed to use two pages of crib sheet with handwritten or printed notes on them.

Mentimeter

6647 386

Mentimeter

```cpp
#include <iostream>
#include <vector>

int sequence(int n)
{
    std::vector<int> dp1(n + 1, 0);
    std::vector<int> dp2(n + 2, 0);
    dp1[1] = 1;
    dp2[1] = 1;

    for(int i = 2; i <= n; i++)
    {
        dp1[i] = dp1[i - 1] + dp1[i - 2];
        dp2[i] = dp1[i] * dp1[i];
    }

    return dp2[n];
}

int main()
{
    std::cout << sequence(6);
    return 0;
}
```

Mentimeter

	0	1	2	3	4	5	6	7
{}	0	0	0	0	0	0	0	0
{1}	0							
{1, 2}	0							
{1, 2, 3}	0							
{1, 2, 3, 4}	0							

	v_i	w_i
1	3	1
2	6	3
3	28	5
4	30	7

$$OPT(i, W) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1, w) & \text{if } w_i > W \\ \max\{OPT(i-1, w), v_i + OPT(i-1, W-w_i)\} & \text{otherwise} \end{cases}$$

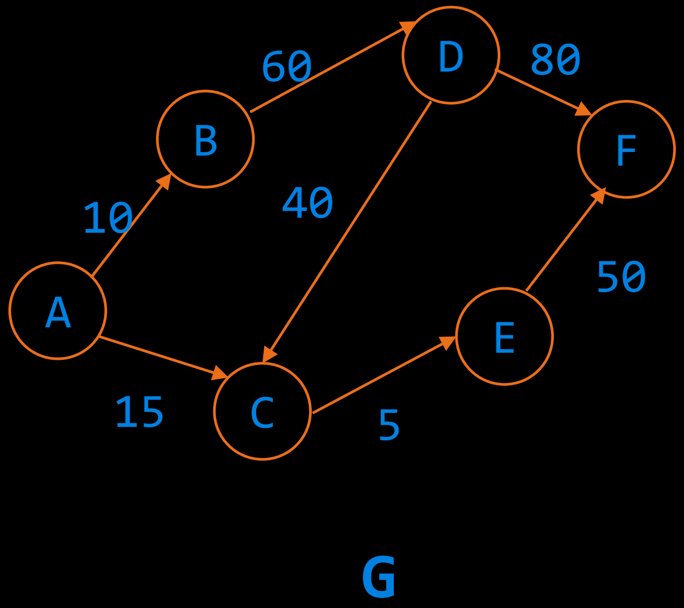
Mentimeter

	0	1	2	3	4	5	6	7
{}	0	0	0	0	0	0	0	0
{1}	0	3	3	3	3	3	3	3
{1, 2}	0	3	3	6	9	9	9	9
{1, 2, 3}	0	3	3	6	9	28	31	31
{1, 2, 3, 4}	0	3	3	6	9	28	31	31

	v_i	w_i
1	3	1
2	6	3
3	28	5
4	30	7

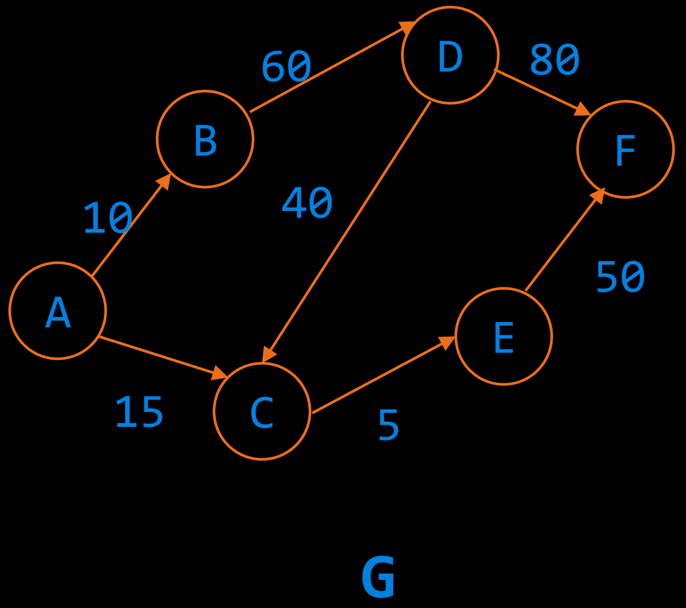
$$OPT(i, W) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1, w) & \text{if } w_i > W \\ \max\{OPT(i-1, w), v_i + OPT(i-1, W-w_i)\} & \text{otherwise} \end{cases}$$

Common Representations



- Edge List
- Adjacency Matrix
- Adjacency List

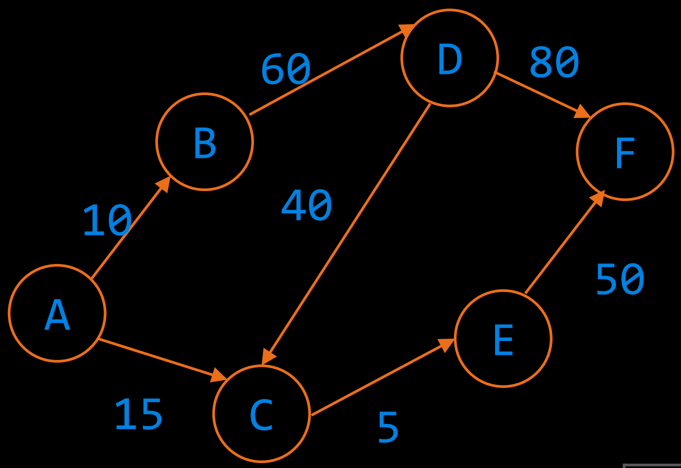
Edge List



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

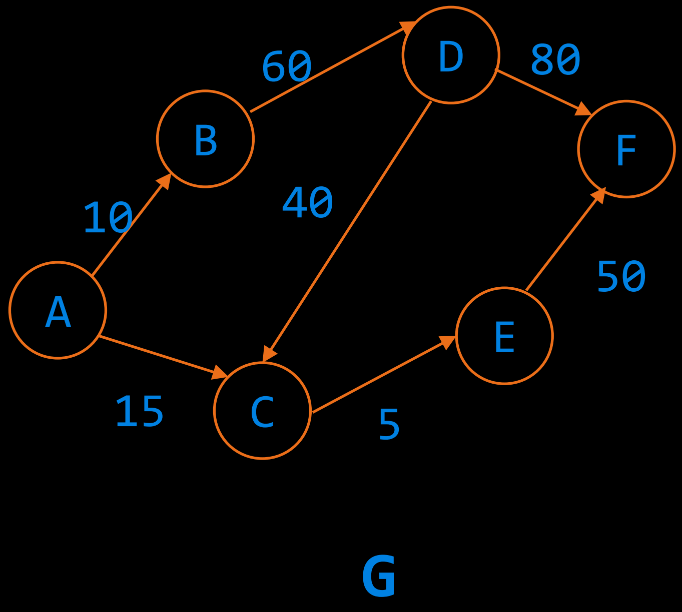
What are A's adjacent nodes?

~ O(E)

$O(|E|) \sim O(|V| * |V|)$

Space: O(E)

Adjacency Matrix



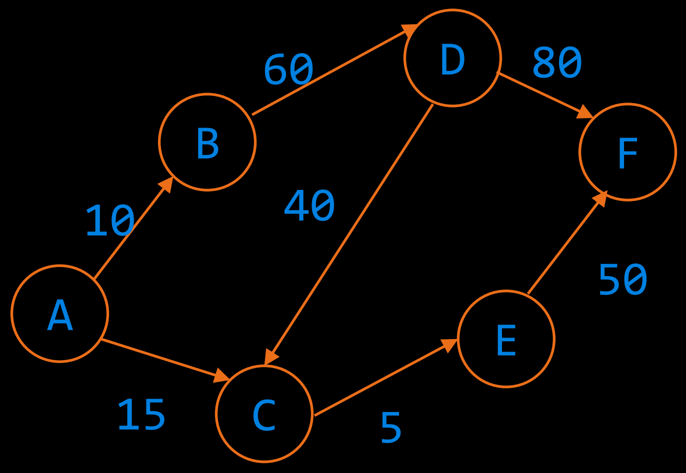
	A	B	C	D	E	F
A	0	10	15	0	0	0
B	0	0	0	60	0	0
C	0	0	0	0	5	0
D	0	0	40	0	0	80
E	0	0	0	0	0	50
F	0	0	0	0	0	0

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

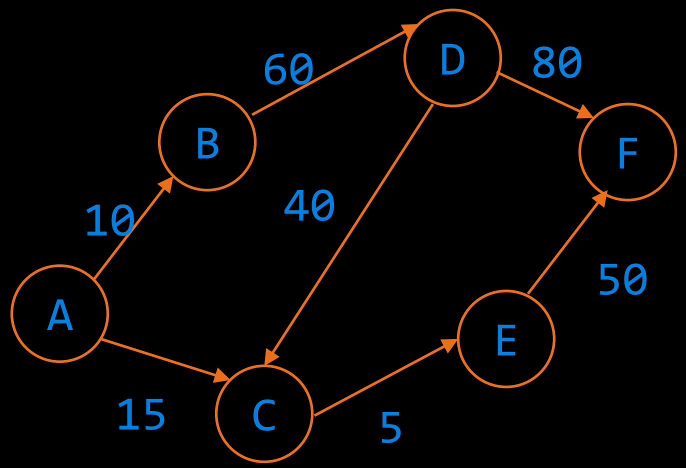
Map
A 0
B 1
C 2
D 3
E 4
F 5

	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)

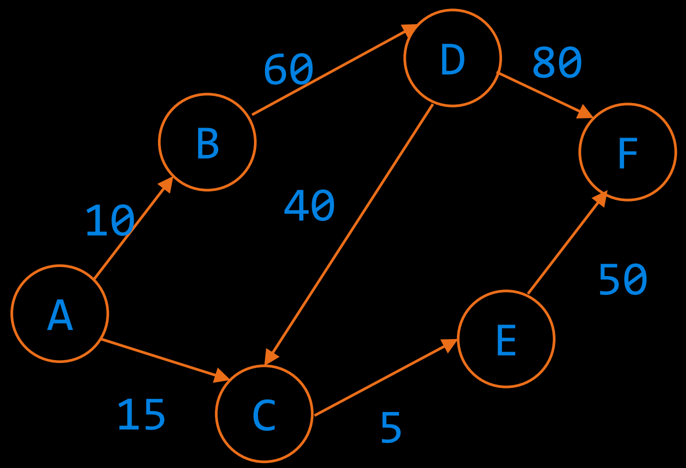
2. Adjacency

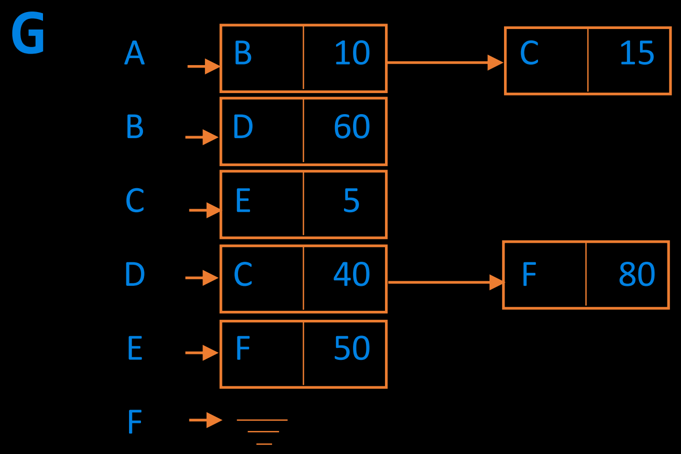
What are A’s adjacent nodes?

```
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|)
```

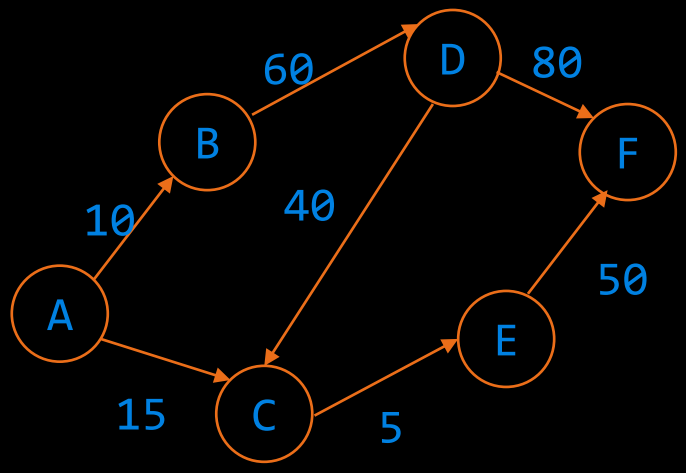
2. Adjacency

What are A’s adjacent nodes?

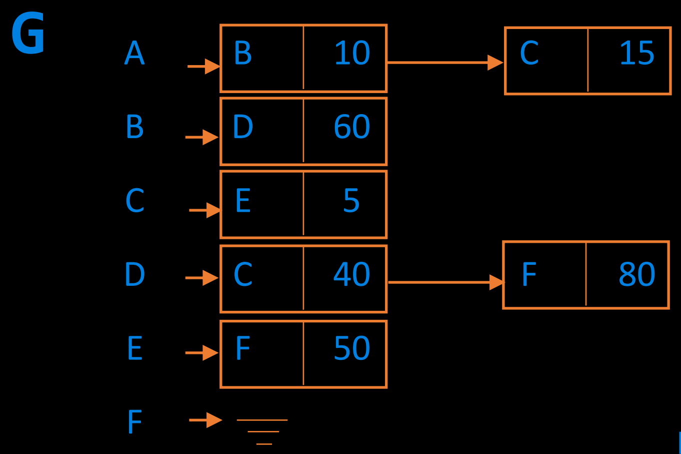
G[“A”] ~ O(outdegree|V|)

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?

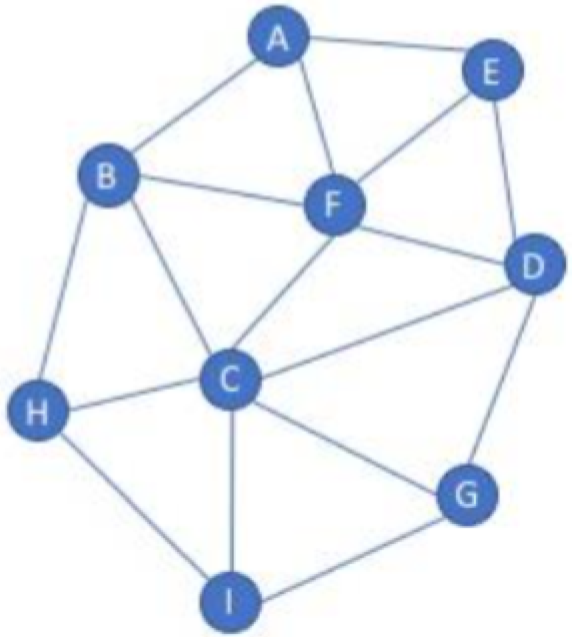
a) A F B E D C H G I

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

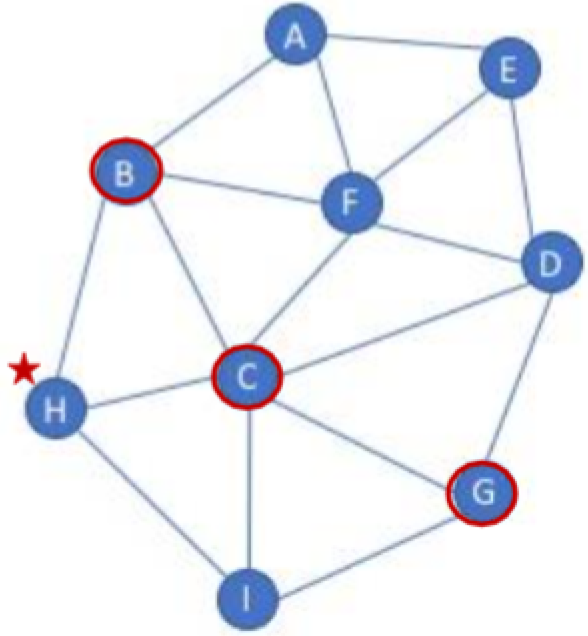
d) E A F D B H C I G

e) F A E D C B G I H

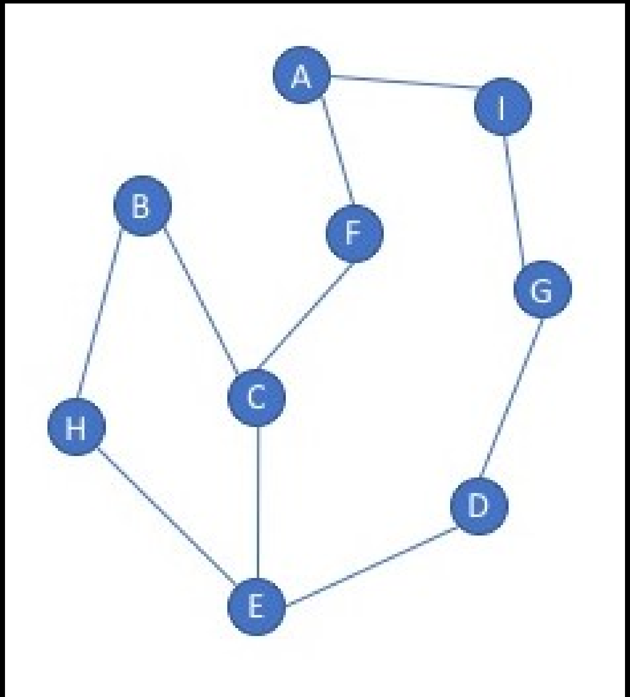
All the options except for d

Why not d?

** H is visited before C and G

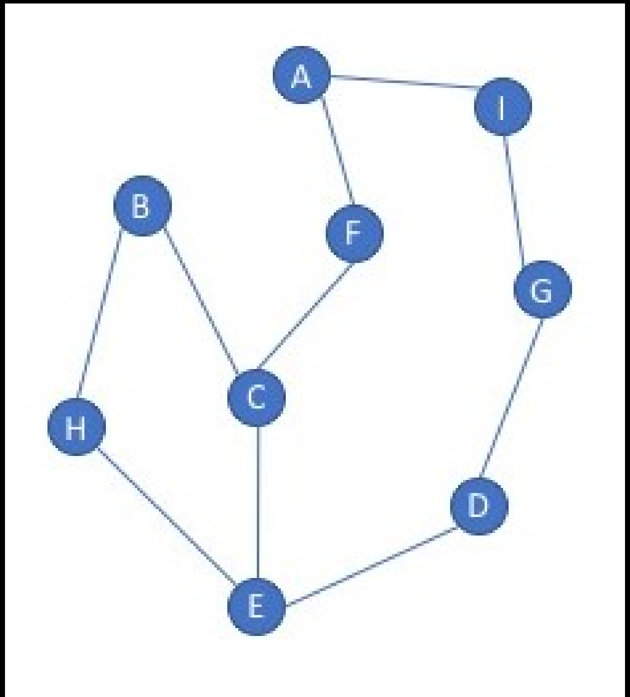


Valid DFS: Which DFS are valid?

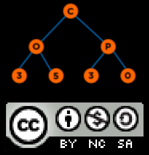


- H E C B D G I A F
- C E H B D G I A F
- A F C E H B I G D
- D E C B H F A I G

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



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

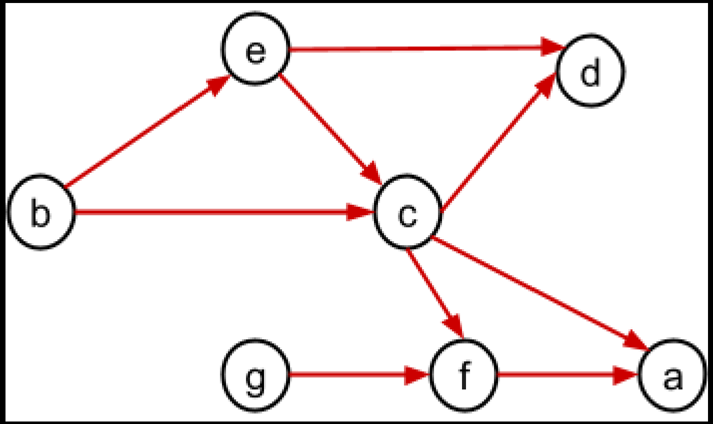
Prim’s or Kruskals
BFS
DFS
Topological Sort
Dijkstra’s Algorithm

Graph Algorithm Mix n Match

- 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

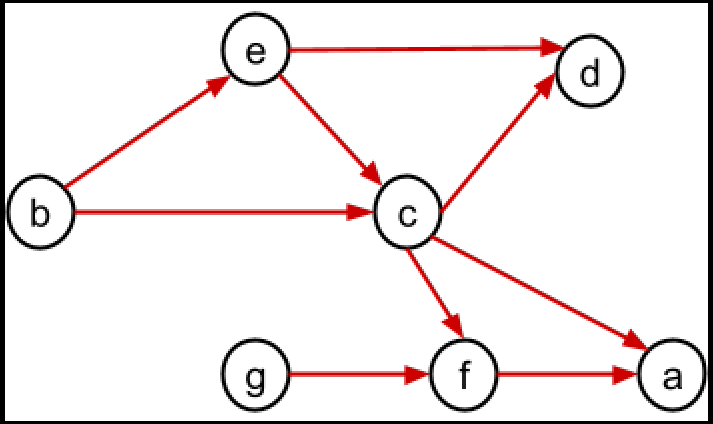
Prim's or Kruskals
BFS
DFS
Topological Sort
Dijkstra's Algorithm

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

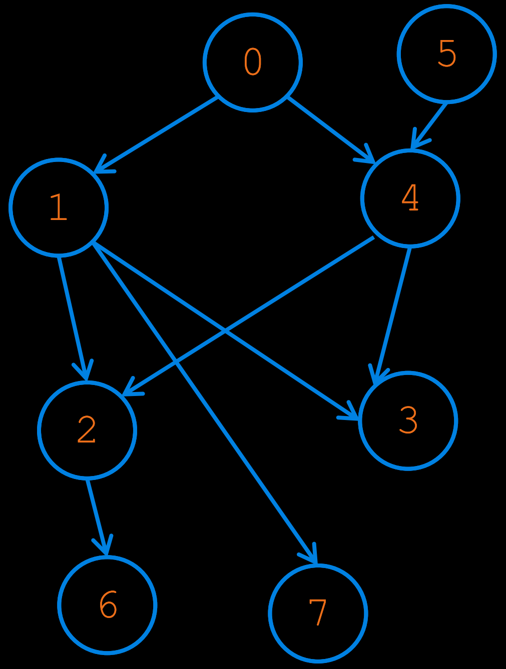
Which of the choices below represent a valid topological sort ordering of this graph?



- 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?

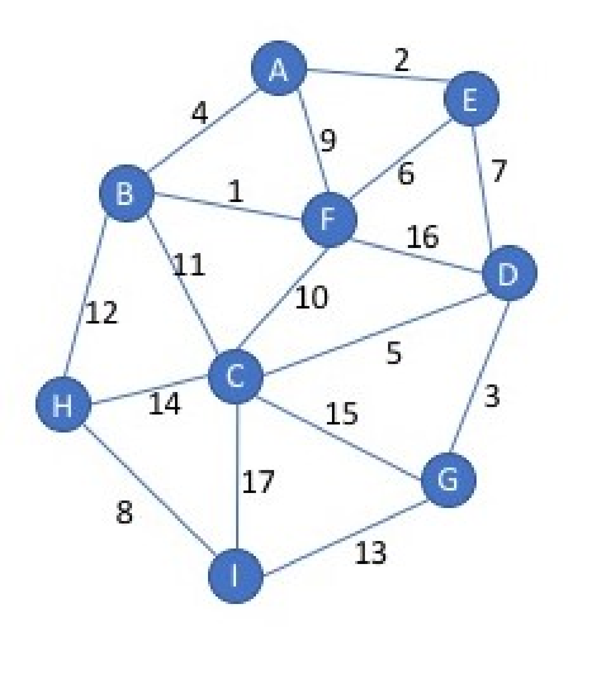
Scenario

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.

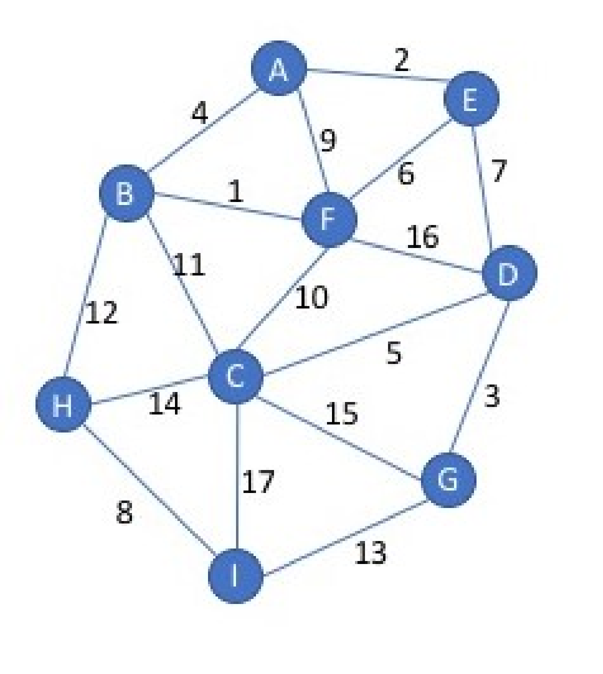
Which graph algorithm that we discussed in class could they use to solve this problem? What are the vertices, what are the edges, what are the edge values?

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

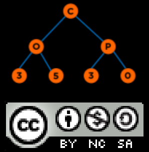
MST using Prims starting from “I”



MST using Prims starting from “I”

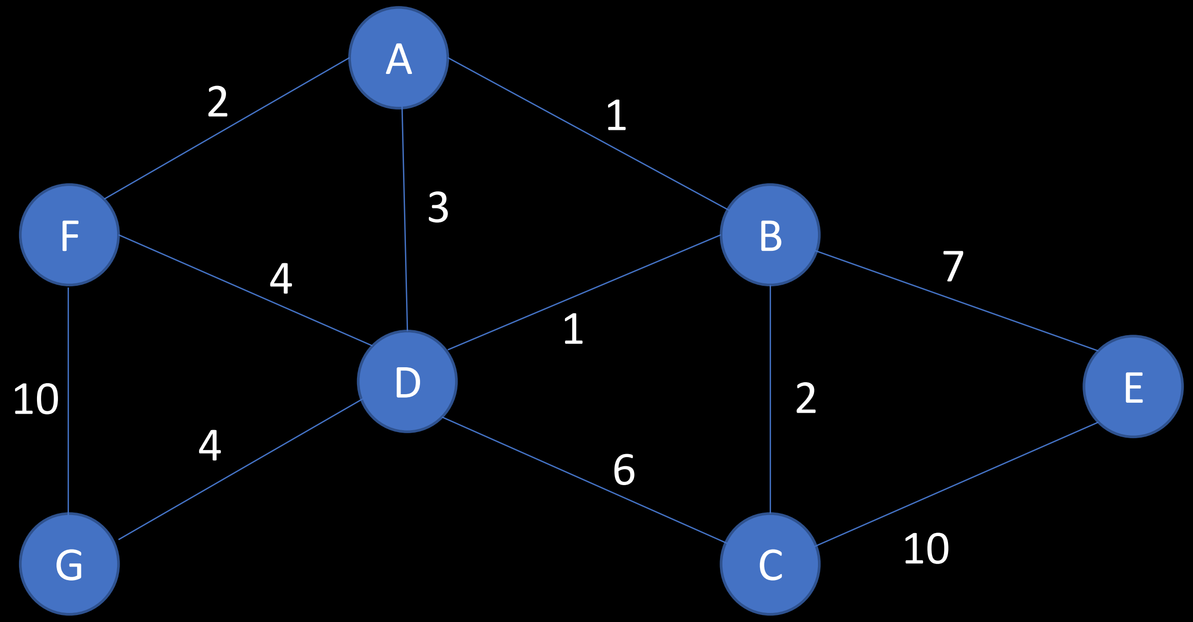


I H B F A E D G C



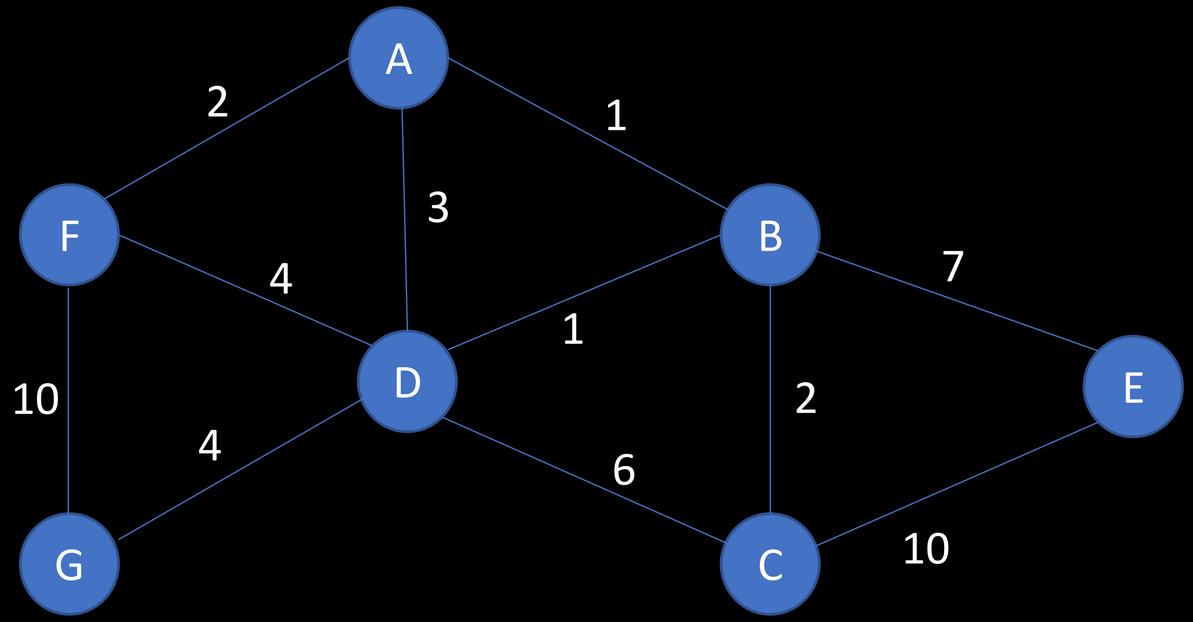
Dijkstra with A as source

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

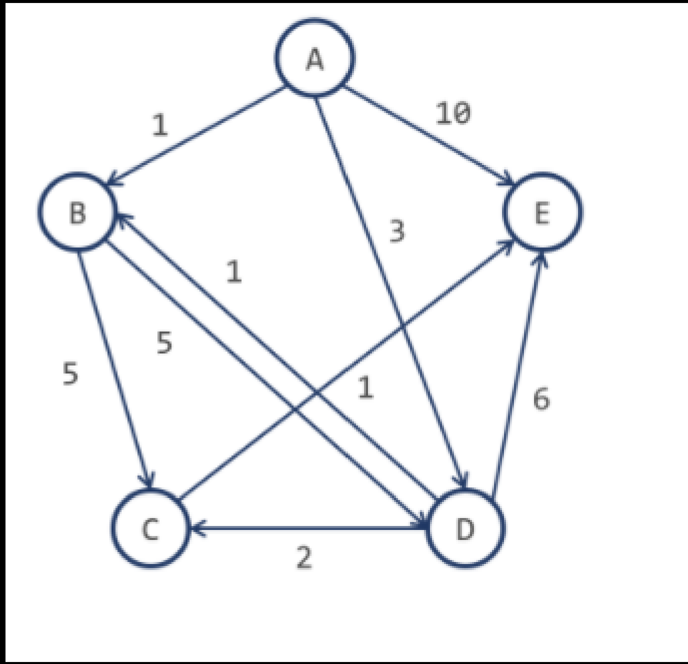


Dijkstra with A as source

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

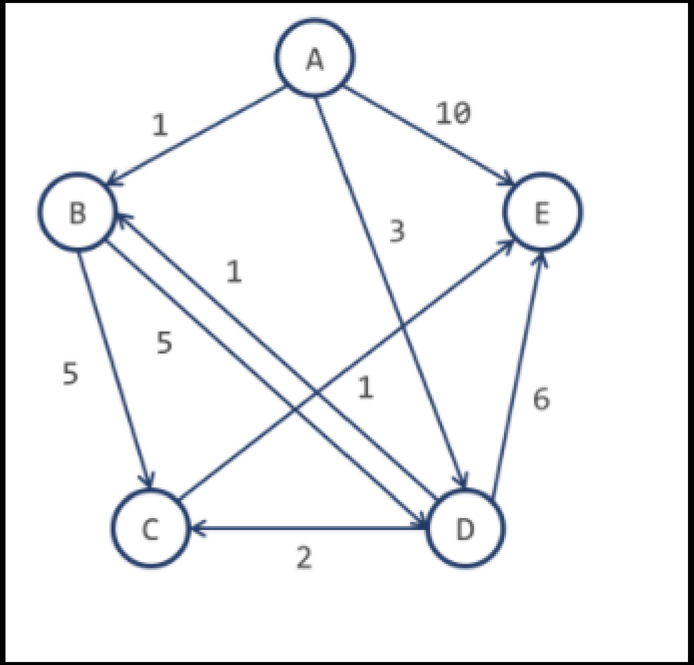


Dijkstra with A as source



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

Algorithm for Huffman Encoding

Given this file, generate a Huffman Tree and identify the codes of each character.

```
care racecar era
```

Algorithm for Huffman Encoding

1. Create a table with symbols and their frequencies

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 min priority queue.

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.

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

Algorithm for Huffman Encoding

1. Create a table with symbols and their frequencies

```
care racecar era
```

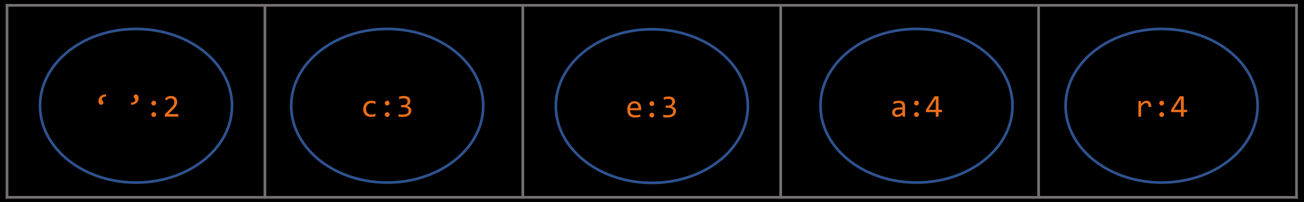
Character	Frequency
a	4
r	4
c	3
e	3
‘ ’	2

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 min priority queue.

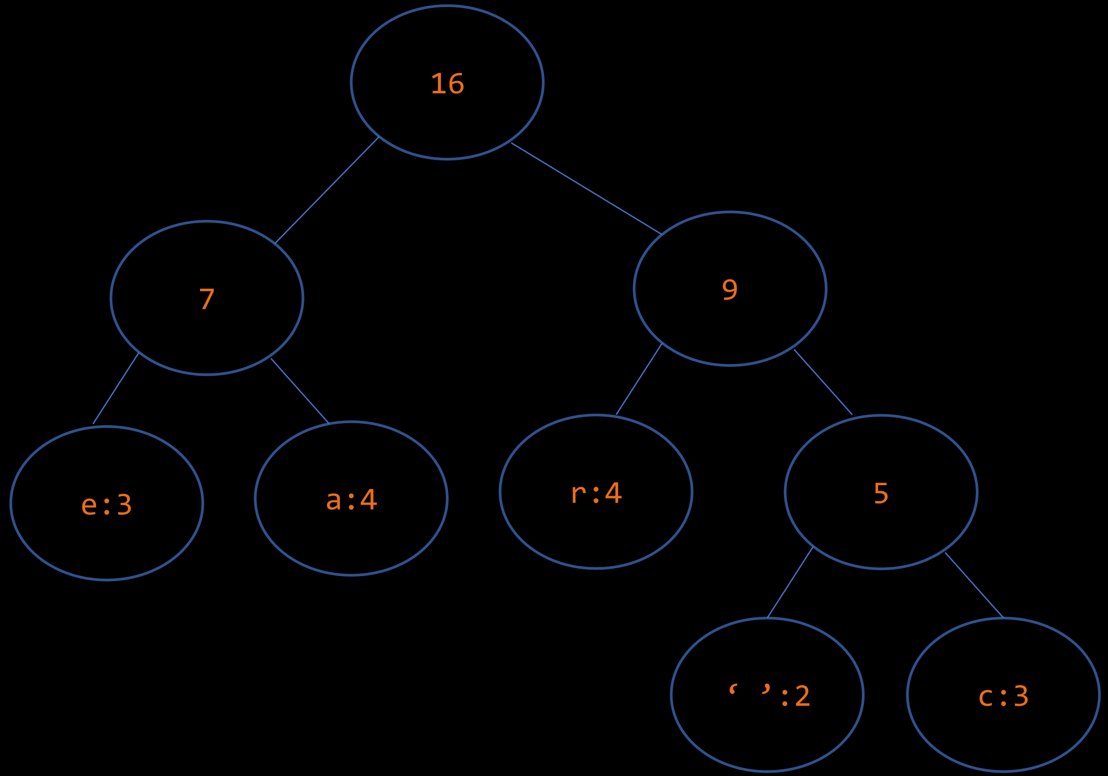
Character	Frequency
a	4
r	4
c	3
e	3
‘ ’	2



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.

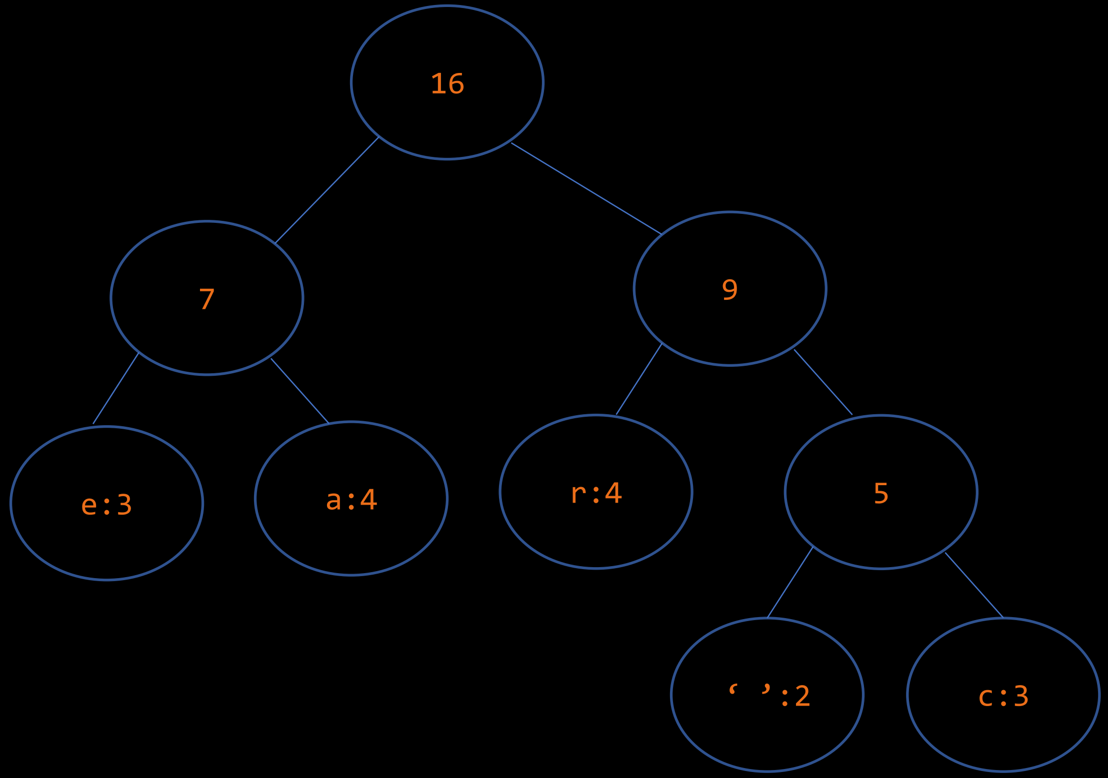
Character	Frequency
a	4
r	4
c	3
e	3
‘ ’	2



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.

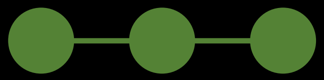
Character	Frequency	Huffman Code
a	4	01
r	4	10
c	3	111
e	3	00
‘ ’	2	110

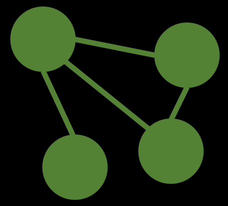


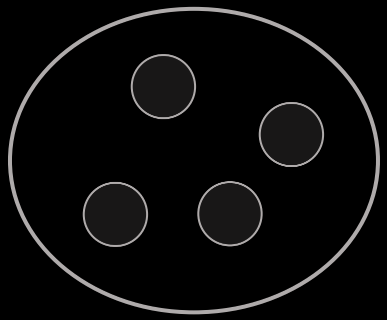
Questions

Categories of Data Structures

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







Categories of Algorithms

Brute Force	Divide & Conquer	Greedy	Dynamic Programming
Selection Sort	Binary Search	Minimum Spanning Tree	Knapsack
Bubble Sort	Merge Sort	Shortest Paths	Fibonacci
Insertion Sort	Quick Sort		
NP Complete Problems