Graph Representations

Mattox Beckman

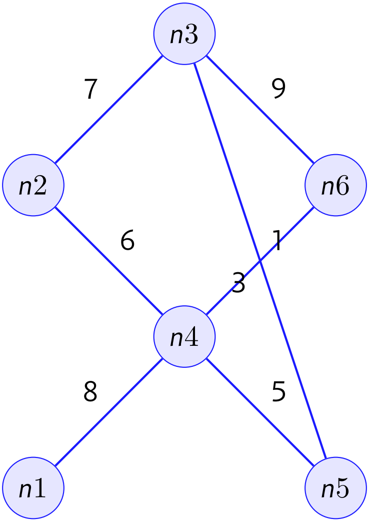
University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2023

Objectives

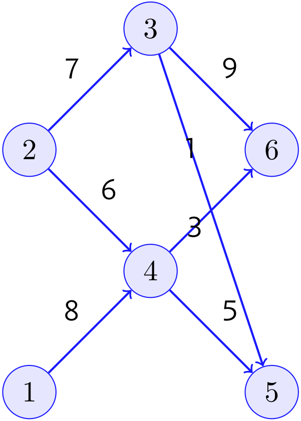
- Use three methods of representing a graph
  - Adjacency Matrix
  - Adjacency List
  - Edge List
- Explain the time complexities and tradeoffs

Graph Properties



Directed Graphs

Directed Graph

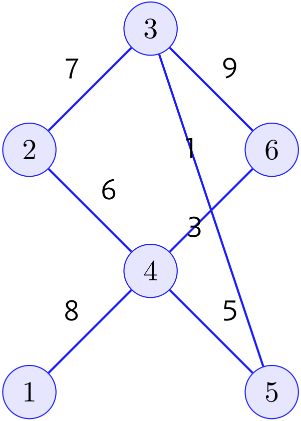


Adjacency Matrix

	1	2	3	4	5	6
1				8		
2			7	6		
3					1	9
4					5	3
5						
6						

Undirected Graphs

Undirected Graph

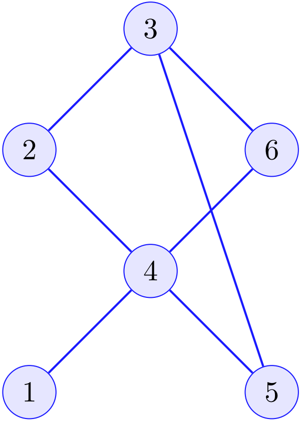


Adjacency Matrix

	1	2	3	4	5	6
1				8		
2			7	6		
3		7			1	9
4	8	6			5	3
5			1	5		
6			9	3		

Unweighted Graphs

Unweighted Graph



Adjacency Matrix

	1	2	3	4	5	6
1				1		
2			1	1		
3		1			1	1
4	1	1			1	1
5			1	1		
6			1	1		

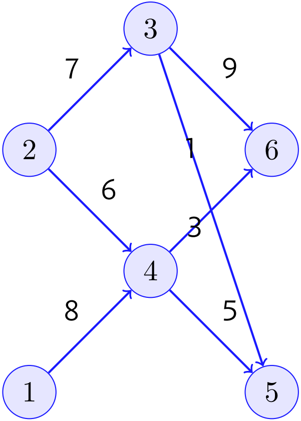
Adjacency Matrix Implementation

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

cin >> N; // number of nodes
cin >> E; // number of edges

vvii graph = vvii(N,vi(N));
for(int i=0; i<e; ++i) {
  cin >> s >> d >> w ;  // source, destination, weight
  graph[s][d] = w;
  // and if undirected, do this too
  graph[d][s] = w;
}
```

- $\mathcal{O}(n^2)$ memory — expensive!
- What kind of queries are good for this?

Directed Graphs

Directed Graph

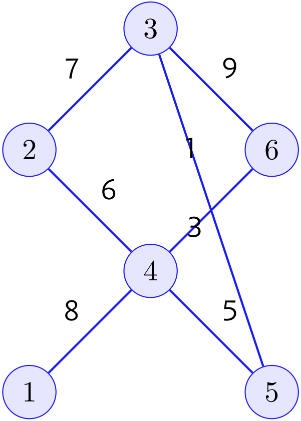


Adjacency List

1	(4,8)	
2	(4,6)	(3,7)
3	(5,1)	(6,9)
4	(6,3)	(5,5)
5		
6		

Undirected Graphs

Undirected Graph

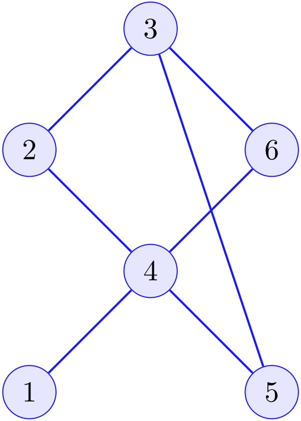


Adjacency List

1	(4,8)			
2	(4,6)	(3,7)		
3	(5,1)	(2,7)	(6,9)	
4	(6,3)	(5,5)	(1,8)	(2,6)
5	(3,1)	(4,5)		
6	(3,9)	(4,3)		

Unweighted Graphs

Unweighted Graph



Adjacency List

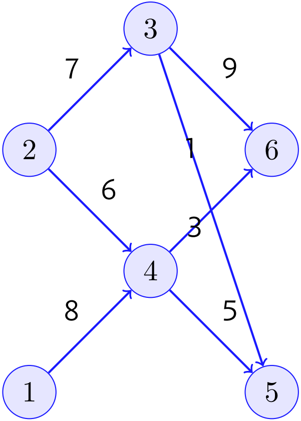
1	4			
2	4	3		
3	5	2	6	
4	6	5	1	2
5	3	4		
6	3	4		

Adjacency List Implementation

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

cin >> N; // number of nodes
cin >> E; // number of edges

vvii graph = vii(N);
for(int i=0; i<e; ++i) {
  cin >> s >> d >> w ;  // source, destination, weight
  graph[s].push_back(ii(d,w));
  // and if undirected, do this too
  graph[d].push_back(ii(s,w));
}

// accessing edges of node a:
for(auto it=graph[a].begin(); it != graph[a].end(); ++it)
  cout << a << " goes to " << it->first() << " with weight
```

Directed Graph

Directed Graph

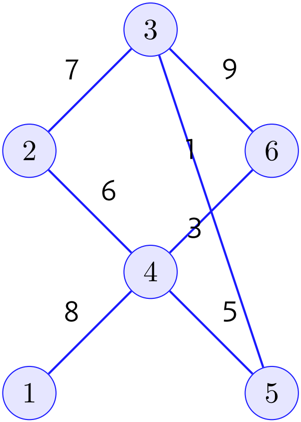


Edge List

(1,4,8)
(2,4,6)
(3,5,1)
(4,6,3)
(2,3,7)
(3,6,9)
(4,5,5)

Undirected graph

Undirected Graph

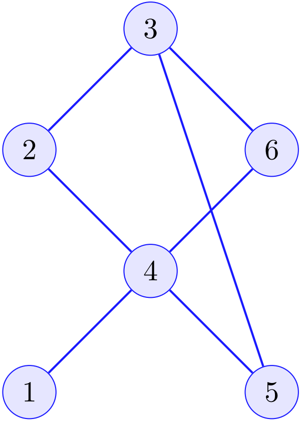


Edge List

(1,4,8)
(2,4,6)
(3,5,1)
(4,6,3)
(2,3,7)
(3,6,9)
(4,5,5)
(4,1,8)
(4,2,6)
(5,3,1)
(6,4,3)
(3,2,7)
(6,3,9)
(5,4,5)

Unweighted Graph

Unweighted Graph



Edge List

(1,4)
(2,4)
(3,5)
(4,6)
(2,3)
(3,6)
(4,5)
(4,1)
(4,2)
(5,3)
(6,4)
(3,2)
(6,3)
(5,4)

Edge List Implementation

```
typedef pair<int,int> ii;
typedef pair<ii,int> edge;
typedef vector<edge> vedge;

cin >> N; // number of nodes
cin >> E; // number of edges

vedge graph;
for(int i=0; i<e; ++i) {
  cin >> s >> d >> w ;  // source, destination, weight
  graph.push_back(edge(ii(s,d),w));
  // and if undirected, do this too
  graph.push_back(edge(ii(d,s),w));
}
```

- $\mathcal{O}(e)$ memory — Not bad!
- What kind of queries are good for this?