Data Structures
Chapter 7: Graph

1. Introduction
   - **Terminology, Representation, ADT**
2. Basic Operations
   - DFS, CC, BFS, Processing
3. Digraph and Applications
4. Minimum Spanning Tree(MST)

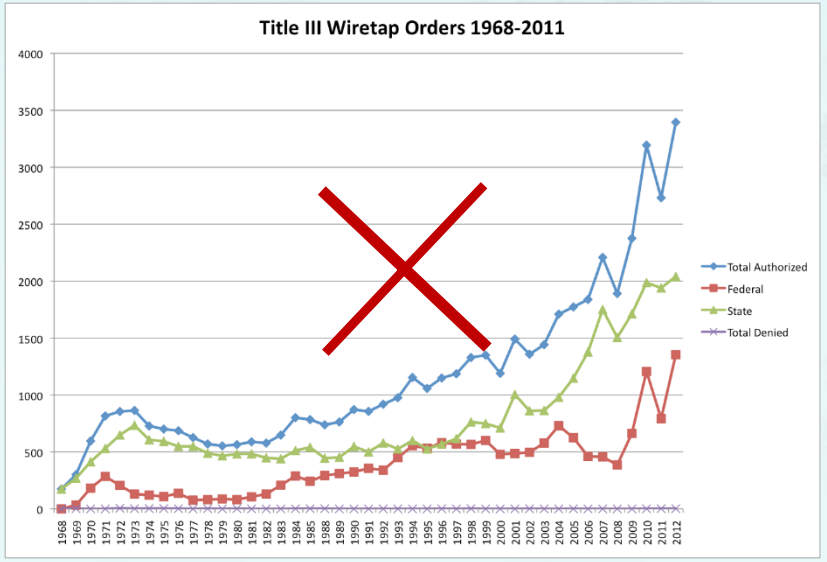
Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

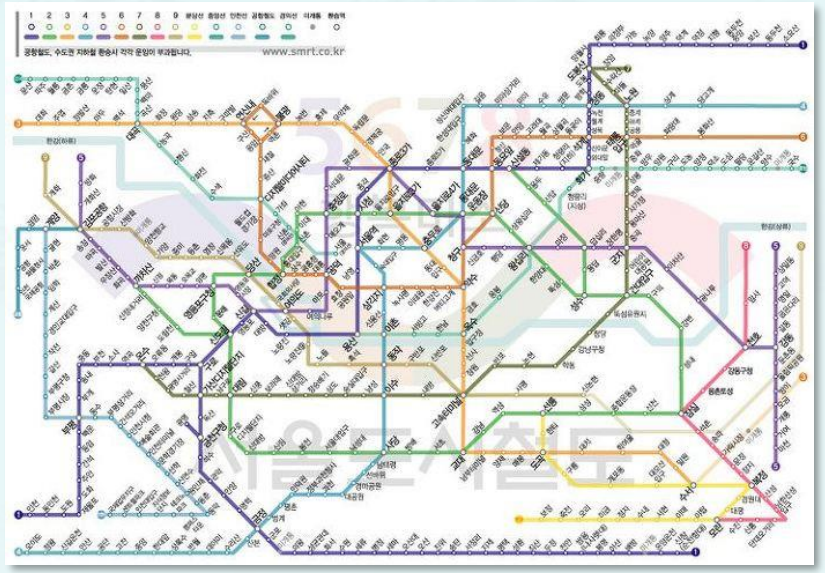
너는 청년의 때에 너의 창조주를 기억하라. 곧 곤고한 날이 이르기 전에, 나는 아무 낙이 없다고 할 해들이 가깝기 전에 (전도서 12:1)

모든 사람이 죄를 범하였으매 하나님의 영광에 이르지 못하더니 그리스도 예수 안에 있는 속량으로 말미암아 하나님의 은혜로 값없이 의롭다 하심을 얻은 자 되었느니라 (로마서 3:23-24)

Undirected graphs

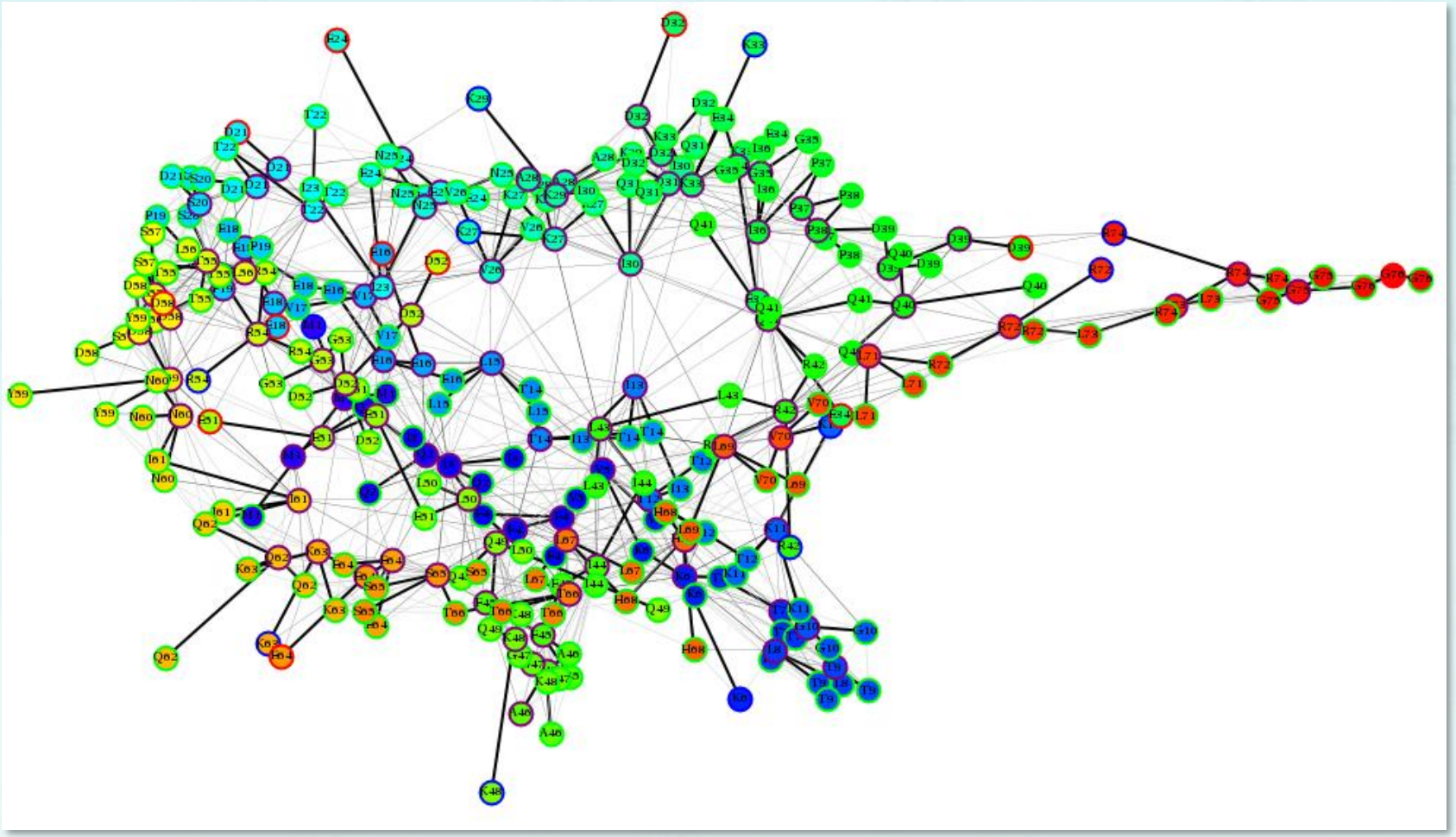
- **Graph: Set of vertices connected pairwise by edges**
- *Why study graph algorithms?*
  - Thousands of practical applications.
  - Hundreds of graph algorithms known.
  - Interesting and broadly useful abstraction.
  - Challenging branch of computer science and discrete math.





Undirected graphs

- **Chemical Environments: Protein Graphs**



Reference: **Benson NC**, Daggett V (2012) A comparison of methods for the analysis of molecular dynamics simulations. *J. Phys. Chem. B* **116**(29): 8722-31.

Undirected graphs

- The Spread of Obesity in a Large Social Network over 32 Years

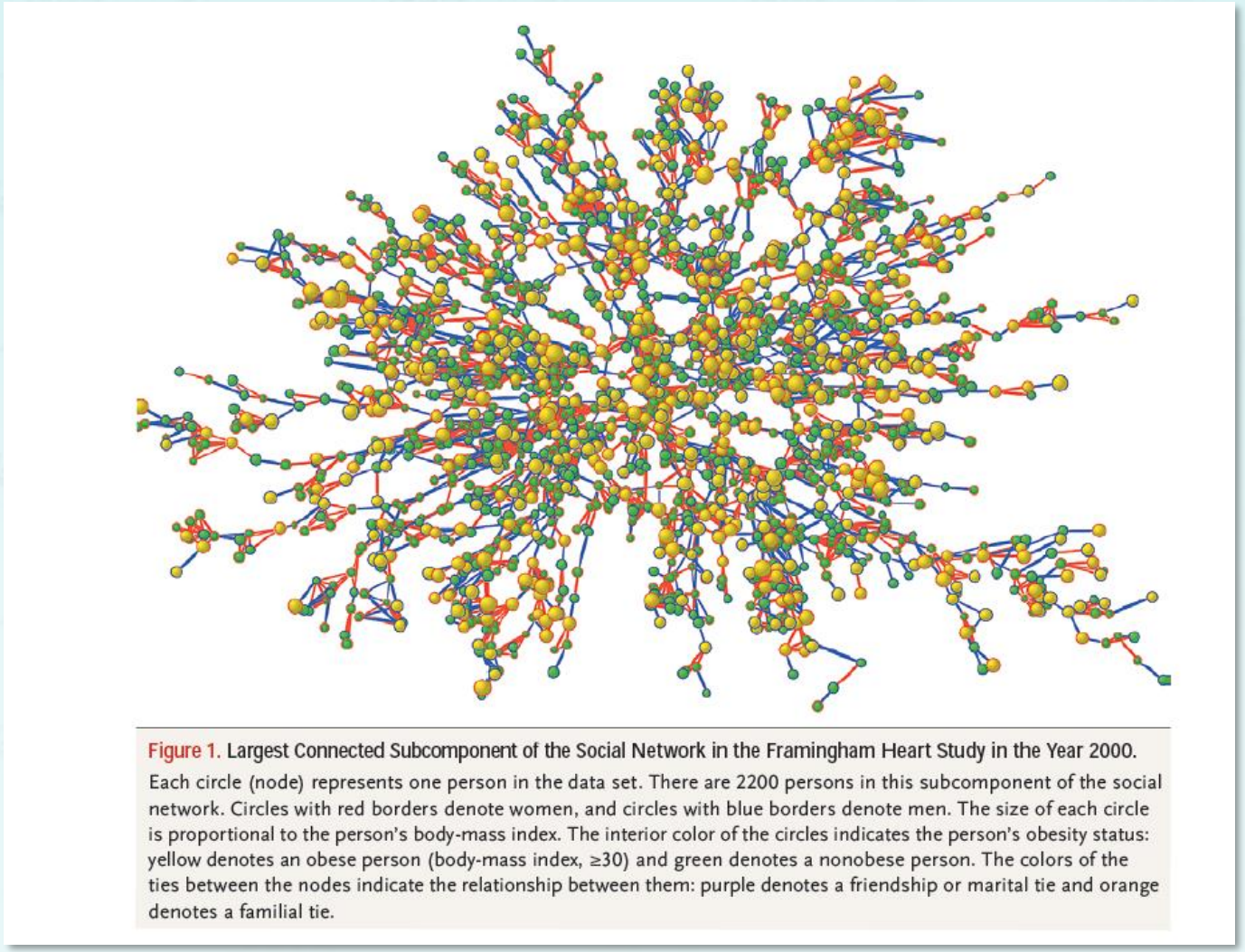


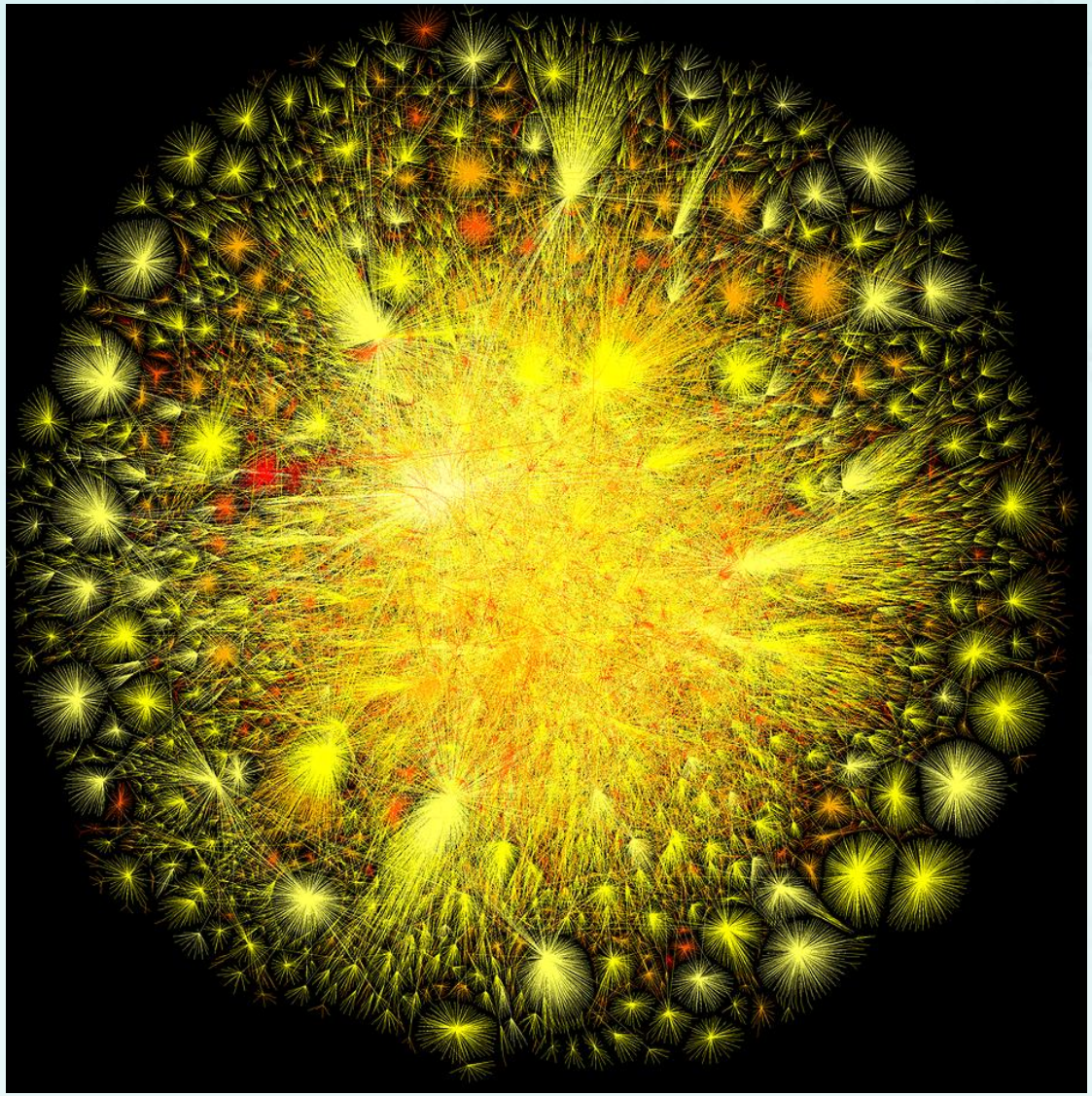
Figure 1. Largest Connected Subcomponent of the Social Network in the Framingham Heart Study in the Year 2000.
Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of the circles indicates the person's obesity status: yellow denotes an obese person (body-mass index, ≥30) and green denotes a nonobese person. The colors of the ties between the nodes indicate the relationship between them: purple denotes a friendship or marital tie and orange denotes a familial tie.

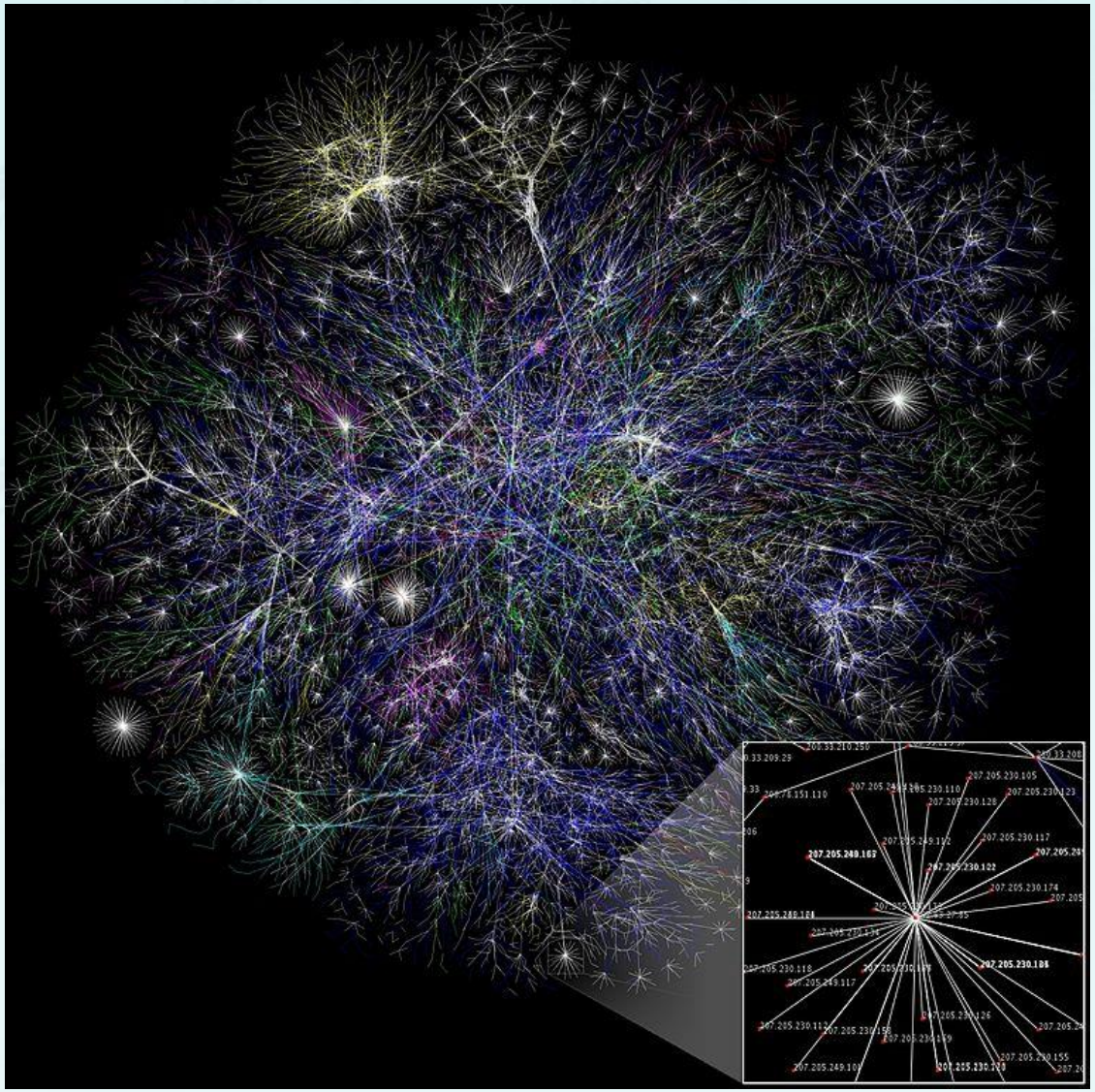
http://www.youtube.com/watch?v=pJfq-o5nZQ4

http://www.nejm.org/doi/full/10.1056/NEJMsa066082

Undirected graphs

- **the Opte Project:** Visualization of the various routes through a portion of the Internet





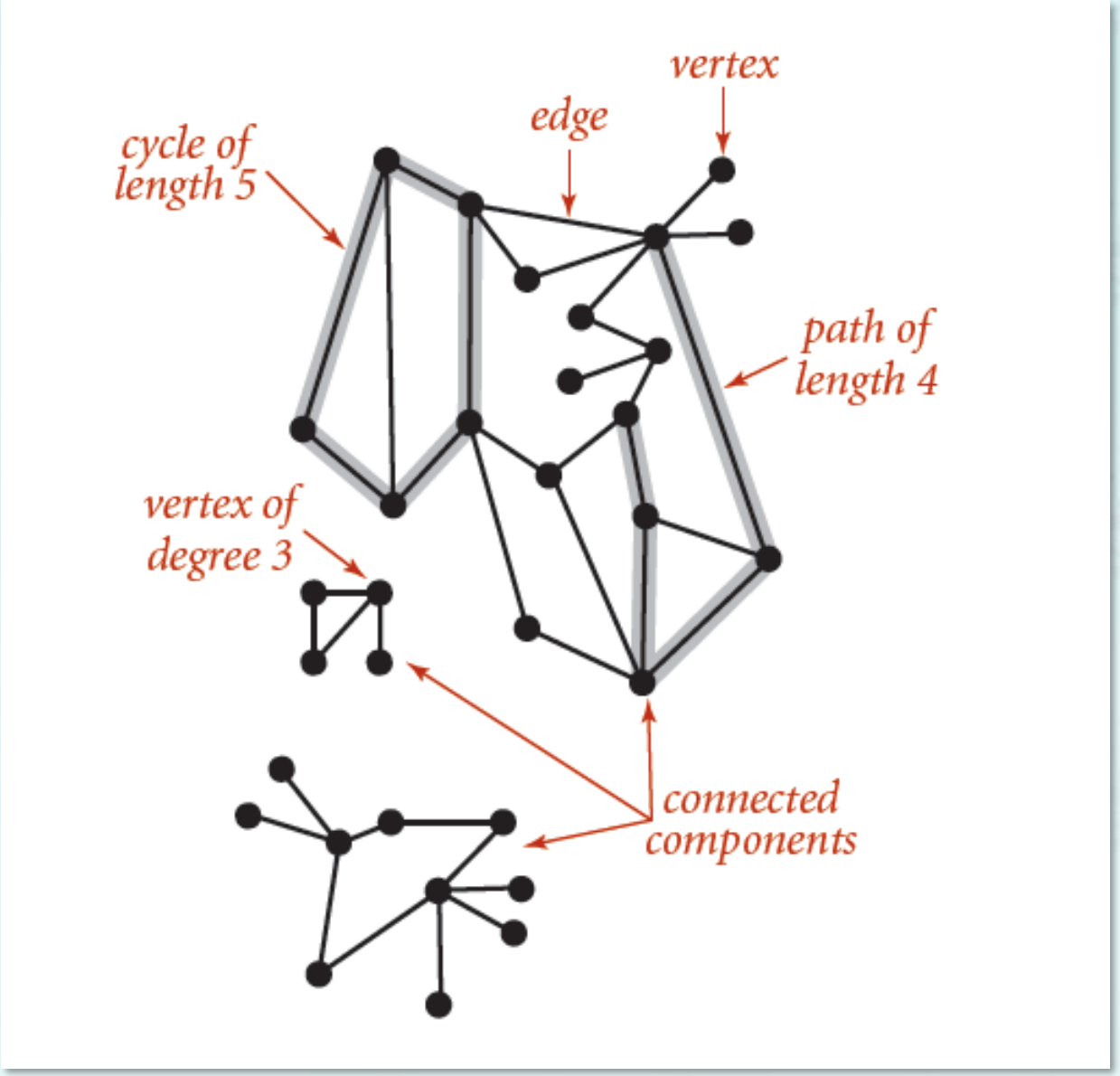
Undirected graphs

- "Visualizing Friendships" by Paul Butler – an intern at Facebook

Graph Applications

graph	vertex	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	street intersection, airport	highway, airway route
internet	class C network	connection
social relationship	person, actor	friendship, movie cast
neural network	neuron	synapse
protein network	protein	protein-protein interaction
molecule	atom	bond

Graph Terminology



Graph Terminology

- **Path:** Sequence of vertices connected by edges.
- **Cycle:** Path whose first and last vertices are the same.
- Two vertices are connected if there is a path between them.
- **Cycle** Is there a cycle in the graph?
- **Euler path** Is there a path that uses each edge exactly once?
- **Hamilton cycle** Is there a cycle that uses each vertex exactly once?

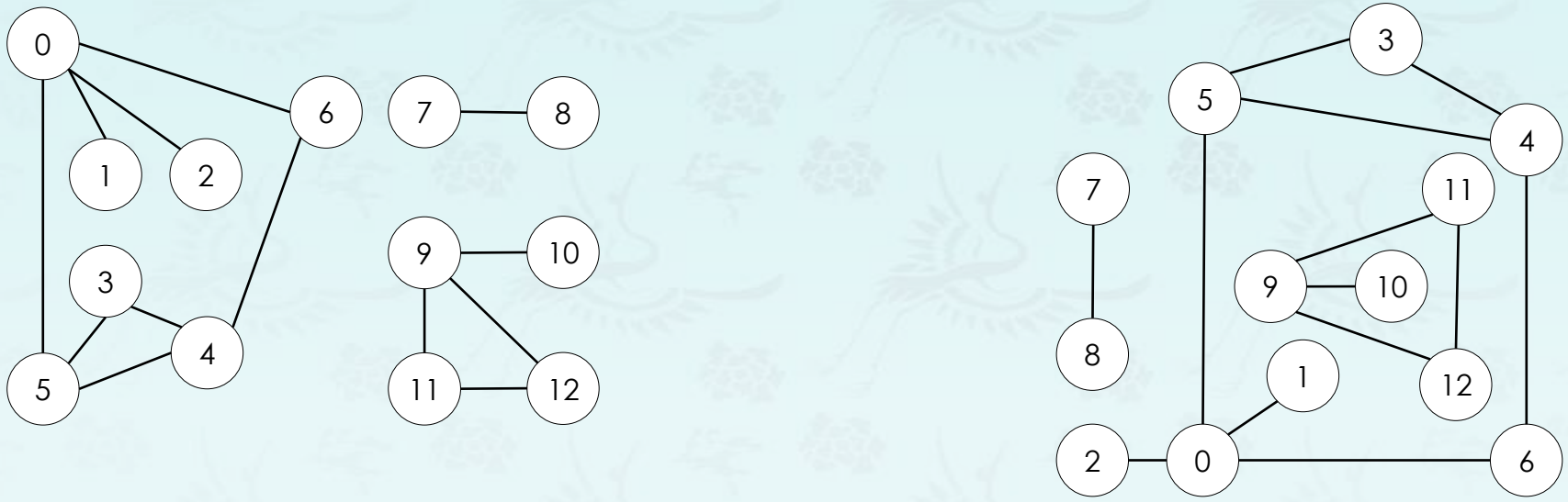
- **Connectivity** Is there a way to connect all of the vertices?
- **MST** What is the best way to connect all of the vertices?
- **BiConnectivity** Is there a vertex whose removal disconnects the graph?

- **Planarity** Can you draw the graph in the plane with no crossing edges
- **Graph isomorphism** Do two adjacency lists represent the same graph?

- **Challenge** Which of these problems are easy? difficult? intractable?

Graph Representation

- **Graph drawing.** Provides intuition about the structure of the graph.

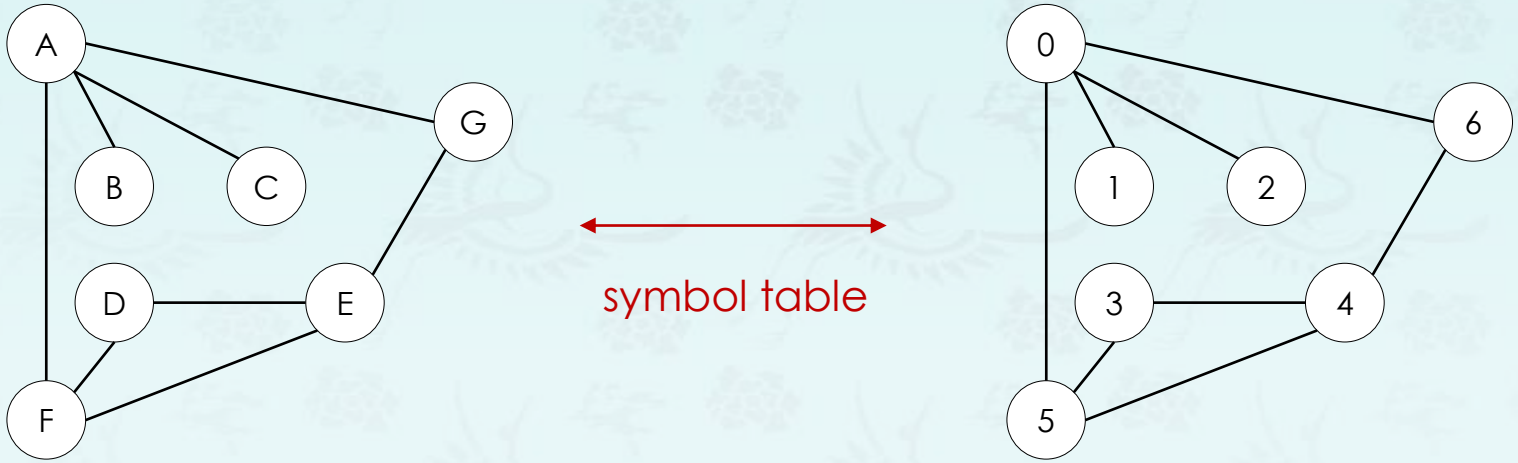


two drawings of the same graph

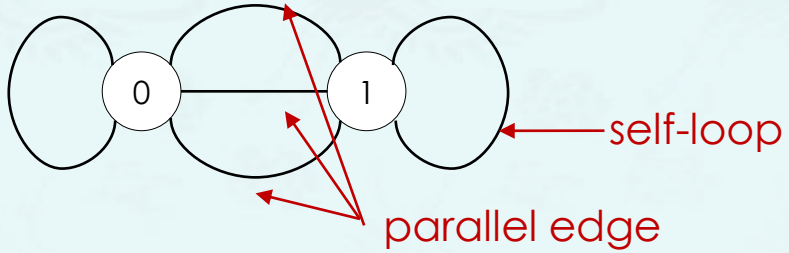
Graph Representation

Vertex representation.

- We use integers between **0** and ***V* – 1**.
- Applications: convert between names and integers with symbol table.



Anomalies

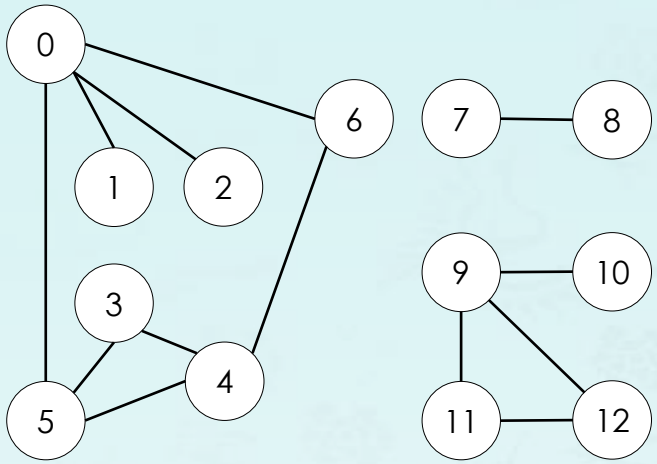


Graph ADT

.

return type	Graph	
	Graph(int V)	*create an empty graph with V vertices*
	Graph(char *fname)	*create a graph from input stream*
void	addEdge(int v, int w)	*add an edge v-w*
vector<Int>	adjacent(int V)	*vertices adjacent to v*
int	V()	*number of vertices*
int	E()	*number of edges*
	toString()	*string representation*

Graph Input Format



graph6cc.txt

```
13
13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

V

E

graph.cpp

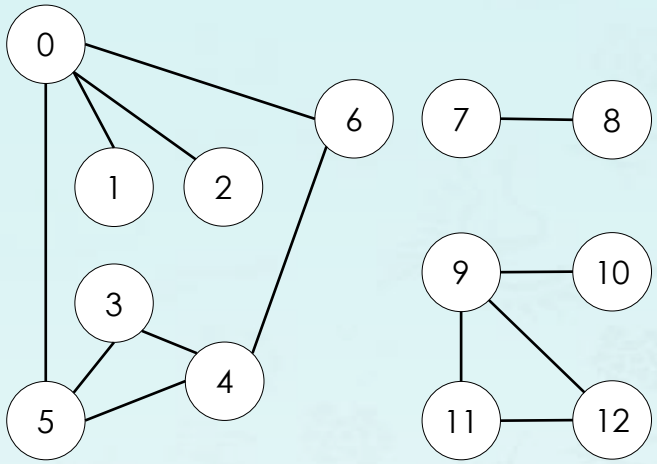
```
Graph g = graph_by_file(argv[1]);

for (int v = 0; v < V(g); ++v) {
  cout << "V[" << v << "]: ";
  for (gnode w = g->adj[v].next; w; w = w->next) {
    cout << w->item << " ";
    (w->next == nullptr) ? (cout << endl) : (cout << "-> ");
  }
}
```

read graph from input stream

print out edge list by vertices

Graph Input Format



graph6cc.txt

```
13
13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

V

E

C:₩GitHub₩nowicx₩Debug₩graph.exe

```
Adjacency-list:
V[0]: 6 -> 2 -> 1 -> 5
V[1]: 0
V[2]: 0
V[3]: 5 -> 4
V[4]: 5 -> 6 -> 3
V[5]: 3 -> 4 -> 0
V[6]: 0 -> 4
V[7]: 8
V[8]: 7
V[9]: 11 -> 10 -> 12
V[10]: 9
V[11]: 9 -> 12
V[12]: 11 -> 9

 [0] ----------------[6]          [7]--------[8]
  | \------------[2]  |
  |  \--------[1]     |          [9]--------[10]
  |                  /            | \
  |    [3]       ---/             |   \
  |   /   \     /                 |     \
  |  /     \   /                  |      \
 [5]--------[4]                  [11]----[12]

    vertex[0..12] =   0  1  2  3  4  5  6  7  8  9 10 11 12
     color[0..12] =   0  0  0  0  0  0  0  0  0  0  0  0  0
```

graph.cpp

```
Graph g = graph_by_file(argv[1]);

for (int v = 0; v < V(g); ++v) {
  cout << "V[" << v << "]: ";
  for (gnode w = g->adj[v].next; w; w = w->next) {
    cout << w->item << " ";
    (w->next == nullptr) ? (cout << endl) : (cout << "-> ");
  }
}
```

Graph Coding

Compute the degree of V

```
int degree(graph g, int v) {
  if (!validVertex(g, v)) return -1;
  int deg = 0;
  for (gnode w = g->adj[v].next; w; w = w->next, deg++);
  return deg;
}
```

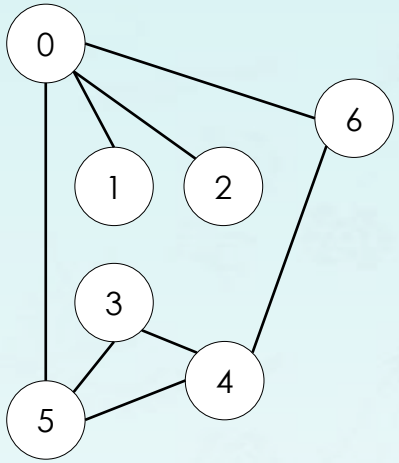
Compute maximum degree

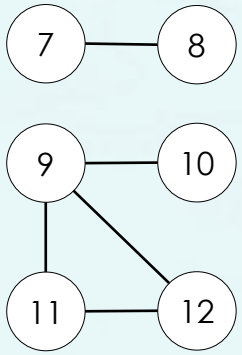
```
int degree(graph g) {
  int max = 0;
  for (int v = 0; v < V(g); ++v) {
    int deg = degree(g, v);
    if (deg > max) max = deg;
  }
  return max;
}
```

Compute average degree

```
double degree_average(graph g) {
  int return 2.0 * E(g) / V(g);
}
```

Graph Coding – edge list





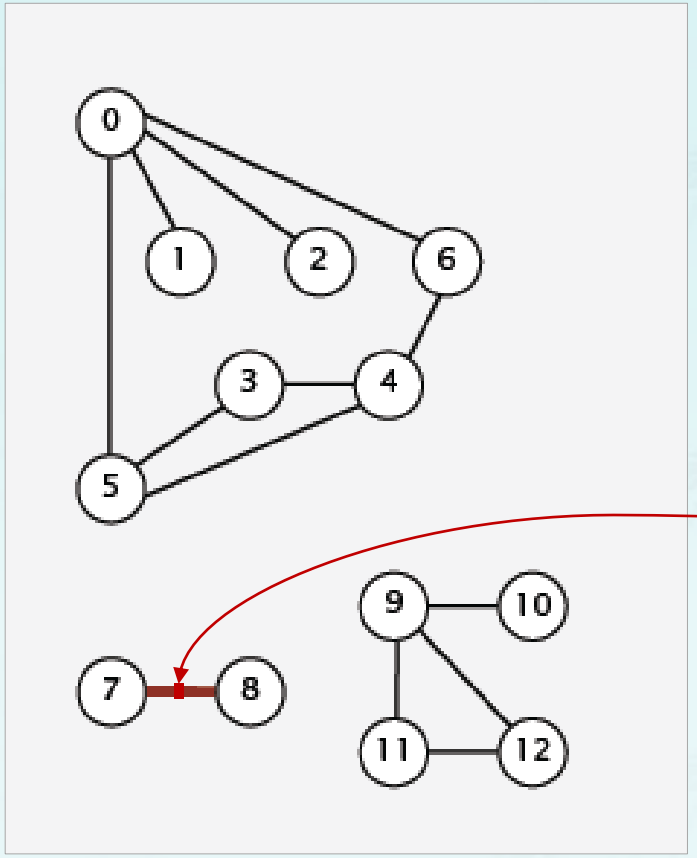
graph6cc.txt

```
13          <- V
13          <- E
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

1. **Edge list**
2. **Adjacency matrix**
3. **Adjacency list**

Graph Coding – edge list

1. Maintain a list of the edges (linked list or array)

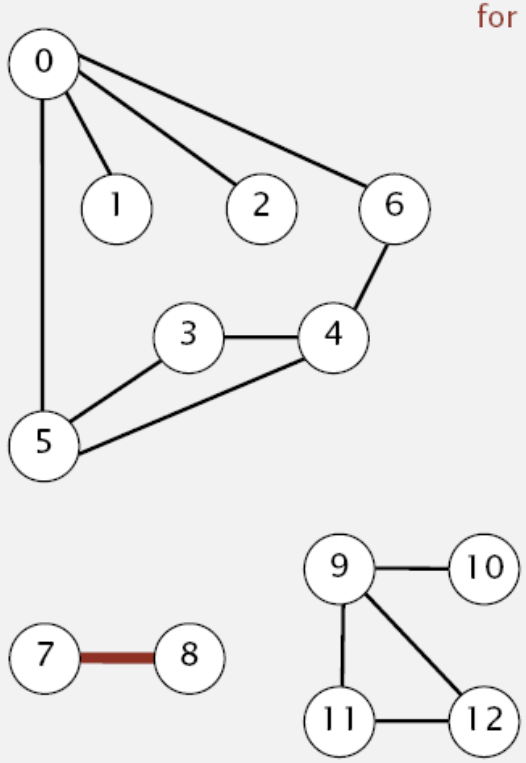


Edge list

```
0 1
0 2
0 5
0 6
3 4
3 5
4 5
4 6
7 8
9 10
9 11
9 12
11 12
```

Graph Coding – Adjacency-matrix 인접행렬

2. Maintain a two-dimensional V-by-V Boolean array;
   for each edge v-w in graph: adj[v][w] = adj[w][v] = true.



two entries for each edge

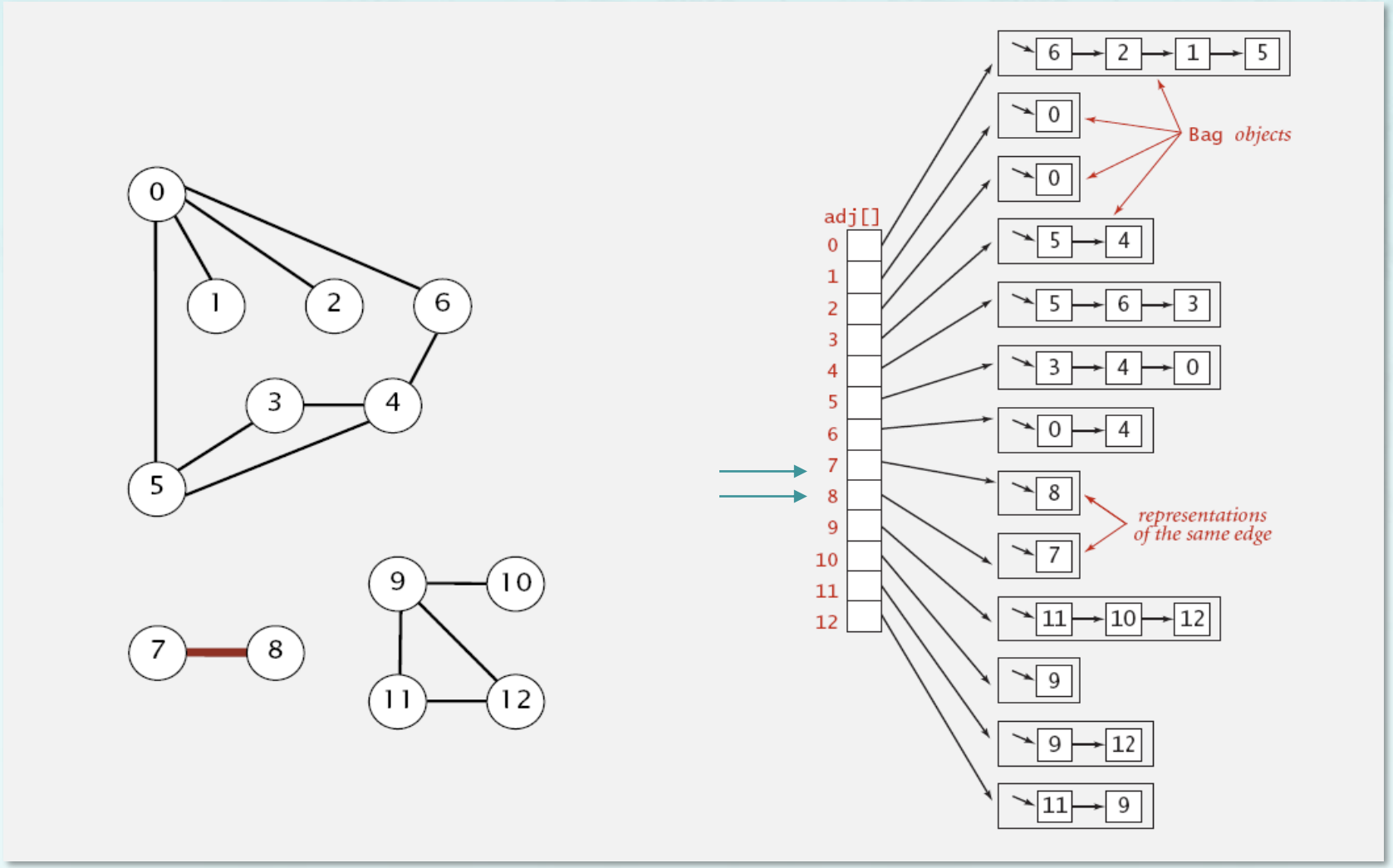
Since it is undirected edge (or graph)

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Graph Coding – Adjacency list 인접리스트

3. Maintain vertex-index array of lists.

use Bag in Java.
use a linked list in C/C++.



Graph Coding – graph.h

```
// a structure to represent an adjacency list of vertices
struct Gnode {
  int    item;
  Gnode* next;
  Gnode (int i, Gnode *p = nullptr) {
    item = i;  next = p;
  }
  ~Gnode() {}
};

using gnode = Gnode *;
```

adjacent vertices using a singly linked list

next vertex to link if any.

Graph Coding – graph.cpp

```
struct Graph {
  int V;              // N vertices
  int E;              // N edges
  gnode adj;          // array of linked lists of vertices
  Graph(int v = 0) { // constructs a graph with v vertices
    V = v;
    E = 0;
    adj = new (nothrow) Gnode[v];
    assert(adj != nullptr);

    for (int  i = 0; i < v; i++) { // initialize adj list as empty;
        adj[i].next = nullptr;
        adj[i].item = i;
    }
  }
  ~Graph() {}
};
using graph = Graph *;
```

set each adj list nullptr to begin with

unused;
but may store the degree of vertex i.

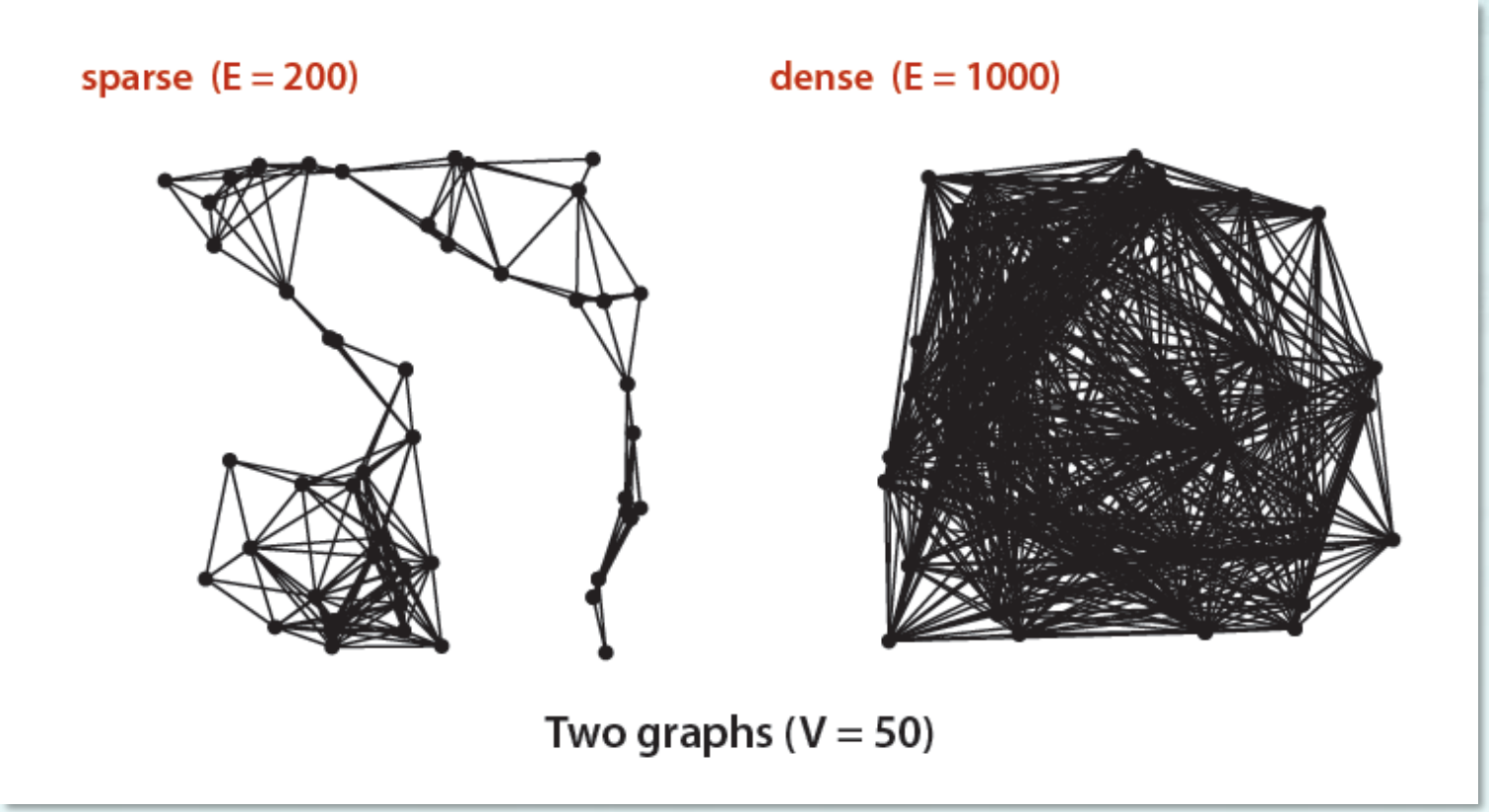
```
graph g = new Graph(v);
for (int i = 0; i < E; i++)
  addEdge(g, from[i], to[i]);
```

Graph Coding – Adjacency-matrix 인접행렬

In practice: Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree



Graph Coding – Adjacency-matrix 인접행렬

In practice: Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1	1	V
adjacency lists	$E + V$	1	$degree(v)$	$degree(v)$

Data Structures
Chapter 7: Graph

1. **Introduction**
   - **Terminology, Representation, ADT**
2. Basic Operations
   - DFS, CC, BFS, Processing
3. Digraph and Applications
4. Minimum Spanning Tree(MST)

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University