Data structures and algorithms

Tutorial 7 - Introduction to graph theory

Amr Keleg

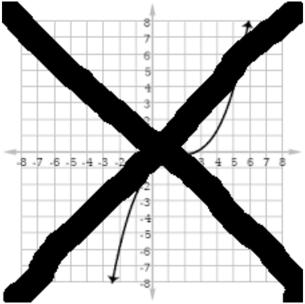
Faculty of Engineering, Ain Shams University

April 10, 2020

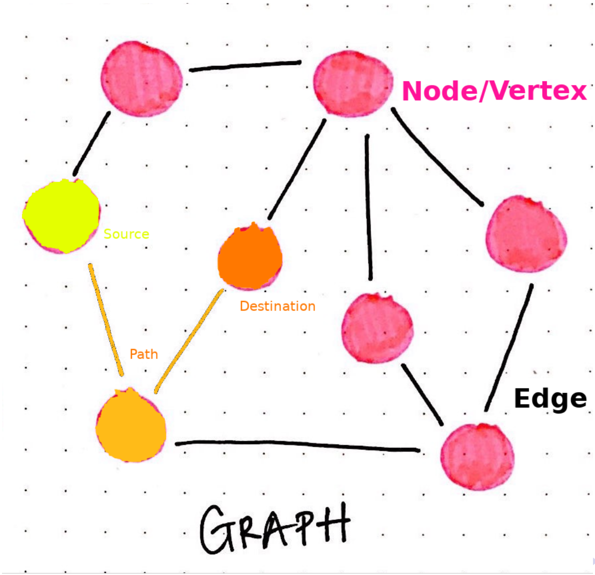
Contact: amr_mohamed@live.com

Outline

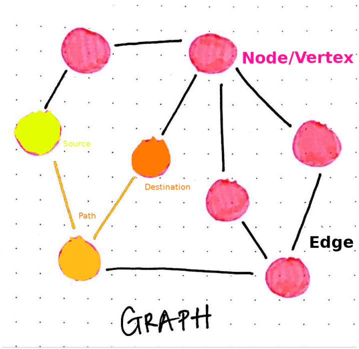
Graph? NO



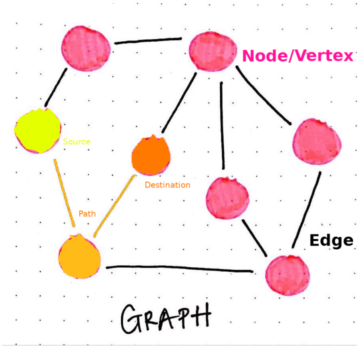
Graph? YES



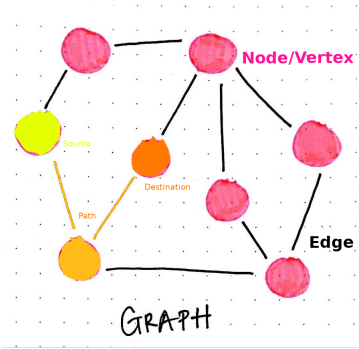
Outline



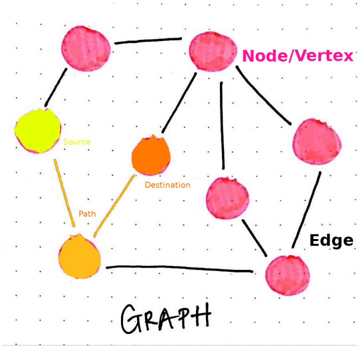
- Node



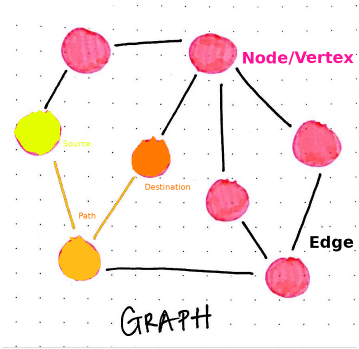
- Node
- Edge:



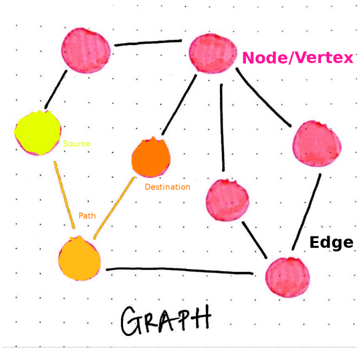
- Node
- Edge: A direct connection between two nodes



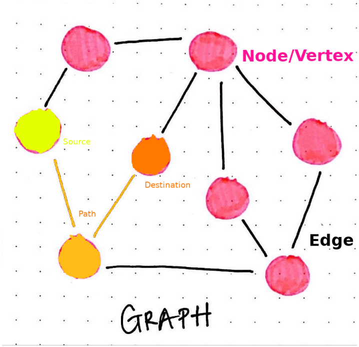
- Node
- Edge: A direct connection between two nodes
- Path:



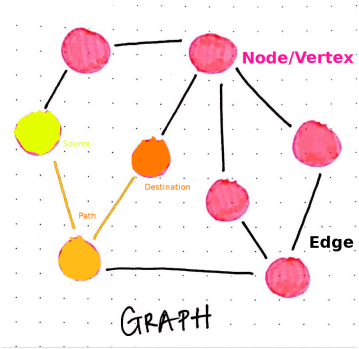
- Node
- Edge: A direct connection between two nodes
- Path: A set of edges connecting source and destination nodes



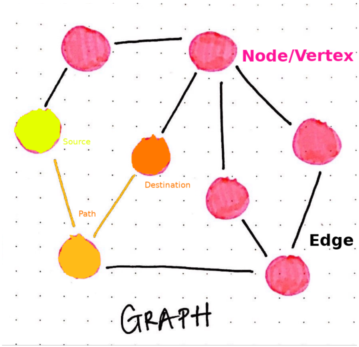
- Node
- Edge: A direct connection between two nodes
- Path: A set of edges connecting source and destination nodes
- Weight:



- Node
- Edge: A direct connection between two nodes
- Path: A set of edges connecting source and destination nodes
- Weight: A value give to each edge



- Node
- Edge: A direct connection between two nodes
- Path: A set of edges connecting source and destination nodes
- Weight: A value give to each edge
- Degree:



- Node
- Edge: A direct connection between two nodes
- Path: A set of edges connecting source and destination nodes
- Weight: A value give to each edge
- Degree: No. of edges for each node

Outline

Undirected vs Directed

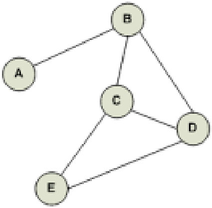


Fig 1. Undirected Graph

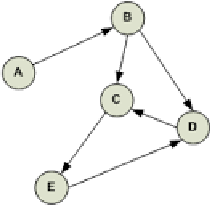
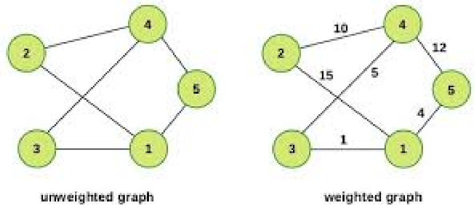


Fig 2. Directed Graph

Unweighted vs Weighted



Directed or undirected?

- Followers on Twitter

Directed or undirected?

- Followers on Twitter (Directed)

Directed or undirected?

- Followers on Twitter (Directed)
- Friends on facebook

Directed or undirected?

- Followers on Twitter (Directed)
- Friends on facebook (Undirected)

Directed or undirected?

- Followers on Twitter (Directed)
- Friends on facebook (Undirected)
- Roads of Google Maps

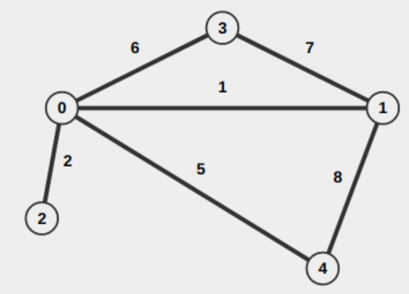
Directed or undirected?

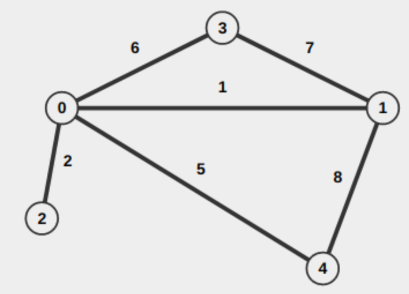
- Followers on Twitter (Directed)
- Friends on facebook (Undirected)
- Roads of Google Maps (Directed)

The two properties aren't mutually exclusive.
E.G: A graph can be:

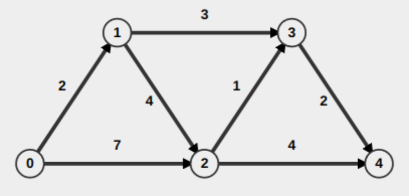
- Directed and Unweighted
- Undirected and Unweighted
- Directed and Weighted
- Undirected and Weighted

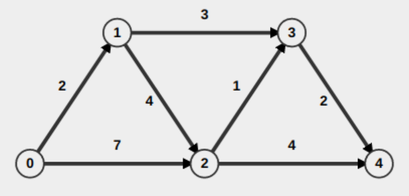
Outline


3
6
7
1
0
1
2
5
8
2
4



Node	Degree
0	4
1	3
2	1
3	2
4	2


3
1
3
2
4
1
2
7
4
0
2
4



Node	In-degree	Out-degree
0	0	2
1	1	2
2	2	2
3	2	1
4	2	0

Outline

1 Basic concepts of graph theory

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search
- Sheet 4 - Question 3
- DFS using stack instead of recursion
- Sheet 4 - Question 3 - using stack

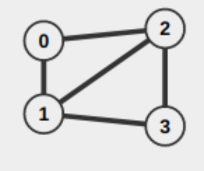
For simplicity and without loss of generality, let's assume that the graph isn't dynamic (We won't need to add or delete nodes after the construction of the graph).

Outline

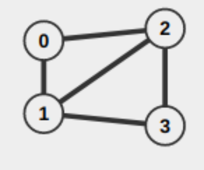
1. Basic concepts of graph theory
   - Terminology
   - Types of graphs
   - Degree in graphs
2. Adjacency Matrix
   - How to represent a graph using basic data-structures?
   - Searching algorithms in graph - DFS - Depth First Search
   - Sheet 4 - Question 3
   - DFS using stack instead of recursion
   - Sheet 4 - Question 3 - using stack

- Wikipedia definition: "In graph theory and computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph."
- For a graph of N nodes, build a 2D array (matrix) of size N*N.
- The values stored in the adjacency matrix differs according to the type of the graph.

Generate the adjacency matrix for the following graph.



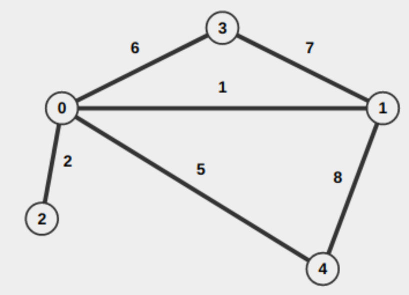
Generate the adjacency matrix for the following graph.



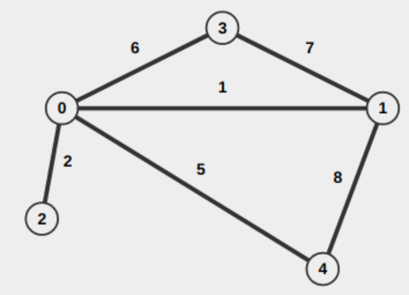
Adjacency Matrix

	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

Generate the adjacency matrix for the following graph.

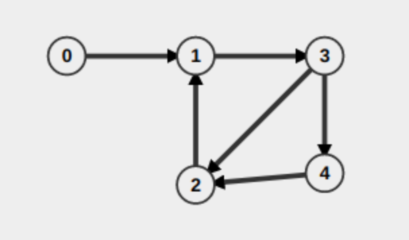


Generate the adjacency matrix for the following graph.

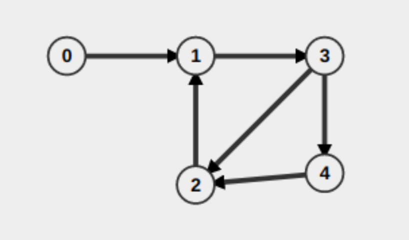


Adjacency Matrix					
	0	1	2	3	4
0	0	1	2	6	5
1	1	0	0	7	8
2	2	0	0	0	0
3	6	7	0	0	0
4	5	8	0	0	0

Generate the adjacency matrix for the following graph.



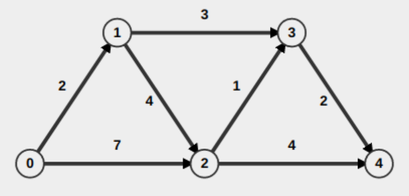
Generate the adjacency matrix for the following graph.



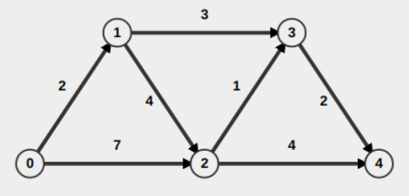
Adjacency Matrix

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

Generate the adjacency matrix for the following graph.



Generate the adjacency matrix for the following graph.



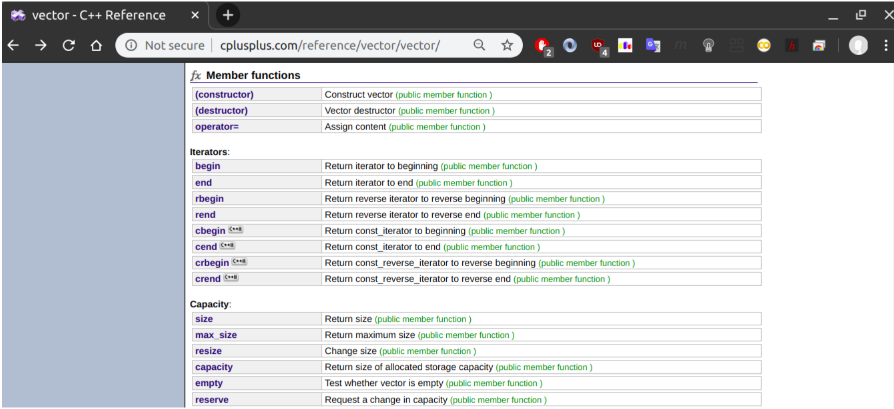
Adjacency Matrix					
	0	1	2	3	4
0	0	2	7	0	0
1	0	0	4	3	0
2	0	0	0	1	4
3	0	0	0	0	2
4	0	0	0	0	0

How to read a description of the graph and load it into an adjacency matrix?

```
// An undirected unweighted graph
3 3 // 3 nodes and 3 edges

0 1
0 2
1 2
```

```
#include<vector>

int main(){
  // An empty vector of size 0
  vector<int> v1;

  // A vector of size 10
  vector<int> v2(10);

  // A vector of size 10 and initial value 5
  vector<int> v3(10, 5);
}
```

http://www.cplusplus.com/reference/vector/vector/


vector - C++ Reference

Not secure | cplusplus.com/reference/vector/vector/

fx **Member functions**

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

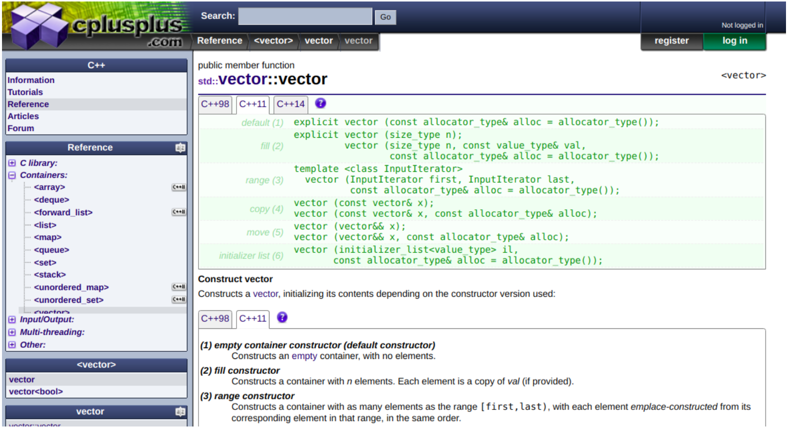
Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)

http://www.cplusplus.com/reference/vector/vector/



cplusplus.com

Search: Go

Reference <vector> vector vector

Not logged in

register log in

C++

Information
Tutorials
Reference
Articles
Forum

Reference

C library:
Containers:
<array>
<deque>
<forward_list>
<list>
<map>
<queue>
<set>
<stack>
<unordered_map>
<unordered_set>
<vector>
Input/Output:
Multi-threading:
Other:

<vector>

vector
vector<bool>

vector

public member function

std::**vector::vector**

<vector>

C++98 C++11 C++14

default (1)	`explicit vector (const allocator_type& alloc = allocator_type());`
fill (2)	`explicit vector (size_type n);` `vector (size_type n, const value_type& val, const allocator_type& alloc = allocator_type());`
range (3)	`template <class InputIterator> vector (InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type());`
copy (4)	`vector (const vector& x);` `vector (const vector& x, const allocator_type& alloc);`
move (5)	`vector (vector&& x);` `vector (vector&& x, const allocator_type& alloc);`
initializer list (6)	`vector (initializer_list<value_type> il, const allocator_type& alloc = allocator_type());`

Construct vector

Constructs a vector, initializing its contents depending on the constructor version used:

C++98 C++11

(1) empty container constructor (default constructor)
Constructs an empty container, with no elements.

(2) fill constructor
Constructs a container with *n* elements. Each element is a copy of *val* (if provided).

(3) range constructor
Constructs a container with as many elements as the range `[first,last)`, with each element *emplace-constructed* from its corresponding element in that range, in the same order.

---

http://www.cplusplus.com/reference/vector/vector/

Example

```
// constructing vectors
#include <iostream>
#include <vector>

int main ()
{
  // constructors used in the same order as described above:
  std::vector<int> first;                                // empty vector of ints
  std::vector<int> second (4,100);                       // four ints with value 100
  std::vector<int> third (second.begin(),second.end());  // iterating through second
  std::vector<int> fourth (third);                       // a copy of third

  // the iterator constructor can also be used to construct from arrays:
  int myints[] = {16,2,77,29};
  std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

  std::cout << "The contents of fifth are:";
  for (std::vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Edit & Run

Output:

```
The contents of fifth are: 16 2 77 29
```

Complexity

Constant for the *default constructor (1)*, and for the *move constructors (5)* (unless *alloc* is different from *x*'s allocator).
For all other cases, linear in the resulting container size.
Additionally, if `InputIterator` in the *range constructor (3)* is not at least of a forward iterator category (i.e., it is just an input iterator), the new capacity cannot be determined beforehand and the construction incurs in additional logarithmic complexity in size (reallocations while growing).

http://www.cplusplus.com/reference/vector/vector/

```
// Vector of strings
vector<string>vs(100, "ABC");
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> >
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> >  vi
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> >  vi(100,
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> >  vi(100,  vector<int >(10, -1) )
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> >  vi(100,  vector<int >(10, -1) );
// 100 rows and 10 columns
// vi[0][0] to vi[99][9]
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> > vi(100, vector<int>(10, -1) );
// 100 rows and 10 columns
// vi[0][0] to vi[99][9]

int * * arr; // or int ** arr;
arr = new int*[100];
```

```
// Vector of strings
vector<string>vs(100, "ABC");

// Vector of vectors
vector< vector<int> > vi(100, vector<int>(10, -1) );
// 100 rows and 10 columns
// vi[0][0] to vi[99][9]

int * * arr; // or int ** arr;
arr = new int*[100];
for(int i=0; i<100; i++)
{
  arr[i] = new int[10];
  for(int j=0;j<10;j++)
  {
    arr[i][j] = -1;
  }
}
```

How to read a description of the graph and load it into an adjacency matrix?

Note: The lecture uses an array of pointers.

```
// Undirected unweighted
int n, m;
cin>>n>>m;
```

How to read a description of the graph and load it into an adjacency matrix?
Note: The lecture uses an array of pointers.

```
// Undirected unweighted
int n, m;
cin>>n>>m;
// 1) A static array (not preferable)
int adj_matrix[1000][1000]; // 1000 is just a large no.

// 2) Using dynamic arrays
// without getting our hands dirty
vector<vector<int> > adj_matrix(n, vector<int> (n, 0));
```

How to read a description of the graph and load it into an adjacency matrix?
Note: The lecture uses an array of pointers.

```
// Undirected unweighted
int n, m;
cin>>n>>m;
// 1) A static array (not preferable)
int adj_matrix[1000][1000]; // 1000 is just a large no.

// 2) Using dynamic arrays
// without getting our hands dirty
vector<vector<int> > adj_matrix(n, vector<int> (n, 0));

int a, b;
```

How to read a description of the graph and load it into an adjacency matrix?
Note: The lecture uses an array of pointers.

```
// Undirected unweighted
int n, m;
cin>>n>>m;
// 1) A static array (not preferable)
int adj_matrix[1000][1000]; // 1000 is just a large no.

// 2) Using dynamic arrays
// without getting our hands dirty
vector<vector<int> > adj_matrix(n, vector<int> (n, 0));

int a, b;
for (int i=0; i<m ;i++){
  cin>>a>>b;
  adj_matrix[a][b] = 1;
  adj_matrix[b][a] = 1;
}
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Directed unweighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Directed unweighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));

int a, b;
for (int i=0; i<m ;i++){
  cin>>a>>b;
  adj_matrix[a][b] = 1;
}
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Undirected Weighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Undirected Weighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));

int a, b, c;
for (int i=0; i<m ;i++){
  cin>>a>>b>>c;
  adj_matrix[a][b] = c;
  adj_matrix[b][a] = c;
}
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Directed Weighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));
```

How to read a description of the graph and load it into an adjacency matrix?

```
// Directed Weighted
int n, m;
cin>>n>>m;

vector<vector<int> > adj_matrix(n, vector<int> (n, 0));

int a, b, c;
for (int i=0; i<m ;i++){
  cin>>a>>b>>c;
  adj_matrix[a][b] = c;
}
```

Outline

One of the questions that DFS can solve is:
"Are nodes src and des connected?"

```
bool dfs(int node, int des,
  const vector<vector<int> > & adj_matrix);
// Return true if there is a path between node and des
```

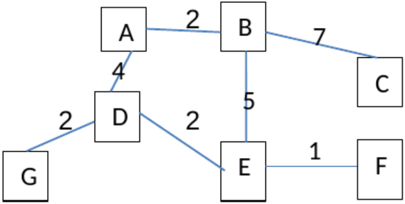
```
bool dfs(int node, int des,
  const vector<vector<int> > & adj_matrix){

  if (node ==des){
    return true;
  }
  for (int next_node=0;
      next_node<adj_matrix[node].size();
      next_node++){
    if (adj_matrix[node][next_node] == 0)
      continue;
    if (dfs(next_node, des, adj_matrix))
      return true;
  }
  return false;
}
```

```
bool dfs(int node, int des,
  const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  if (node ==des){
    return true;
  }
  visited[node] = true;
  for (int next_node=0;
      next_node<adj_matrix[node].size();
      next_node++){
    if (adj_matrix[node][next_node] == 0
      || visited[next_node])
      continue;
    if (dfs(next_node, des, adj_matrix, visited))
      return true;
  }
  return false;
}
```

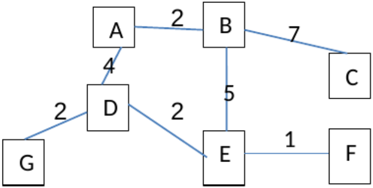
```
bool dfs(int node, int des,
  const vector<vector<int> > & adj_matrix,
  vector<bool> & visited);

int main(){
  int n, m;
  cin>>n>>m;
  vector<vector<int> > adj_matrix(n, vector<int> (n, 0))
  ... // Build the graph
  int src, des;
  cin>>src>>des;
  vector<bool>visited(n, false);
  if(dfs(src, des, adj_matrix, visited))
    cout<<"Connected";
  else
    cout<<"Not connected";
}
```

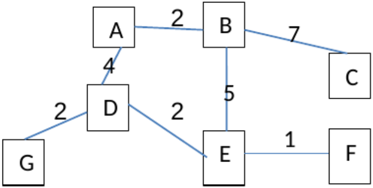
Outline

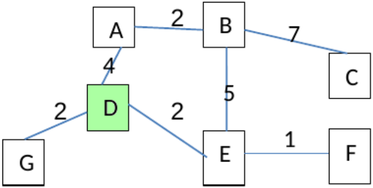
Q3. Apply both depth first and breadth first algorithms starting from vertex D

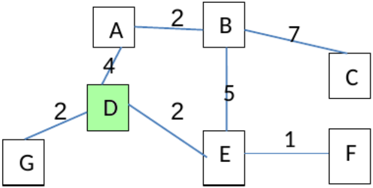


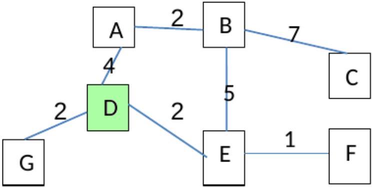
Note: The paths will depend on the way nodes are pushed to the queue/ stack and whether you mark the node on push/ on pop. State your assumptions clearly when you answer such questions unless the rules are stated in the question's statement.

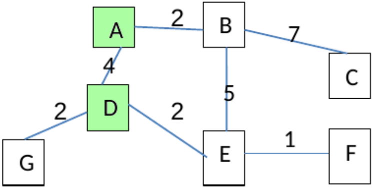

A
B
C
D
E
F
G
2
7
4
5
2
2
1

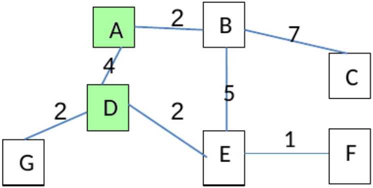

A
2
B
7
C
4
5
D
2
2
G
E
1
F

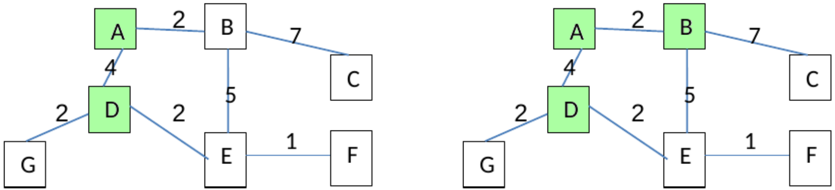

A
2
B
7
C
4
5
D
2
2
G
E
1
F

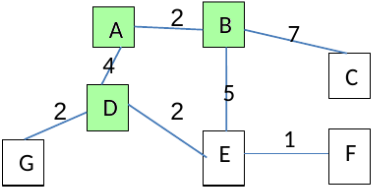

A
B
C
D
E
F
G
2
7
4
5
2
2
1

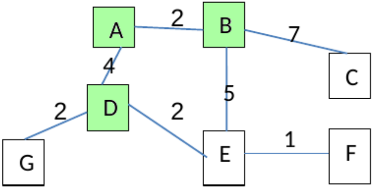

A
B
C
D
E
F
G
2
7
4
5
2
2
1

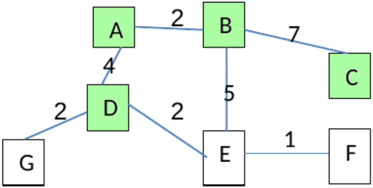

A
B
C
D
E
F
G
2
7
4
5
2
2
1

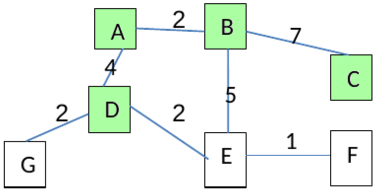

A
B
C
D
E
F
G
2
7
4
5
2
2
1

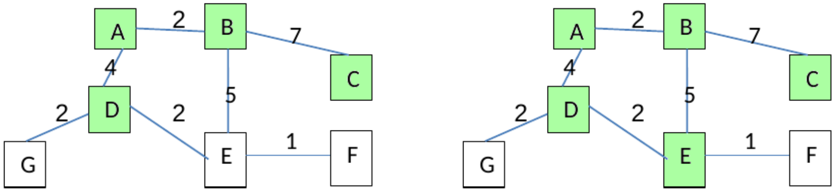

A
B
C
D
E
F
G
2
7
4
5
2
2
1
A
B
C
D
E
F
G
2
7
4
5
2
2
1

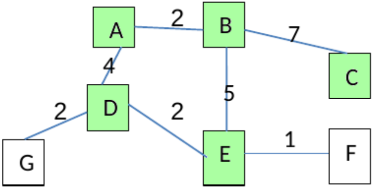

A
2
B
7
C
4
5
D
2
2
G
E
1
F

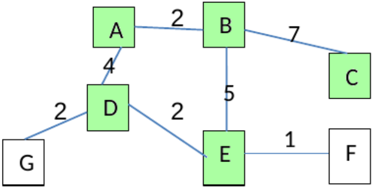

A
B
C
D
E
F
G
2
7
4
5
2
2
1

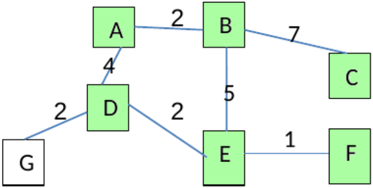

A
B
C
D
E
F
G
2
7
4
5
2
2
1

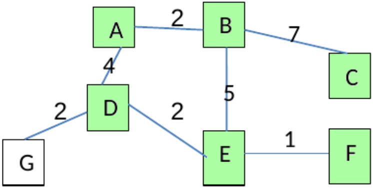

A
B
C
D
E
F
G
2
7
4
5
2
2
1

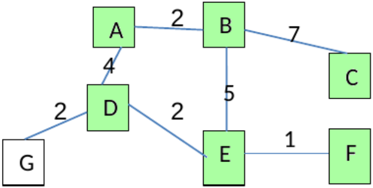

A
B
C
D
E
F
G
2
7
4
5
2
2
1
A
B
C
D
E
F
G
2
7
4
5
2
2
1

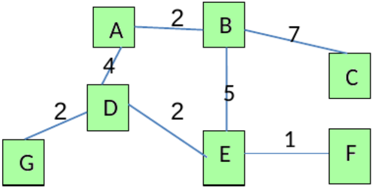

A
2
B
7
C
4
5
D
2
2
G
E
1
F


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


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


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


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


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

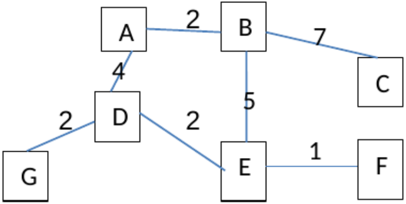
DFS: D - A - B - C - E - F - G

Outline

```
bool dfs(int node, int des, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    visited[cur_node] = 1;

    if (cur_node==des)
      return true;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
  return false;
}
```

Outline

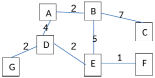
Q3. Apply both depth first and breadth first algorithms starting from vertex D



Note: The paths will depend on the way nodes are pushed to the queue/ stack and whether you mark the node on push/ on pop. State your assumptions clearly when you answer such questions unless the rules are stated in the question's statement.

```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

[X] stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

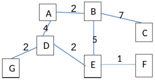
Explored Nodes: -



```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
[X]  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

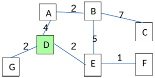
Explored Nodes: -

D



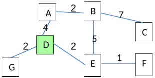
```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
[X]   visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

Explored Nodes:
D



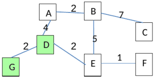
```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
[X]   }
  }
}
```

Explored Nodes:
D



```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;
[X]
    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

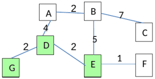
Explored Nodes:
D - G



```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;
[X]
    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

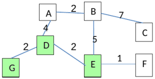
Explored Nodes:

D - G - E



```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
[X]   }
  }
}
```

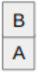
Explored Nodes:
D - G - E

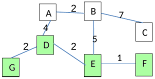


```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;
[X]
    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

Explored Nodes:

D - G - E - F



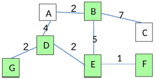


```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;
[X]
    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
    }
  }
}
```

Explored Nodes:

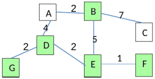
D - G - E - F - B

A



```
void dfs(int node, const vector<vector<int> > & adj_matrix,
  vector<bool> & visited){

  stack<int> nodes_stack;
  nodes_stack.push(node);

  while(!nodes_stack.empty()){

    int cur_node = nodes_stack.top();
    nodes_stack.pop();
    cout<<cur_node<<" ";
    visited[cur_node] = 1;

    for (int next_node=0; next_node<adj_matrix[cur_node].size(); next_node++){

      if (adj_matrix[cur_node][next_node] == 0 || visited[next_node])
        continue;

      nodes_stack.push(next_node);
[X]   }
  }
}
```

Explored Nodes:
D - G - E - F

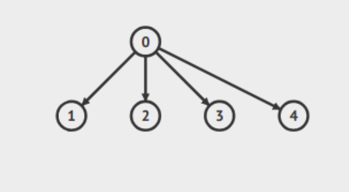


- Visit/Mark on pop (as we have done)
  OR
- Visit/Mark on push (as long as node is pushed to the stack, mark it as visited)

What is the complexity of the DFS on adjacency matrix?
The graph has N nodes and E edges.

What is the complexity of the DFS on adjacency matrix?
The graph has N nodes and E edges.

$$O(N^2) \tag{1}$$



For a simple graph like this one, DFS using adjacency matrix will make 25(5*5) iterations/checks.

To be continued!

Feedback form: `https://forms.gle/hTuHcEMs87vLNrUL9`