ITP20001/ECE20010 Data Structures

Chapter 6 Graph

- ***Introduction***
- ***Graph API***
- *Elementary Graph Operations*
    - *DFS: Depth first search*
    - *BFS: Breadth first search*
    - *CC: Connected components*

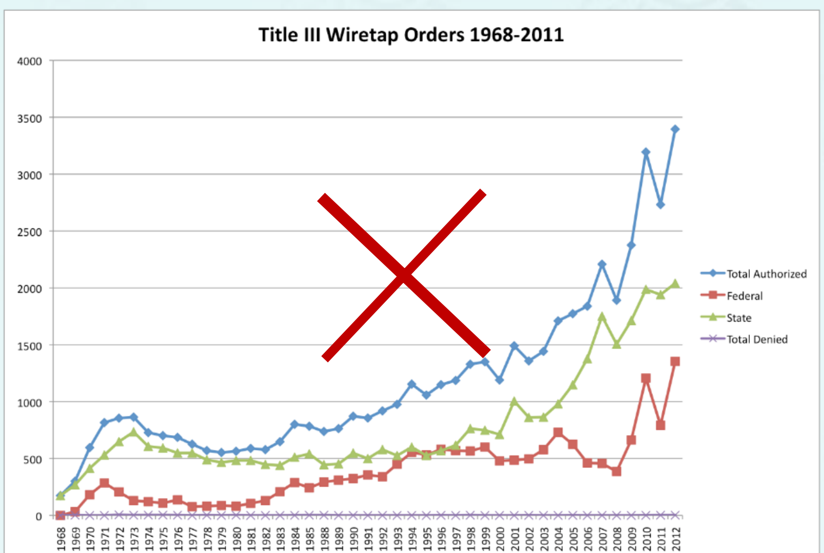
Major references:
1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4th edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet

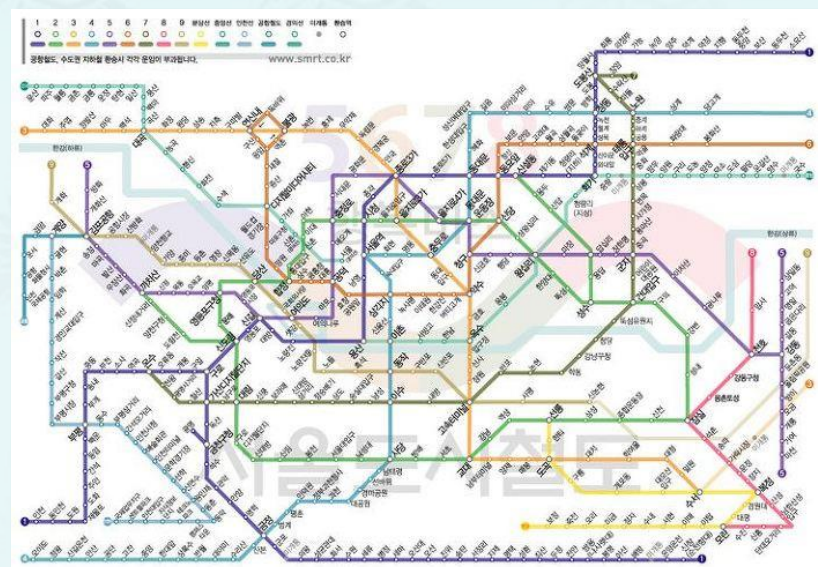
Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

Chapter 6 Undirected graphs

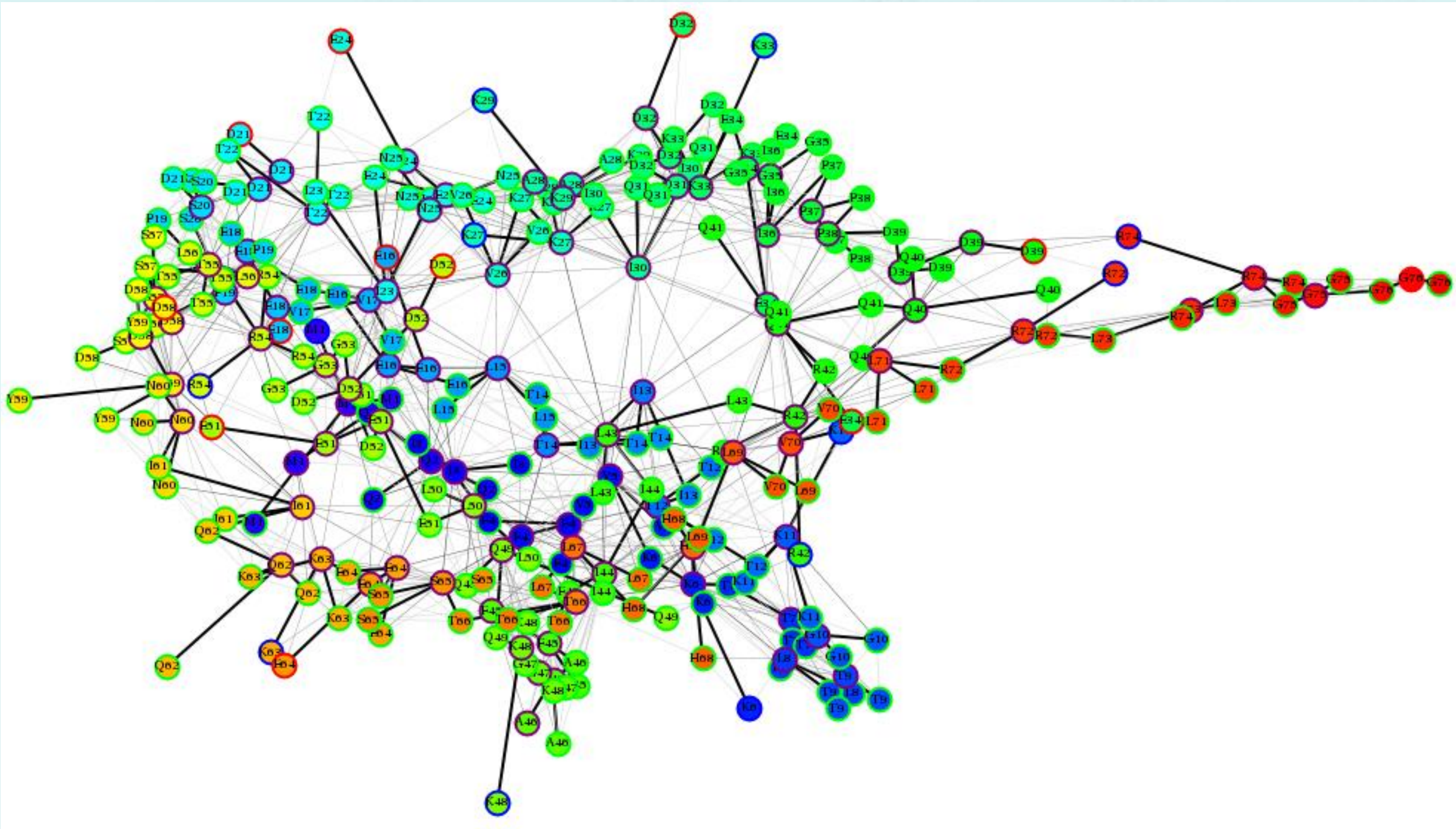
Graph: Set of vertices connected pairwise by edges.

- *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.



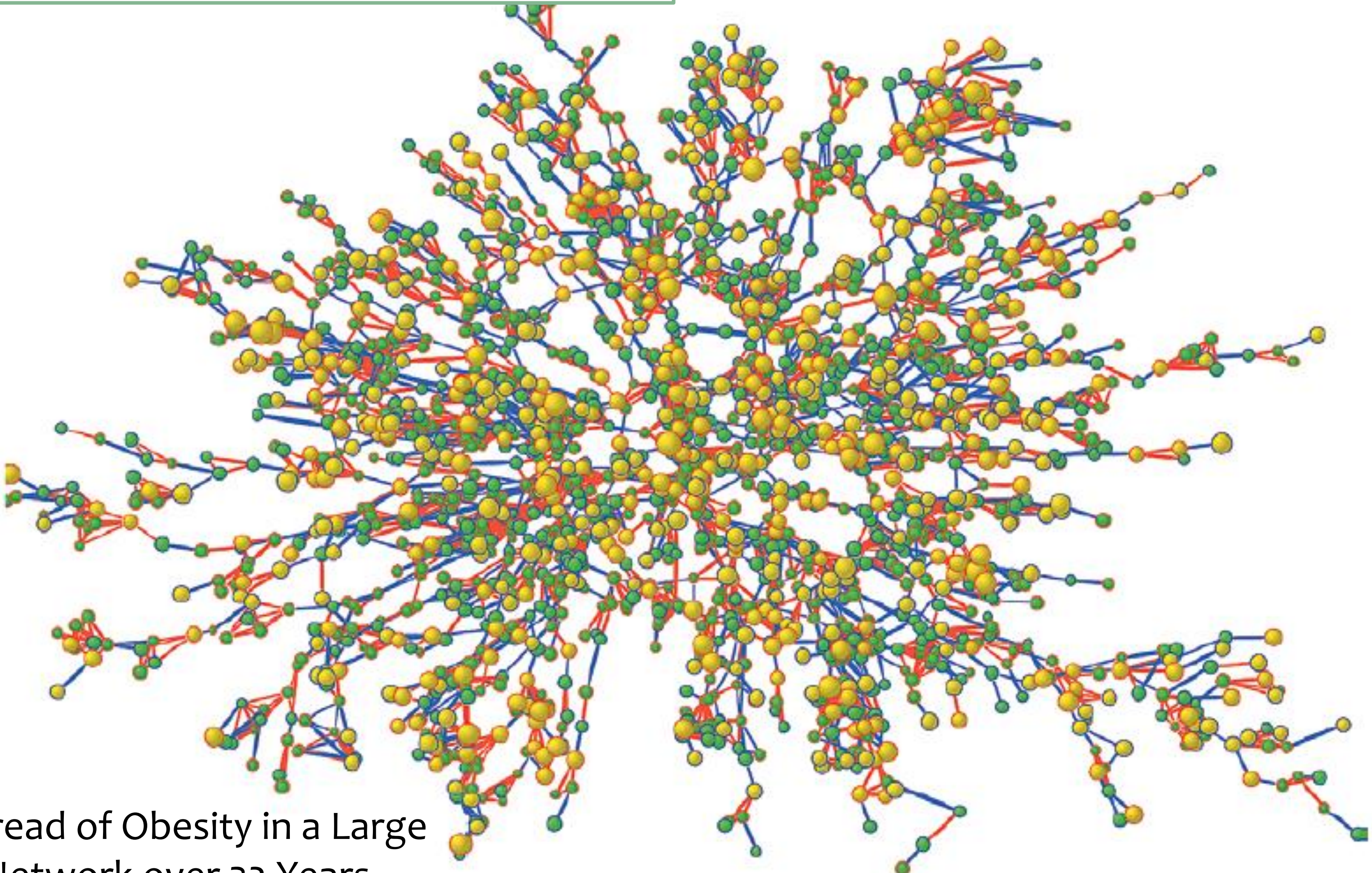


Chemical Environments: Protein Graphs



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.

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



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.

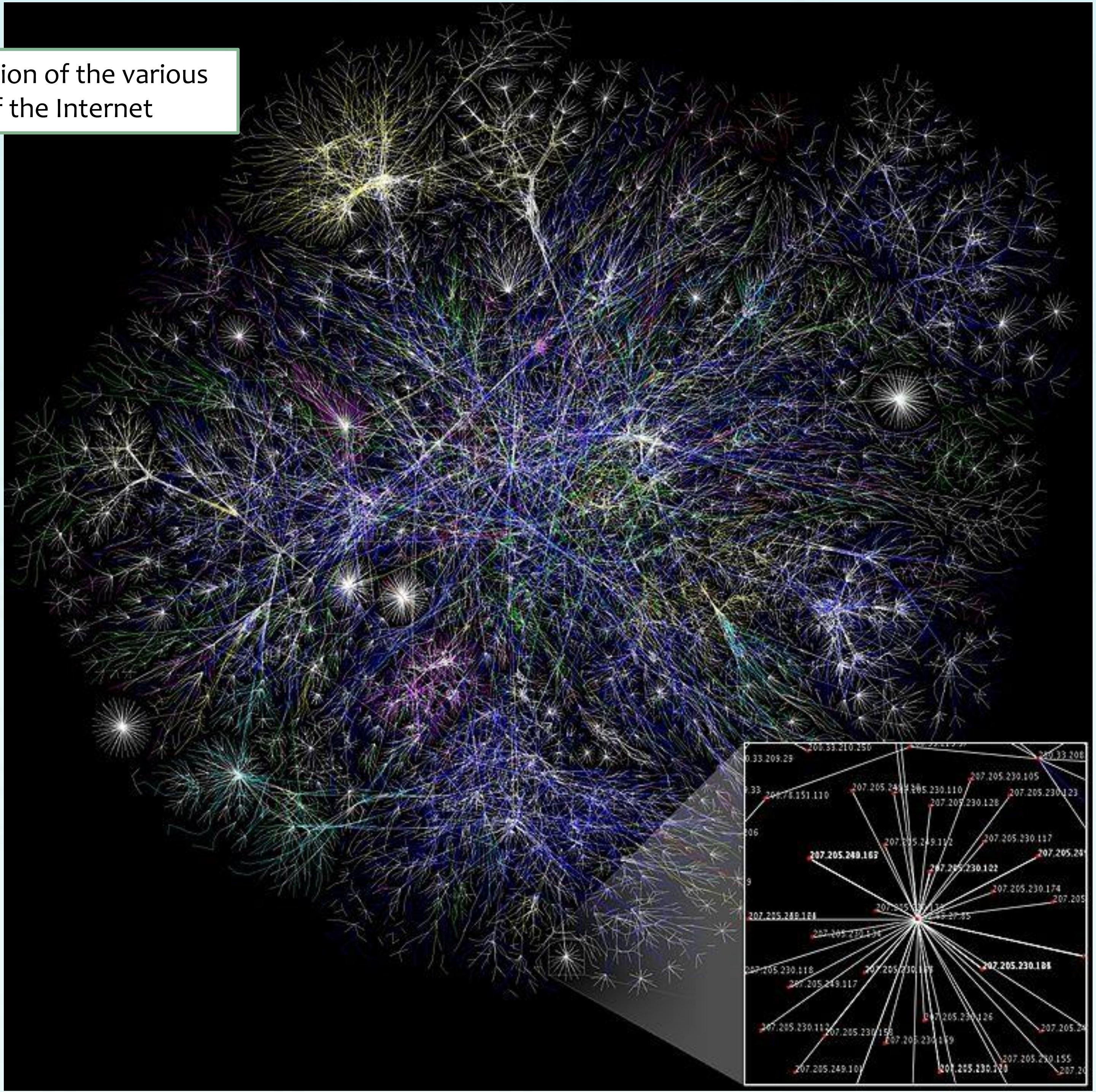
Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of 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.

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

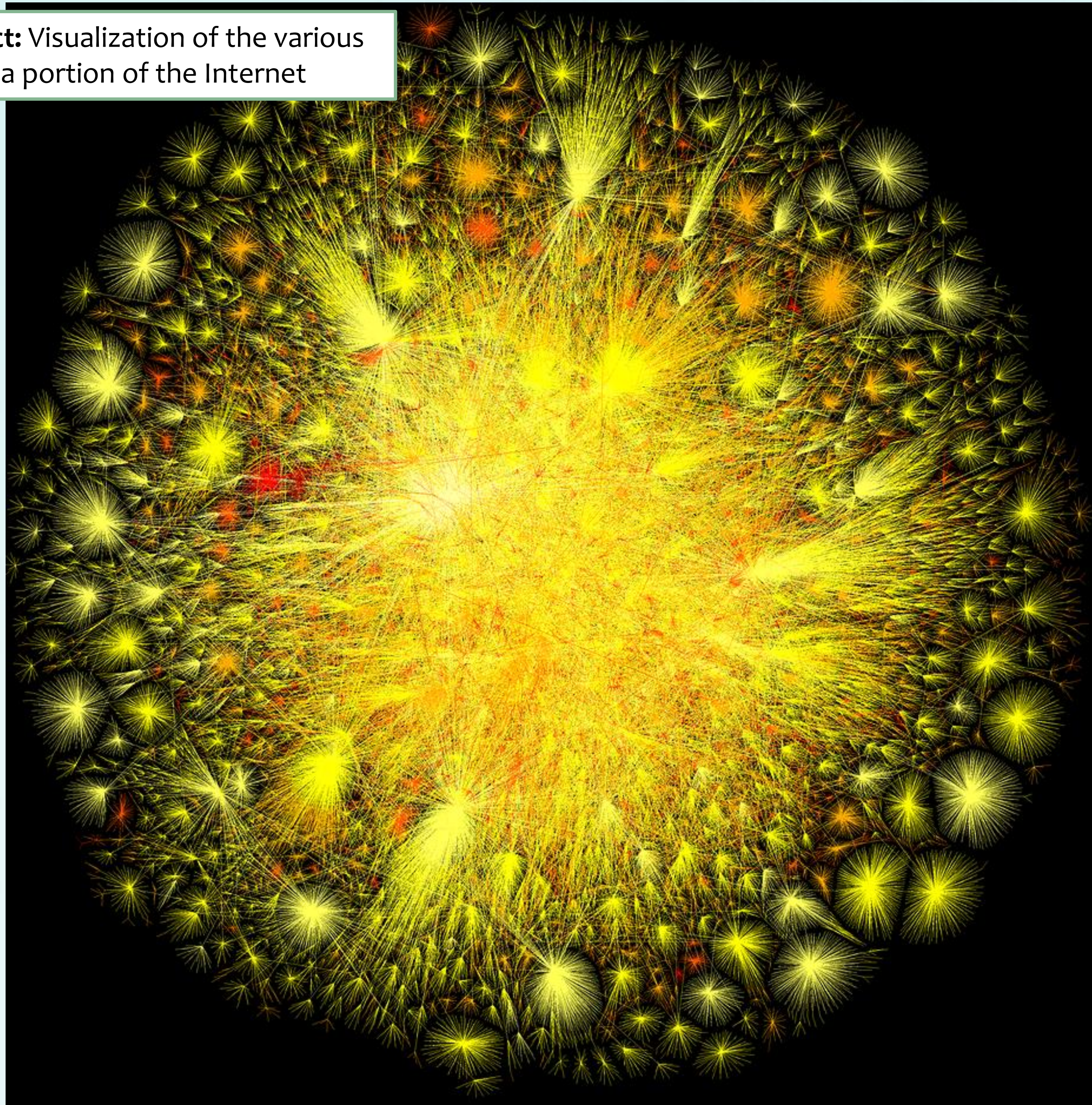
http://www.nejm.org/doi/full/10.1056/NEJMsa066082

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

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



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



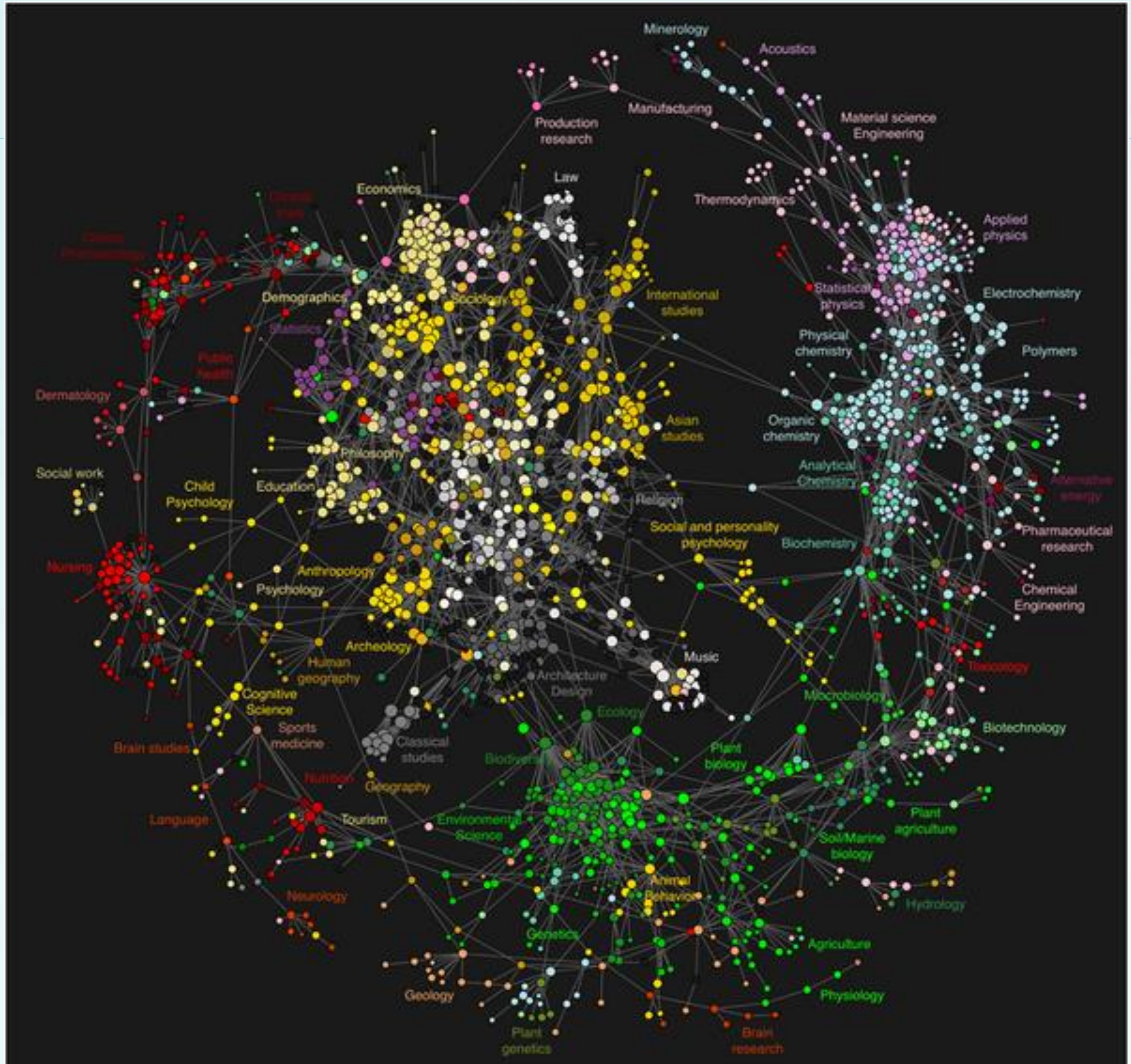


Figure 5. Map of science derived from clickstream data.

Bollen J, Van de Sompel H, Hagberg A, Bettencourt L, et al. (2009) Clickstream Data Yields High-Resolution Maps of Science. http://www.plosone.org/article/info:doi/10.1371/journal.pone.0004803

10 million Facebook friends

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

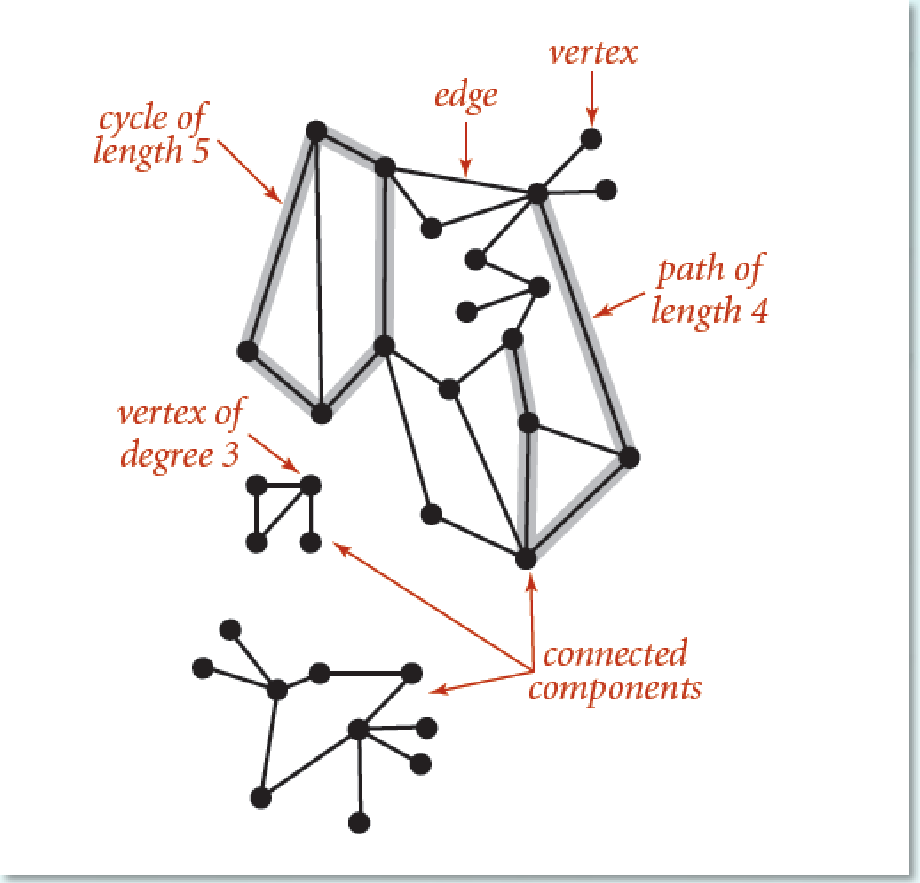
Graph applications

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

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.



Some graph-processing problems

Path.	Is there a path between *s* and *t* ?
Shortest path.	What is the shortest path between *s* and *t* ?
Cycle.	Is there a cycle in the graph?
Euler tour.	Is there a cycle that uses each edge exactly once?
Hamilton tour.	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?

ECE 20010 Data Structures

Data Structures

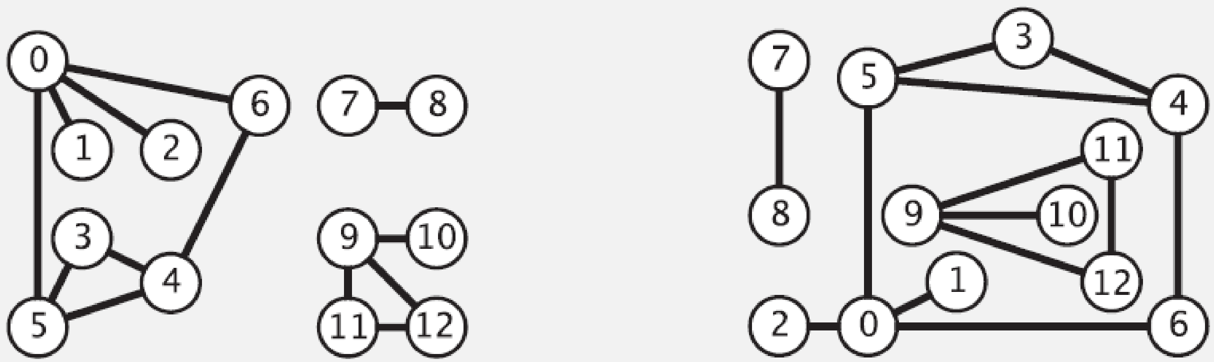
Chapter 6

- *Introduction*
- ***Graph API***
- *Elementary Graph Operations*
  - *DFS: Depth first search*
  - *BFS: Breadth first search*
  - *CC: Connected components*

Major references:
1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4th edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet

Graph representation

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

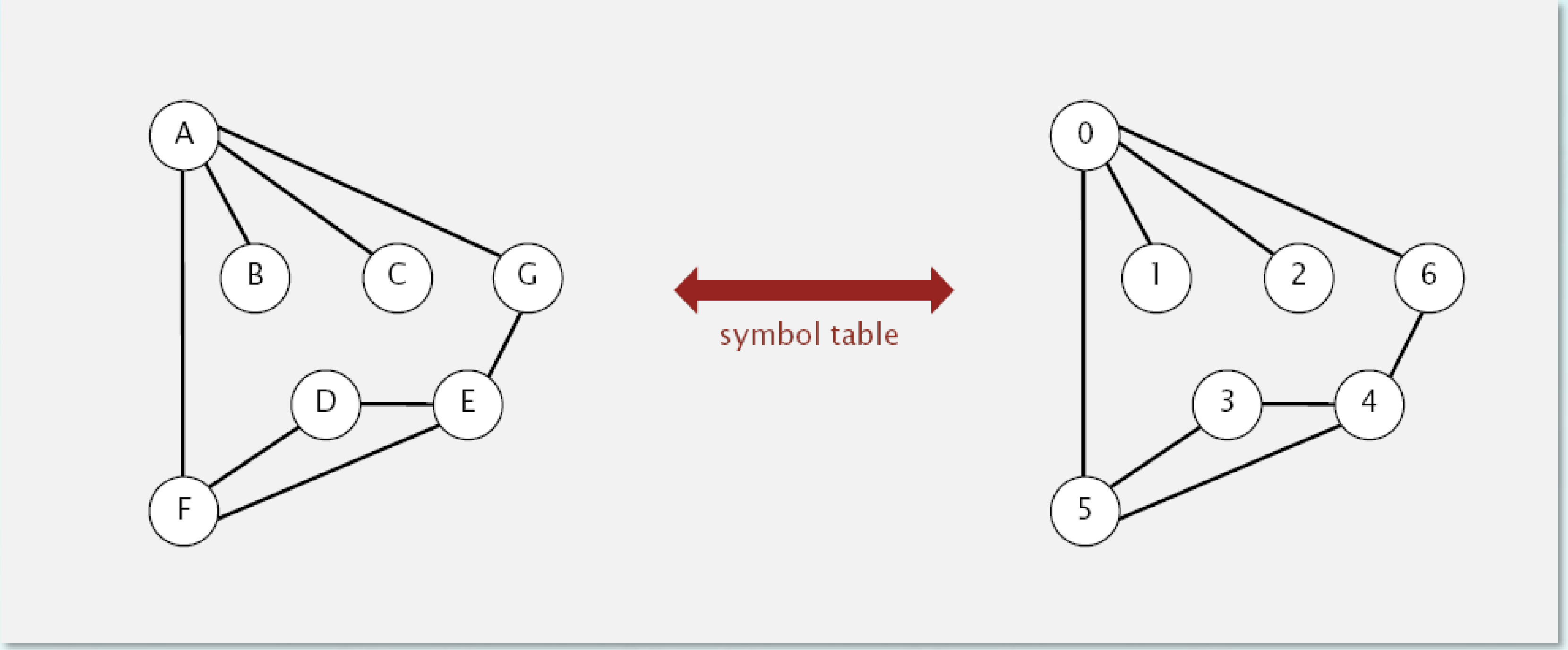


two drawings of the same graph

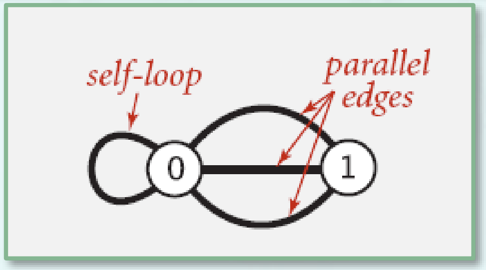
Graph representation

Vertex representation.

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



Anomalies.



Graph ADT - Java

public class	Graph	
	Graph(int V)	*create an empty graph with V vertices*
	Graph(In in)	*create a graph from input stream*
void	addEdge(int v, int w)	*add an edge v-w*
Iterable<Integer>	adj(int V)	*vertices adjacent to v*
int	V()	*number of vertices*
int	E()	*number of edges*
	toString()	*string representation*

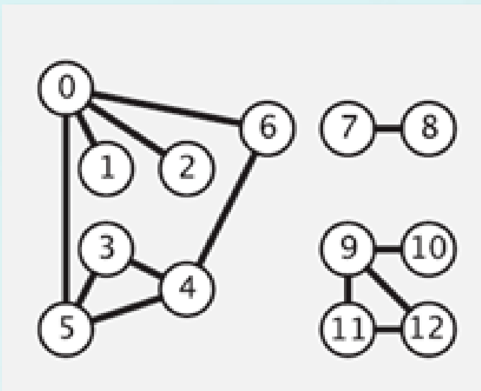
```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
  for (int w: G.adj(v))
    StdOut.println(v + “-” + w);
```

read graph from input stream

print out each edge (twice)

Notice that this prints v-w edge twice for undirected graph.

Graph input format



myG.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

$java myG myG.txt

```
0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
…
12-11
12-9
```

myG.java

```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
  for (int w: G.adj(v))
    StdOut.println(v + "-" + w);
```

read graph from input stream

print out each edge (twice)

Typical graph-processing code

Compute the degree of V

```
public static int degree(Graph G, int v)
{
   int degree = 0;
   for (int w : G.adj(v)) degree++;
   return degree;
}
```

Compute maximum degree

```
public static int maxDegree(Graph G)
{
   int max = 0;
   for (int v = 0; v < G.V(); v++)
      if (degree(G, v) > max)
         max = degree(G, v);
   return max;
}
```

Compute average degree

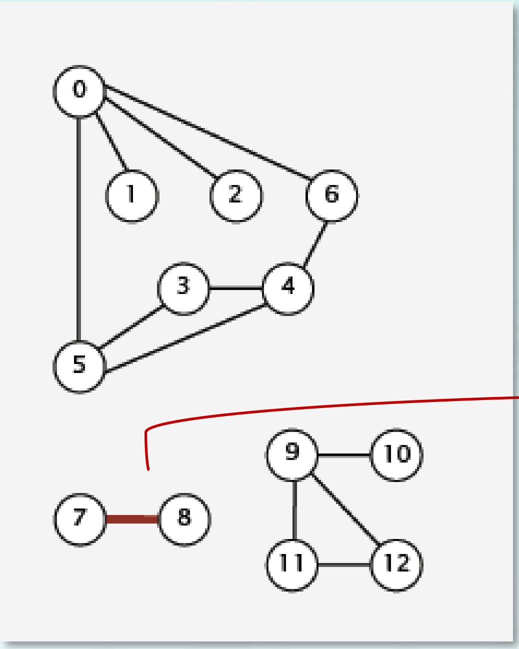
```
public static double averageDegree(Graph G)
{  return 2.0 * G.E() / G.V();  }
```

Count self-loop

```
public static int numberOfSelfLoops(Graph G)
{
   int count = 0;
   for (int v = 0; v < G.V(); v++)
      for (int w : G.adj(v))
         if (v == w) count++;
   return count/2;   // each edge counted twice
}
```

How to implement? Set-of-edges graph representation

- Maintain a list of the edges (linked list or array)

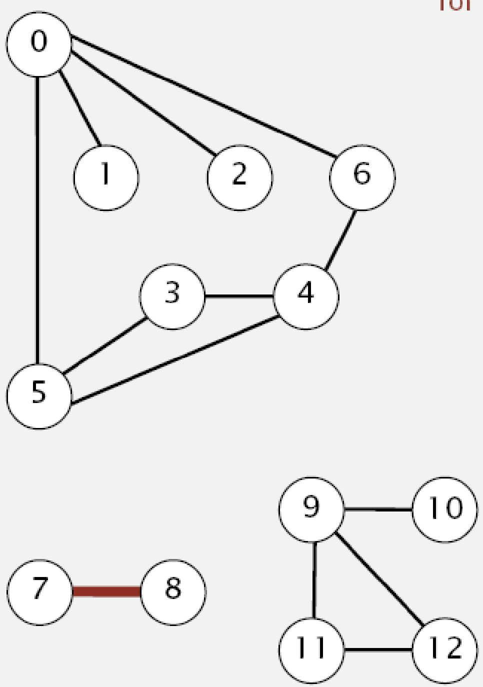


Edge list

```
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
```

How to implement? Adjacency-matrix graph representation 인접행렬

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



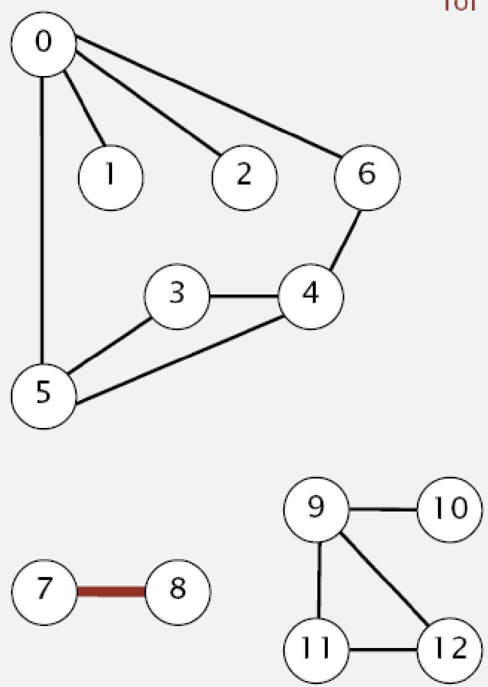
two entries
for each edge

Why two entries?

	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

How to implement? Adjacency-matrix graph representation 인접행렬

- 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

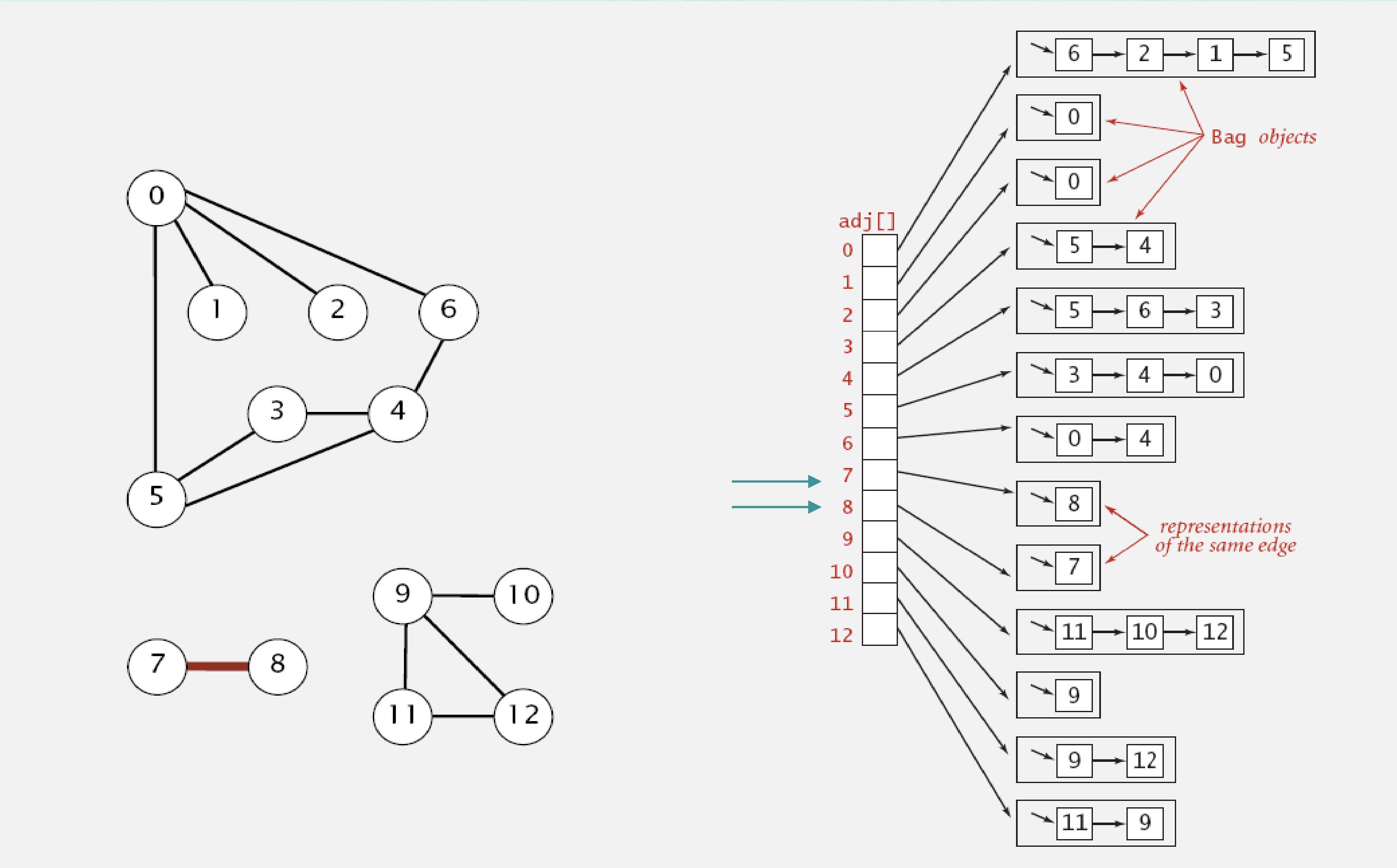
Since it is undirected edge (or graph)

	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

How to implement? Adjacency-list graph representation 인접리스트

- Maintain vertex-index array of lists.

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



Adjacency-list graph representation: C implementation 인접리스트

```
// graph.h

// a structure to represent an adjacency list node
typedef struct GNode *pGNode;
typedef struct GNode {
  int         item;
  pGNode   next;
} GNode;
```

adjacency list nodes (using a linked list)

```
// a structure to represent a graph.
// a graph is an array of adjacency lists.
// size of will be V (number of vertices in graph)

typedef struct Graph *pGraph;
typedef struct Graph {
  int          V;           // number of vertices in the graph
  int          E;           // number of edges in the graph
  pGNode adj;           // an array of adjacency lists
} Graph;
```

graph representation

adjacency list (using an array)

create empty graph with V vertices

add edge v-w (parallel edges and self-loops allowed)

iterator for vertices adjacent to v

Adjacency-list graph representation: C implementation

인접리스트

```
// create a new adjacency list node

pGNode newGNode(int item) {
   pGNode node = (pGNode)malloc(sizeof(GNode));
   assert(node != NULL);
   node->item = item;
   node->next = NULL;
   return node;
}
```

```
// a structure to represent an adjacency list node
typedef struct GNode *pGNode;
typedef struct GNode {
  int          item;
  pGNode   next;
} GNode;
```

Adjacency-list graph representation: C implementation 인접리스트

create an empty graph with V vertices

```
pGraph  newGraph(int V) {
   pGraph  g = (pGraph) malloc(sizeof(Graph));
   assert(g != NULL) ;
   g->V = V;
   g->E = 0;

   // create an array of adjacency list. size of array will be V
   g->adj = (pGNode)malloc(V * sizeof(GNode));
   assert(g->adj != NULL);

   // initialize each adjacency list as empty by making head as NULL;

   for (int  i = 0; i < V; i++)
      g->adj[i].next = NULL;
      g->adj[i].item = i
   return g;
}
```

```
typedef struct Graph *pGraph;
typedef struct Graph {
  int        V;        // num of vertices in G
  int        E;        // num of edges G
  pGNode     adj;      // an array of adj lists
} Graph;
```

adjacency list
(using an array)

adjacency list
set head node NULL

unused; but may store the size of degree.

Adjacency-list graph representation: C implementation

인접리스트

```
// add an edge to an undirected graph

void addEdgeUniDirection(pGraph g, int v, int w) {
   // add an edge from v to w.
   // A new node is added to the adjacency list of v.
   // The node is added at the beginning

   pGNode node = newGNode(w);
   node->next = g->adj[v].next;
   g->adj[v].next = node;
}

// add an edge to an undirected graph

void addEdge(pGraph g, int v, int w) {
   addEdgeUniDirection(g, v, w);          // add an edge from v to w.
   addEdgeUniDirection(g, w, v);          // if graph is undirected, add both
}
```

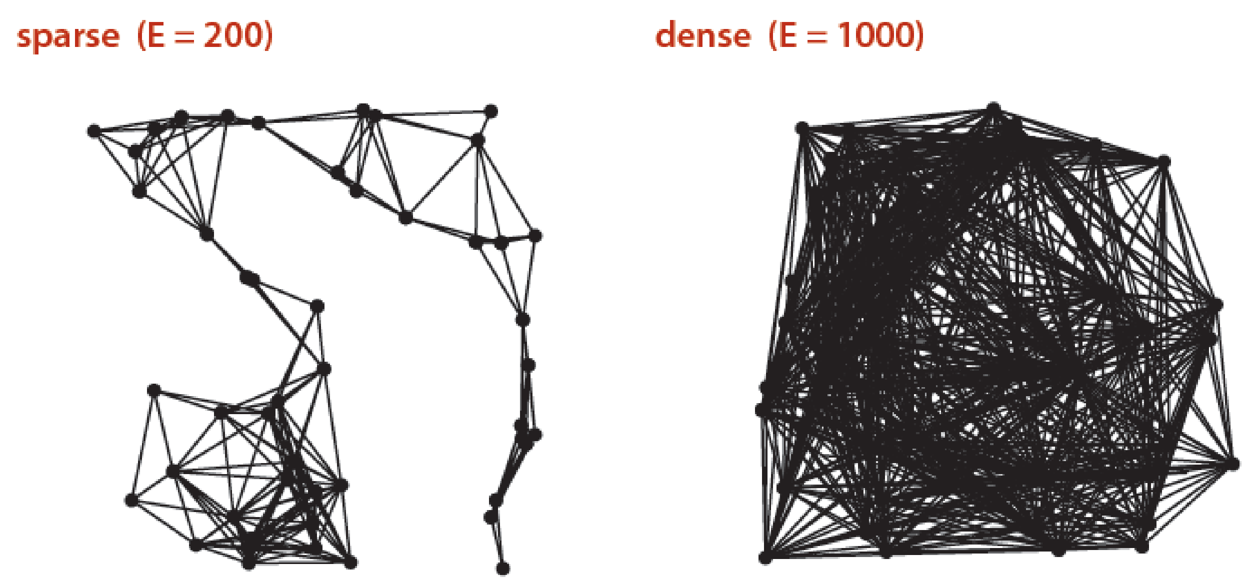
instantiate a node w insert it **at the front** of adjacency list[v]

add an edge for undirected graph

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



Two graphs (V = 50)

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	$degree(v)$	$degree(v)$

ITP20001/ECE20010 Data Structures

Chapter 6

- ***Introduction***
- ***Graph API***
- *Elementary Graph Operations*
  - *DFS: Depth first search*
  - *BFS: Breadth first search*
  - *CC: Connected components*

Major references:

1. Fundamentals of Data Structures by Horowitz, Sahni, Anderson-Freed,
2. Algorithms 4th edition - Part 1 & Part 2 by Robert Sedgewick and Kevin Wayne
3. Wikipedia and many resources available from internet