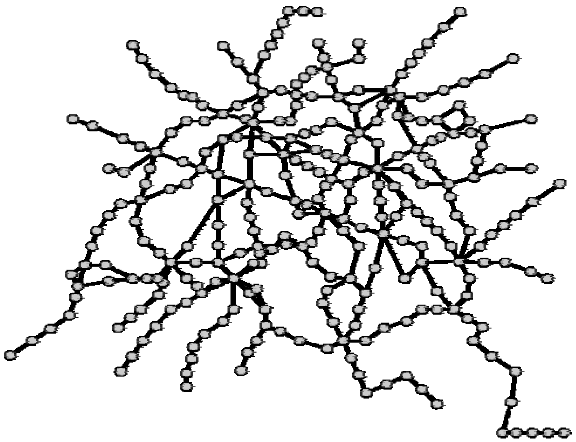
# Undirected Graphs

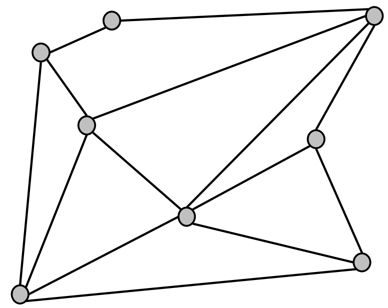


Some of these lecture slides are adapted from material in:
- *Algorithms in C, Part 5*, R. Sedgwick.

## Undirected Graphs

**GRAPH. Set of OBJECTS with pairwise CONNECTIONS.**
- **Interesting and broadly useful abstraction.**

**Why study graph algorithms?**
- **Challenging branch of computer science and discrete math.**
- **Hundreds of graph algorithms known.**
- **Thousands of practical applications.**





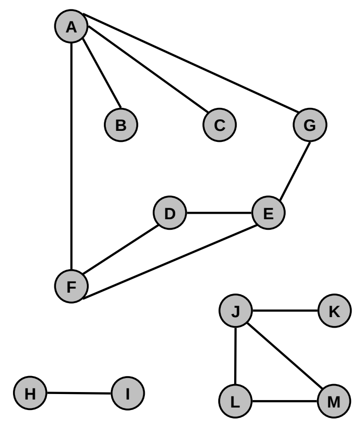
## Graphs

| Graph | Vertices | Edges |
|---|---|---|
| communication | telephone exchanges, computers, satellites | cables, fiber optics, microwave relays |
| circuits | gates, registers, processors | wires |
| mechanical | joints | rods, beams, springs |
| hydraulic | reservoirs, pumping stations | pipelines |
| financial | stocks, currency | transactions |
| transportation | street intersections, airports | highways, airway routes |
| scheduling | tasks | precedence constraints |
| software systems | functions | function calls |
| internet | web pages | hyperlinks |
| games | board positions | legal moves |
| social relationship | people, actors | friendships, movie casts |



## Graph Jargon

**Terminology.**
- **Vertex: `v`.**
- **Edge: `e = v-w`.**
- **Graph: `G`.**
- **`V` vertices, `E` edges.**
- **Parallel edge, self loop.**
- **Directed, undirected.**
- **Sparse, dense.**
- **Path.**
- **Cycle, tour.**
- **Tree, forest.**
- **Connected, connected component.**



# A Few Graph Problems

PATH. Is there a path from s to t?
SHORTEST PATH. What is the shortest path between two vertices?
LONGEST PATH. What is the longest path between two vertices?

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?
BI-CONNECTIVITY. Is there a vertex whose removal disconnects graph?

PLANARITY. Can graph be drawn in plane with no crossing edges?
ISOMORPHISM. Do two adjacency matrices represent the same graph?



# Graph ADT in C

Standard method to separate clients from implementation.
- Opaque pointer to `Graph` ADT.
- Plus simple `typedef` for `Edge`.

**GRAPH.h**

```
typedef struct graph *Graph;
typedef struct { int v, w; } Edge;
Edge EDGEinit(int v, int w);

Graph GRAPHinit(int V);
Graph GRAPHrand(int V, int E);
void  GRAPHdestroy(Graph G);
void  GRAPHshow(Graph G);
void  GRAPHinsertE(Graph G, Edge e);
void  GRAPHremoveE(Graph G, Edge e);
int   GRAPHcc(Graph G);
int   GRAPHisplanar(Graph G);
. . .
```



# Graph ADT in C

Typical client program.
- Call `GRAPHinit()` or `GRAPHrand()` to create instance.
- Uses `Graph` handle as argument to ADT functions.
- Calls ADT function to do graph processing.

**client.c**

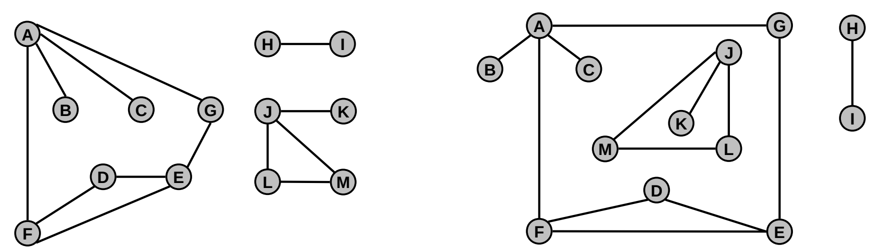
```
#include <stdio.h>
#include "GRAPH.h"

int main(int argc, char *argv[]) {
   int V = atoi(argv[1]);
   int E = atoi(argv[2]);
   Graph G = GRAPHrand(V, E);
   GRAPHshow(G);
   printf("%d component(s)\n", GRAPHcc(G));
   return 0;
}
```



# Graph Representation

Vertex names. (A B C D E F G H I J K L M)
- C program uses integers between `0` and `V-1`.
- Convert via implicit or explicit symbol table.

Two drawing represent same graph.

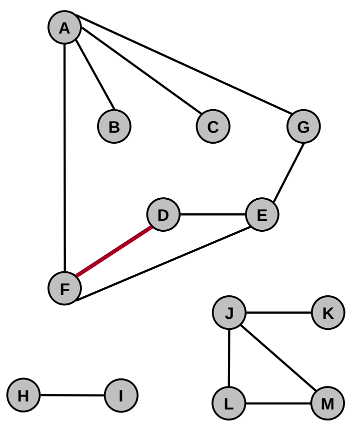



Set of edges representation.
- { A-B, A-G, A-C, L-M, J-M, J-L, J-K, E-D, F-D, H-I, F-E, A-F, G-E }.

# Adjacency Matrix Representation

**Adjacency matrix representation.**

- Two-dimensional V × V array.
- Edge v-w in graph: `adj[v][w] = adj[w][v] = 1`.




| | | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | B | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | C | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | D | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | E | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | F | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | G | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 8 | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | J | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 10 | K | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 11 | L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 12 | M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Adjacency Matrix



# Graph ADT Implementation: Adjacency Matrix

**GRAPH.h**

```
#include <stdlib.h>
#include "GRAPH.h"
struct graph {
   int V;          // # vertices
   int E;          // # edges
   int **adj;      // V × V adjacency matrix
};

Graph GRAPHinit(int V) {
   Graph G = malloc(sizeof *G);
   G->V = V; G->E = 0;
   G->adj = MATRIXinit(V, V, 0);
   return G;
}

void GRAPHinsertE(Graph G, Edge e) {
   int v = e.v, w = e.w;
   if (G->adj[v][w] == 0) G->E++;
   G->adj[v][w] = G->adj[w][v] = 1;
}
```

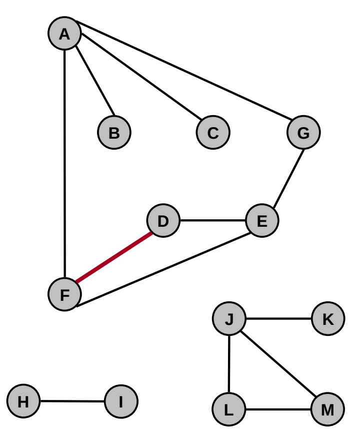
no parallel edges

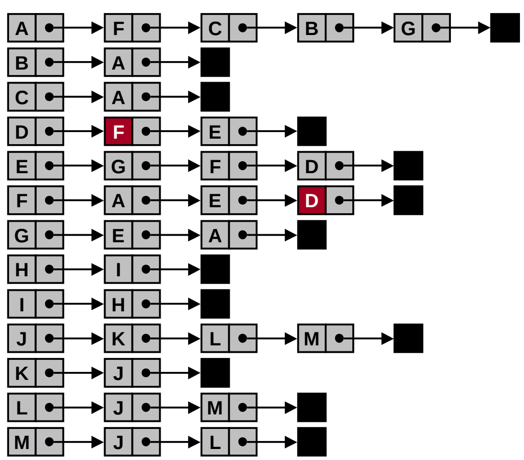


# Adjacency List Representation

**Vertex indexed array of lists.**

- Space proportional to number of edges.
- Two representations of each undirected edge.






# Graph ADT Implementation: Adjacency List

**GRAPH.h**

```
#include "GRAPH.h"

typedef struct node *link;
struct node {
   int v;        // current vertex in adjacency list
   link next;    // next node in adjacency list
};

struct graph {
   int V;        // # vertices
   int E;        // # edges
   link *adj;    // array of V adjacency lists
};

link NEWnode(int v, link next) {
   link x = malloc(sizeof *x);
   x->v = v;
   x->next = next;
   return x;
}
```

# Adjacency List Graph ADT Implementation

**GRAPH.h**

```
// initialize a new graph with V vertices
Graph GRAPHinit(int V) {
   int v;
   Graph G = malloc(sizeof *G);
   G->V = V; G->E = 0;
   G->adj = malloc(V * sizeof(link));
   for (v = 0; v < V; v++) G->adj[v] = NULL;
   return G;
}

// insert an edge e = v-w into Graph G
void GRAPHinsertE(Graph G, Edge e) {
   int v = e.v, w = e.w;
   G->adj[v] = NEWnode(w, G->adj[v]);
   G->adj[w] = NEWnode(v, G->adj[w]);
   G->E++;
}
```



# Graph Representations

Graphs are abstract mathematical objects.

- ADT implementation requires specific representation.
- Efficiency depends on matching algorithms to representations.

| Representation | Space | Edge between v and w? | Edge from v to anywhere? | Enumerate all edges |
|---|---|---|---|---|
| Adjacency matrix | $O(V^2)$ | O(1) | O(V) | $O(V^2)$ |
| Adjacency list | O(E + V) | O(E) | O(1) | O(E + V) |

Most real-world graphs are sparse $\Rightarrow$ adjacency list.



# Graph Search

Goal. Visit every node and edge in Graph.

A solution. Depth-first search.

- To visit a node v:
  - mark it as visited
  - recursively visit all unmarked nodes w adjacent to v
- To traverse a Graph G:
  - initialize all nodes as unmarked
  - visit each unmarked node

Enables direct solution of simple graph problems.

- Connected components.
- Cycles.

Basis for solving difficult graph problems.

- Biconnectivity.
- Planarity.



# Depth First Search: Connected Components

**Depth First Search**

```
#define UNMARKED -1
static int mark[MAXV];

// traverse component of graph
int GRAPHcc(Graph G) {
    int v, id = 0;
    // initialize all nodes as unmarked
    for (v = 0; v < G->V; v++) mark[v] = UNMARKED;
    // visit each unmarked node
    for (v = 0; v < G->V; v++)
      if (mark[v] == UNMARKED) dfsR(G, v, id++);
    return id;
}

// return 1 if s and t in same connected component
int GRAPHconnect(int s, int t) {
    return mark[s] == mark[t];
}
```

# Depth First Search: Connected Components

**Depth First Search: Adjacency Matrix**

```
void dfsR(Graph G, int v, int id) {
   int w;
   mark[v] = id;
   for (w = 0; w < G->V; w++)
      if (G->adj[v][w] != 0 && mark[w] == UNMARKED)
         dfsR(G, w, id);
}
```

**Depth First Search: Adjacency List**

```
void dfsR(Graph G, int v, int id) {
   link t;
   int w;
   mark[v] = id;

   // iterate over all nodes w adjacent to v
   for (t = G->adj[v]; t != NULL; t = t->next) {
      w = t->v;
      if (mark[w] == UMARKED) dfsR(G, w, id);
   }
}
```



# Connected Components

PATHS. Is there a path from s to t?

| Method | Preprocess | Query | Space |
|---|---|---|---|
| Union Find | O(E log* V) | O(log* V) | O(V) |
| DFS | O(E + V) | O(1) | O(V) |

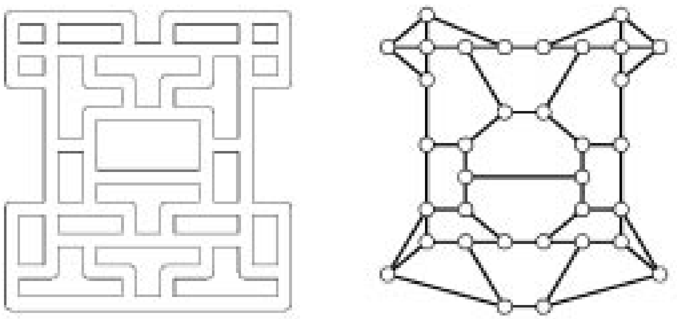
UF advantage.
- Dynamic: can intermix query and edge insertion.

DFS advantage.
- Can get path itself in same running time.
  - maintain parent-link representation of tree
  - change DFS argument to pass EDGE taken to visit vertex
- Extends to other problems.



# Graphs and Mazes

Maze graphs.
- Vertices = intersections
- Edges = hallways.



DFS.
- Mark ENTRY and EXIT halls at each vertex.
- Leave by ENTRY when no unmarked halls.



# Breadth First Search

Depth-first search.
- Visit all nodes and edges recursively.
- Put unvisited nodes on a STACK.

Breadth-first search.
- Put unvisited nodes on a QUEUE.

SHORTEST PATH. What is fewest number of edges to get from s to t?

Solution. BFS.
- Initialize `mark[s] = 0`.
- When considering edge `v-w`:
  - if `w` is marked then ignore
  - if `w` not marked, set `mark[w] = mark[v] + 1`

# Breadth First Search

**Breadth First Search**

```
bfs(Graph G, int s) {
   link t;
   int v, w;
   QUEUEput(s);
   mark[s] = 0;
   while (!QUEUEempty()) {
      v = QUEUEget();
      for (t = G->adj[v]; t != NULL; t = t->next) {
         w = t->v;
         if (mark[w] == UNMARKED) {
             mark[w] = mark[v] + 1;
             QUEUEput(w);
         }
      }
   }
}
```



# Related Graph Search Problems

PATHS. Is there a path from s to t?
- Solution: DFS, BFS, any graph search.

SHORTEST PATH. Find shortest path (fewest edges) from s to t.
- Solution: BFS.

CYCLE. Is there a cycle in the graph?
- Solution: DFS. See textbook.

EULER TOUR. Is there a cycle that uses each edge exactly once?
- Yes if connected and degrees of all vertices are even.
- See textbook to find tour.

HAMILTON TOUR. Is there a cycle that uses each vertex exactly once?
- Solution: ??? (NP-complete)