# Final Exam Review

Data Structures & Algorithm

# Categories of Data Structures

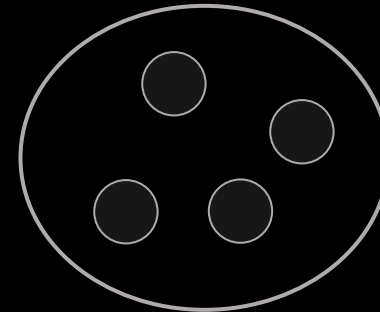| Linear Ordered | Non-linear Ordered | Not Ordered |
|:---:|:---:|:---:|
| Lists | Trees | Sets |
| Stacks | Graphs | Tables/Maps |
| Queues | | |
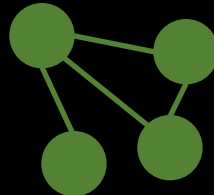
# Announcements

- You can take the Exam 2 between July 29-July 31.

- The exam will be on Honorlock.

- It will cover Modules 1-3, 5-8 (everything till Topological Sort).

- Topics and expectations guide on Canvas.

- You are allowed to use one page of crib sheet with handwritten notes + 4 sheets of blank scratch paper.

# Mini Review – Linked Lists

Consider a class List that implements an ordered list backed by a singly linked list with a head pointer. The invariant "ordered" is maintained always. Given that representation, what is the worst-case time complexity of the following operations? Assume the list is sorted in ascending order.

A. Insert an item
B. Finding the minimum element
C. Delete the largest element from list
D. Finding the largest element
E. Finding a random element, n
F. Deleting the minimum element in the list

# Mini Review – Linked Lists

Consider a class List that implements an ordered list backed by a singly linked list with a head pointer. The invariant "ordered" is maintained always. Given that representation, what is the worst-case time complexity of the following operations? Assume the list is sorted in ascending order.

A. Insert an item : O(n)
B. Finding the minimum element : O(1)
C. Delete the largest element from list : O(n)
D. Finding the largest element : O(n)
E. Finding a random element, n : O(n)
F. Deleting the minimum element in the list : O(1)

# Mini Review – Stacks

Worst case time complexity for an array-based queue for following operations:

1. enqueue()
2. dequeue()
3. isEmpty()
4. top()

# Mini Review – Stacks

Worst case time complexity for an array-based queue for following operations:

1. enqueue() – O(1)
2. dequeue() – O(1)
3. isEmpty() – O(1)
4. top()    – O(1)

# Mini Review – Lists

What is the output of the following code:

```cpp
#include <iostream>
#include <list>

int main ()
{
  std::list<int> mylist = {34, 77, 16, 2};

  std::cout << "List contains: ";
  auto it = mylist.end();
  while(it != mylist.begin())
  {
    std::cout << *(--it) << " ";
  }

  return 0;
}
```

# Mini Review – Lists

What is the output of the following code:

```cpp
#include <iostream>
#include <list>

int main ()
{
  std::list<int> mylist = {34, 77, 16, 2};

  std::cout << "List contains: ";
  auto it = mylist.end();
  while(it != mylist.begin())
  {
    std::cout << *(--it) << " ";
  }

  return 0;
}
```
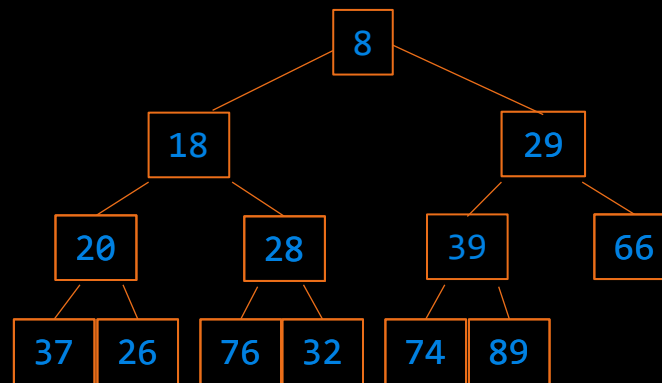
List contains: 2 16 77 34

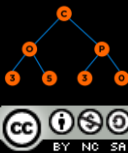# Binary Heap

## Heap Representation

```
int Heap[];
```

```
          8

    18         29

  20   28    39    66

37 26 76 32 74 89
```

For a node at position *p*,

  L. child position:  2*p* + 1
  R. child position:  2*p* + 2

 A node at position *c* can find
its parent at floor((*c* – 1)/2)

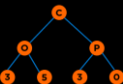| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 8 | 18 | 29 | 20 | 28 | 39 | 66 | 37 | 26 | 76 | 32 | 74 | 89 | |

# Binary Heap Insertion
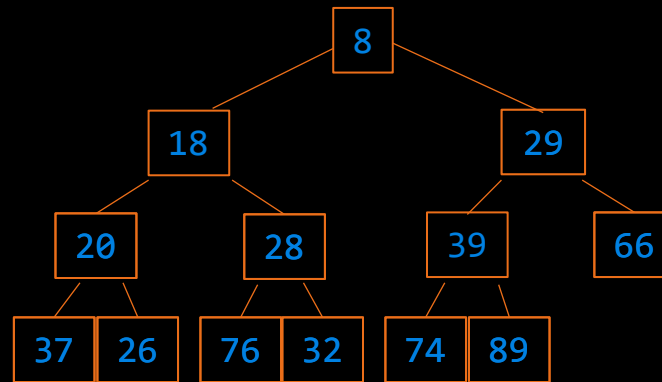
**Heap Insertion**



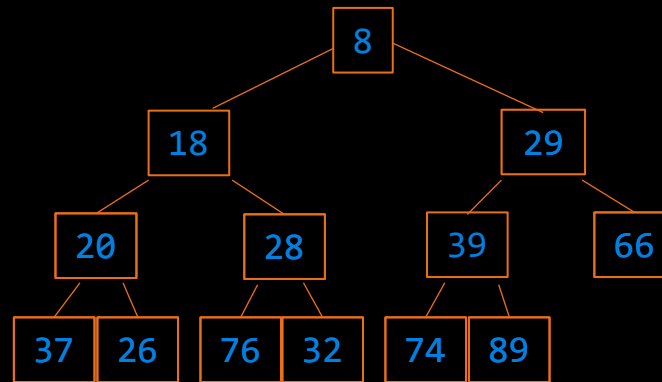## Algorithm for Inserting in a Heap

1. Insert the new item in the next position at the bottom of the heap.
2. `while` new item is not at the root and new item is smaller than its parent
3.     Swap the new item with its parent, moving the new item up the heap.
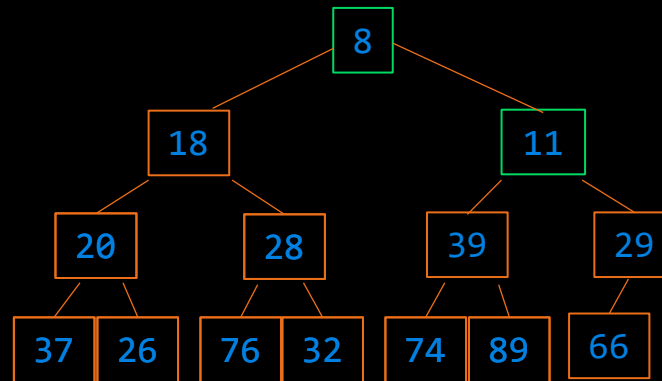
# Binary Heap Insertion

## Heap Insertion

```
8
18          29
20    28    39    66
37 26  76 32  74 89
```

1. Insert the new element at the end of the array and set child to arr.size() - 1
2. Set parent to (child - 1)/ 2
3. while (parent >= 0 and arr[parent] > arr[child])
    Swap arr[parent] and arr[child]
    Set child equal to parent
    Set parent equal to (child-1)/2

insert 11

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 8 | 18 | 29 | 20 | 28 | 39 | 66 | 37 | 26 | 76 | 32 | 74 | 89 | |

# Binary Heap Insertion

**Heap Insertion**



```
1. Insert the new element at the end of the array and set
   child to arr.size() – 1
2. Set parent to (child – 1)/ 2
3. while (parent >= 0 and arr[parent] > arr[child])
      Swap arr[parent] and arr[child]
      Set child equal to parent
      Set parent equal to (child-1)/2
```
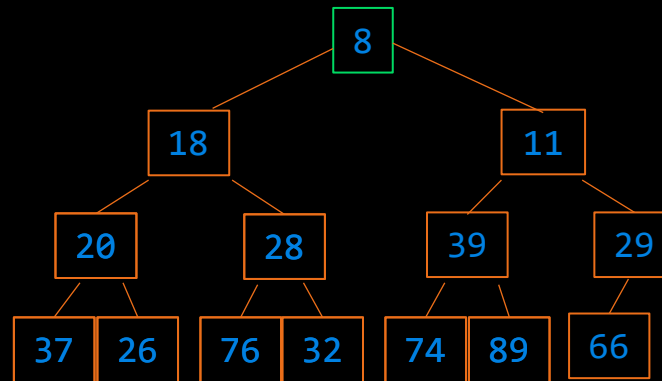
```
child = 13 | 6 | 2
parent = 6 | 2 | 0
```

O(log n) time to insert!

# Binary MinHeap Deletion
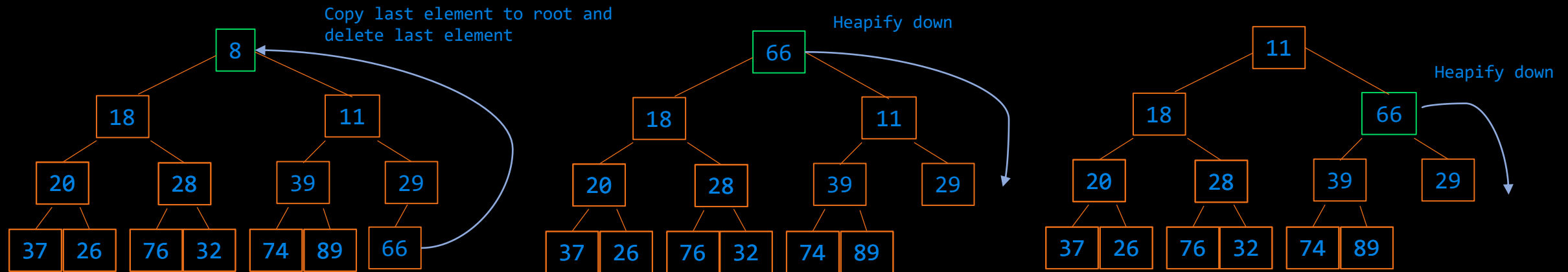
**Heap Deletion (ExtractMin)**



**Algorithm for Removal from a Heap**

1. Remove the item in the root node by replacing it with the last item in the heap (LIH).
2. **while** item LIH has children and item LIH is larger than either of its children
3.     Swap item LIH with its smaller child, moving LIH down the heap.

# Binary MinHeap Deletion

## Heap Deletion (ExtractMin)



Copy last element to root and delete last element

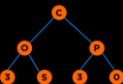Heapify down

Heapify down

**Algorithm for Removal from a Heap**

1. Remove the item in the root node by replacing it with the last item in the heap (LIH).
2. while item LIH has children and item LIH is larger than either of its children
3.     Swap item LIH with its smaller child, moving LIH down the heap.
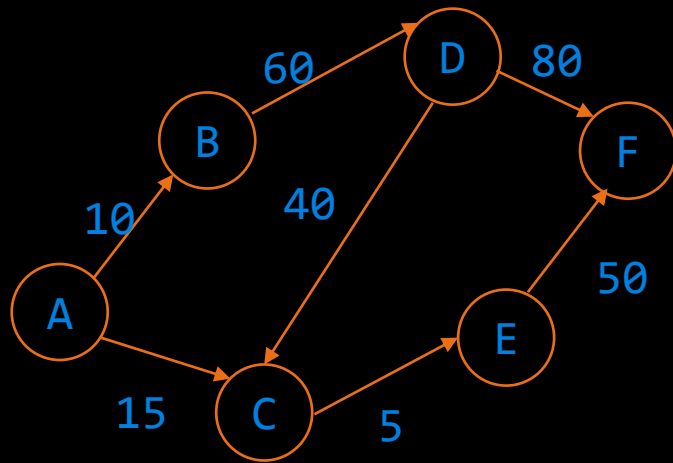
# Binary MinHeap Deletion

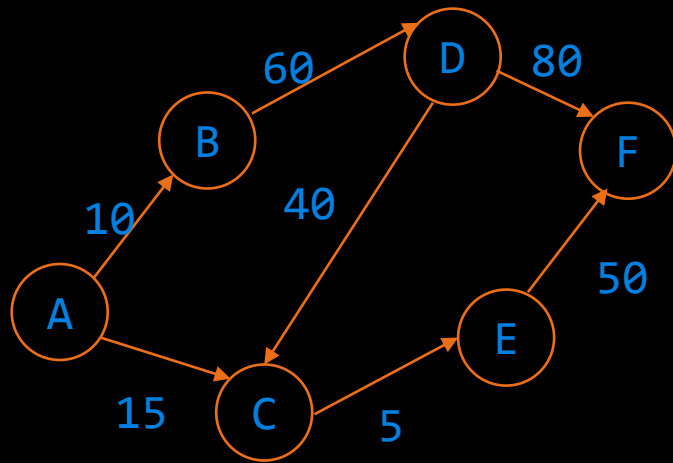**Heap Deletion (ExtractMin)**
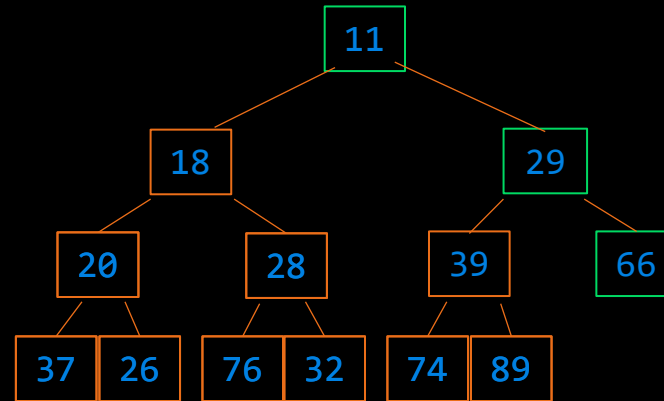


O(log n) time to ExtractMin!

# Common Representations



- Edge List

- Adjacency Matrix
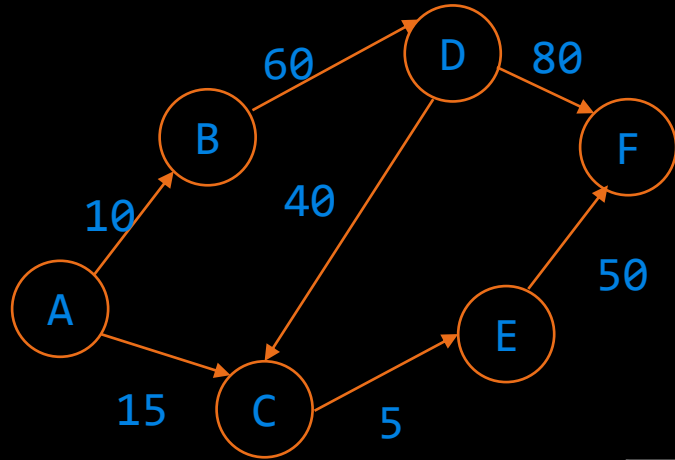
- Adjacency List

# 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



60    D    80

B

F

10

40

A

50

E

15    C    5

**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
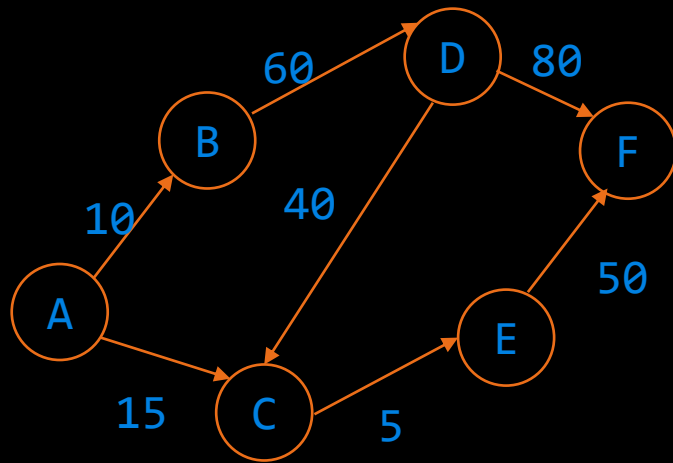


|   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 10 | 15 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 60 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 5 | 0 |
| D | 0 | 0 | 40 | 0 | 0 | 80 |
| E | 0 | 0 | 0 | 0 | 0 | 50 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |

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)

```
01  #include <iostream>
02  #include<map>
03  #define VERTICES 6
04  using namespace std;
05  int main()
06  {
07        int no_lines, wt, j=0;
08        string from, to;
09        int graph [VERTICES][VERTICES] = {0};
10        map<string, int> mapper;
11        cin >> no_lines;
12        for(int i = 0; i < no_lines; i++)
13        {
14              cin >> from >> to >> wt;
15              if (mapper.find(from) == mapper.end())
16                    mapper[from] = j++;
17              if (mapper.find(to) == mapper.end())
18                    mapper[to] = j++;
19              graph[mapper[from]][mapper[to]] = wt;
20        }
21        return 0;
22  }
```
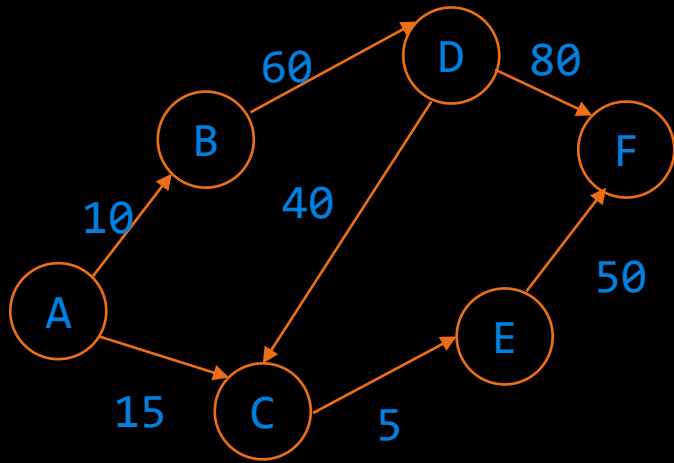
D    B    F    A    C    E    G

60   80   10   40   50   15   5

Input

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

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 |

https://www.onlinegdb.com/Hy8M0CnsS

# Adjacency Matrix

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"]
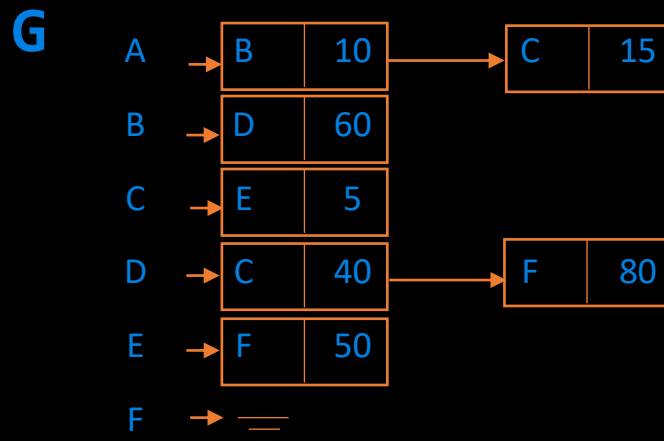       if x ! = 0

                          ~ O(|V|)

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

**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  |

# Adjacency List



**G**

A → | B | 10 | → | C | 15 |

B → | D | 60 |

C → | E | 5 |

D → | C | 40 | → | F | 80 |

E → | F | 50 |

F → 
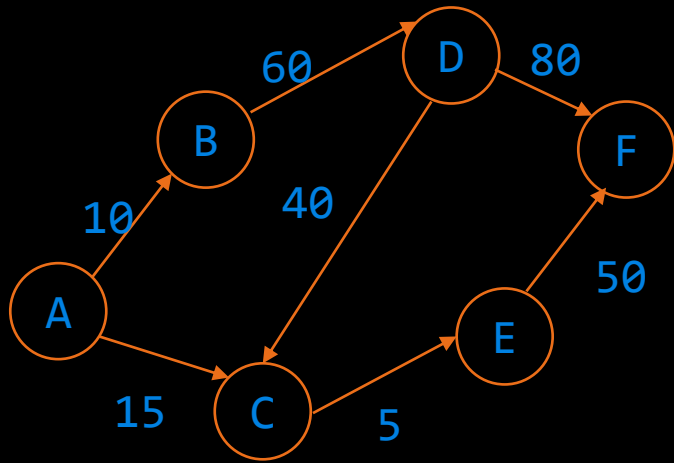
**Sparse Graph:**
Edges ~ Vertices

Common Operations:

1. Connectedness

   Is A connected to B?

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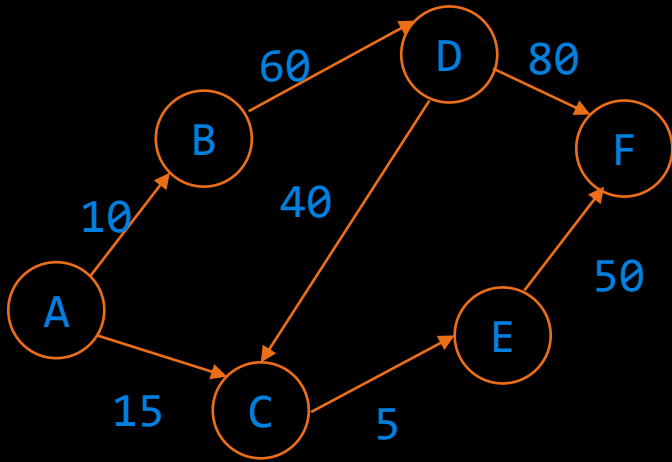   **if x ! = 'B'**
   **~ O(outdegree|V|)**

2. Adjacency

   What are A's adjacent nodes?

   **G["A"] ~ O(outdegree|V|)**
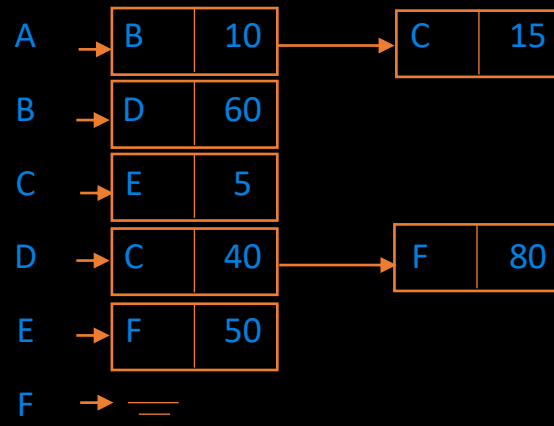
Space: **O(|V| + |E|)**

# Adjacency List Implementation

If to or from vertex not present add vertex

Otherwise add edge at the end of the list

```
01  #include <iostream>
02  #include<map>
03  #include<vector>
04  #include<iterator>
05  using namespace std;
06
07  int main()
08  {
09          int no_lines;
10          string from, to, wt;
11          map<string, vector<pair<string,int>>> graph;
12          cin >> no_lines;
13          for(int i = 0; i < no_lines; i++)
14          {
15                  cin >> from >> to >> wt;
16                  graph[from].push_back(make_pair(to, stoi(wt)));
17                  if (graph.find(to)==graph.end())
18                          graph[to] = {};
19          }
20  }
```

60    D    80
  B           F

10         40
                   50
A                E

15    C    5

G

A → | B | 10 | → | C | 15 |

B → | D | 60 |

C → | E | 5 |

D → | C | 40 | → | F | 80 |

E → | F | 50 |

F → ⎯

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

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
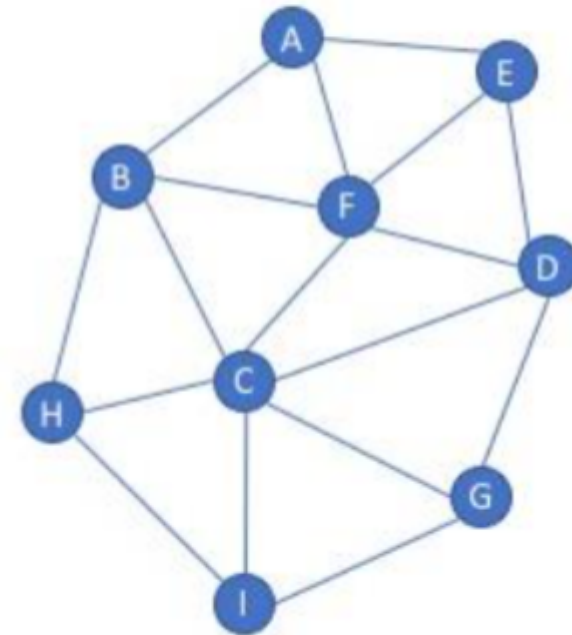
e)   F A E D C B G I H

# 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
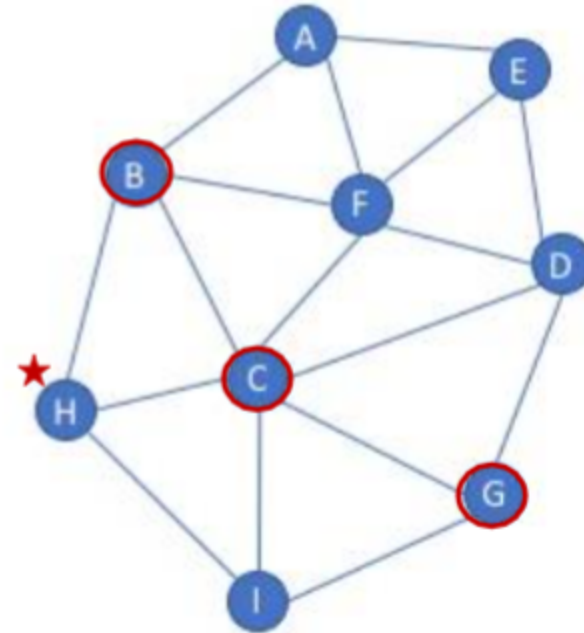
# Valid DFS: Which DFS are valid?
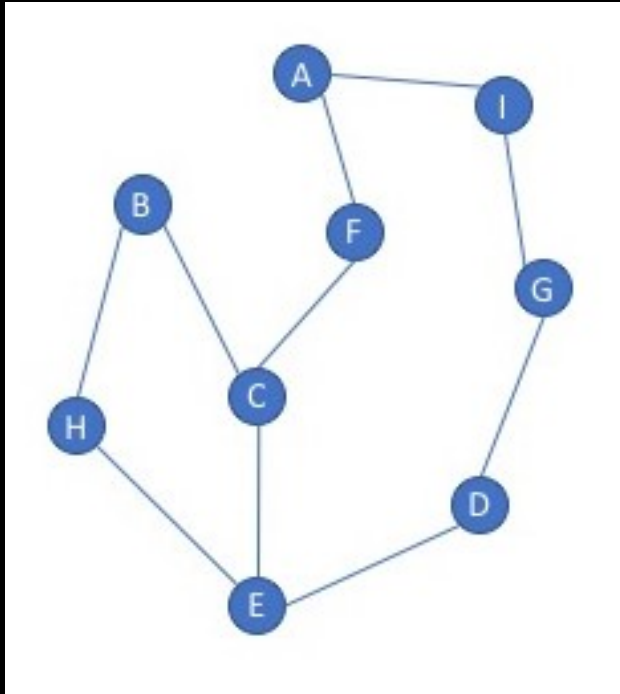


- H E C B D G I A F
- C E H B D G I A F
- A F C E H B I G D
- D E C B H F A I G

# Valid DFS: Which DFS are valid?
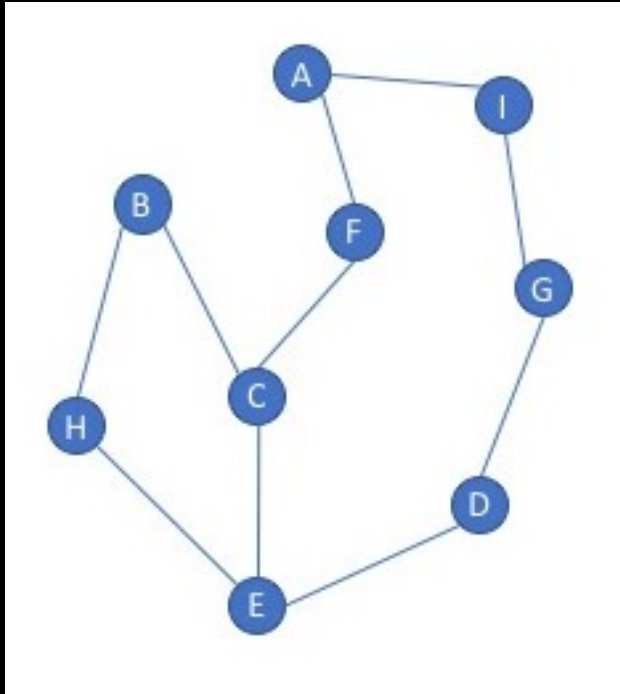


- H E C B D G I A F
- C E H B D G I A F
- A F C E H B I G D
- D E C B H F A I G

# 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

```
01  string source = "A";
02  std::set<string> visited;
03  std::queue<string> q;
04
05  visited.insert(source);
06  q.push(source);
07  cout<<"BFS: ";
08
09  while(!q.empty())
10  {
11      string u = q.front();
12      cout << u;
13      q.pop();
14      vector<string> neighbors = graph[u];
15      for(string v: neighbors)
16      {
17          if(visited.count(v)==0)
18          {
19              visited.insert(v);
20              q.push(v);
21          }
22      }
23  }
```

```
01  string source = "A";
02  std::set<string> visited;
03  std::stack<string> s;
04
05  visited.insert(source);
06  s.push(source);
07  cout<<"DFS: ";
08
09  while(!q.empty())
10  {
11      string u = s.top();
12      cout << u;
13      s.pop();
14      vector<string> neighbors = graph[u];
15      for(string v: neighbors)
16      {
17          if(visited.count(v)==0)
18          {
19              visited.insert(v);
20              s.push(v);
21          }
22      }
23  }
```

Theoretical Complexity: O(V+E)
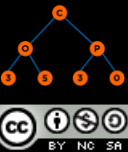
# Graph Algorithm Mix n Match

- Finds the shortest paths in a weighted graph
- Find the minimum cost connected network
- Scheduling algorithm, list steps in a process
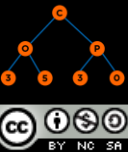- Finds the shortest path in an unweighted graph
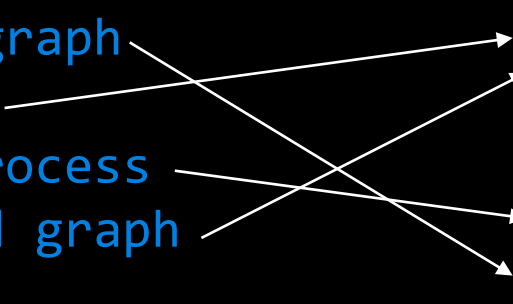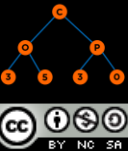
Prim's or Kruskals
BFS
DFS
Topological Sort
Dijkstra's Algorithm

# Graph Algorithm Mix n Match

- 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

# 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?

```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;
}
```
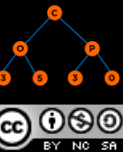
Returns whether a given vertex is reachable from another vertex using DFS
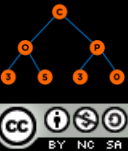
# Scenario

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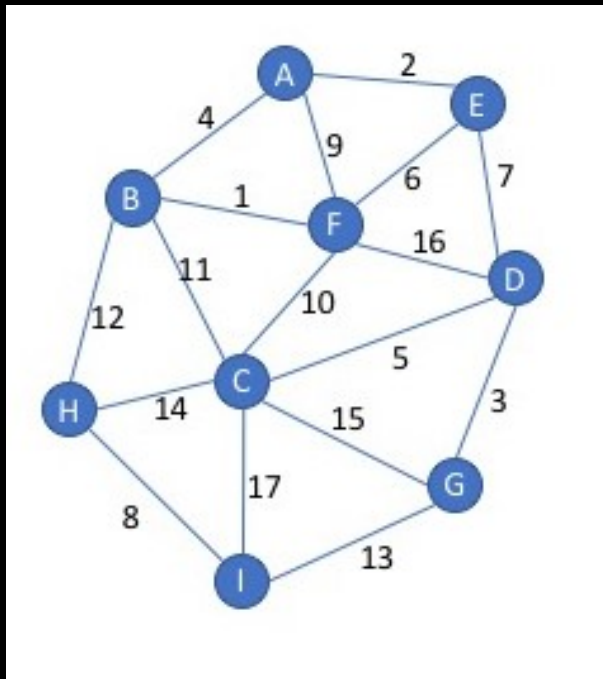
Which graph algorithm that we discussed in class could they use to solve this problem? What are the vertices, what are the edges, what are the edge values?

- Prim's or Kruskals algorithm for minimum spanning tree.
- Roads are edges.
- Ends of roads are vertices.
- Edge weights are cost for maintaining roads.
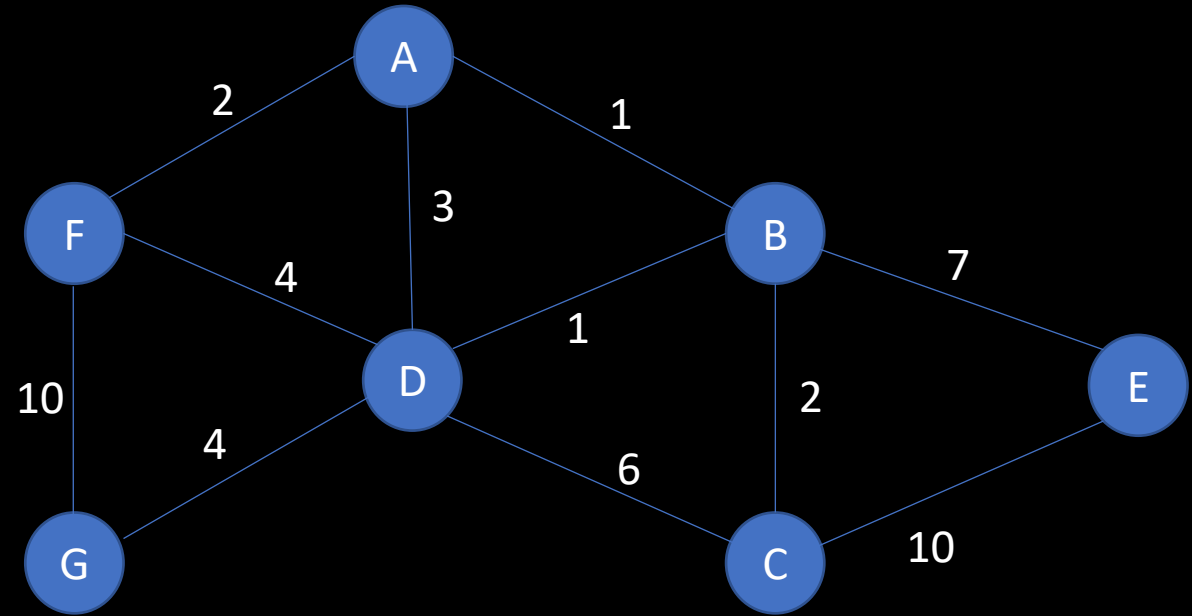
# MST using Prims starting from "I"
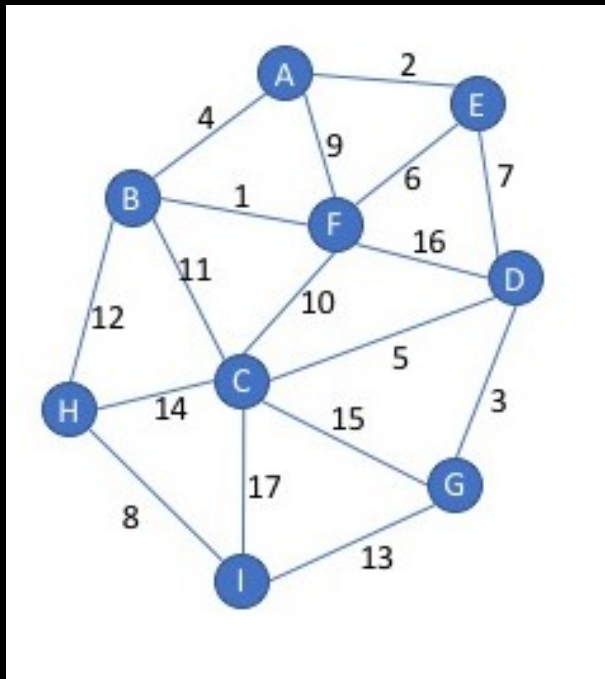
# MST using Prims starting from "I"
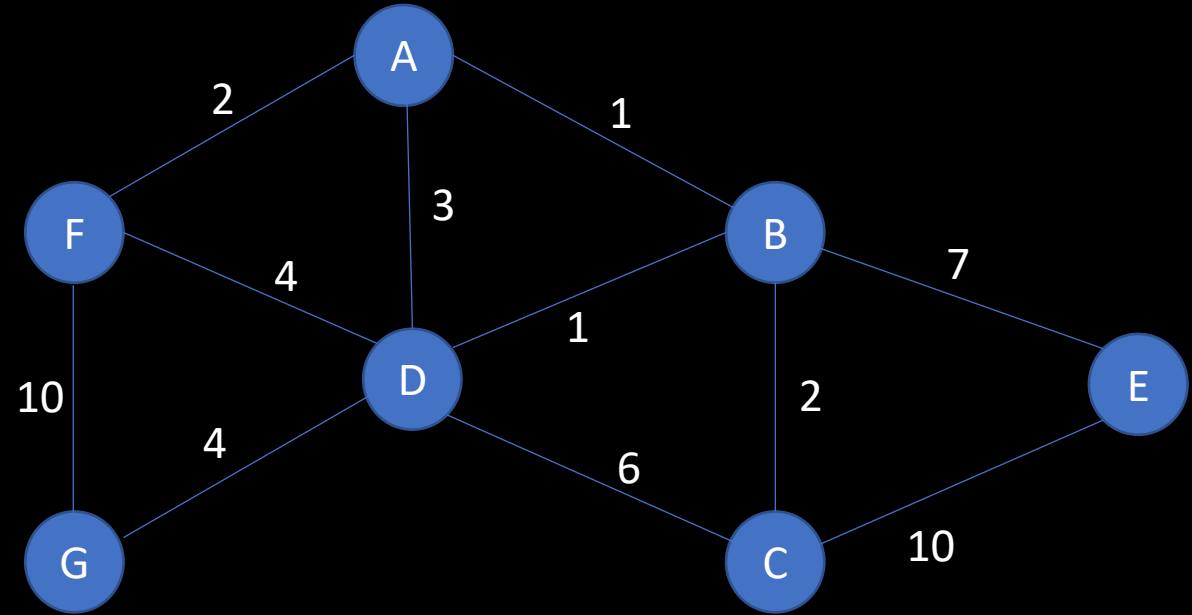


**I H B F A E D G C**
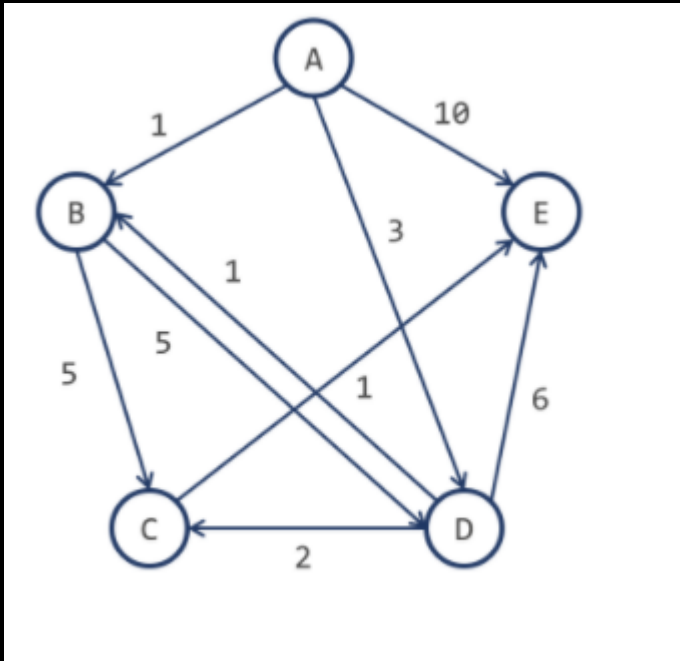
# Dijkstra with A as source

| 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 |