SIT221: Data Structures and Algorithms

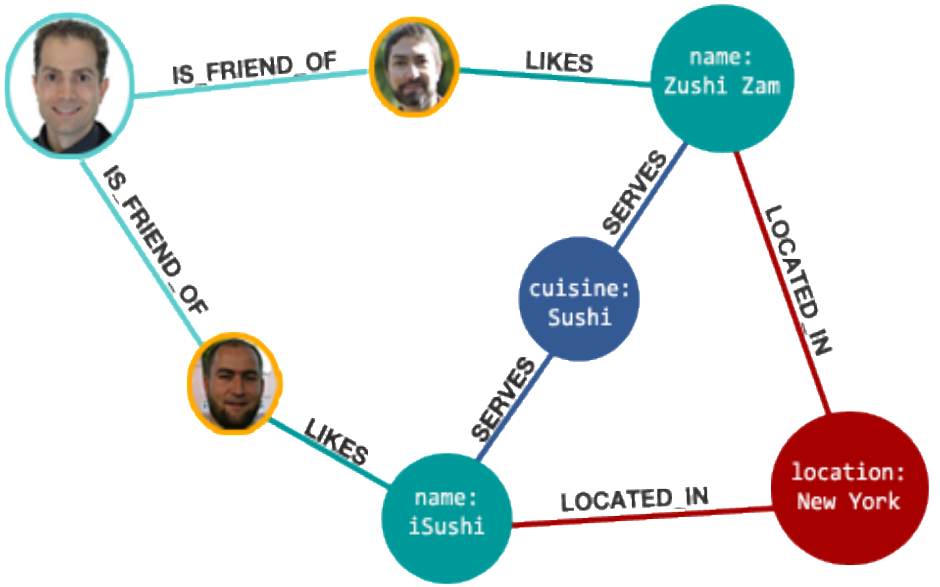
Lecture 8: Graphs

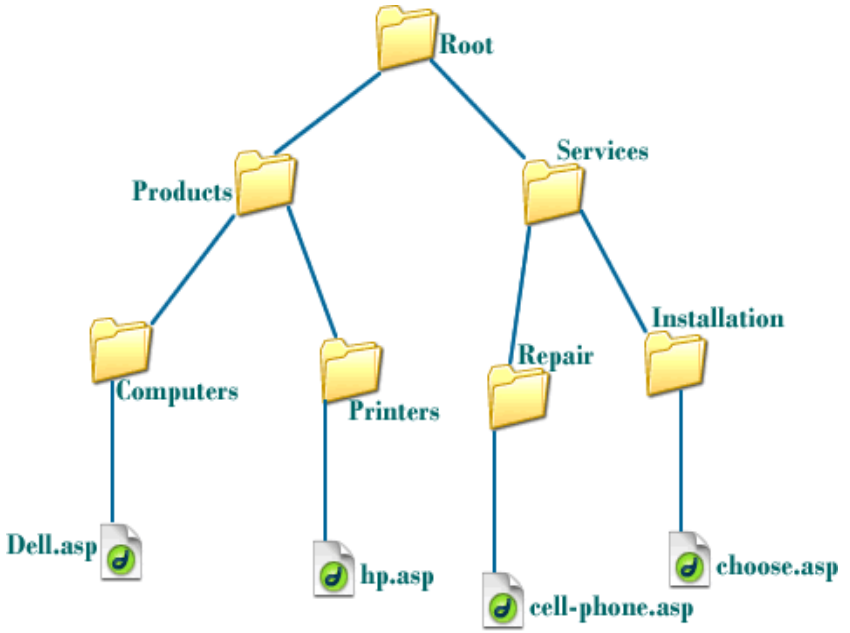
Week 7 recording for prac

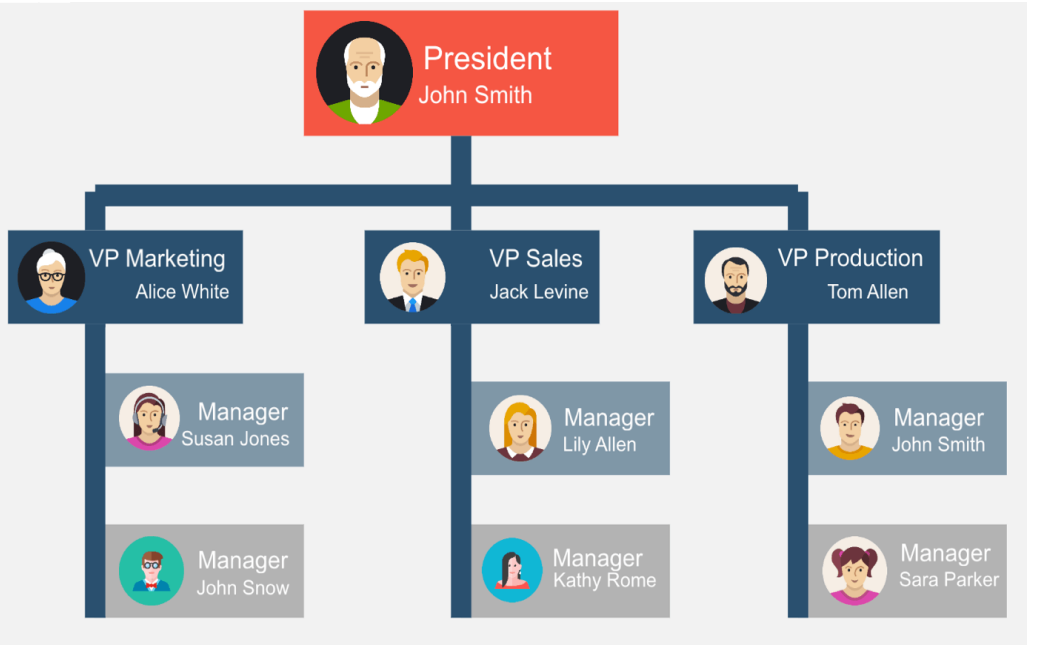
Week 7	Week 7 - prac - recording_1	Week 7 - prac	26/08/2017 4:11 pm	00:39:42	···
Week 4	SIT221 - Data Structures And Algorithms - recording_1	SIT221 - Data Structures And Algorithms	3/08/2017 11:22 am	00:42:34	···
Week 3	SIT221 - Data Structures And Algorithms - recording_1	SIT221 - Data Structures And Algorithms	27/07/2017 10:55 am	00:40:01	···
Week 2	Week 2 - prac - recording_1	Week 2 - prac	20/07/2017 11:01 am	00:31:17	···

Non-linear data structures

- Values are not arranged in order, many next/previous nodes.
  - Hierarchical data (last week)
  - Network data (this week)

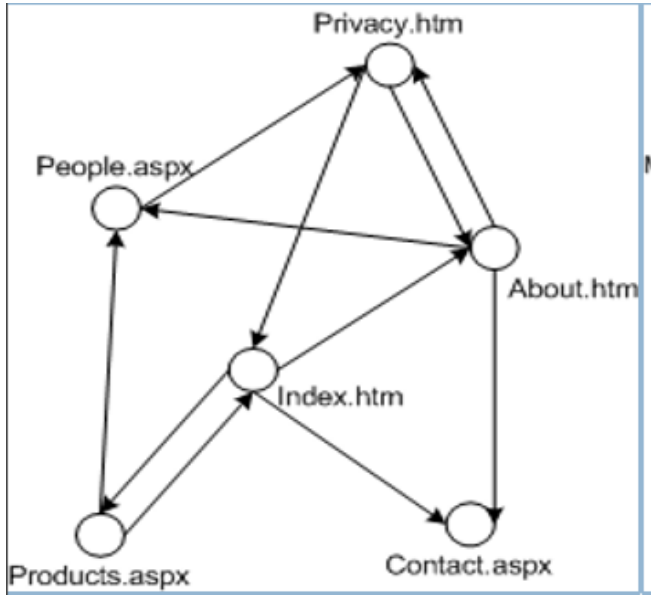


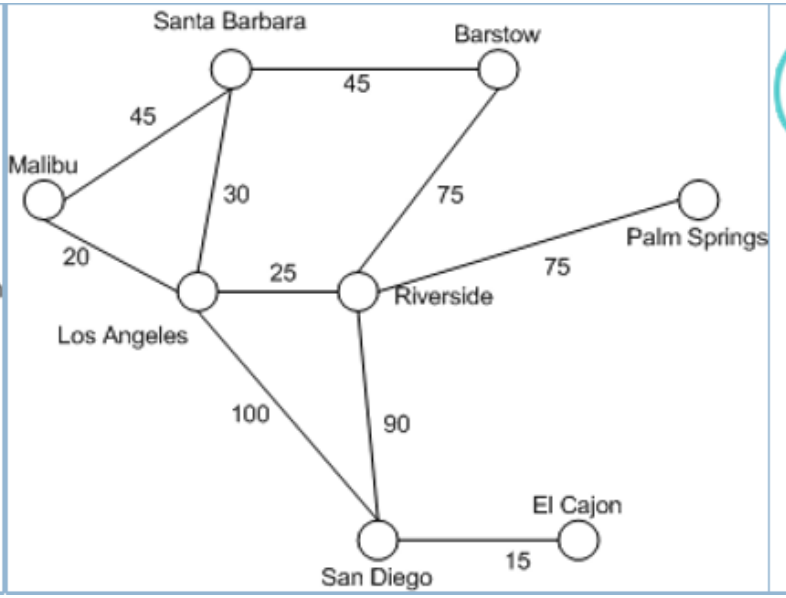


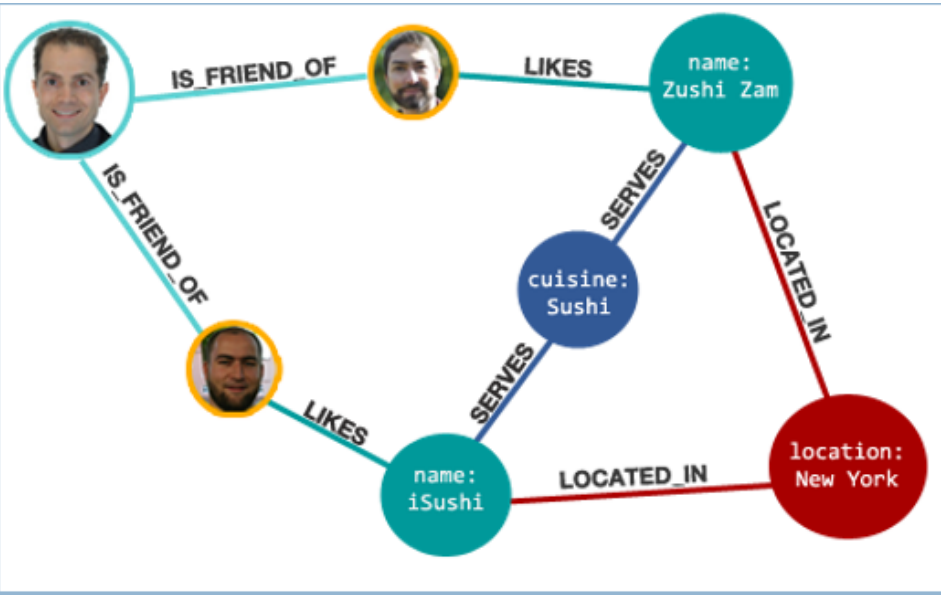


Motivation

- How can we store data of this type in memory?

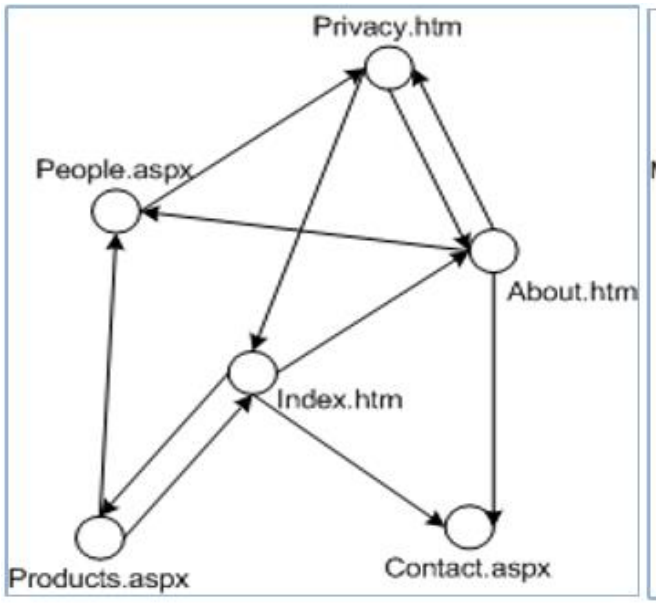


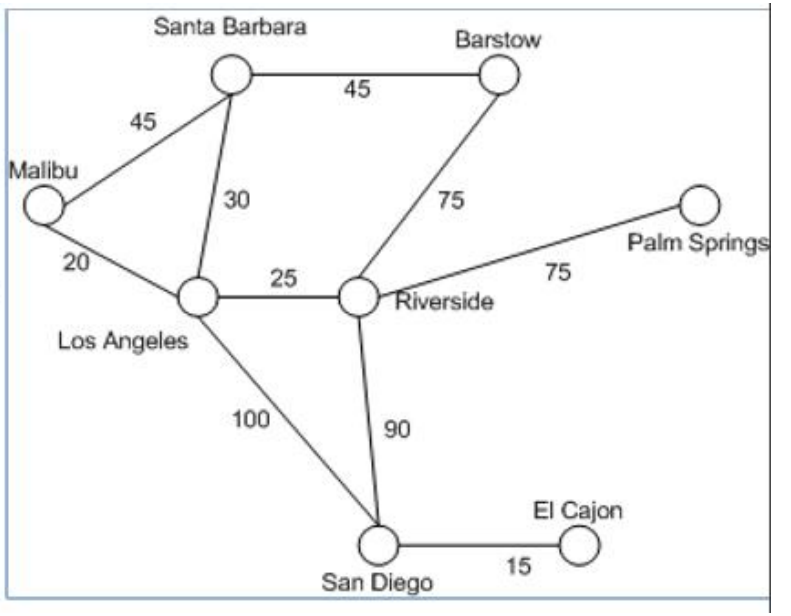




Graph data

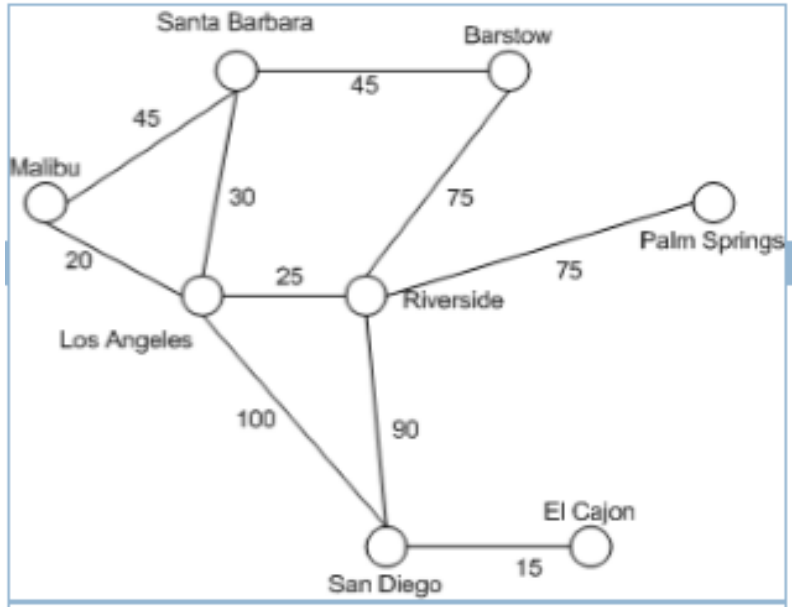
- What are the key elements in this diagram?
  - Nodes/vertices, e.g. pages, individuals, etc.
  - Links/edges, e.g. to represent navigations from one page to another, route from a city to another, etc.
- Thus, any graph can be described as: Vertices & edges
- **G = (V, E)**

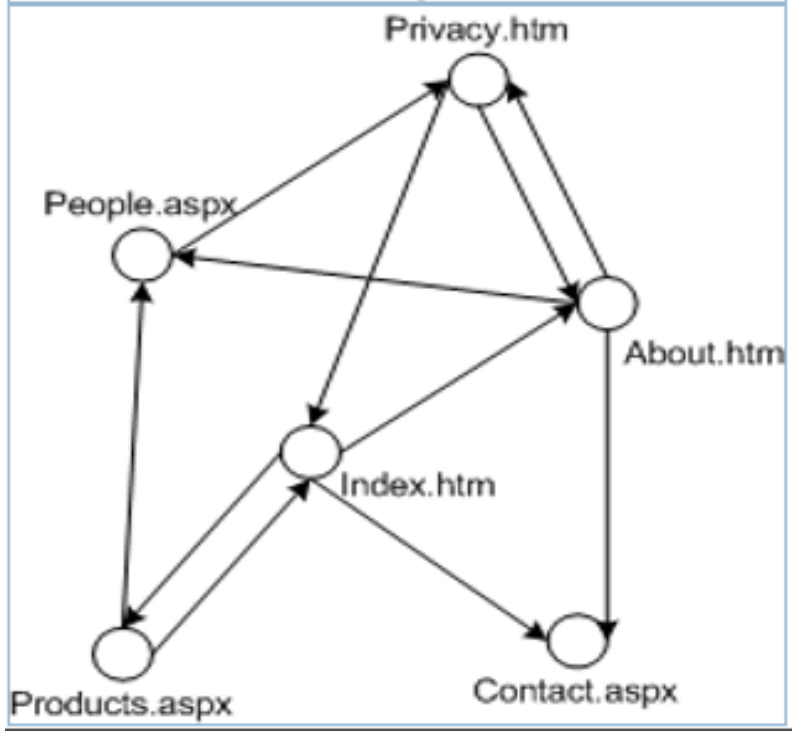




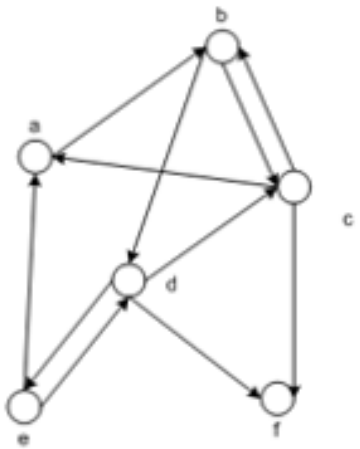
Graph data (cont)

- Are all graphs the same? NO
- Graphs can be directed or undirected
- Edges may have weights
- Graphs can be sparse (few links/edges) or dense (to many edges) or even fully connected (every two nodes are connected by a link)

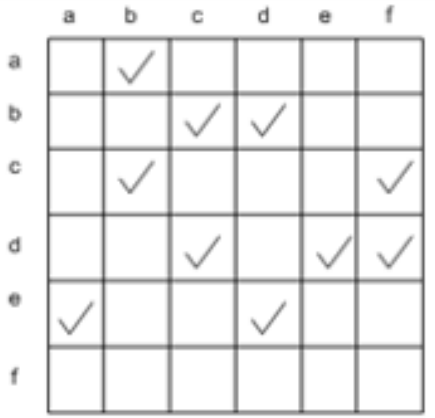




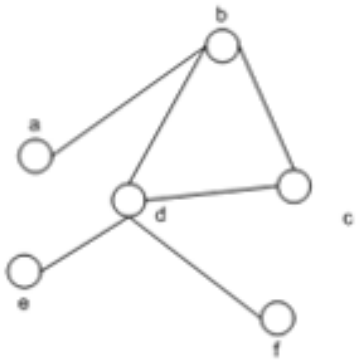
How to represent a graph?



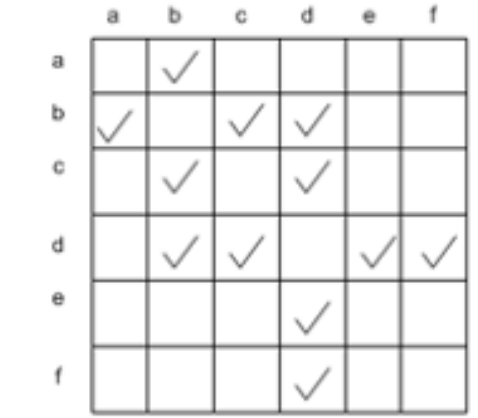
Directed Graph (a)



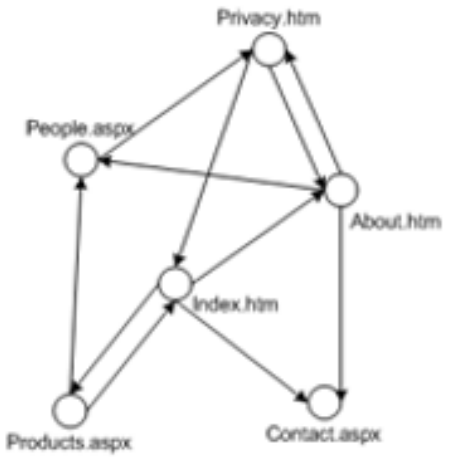
Adjacency Matrix Representation (a)



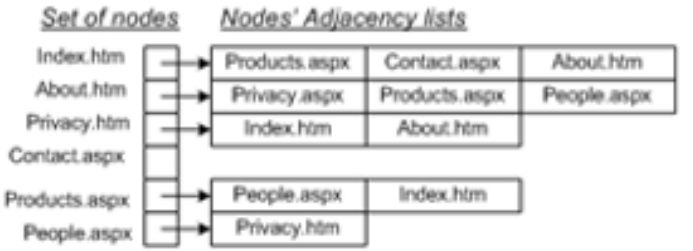
Undirected Graph (b)



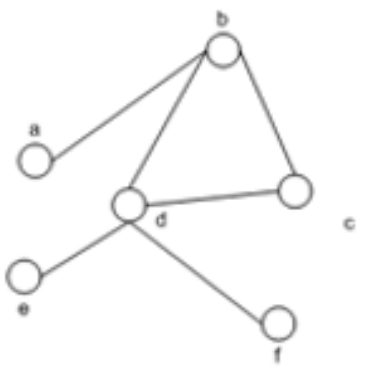
Adjacency Matrix Representation (b)



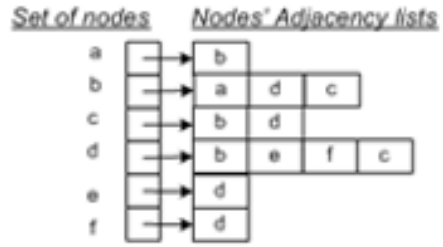
Directed Graph (a)



Adjacency List Representation (a)



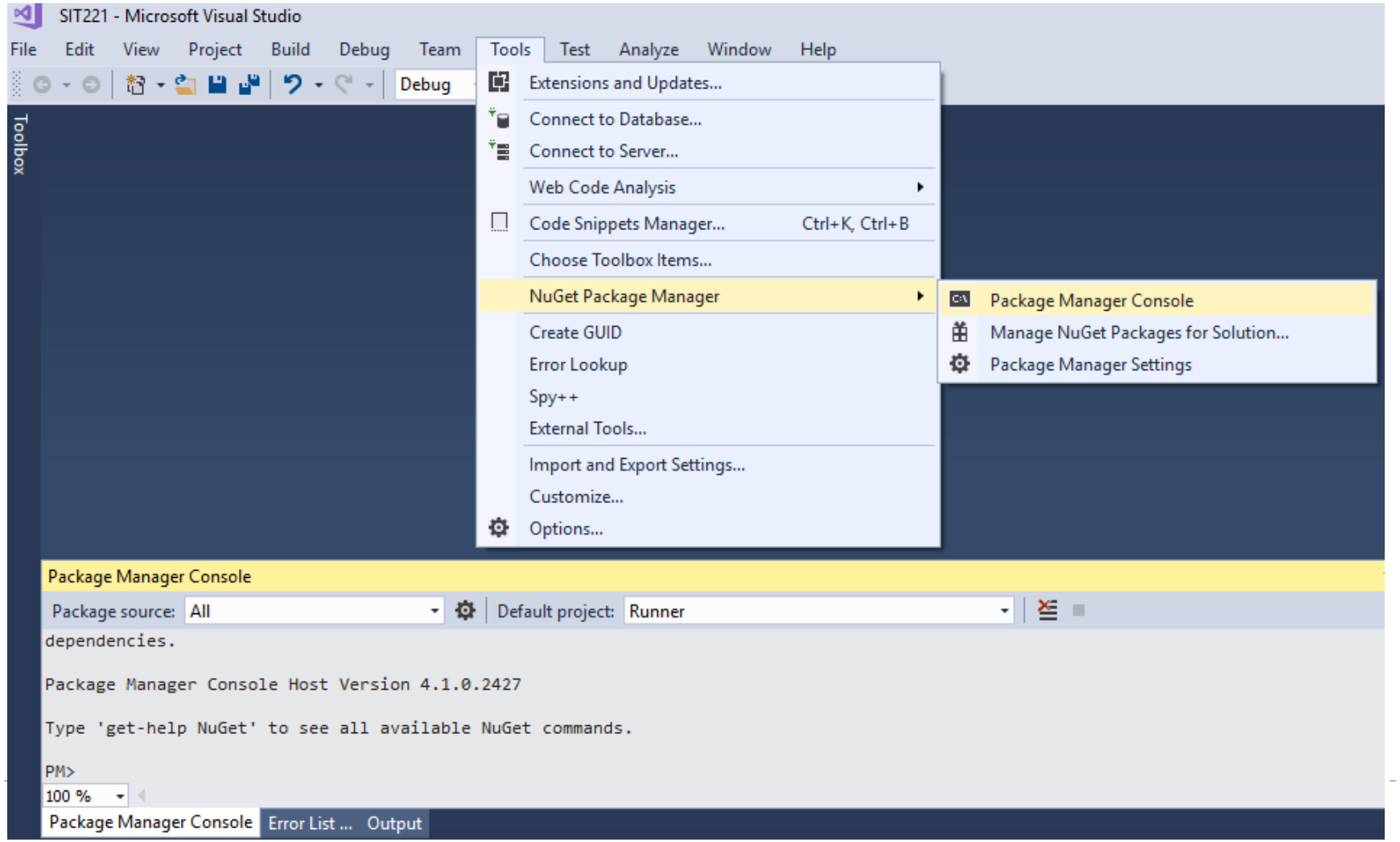
Undirected Graph (b)



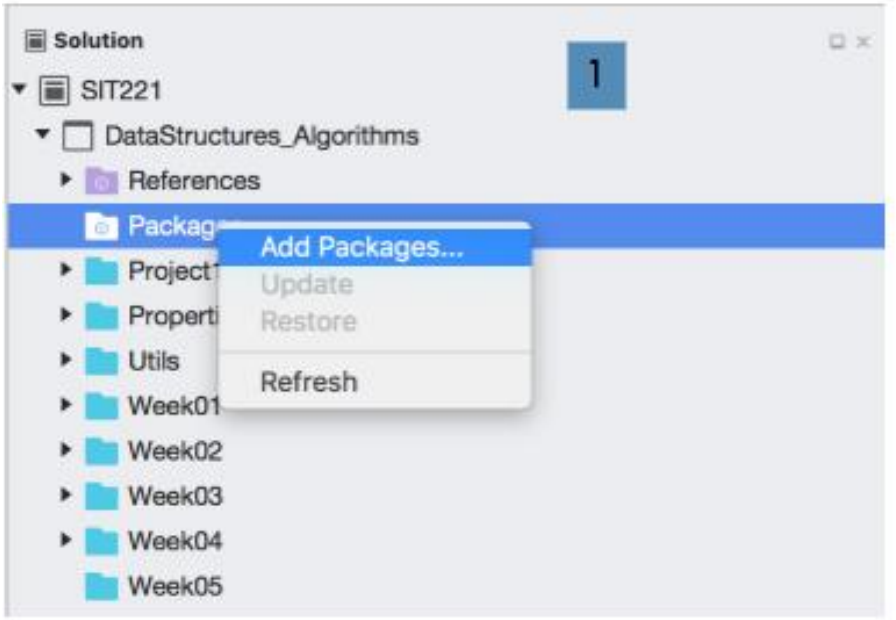
Adjacency List Representation (b)

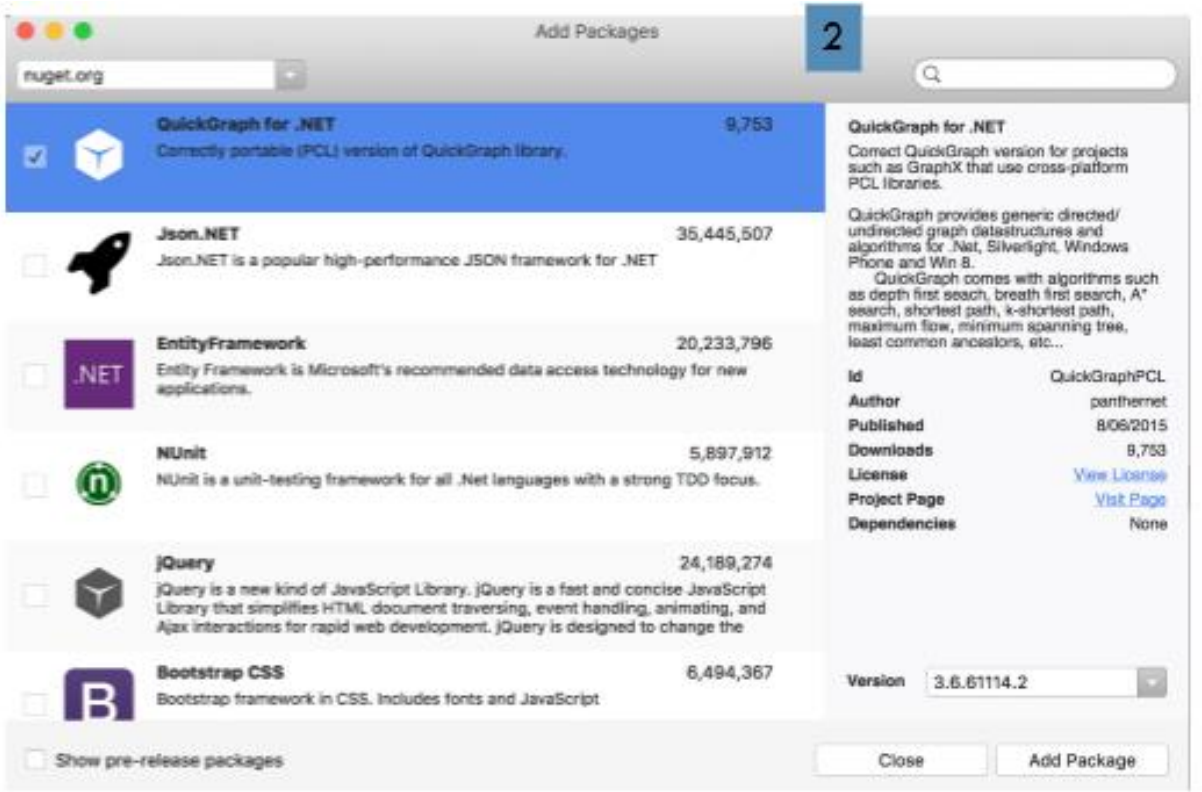
Using library

- Visual Studio 2017 http://quickgraph.codeplex.com/



Using library





Example 1

- A graph of cities
  - Let's create a bi-directional graph

```
var trafficNetwork = new BidirectionalGraph<NODETYPE, Edge<NODETYPE>>();
```

  - If we want each node is named by a city name
    - NOTETYPE: string

```
var trafficNetwork = new BidirectionalGraph<string, Edge<string>>();
```

Example 2

- A graph of integers

```
BidirectionalGraph<int, Edge<int>> integersGraph = new
BidirectionalGraph<int , Edge<int>>();
```

Example 3

- A graph of POIs

BidirectionalGraph<PointOfInterest, GraphEdge<PointOfInterest>> POIsGraph = new BidirectionalGraph<PointOfInterest , GraphEdge<PointOfInterest>>();

- GraphEdge class is a user-defined class that enables adding description and distance/time to any edge (you will see this in next week prac)

Adding nodes in a graph

- trafficNetwork

```
//Let's add Australia big cities
trafficNetwork.AddVertex("Melbourne");
trafficNetwork.AddVertex("Sydney");
trafficNetwork.AddVertex("Brisbane");
trafficNetwork.AddVertex("Adelaide");
trafficNetwork.AddVertex("Perth");
trafficNetwork.AddVertex("Melbourne"); //what would happen here?
```

Adding nodes in a graph

- integersGraph

```
integersGraph.AddVertex(1);
integersGraph.AddVertex(2);
integersGraph.AddVertex(3);
integersGraph.AddVertex(4);
```

Adding nodes in a graph

- POIsGraph

```
var poi = … // similar to week07 prac
POIsGraph.AddVertex(poi);
```

Adding edges in a graph

```
//new Edge<string>(Source, Target)
trafficNetwork.AddEdge(new Edge<string>("Melbourne", "Sydney"));
trafficNetwork.AddEdge(new Edge<string>("Melbourne", "Brisbane"));
trafficNetwork.AddEdge(new Edge<string>("Sydney", "Brisbane"));

//Let's add Edges between two POIs
POIsGraph.AddEdge(new GraphEdge<PointOfInterest> { Source = OBJ
_POI1 , Target =  OBJ_POI2 , Description = "Description of route from
POI1 to POI2", Distance = 100 });
```

Displaying graph nodes and edges

```
// Let's display all graph nodes
foreach (var vertex in trafficNetwork.Vertices)
     Console.WriteLine(vertex);

 // Let's display all edges
foreach (var edge in trafficNetwork.Edges)
     Console.WriteLine(edge);


// Let's display each node and it's edges
foreach(var vertex in trafficNetwork.Vertices)
   foreach (var edge in trafficNetwork.OutEdges(vertex)) //you can use InEdges
         Console.WriteLine(edge);
```

Graph Applications

- Traffic/Navigation
- PageRanking – Search Engines
- Social media – Facebook/Twitter
- Computer Networks/routing
- Dependencies – in your code/unit pre-requisites
- Project management – critical path analysis
- Graphical models (machine learning)