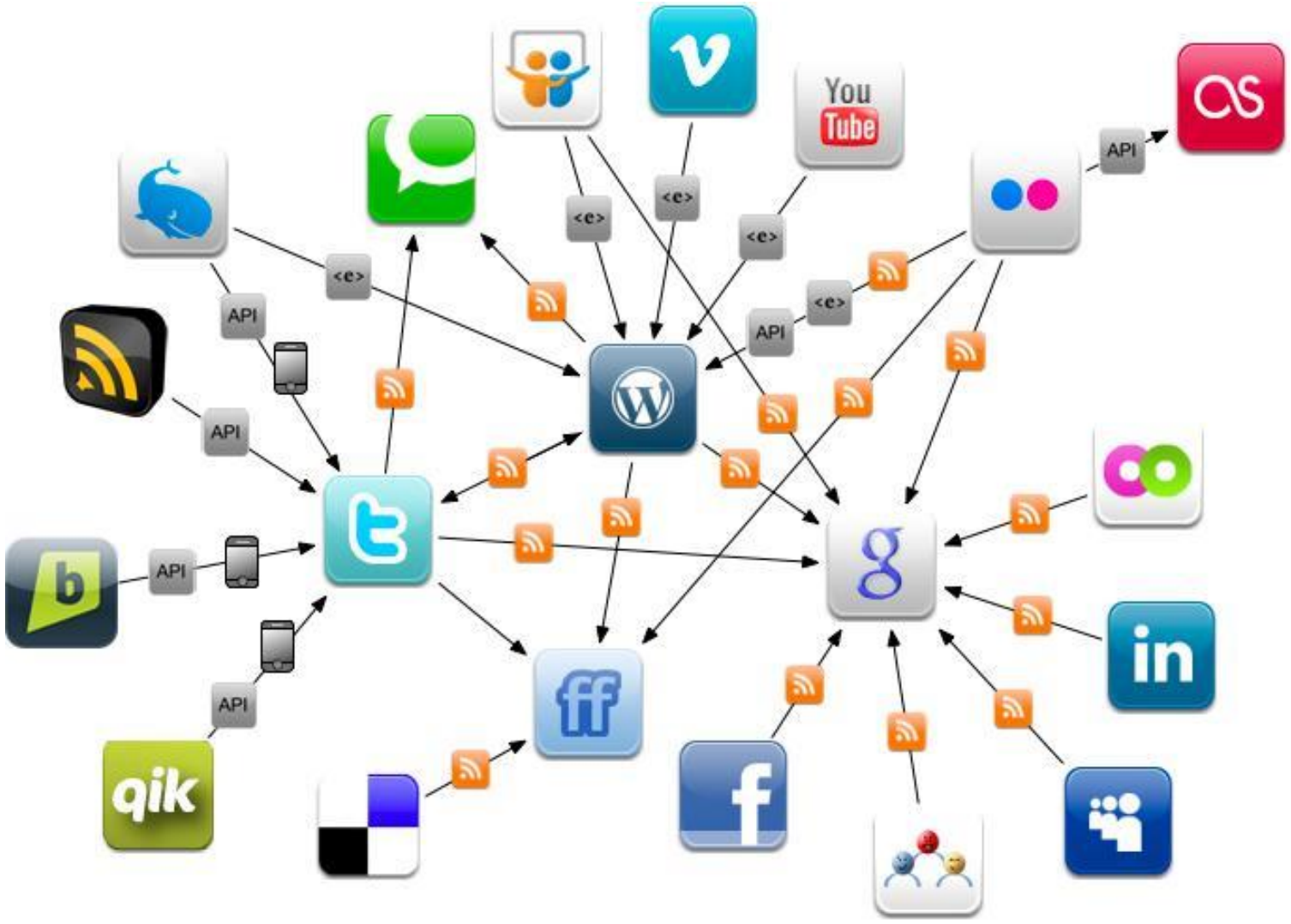
GRAPHS

DATA STRUCTURES AND ALGORITHMS

GRAPHS DATA STRUCTURE AND ALGORITHMS

Graph

- Graph Overview
  - Definition
  - Types of graph
  - Application
- Graphs Presentations
  - Adjacency Matrix
    - Fulfilling, Printing
  - Adjacency List
    - Fulfilling, Printing
  - Edge list
    - Fulfilling, Printing
  - Converting one type to another

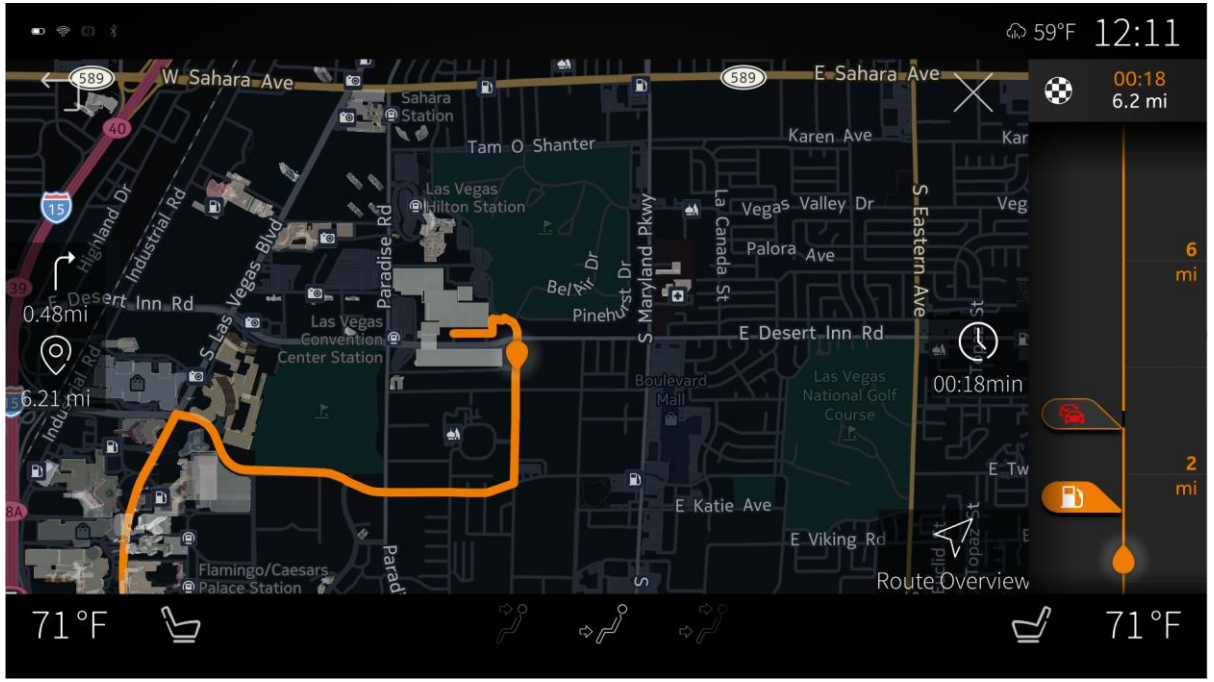


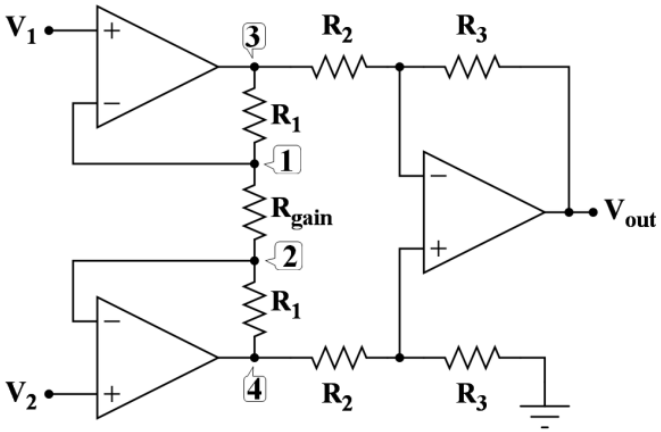
APPLICATIONS AND USAGE

Applications

- Navigation
  - Path planning
- Relations
- Connecting
- Social networks
- Bio Programming
- Graphics Applications
  - Autodesk
- Network Flows
  - Calculating traffic jams
  - Cables calculating

Facebook: a centralized network





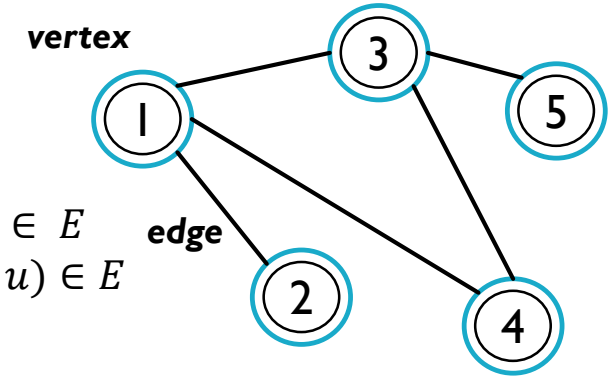
INTRODUCTION TO GRAPHS

DATA STRUCTURES AND ALGORITHMS

GRAPH

$$G = (V, E)$$
$$E \subset V x\, V$$

Set of vertices V and set of edges E



$undirected\ graph\ (u, v) \in E$
$iff\ (v, u) \in E$

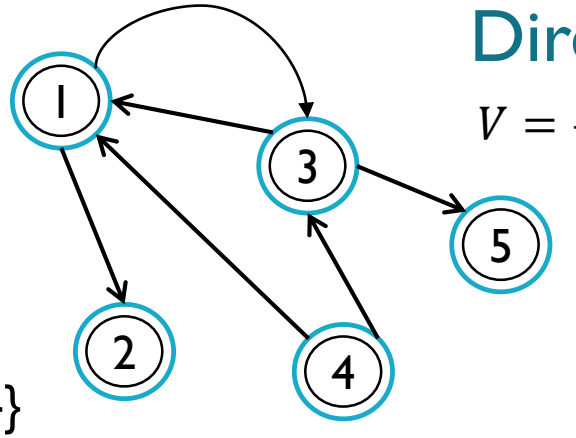
Undirected

$V = \{ 1, 2, 3, 4, 5 \}$

$E = \{\{1,2\}\{2,1\}\{1,4\}\{4,1\}, \{1,3\}, \{3,1\}\{3,4\}\{4,3\}\{3,5\}\{5,3\}\}$

Graphs

- Directed / Undirected
- Weighter vs Unweighted
- Cyclic vs Acyclic



Directed

$V = \{ 1, 2, 3, 4, 5 \}$

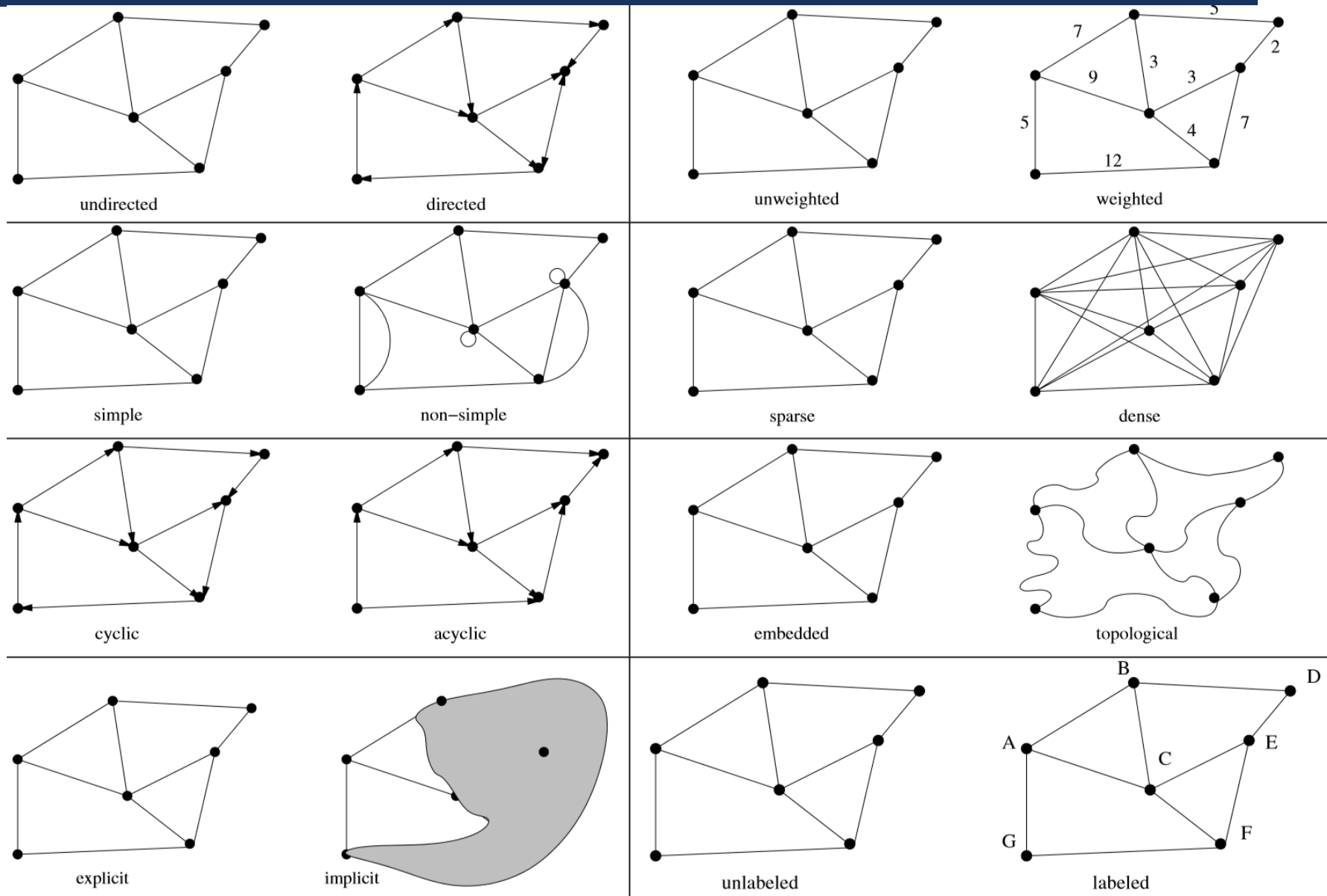
$E = \{\{1,2\},\{4,1\},\{4,3\},\{3,1\},\{1,3\},\{3,5\}\}$

$E = \{\{1,2\}\{3,1\}\{1,3\}\{4,1\}, \{4,3\}, \{3,5\}\}$

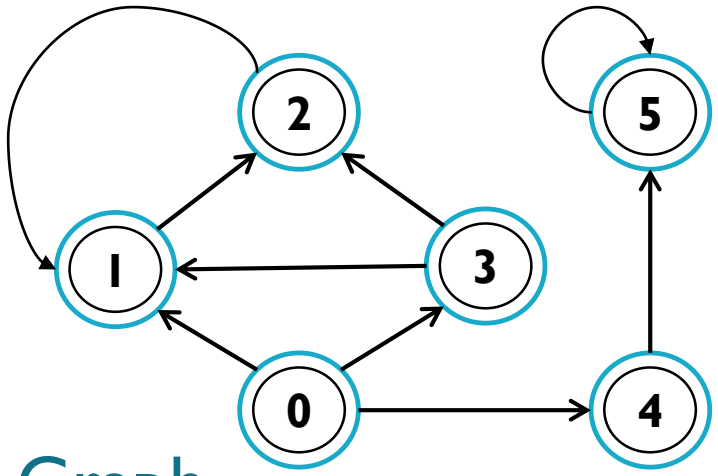
GRAPHS TYPES

Classification

- Directed / Undirected
- Simple / Non-Simple
  - Non-Simple graphs contain self loop {x,x}
- Cyclic / Acyclic
  - Cyclic – node has path to itself
- Explicit / Implicit
  - Implicit graphs usually known by traversing
- Unweighted / Weighted
- Sparse / Dense
  - Dense – quadratic number of edges
- Embedded / Topological
  - Embedded graph has geometric position
- Unlabeled / Labeled



GRAPHS REPRESENTATION



Graph

M	0	1	2	3	4	5
0	0	1	0	1	1	0
1	0	0	1	0	0	0
2	0	1	0	0	0	0
3	0	1	1	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	1

Adjacency Matrix

L	
0	→ 1 → 3 → 4
1	→ 2
2	→ 1
3	→ 1 → 2
4	→ 5
5	→ 5

Adjacency List

E
0 → 1
0 → 3
0 → 4
1 → 2
2 → 1
3 → 1
3 → 2
4 → 5
5 → 5

Adjacency Edge

$$V = \{ 0, 1, 2, 3, 4, 5 \}$$

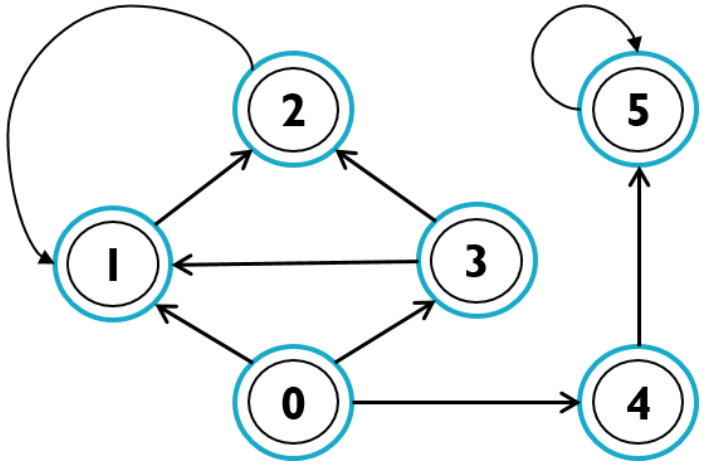
$$E = \{\{0,1\},\{0,3\},\{0,4\},\{1,2\},\{2,1\},\{3,1\},\{3,2\},\{4,5\},\{5,5\}\}$$

MATRIX REPRESENTATION

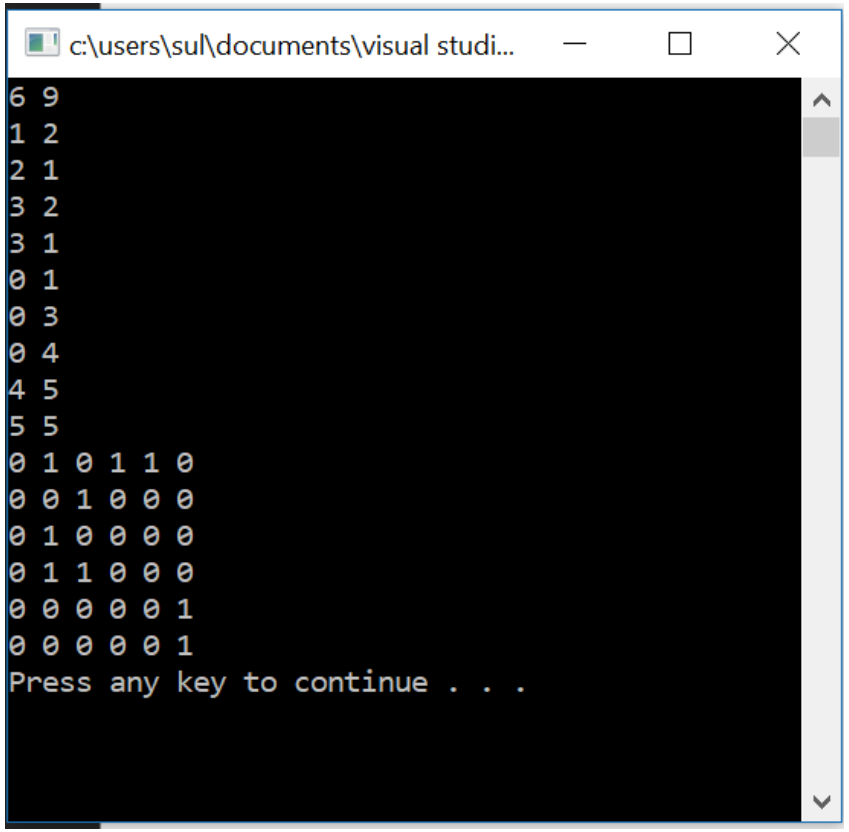
DATA STRUCTURES AND ALGORITHMS

ADJUCENCY MATRIX FULFILL AND PRINT

```
int main(){

  int n, m;
  int a[100][100];
  cin>>n>>m;
  int from, to;
  memset(a, 0, sizeof(a));
  for (int i = 0; i < m; i++){

   cin>>from>>to;
   a[from][to] = 1;

  }

  for (int i = 0; i < n; i++){
     for (int j = 0; j < n; j++){
              cout<<a[i][j]<<" ";
     }
     cout<<endl;
  }
  system("pause");
  return 0;
}
```



Adjacency Matrix Fulfilling



ADJACENCY LIST REPRESENTATION

DATA STRUCTURES AND ALGORITHMS

GRAPH REPRESENTATION: ADJACENCY LIST

```
typedef vector<int> vi;
vector<vi> adj;
```

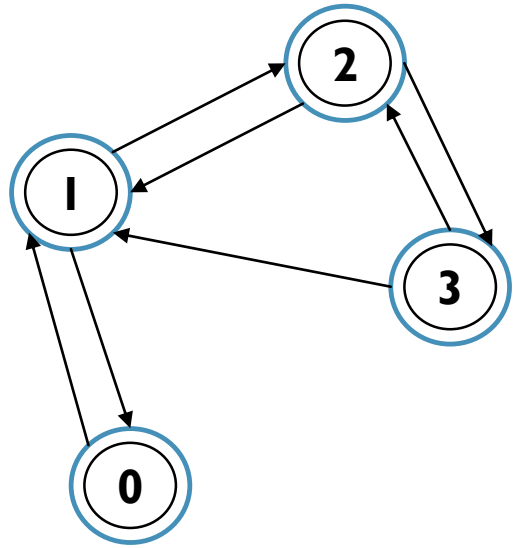
```
vector<vector<int>> adj;
```

List presentation

0: {1}
1: {0}, {2}
2: {1}, {3}
3: {2}, {1}

List presentation

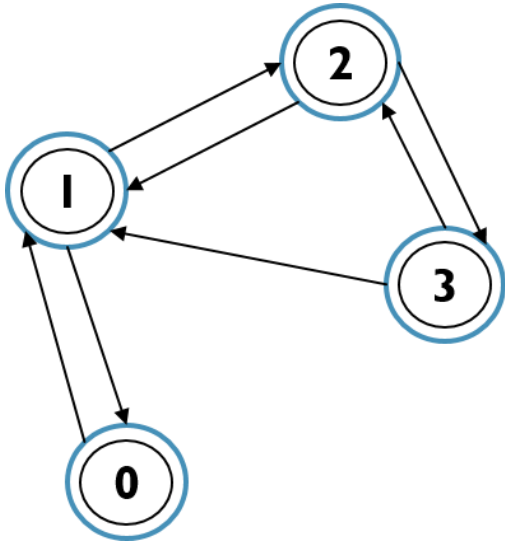
- An adjacency list could be implemented by vectors or by linked list,.
- The inner vector (vi) representing the nodes. Must be assigned while before use the graph.
- The outher vector (Adj) represents list of connections. Must not be assigned, but dynamically adds nodes and resizes own size. push_back( ) method



$$V = \{0, 1, 2, 3\}$$

GRAPH REPRESENTATION: ADJACENCY LIST FULFILLING AND PRINT

```
typedef vector<int> vi;
vector<vi> adj;
int main(){

	int n, m; // nodes, edges
	cin>>n>>m;
	adj.assign(n, vi()); // important !
	int from, to, weight;
	for (int i = 0; i < m; i++)  // 2*m in undirected
	{
		cin>>from>>to;
		adj[from].push_back(to);
		//adj[to].push_back(from); // for undirected
	}
	cout<<endl;
	for (int i = 0; i < adj.size(); i++){
		auto edges = adj[i];
		cout<<i<<": ";
		for (int j = 0; j < edges.size(); j++){
			cout<<edges[j]<<"; ";
		}
		cout<<endl;
	}
	system("pause");
	return 0;
}
```




```
c:\users\sul\documents\visual studio ...
4 7
1 0
0 1
3 1
1 2
2 1
3 2
2 3

0: 1;
1: 0; 2;
2: 1; 3;
3: 1; 2;
Press any key to continue . . .
```

Adjacency List Fulfilling

GRAPH REPRESENTATION: ADJACENCY LIST (WITH WEIGHT)

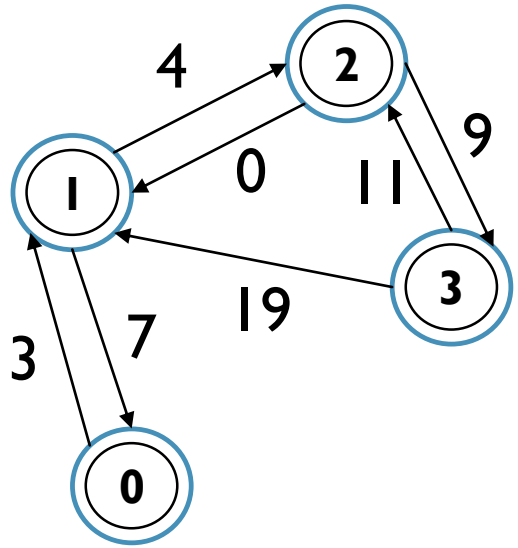
```
typedef pair<int,int> ii; // to, weight
typedef vector<ii> vii;   // from -> to
vector<vii> adj;          // graph
```

```
vector<vector<pair<int,int>>>
```

0: {1, 3}
1: {0, 7}, {2, 4}
2: {1, 0}, {3, 9}
3: {2, 11}, {1, 19}

List presentation with weight

Graph



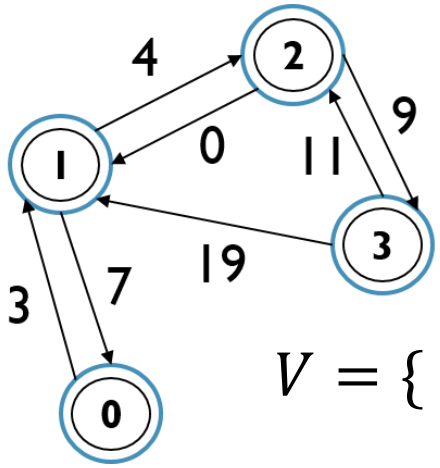
$V = \{0, 1, 2, 3\}$

ADJACENCY LIST WITH WEIGHT: FULFILLING AND PRINT

Weighted Graph Adjacency List Fulfilling

```
typedef pair<int,int> ii;
typedef vector<ii> vii;
vector<vii> adj;

int main(){
     int n, m; // nodes, edges
     cin>>n>>m;
     adj.assign(n, vii()); // important !
     int from, to, weight;
     for (int i = 0; i < m; i++){  // 2*m in undirected
          cin>>from>>to>>weight;
          adj[from].push_back(ii(to,weight));
          //adj[to].push_back(ii(from,weight)); // for undirected
     }
     cout<<endl;
     for (int i = 0; i < adj.size(); i++){
          auto edges = adj[i];
          cout<<i<<": ";
          for (int j = 0; j < edges.size(); j++){
               cout<<edges[j].first<<" "<<edges[j].second<<"; ";
          }
          cout<<endl;
     }
     system("pause");
     return 0;
}
```

```
c:\users\sul\documents\visual studio 2012\...
4 7
1 0 7
0 1 3
1 2 4
2 1 0
2 3 9
3 2 11
3 1 19

0: 1 3;
1: 0 7; 2 4;
2: 1 0; 3 9;
3: 2 11; 1 19;
Press any key to continue . . .
```



$V = \{0, 1, 2, 3\}$

EDGE REPRESENTATION

DATA STRUCTURES AND ALGORITHMS

EDGE PRESENTATION

```
struct edge{

    int from;
    int to;
    int weight;
};
vector<edge> adj;
```

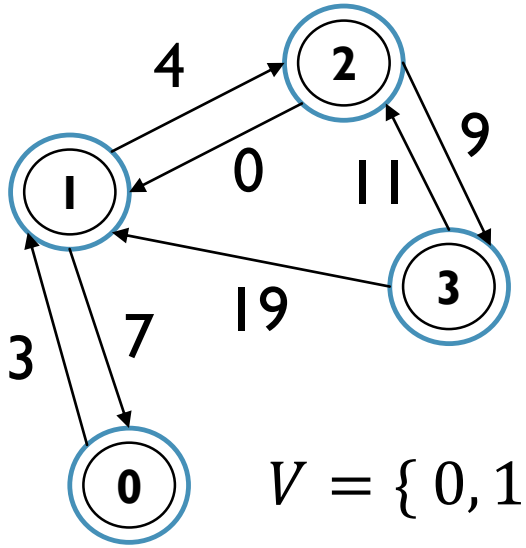
Structure version

{ 7, 1, 0}
{ 3, 0, 1}
{ 4, 1, 2}
{ 0, 2, 1}
{ 11, 3, 9}
{ 9, 2, 3}
{ 19, 3, 1}

Edge presentation

- STL version of edges: The first parameter in weighted graph represents by weight, because sorting is always done by the first parameter that will be used in Kruskal Algorithm.
- In Structure version, each edge represents by structure with three parameters, but to sort them by weight need to use lambda expressions or additional function in edges structure.

Graph



$V = \{ 0, 1, 2, 3\}$

```
vector<pair<int, pair<int,int>> adj;
```

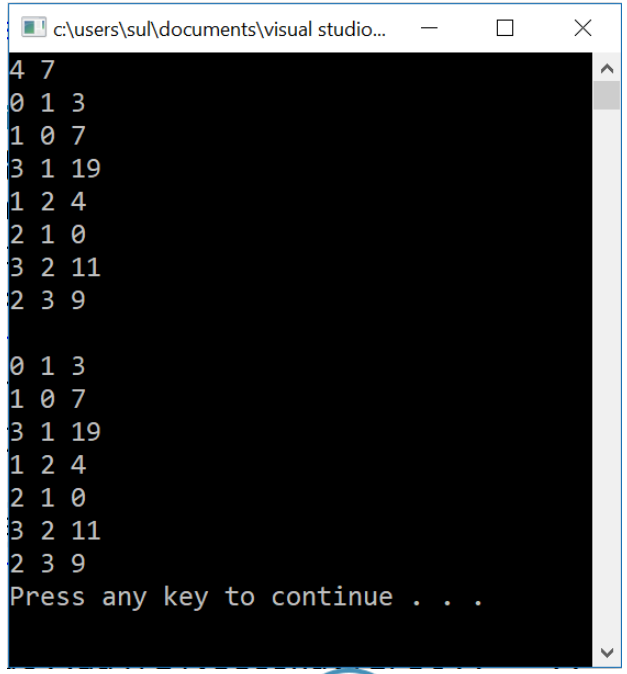
STL version: weighted graph

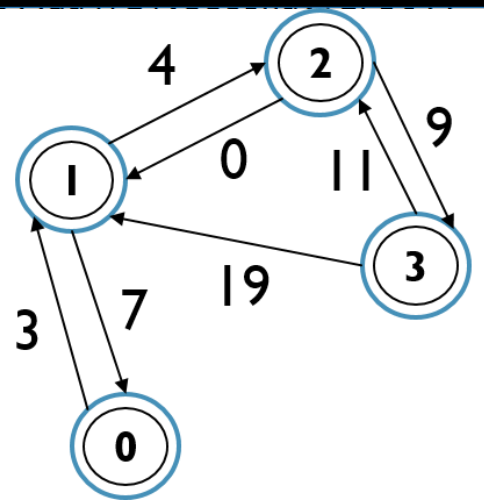
```
vector<pair<int,int>> adj;
```

STL version: unweighted graph

EDGE IMPLEMENTATION

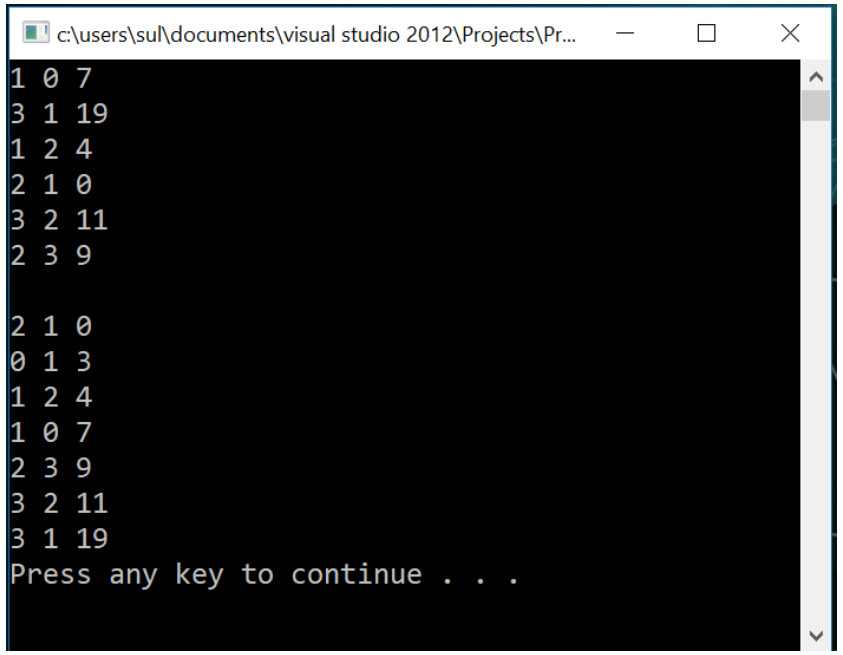
```
vector< pair<int, pair<int,int>>> adj;

int main(){

    int n,m;
    int from, to, weight;
    cin>>n>>m;
    for (int i = 0; i < m; i++){
        cin>>from>>to>>weight;
        adj.push_back(make_pair( weight, make_pair(from,to)));
    }

    cout<<endl;
    for (int i = 0; i < adj.size(); i++){
        cout<<adj[i].second.first<<" "<<
            adj[i].second.second<<" "<<adj[i].first<<endl;
}
system("pause");
return 0;
}
```





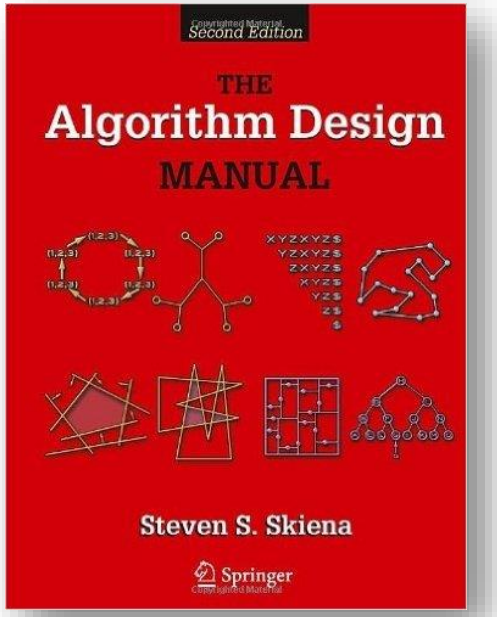
FULFILLING GRAPH BY STRUCTURE

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

struct edge{

    int from;
    int to;
    int weight;
};

vector<edge> adj;

int main(){
    int n,m;
    cin>>n>>m;
    for (int i = 0; i < m; i++){
        int from, to, weight;
        cin>>from>>to>>weight;
        edge e;
        e.to = to;
        e.from = from;
        e.weight = weight;
        adj.push_back(e);
    }
    cout<<endl;

    auto lambda = [](edge e, edge e2){ return e.weight < e2.weight;};
    sort(adj.begin(), adj.end(), lambda);

    for(auto i: adj)
        cout<<i.from<<" "<<i.to<<" "<<i.weight<<endl;
    system("pause");
    return 0;
}
```

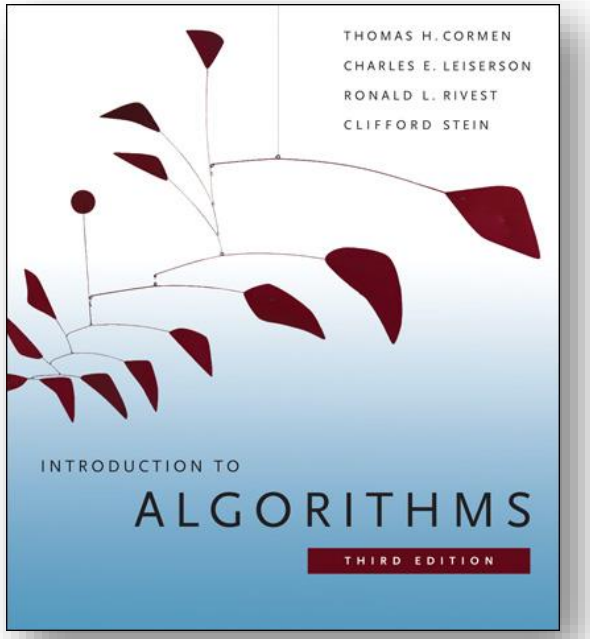


Sorted by weight.

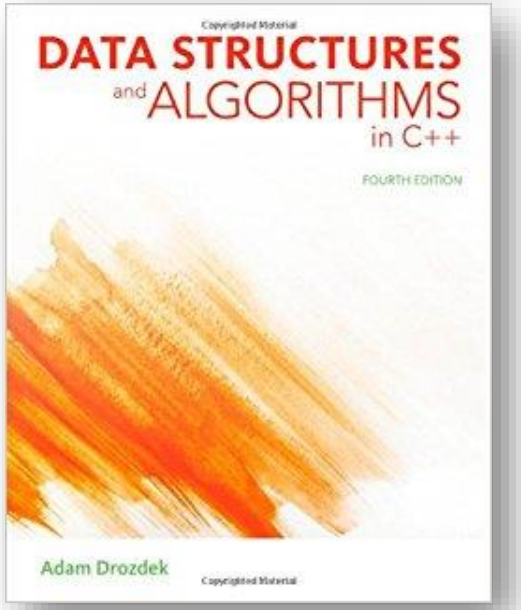
LITERATURE



Stieven Skienna
Algorithms design manual
Chapter 5: Graph Traversal
Page 145



Thomas H. Cormen
Introduction to Algorithms
Chapter VI Graph Algorithms
Page 587.



Adam Drozdek
Data structures and Algorithms in C++
Chapter 8: Graphs
Page 391