Data structures and algorithms

Tutorial 7 - Introduction to graph theory

Amr Keleg

Faculty of Engineering, Ain Shams University

April 10, 2019

Contact: amr_mohamed@live.com

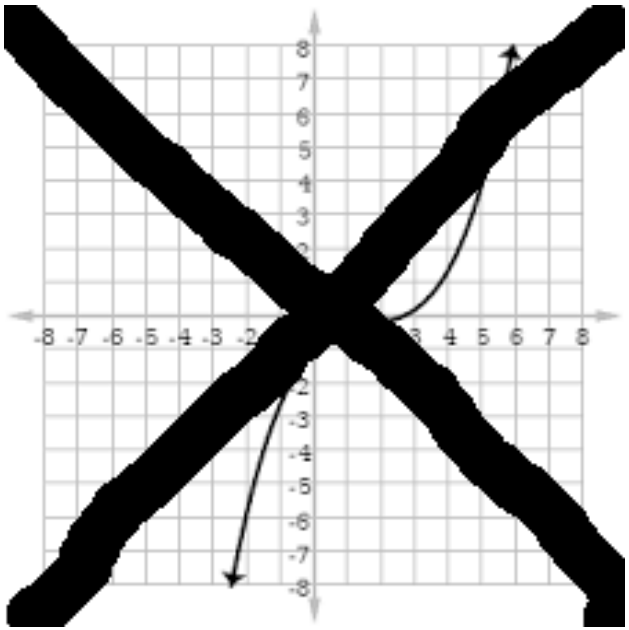
Outline

1 Basic concepts of graph theory
- Terminology
- Types of graphs

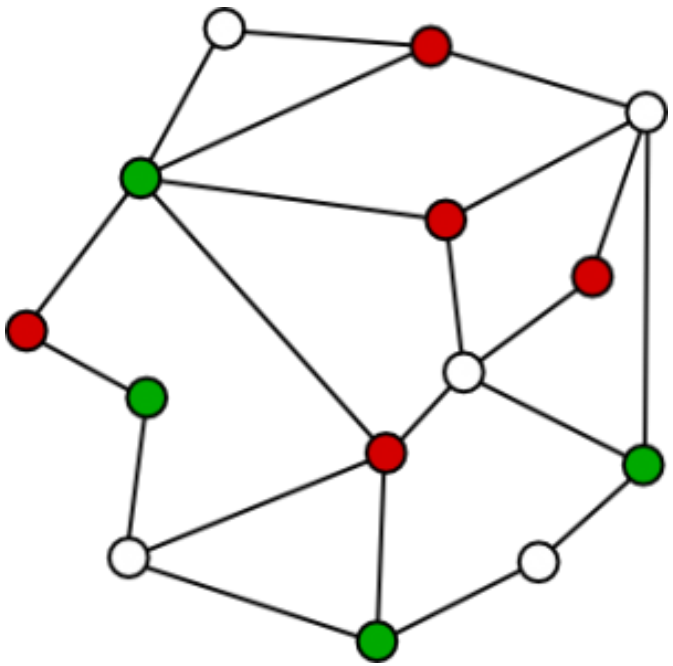
2 Adjacency Matrix

3 Adjacency List

Graph? NO



Graph? YES



Outline

1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

- 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

1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

Undirected vs Directed

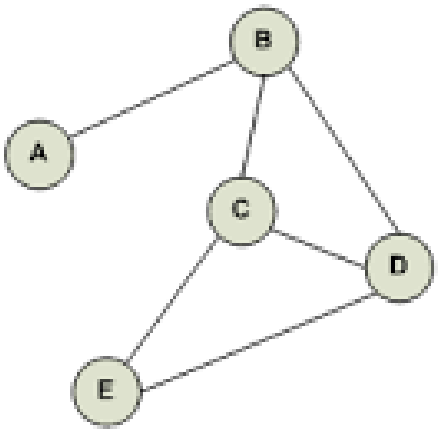


Fig 1. Undirected Graph

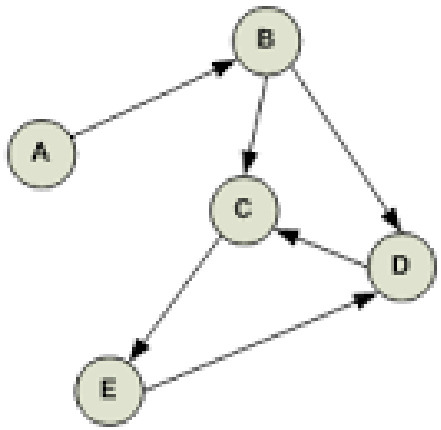
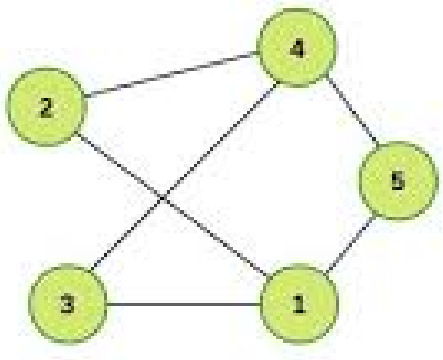
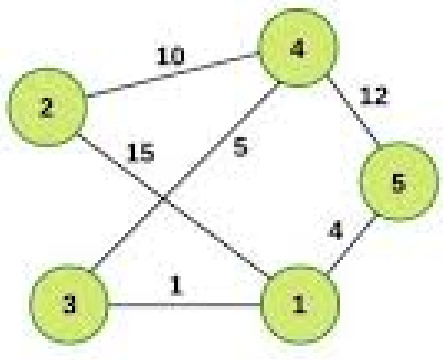


Fig 2. Directed Graph

UnWeighted vs Weighted



unweighted graph



weighted graph

Directed or undirected?

- Followers on Twitter
- Friends on facebook
- Roads of Google Maps

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

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

3 Adjacency List

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

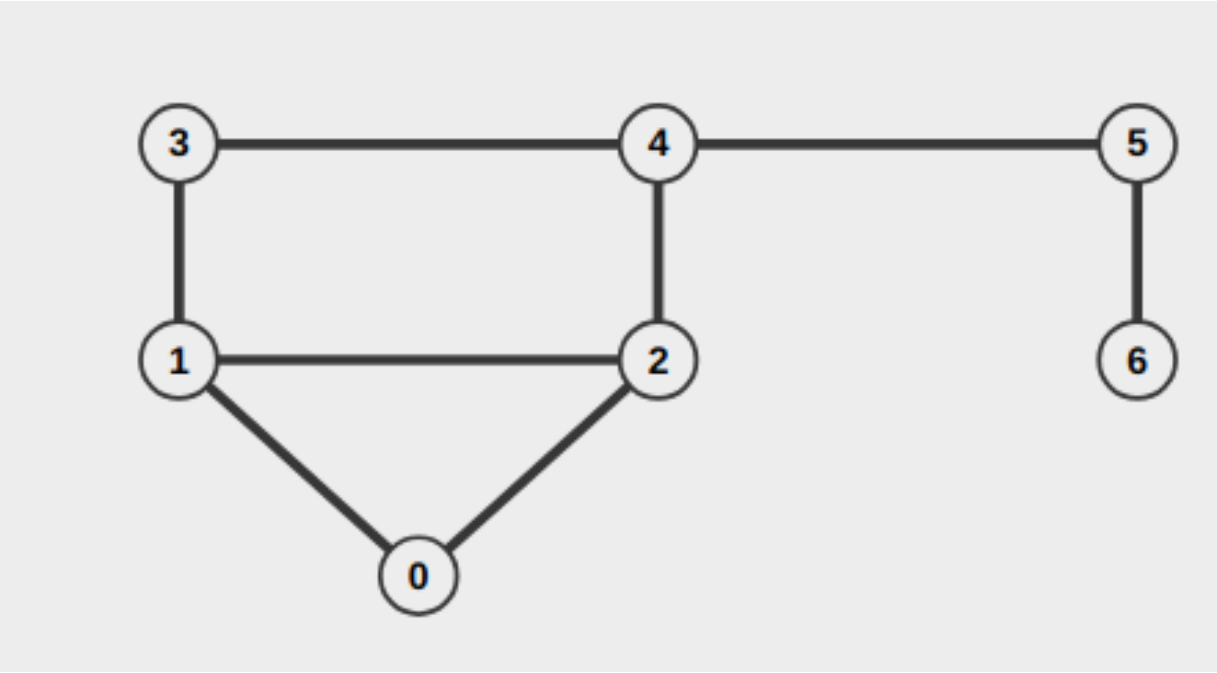
1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

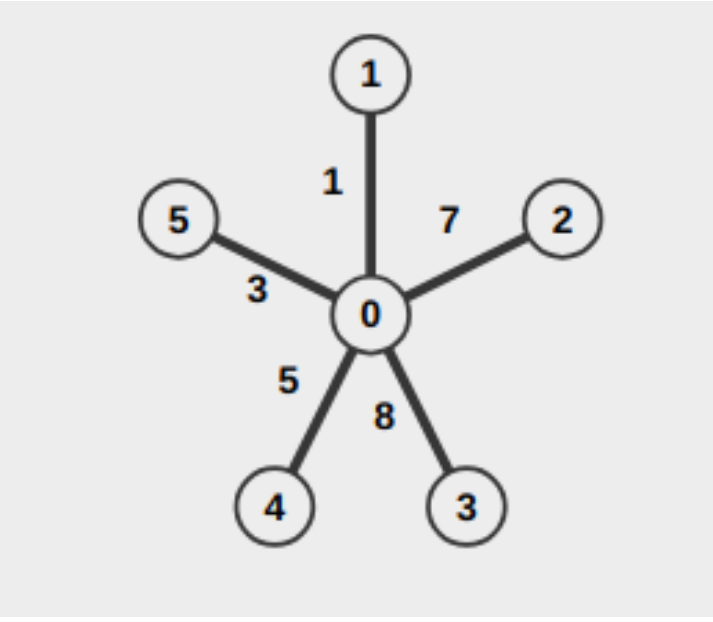
3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

- 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.

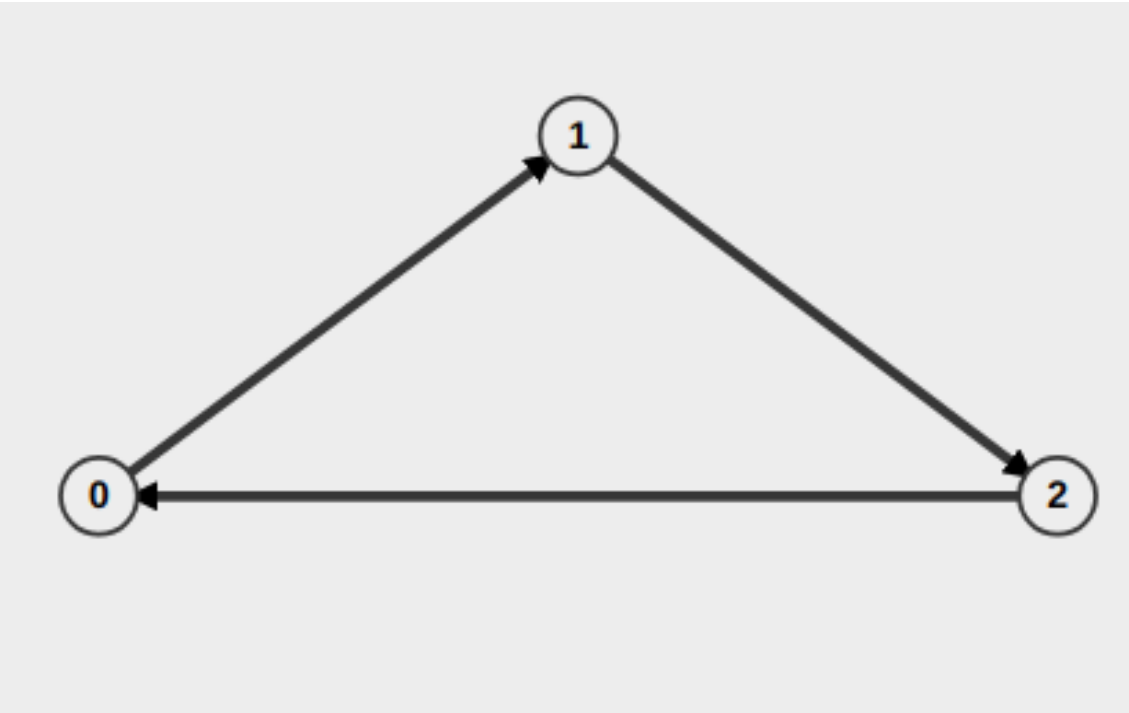
Generate the adjaceny matrix for the following graph.



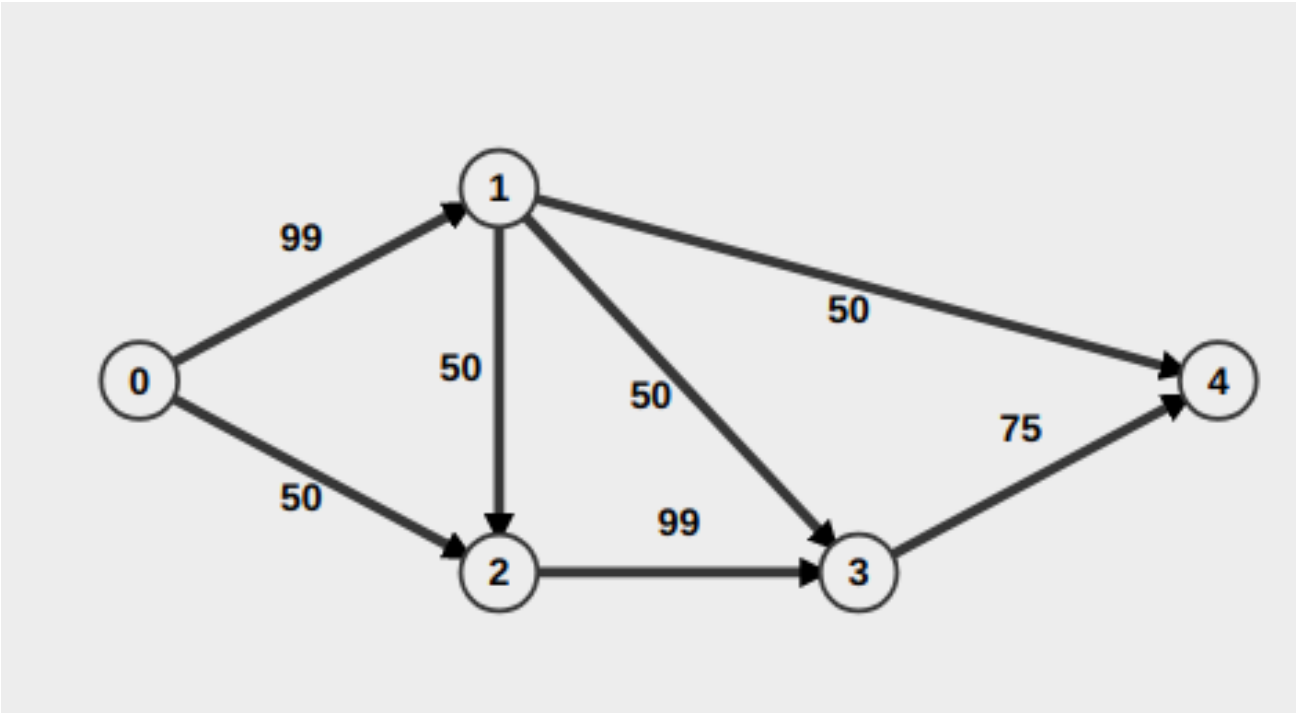
Generate the adjaceny matrix for the following graph.



Generate the adjaceny matrix for the following graph.



Generate the adjaceny matrix for the following graph.



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

```
// An undirected unweighted graph

3 3
0 1
0 2
1 2
```

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

Outline

1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

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 (next_node == node
      || 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 (next_node == node
      || 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";
}
```

```
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[node].size();
     next_node++){
      if (next_node == node
        || adj_matrix[node][next_node] == 0
        || visited[next_node])
        continue;
      nodes_stack.push(next_node);
    }
  }
  return false;
```

Outline

1 Basic concepts of graph theory

2 Adjacency Matrix

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

Outline

1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

- Adjacency matrix isn't suitable for sparse graphs.
- For each node, store only information about its adjacent nodes.

```
// For an Undirected Unweighted graph
int n, m;
cin>>n>>m;
vector<vector<int> > adj_list(n, vector<int> ());
int a, b;
for (int i=0; i<m ;i++){
   cin>>a>>b;
   adj_list[a].push_back(b);
   adj_list[b].push_back(a);
}
```

Outline

1 Basic concepts of graph theory
- Terminology
- Types of graphs

2 Adjacency Matrix
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

3 Adjacency List
- How to represent a graph using basic data-structures?
- Searching algorithms in graph - DFS - Depth First Search

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

  if (node ==des) return true;

  visited[node] = true;
  for (int next_node_index=0;
      next_node_index<adj_list[node].size();
      next_node_index++){

    int next_node = adj_list[next_node_index];
    if (visited[next_node])
      continue;
    if (dfs(next_node, des, adj_list, visited))
      return true;
  }
  return false;
}
```

Feedback form: `https://forms.gle/JG9JY7a6ZrsipWDB9`