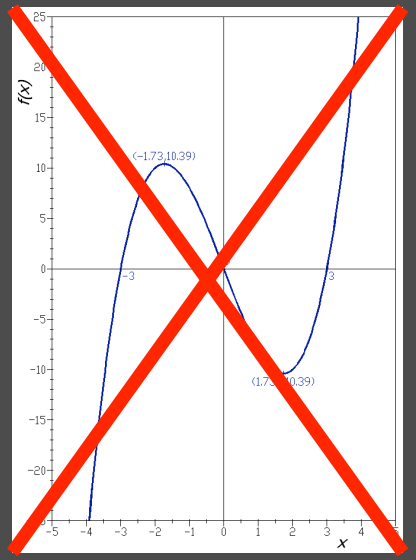
Advanced Computer Contest Preparation

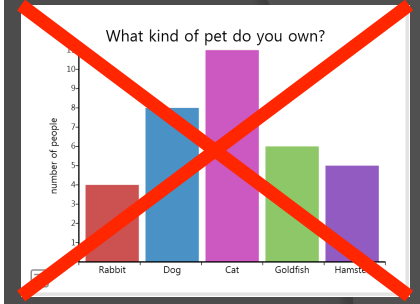
Lecture 5

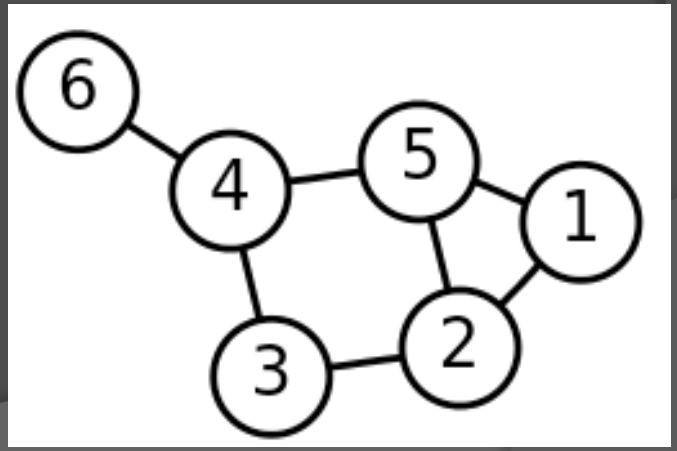
INTRODUCTION TO GRAPH THEORY

What is a Graph?

- Not a graph of a function!
- Not a graph in statistics!
- A collection of:
  - Vertices/nodes
  - Edges that connect two vertices, representing some relationship
- In simpler terms:
  - Circles connected by lines

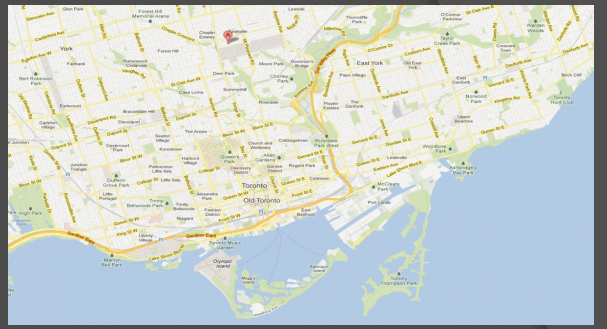


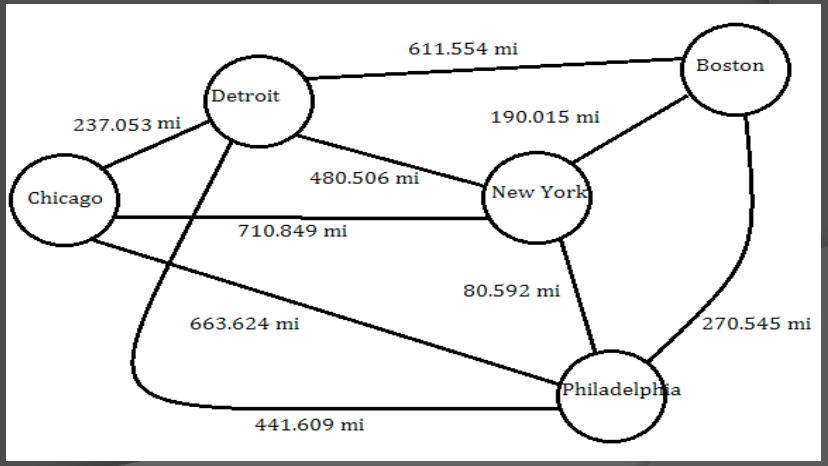




Examples of Graphs in Real Life

- Cities connected by highways
- Google Maps
- Travelling Salesman Problem
- River Systems
- Facebook Friend System
- Wikipedia Links



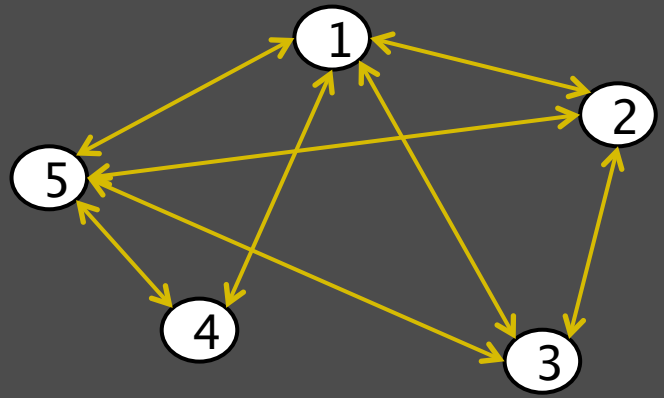


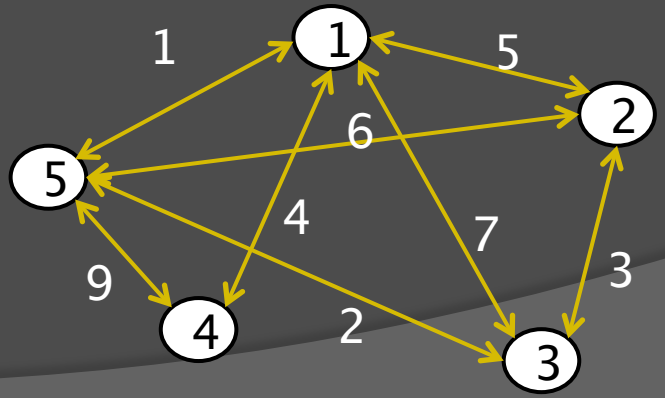
Graph Terminology

- Adjacent
  - Two vertices are adjacent if they are connected by an edge
- Neighboring nodes/Neighbors
  - Nodes that are adjacent
- Degree
  - Degree of vertex is the number of vertices adjacent to it

Types of Graphs

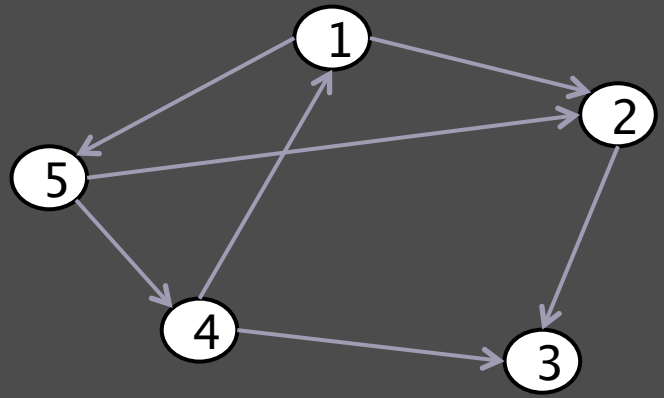
- Unweighted
  - No or uniform weight on nodes and edges
- Weighted
  - Nodes and/or edges have different weights or costs

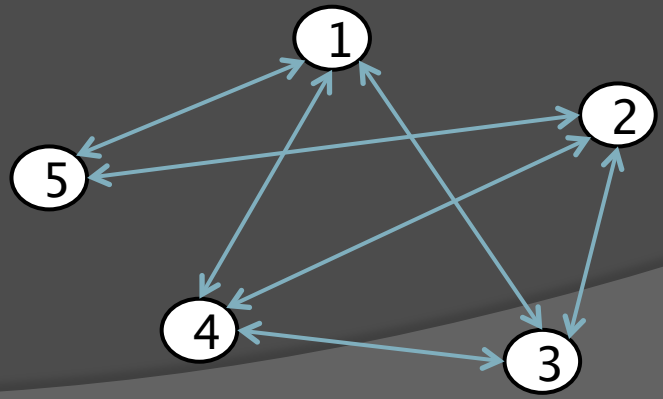




Types of Graphs

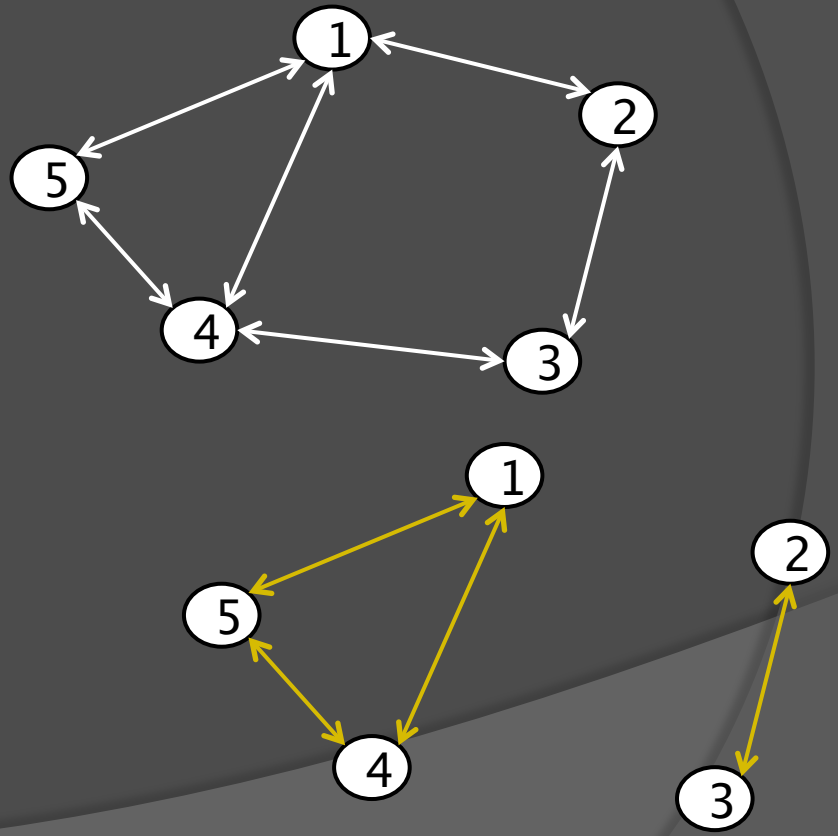
- Directed
  - Each edge can only be traversed in one direction
- Undirected
  - Edges can be traversed in both directions





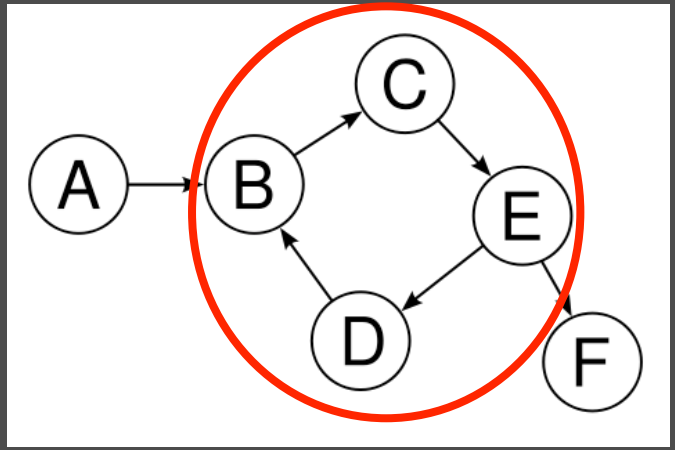
Types of Graphs

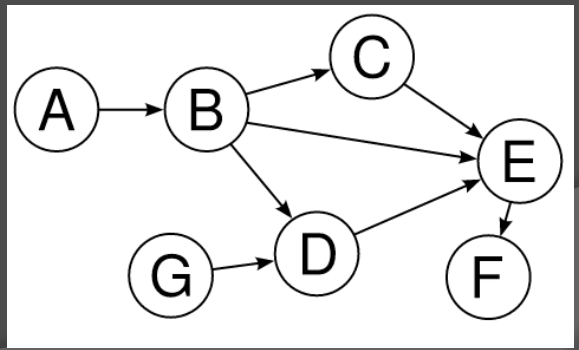
- Connected
  - Each node can be reached from any other node
- Disconnected
  - Might not be possible to reach a node from any other node
  - Assume that a graph is disconnected unless stated otherwise



Types of Graphs

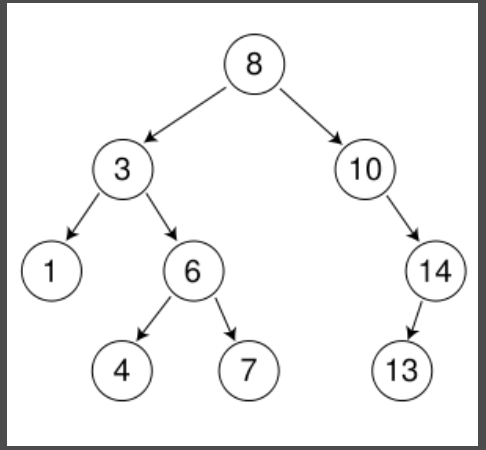
- The following graphs normally only apply for directed graphs:
- Cyclic
  - A graph with at least one path that begins and ends on the same node
  - This path is called a “cycle”
- Acyclic
  - A graph with no cycles
  - DAG = Directed Acyclic Graph

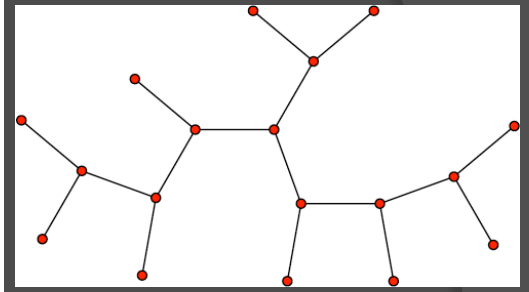


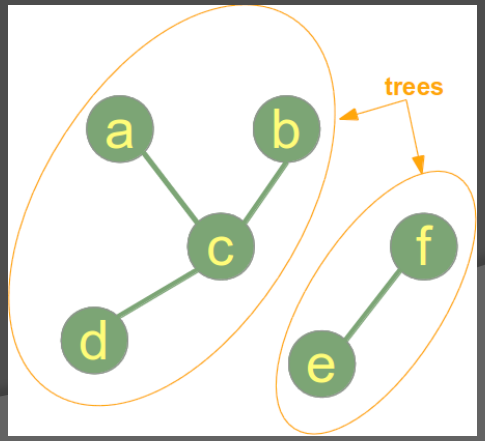


Types of Graphs

- Tree
  - Acyclic graph
  - Connected with at least respect to the root
    - There exists a path from the root to any node
  - Has $V-1$ edges
- Forest
  - An acyclic, disconnected graph
  - A collection of trees
- Difference from DAGs: If there is a path from u to v, there is only one path

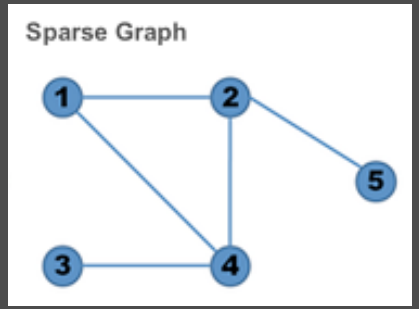


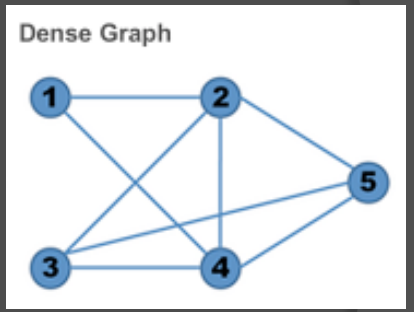


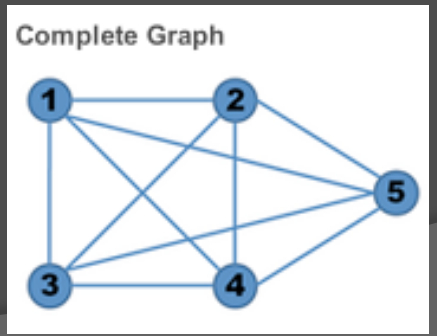


Types of Graphs

- Sparse
  - Low ratio of edges to nodes
- Dense
  - High ratio of edges to nodes
- Distinction between sparse/dense is vague and depends on context
- Complete
  - Each node has an edge to every other node
  - $O(V^2)$ edges

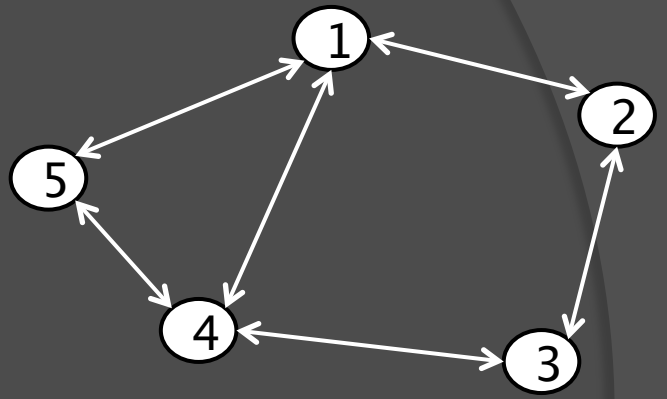






Representation – Edge List

- Use a **`vector`** of **`pairs`**
- One **`pair`** represents one edge
- Numbers in **`pair`** are the nodes that the edge connects
- Not common to use; common as input



	V1	V2
Edge 1	1	2
Edge 2	1	4
Edge 3	1	5
Edge 4	2	3
Edge 5	3	4
Edge 6	4	5

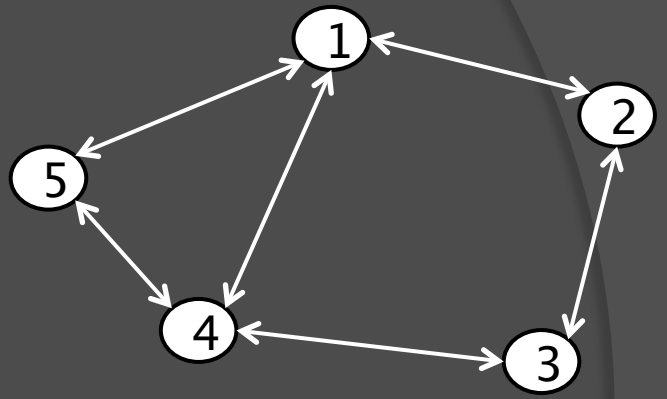
Memory	*O(2*E)*
Adjacency Check	*O(E)*
Iterate Adjacent Vertices	*O(E)*
Add edge	*O(1)*
Delete Edge	*O(E)*

Edge List

```
#include <vector>
vector<pair<int,int> > edges; //edge list
int E;
int main(){
        cin >> E;
        for(int i = 0,n1,n2; i < E; i++){
                cin >> n1 >> n2;
                edges.push_back(make_pair(n1,n2));
        }
        return 0;
}
```

Representation – Adjacency Matrix

- Use a N by N array (`bool`)
- `true/1` if an edge exists, `false/0` otherwise
- Value at row u and column v represents the one-way edge from u to v
- Somewhat common usage, uncommon as input



Memory	$O(N^2)$
Adjacency Check	$O(1)$
Iterate Adjacent Vertices	$O(N)$
Add edge	$O(1)$
Delete Edge	$O(1)$

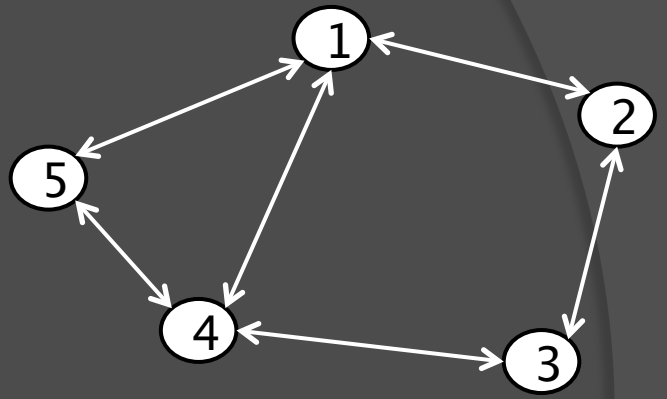
V	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	0
3	0	1	0	1	0
4	1	0	1	0	1
5	1	0	0	1	0

Adjacency Matrix

```
const int MAX_N = 10; //max number of nodes
bool graph[MAX_N][MAX_N]; //adjacency matrix
int E; //number of edges
int main(){
        cin >> E;
        for(int i = 0,n1,n2; i < E; i++){
                cin >> n1 >> n2;
                graph[n1][n2] = true;
                //do not use following line if graph is directed
                graph[n2][n1] = true;
        }
        return 0;
}
```

Representation – Adjacency List

- Every node has a **`vector`** associated with it
- These **`vectors`** store all neighbors of the node it is associated with
- Common usage, uncommon as input



Memory	*O(2*E)*
Adjacency Check	*O(Size of vector)*
Iterate Adjacent vertices	*O(Size of vector)*
Add edge	*O(1)*
Delete Edge	*O(Size of vector)*

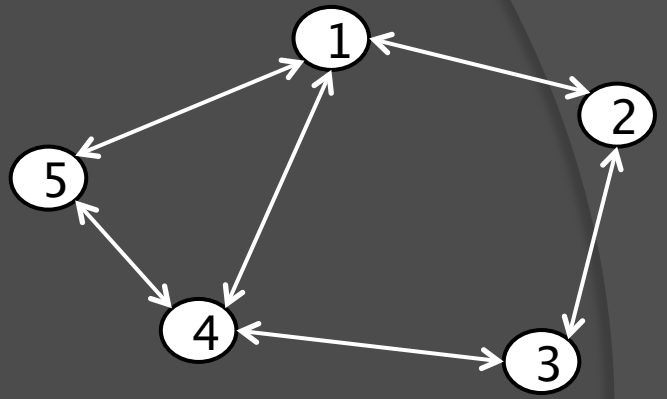
Vertex	Adjacent Vertices (vector)
1	2, 4, 5
2	1, 3
3	2, 4
4	1 ,3, 5
5	1, 4

Adjacency List

```
#include <vector>
const int MAX_N = 10; //max number of nodes
vector<int> graph[MAX_N]; //adjacency list
int E; //number of edges
int main(){
	cin >> E;
	for(int i = 0,n1,n2; i < E; i++){
		cin >> n1 >> n2;
		graph[n1].push_back(n2);
		//do not include below if graph is directed
		graph[n2].push_back(n1);
	}
	return 0;
}
```

Representation – Adjacency List (Set)

- Every node has a **`set`** associated with it
- These **`sets`** store all neighbors of the node it is associated with
- Common usage, uncommon as input
- Slower than **`vector`** version in some operations (use **`set`** only if necessary)



Memory	*O(2*E)*
Adjacency Check	*O(log (size of set))*
Iterate Adjacent vertices	*O(Size of set)*
Add edge	*O(log (size of set))*
Delete Edge	*O(log (size of set))*

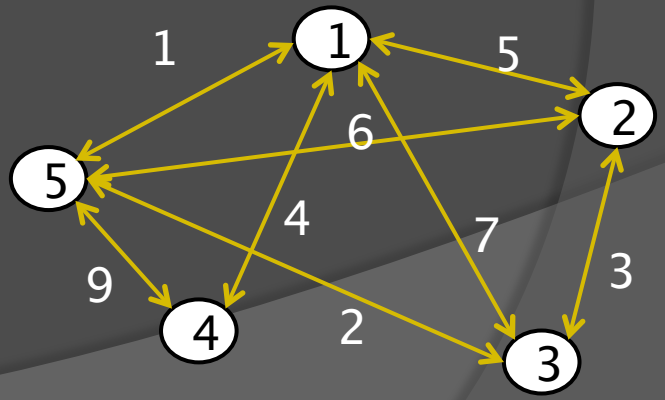
Vertex	Adjacent Vertices (list)
1	2, 4, 5
2	1, 3
3	2, 4
4	1 ,3, 5
5	1, 4

Adjacency List

```
#include <set>
const int MAX_N = 10; //max number of nodes
set<int> graph[MAX_N]; //adjacency list
int E; //number of edges
int main(){
	cin >> E;
	for(int i = 0,n1,n2; i < E; i++){
		cin >> n1 >> n2;
		graph[n1].insert(n2);
		//do not include below if graph is directed
		graph[n2].insert(n1);
	}
	return 0;
}
```

Cost

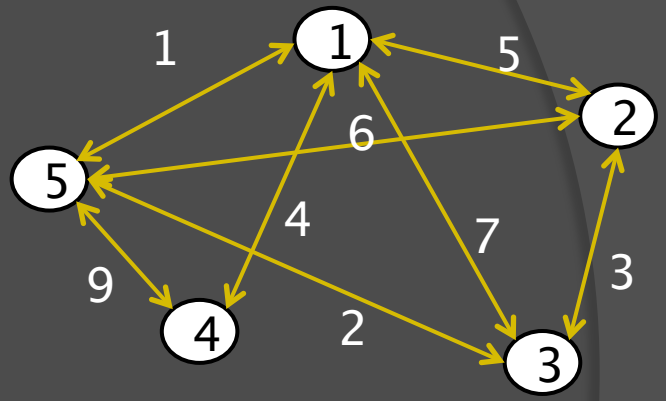
- Sometimes, edges and vertices have additional information (weighted graph)
- Example: Distance between nodes
- Must include in graph representation



Edge List With Cost

- Use a `vector` of `struct` containing variables for two nodes and any costs

	V1	V2	Cost
Edge 1	1	2	5
Edge 2	1	3	7
Edge 3	1	4	4
Edge 4	1	5	1
Edge 5	2	3	3
Edge 6	2	5	6
Edge 7	3	5	2
Edge 8	4	5	9

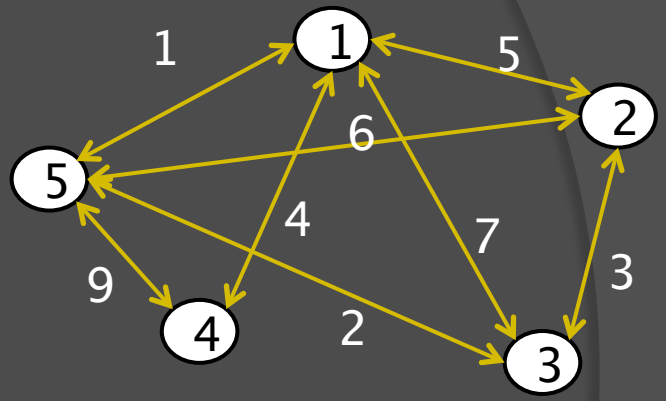


Edge List (int cost)

```
#include <vector>
struct Edge{int node1, node2, cost;};
vector<Edge> edges; //edge list
int E;
int main(){
        cin >> E;
        for(int i = 0,n1,n2,cost; i < E; i++){
                cin >> n1 >> n2 >> cost;
                edges.push_back((Edge){n1,n2,cost});
        }
        return 0;
}
```

Adjacency Matrix With Cost

- Use an N by N array of the data you are storing
- Can be primitive data type (`int`), `struct`, etc.
- 0 or infinity can represent no edge
  - Usage depends on question
  - Be sure to check if edge exists when operating on graph



V	1	2	3	4	5
1	0	5	7	4	1
2	5	0	3	0	6
3	7	3	0	0	2
4	4	0	0	0	9
5	1	6	2	9	0

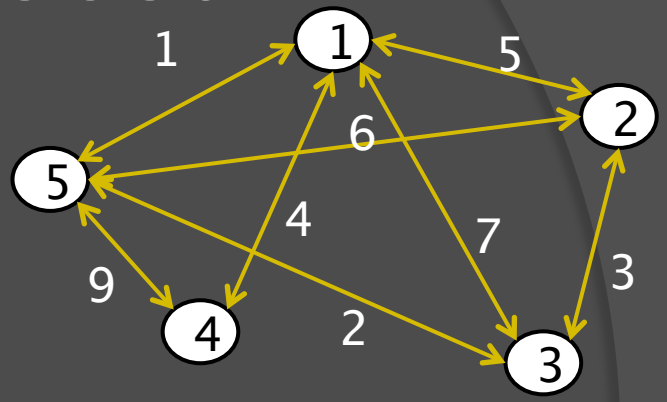
Adjacency Matrix (int cost)

```
const int MAX_N = 10; //max number of nodes
int graph[MAX_N][MAX_N]; //adjacency matrix
int E;

int main(){
        cin >> E;
        for(int i = 0,n1,n2,cost; i < E; i++){
                cin >> n1 >> n2 >> cost;
                graph[n1][n2] = cost;
                //do not include below if graph is directed
                graph[n2][n1] = cost;
        }
        return 0;
}
```

Adjacency List With Cost

- Every node has an associated **`vector`**
- Vector stores **`structs`** or **`pairs`** containing adjacent node and other cost information
- Do not have to worry about non-existing edges
- In the example, left number is node, right number is cost



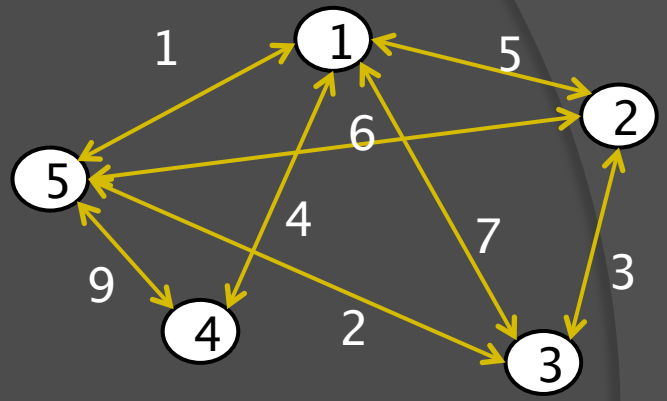
Vertex	Adjacent Vertices (Vector)
1	(2,5),(3,7),(4,4),(5,1)
2	(1,5),(3,3),(5,6)
3	(1,7),(2,3),(5,2)
4	(1,4),(5,9)
5	(1,1),(2,6),(3,2),(4,9)

Adjacency List (int cost)

```
#include <vector>
struct Edge{int other_node, cost;}; //can also be class
//could use pair<int,int> instead of Node
const int MAX_N = 10; //max number of nodes
vector<Edge> graph[MAX_N]; //adjacency list
int E;
int main(){
        cin >> E;
        for(int i = 0,n1,n2,cost; i < E; i++){
                cin >> n1 >> n2 >> cost;
                graph[n1].push_back((Edge){n2,cost});
                //do not include below if graph is directed
                graph[n2].push_back((Edge){n1,cost});
        }
        return 0;
}
```

Adjacency List With Cost (`map`)

- Every node has an associated **`map`**
- Key is the node number
- Value is the cost information
  - Can be **`int`**, **`struct`**, etc.



Vertex	Adjacent Vertices (Map)
1	(2,5),(3,7),(4,4),(5,1)
2	(1,5),(3,3),(5,6)
3	(1,7),(2,3),(5,2)
4	(1,4),(5,9)
5	(1,1),(2,6),(3,2),(4,9)

Adjacency List (int cost)

```
#include <map>
const int MAX_N = 10; //max number of nodes
map<int> graph[MAX_N]; //adjacency list
int E;
int main(){
        cin >> E;
        for(int i = 0,n1,n2,cost; i < E; i++){
                cin >> n1 >> n2 >> cost;
                graph[n1][n2] = cost;
                graph[n2][n1] = cost;
        }
        return 0;
}
```

THANK YOU!