Graph Usage

- I want to visit all the known famous places starting from Seoul ending in Seoul
- Knowledge: distances, costs
- Find the optimal (distance or cost) path
Graph Theory

- Many problems are mapped to graphs
  - traffic
  - VLSI circuits
  - social network
  - communication networks
  - web pages relationship

- Problems
  - how can a problem be represented as a graph?
  - how to solve a graph problem?
Graph Notations

- A graph $G = (V, E)$
  - $V$ is a set of vertices (nodes)
  - $E$ is a set of edges
    - $E = (x, y)$ where $x, y \in V$
    - ordered or unordered pairs of vertices from $V$

- Examples
  - map
    - landmarks or cities are vertices
    - roads are edges
  - program analysis
    - a line of program statement is a vertex
    - the next line to be executed is connected thru edge
- A graph $G = (V, E)$
  - is **undirected** if edge $(x, y) \in E$ implies that $(y, x) \in E$, too.
  - is **directed** if not
- Most graph problems are undirected
Graphs

- weighted or unweighted
Graphs

- **acyclic** – a graph without any cycle
  - undirected (free tree)

- directed (DAG) – the most important one
Graph Representation

- \( G = (V, E), |V|=n \) and \( |E|=m \)

- **adjacency-matrix**
  - \( n \times n \) matrix \( M \)
    - \( M[i, j] = 1 \), if \( (i, j) \in E \)
    - \( 0 \), if \( (i, j) \notin E \)

  - **good**
    - easy to check if an edge \( (i, j) \) is in \( E \)
    - easy to add/remove edges

  - **bad**
    - space overhead if \( n >> m \)
Examples

- the City (Manhattan) – not so big area
  - 15 avenues and 200 streets
  - 3000 vertices and 6000 edges
  - $3000 \times 3000 = 9,000,000$ cells

- VLSI chip with 15 million transistors
Graph Representation 2

- space efficient for sparse graphs
- problems?
Graph Representation 3

- mixed version
  - use array instead of linked lists
  - looks good? alternatives?
Terminologies – adjacency

- clear for undirected graph
- for directed graph
  - 2 is adjacent to 1
  - 1 is NOT adjacent to 2
  - make a formal definition
- if y is adjacent to x, we write $x \rightarrow y$
  - $1 \rightarrow 2$
Terminologies 2 – incident

- **directed**
  - an edge \((x, y)\) is *incident from* (or *leaves*) vertex \(x\)
  - and is *incident to* (or *enters*) vertex \(y\).

- **undirected**
  - an edge \((x, y)\) is *incident on* vertices \(x\) and \(y\)
  - ex.
    - the edges incident on vertex 2: \((1, 2), (2, 5)\)
Terminologies 3

- **Degree of a vertex**
  - **undirected**
    - the number of edges incident on it.
      - ex. vertex 2 in the graph has degree 2.
    - A vertex whose degree is 0, i.e., vertex 4 in the graph, is *isolated*.
  - **directed**
    - out-degree of a vertex : the number of edges leaving it
    - in-degree of a vertex : the number of edges entering it
    - **degree of a vertex** : its in-degree + out-degree
    - ex. – vertex 2 in the right graph
      - in-degree = 2
      - out-degree = 3
      - degree = 2+3 = 5
Adjacency matrix structure

- only if you know MAXDEGREE
- otherwise, MAXV X MAXV
- symmetric for undirected graph
  - waste of space

```c
#define MAXV 100
#define MAXDEGREE 50

typedef struct {
    int edges[MAXV+1][MAXDEGREE]; /* adjacency info */
    int degree[MAXV+1];            /* outdegree of each vertex */
    int nvertices;                /* number of vertices in graph */
    int nedges;                   /* number of edges in graph */
} graph;
```
adding an edge - insert_edge(g, 1, 2, false)

```c
insert_edge(graph *g, int x, int y, bool directed)
{
    if (g->degree[x] > MAXDEGREE)
        printf("Warning: insertion(%d,%d) exceeds max degree\n",x,y);

    g->edges[x][g->degree[x]] = y;
    g->degree[x] ++;

    if (directed == FALSE)
        insert_edge(g,y,x,TRUE);
    else
        g->nedges ++;
}
```
Graph Traversal

- visit vertices and edges
  - all of them for completeness
  - exactly once for efficiency
- Breadth First Search (BFS)
- Depth First Search (DFS)
BFS
Shortest path from $s$
Queue: $s$ 2 3
Queue: $s\ 2\ 3\ 5$
Queue: 2 3 5 4
Queue: 2 3 5 4

5 already discovered: don't enqueue
Queue: 2 3 5 4
Queue: 3 5 4 6
Queue: 4 6
Undiscovered
Discovered
Top of queue
Finished
Queue: 6 8 7 9
Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9
Queue: 7 9
Queue: 7 9
Queue: 9
Since Queue is empty, STOP!
BFS Algorithm

BFS(G, s)
for each vertex u ∈ V – {s}
    do color[u] ← GRAY
        d[u] ← ∞
        π[u] ← NIL
color[s] ← BLUE
d[s] ← 0
π[s] ← NIL
ENQUEUE(Q, s)
while (Q ≠ ∅)
    do u ← DEQUEUE(Q)
    for each v ∈ Adj[u]
        do if color[v] ← GRAY
            then color[v] ← BLUE
                d[v] ← d[u] + 1
                π[v] ← u
                ENQUEUE(Q, v)

color[u] ← GREEN

Initialization
DFS

- similar to Backtracking
  - go as deep as you can
  - next one is your siblings
- stack is an ideal candidate

DFS(G, v)
for all edges e incident on v
  do if edge e is unexplored then
    w ← opposite(v, e) // return the end point of e distant to v
    if vertex w is unexplored then
      mark e as a discovered edge
      recursively call DFS(G, w)
    else
      mark e as a back edge
Adjacency Lists

A:  F  G
B:  A  I
C:  A  D
D:  C  F
E:  C  D  G
F:  E
G:  
H:  B
I:  H
assume "left child first"

Function call stack:
Topological Sort

- **Definition**
  - A *topological sort* of a DAG $G$ is a linear ordering of all its vertices such that if $G$ contains a link $(u,v)$, then node $u$ appears before node $v$ in the ordering.
Algorithm Example

- find source nodes (indegree = 0)
  - if there is no such node, the graph is NOT DAG

```
in_deg=1
  a
  b
  c
  d
  e
  f
```

Sorted: -

Queue

```
Queue
```

in_deg=2

in_deg=0

in_deg=1

in_deg=3
- span c; decrement in_deg of a, b, e
  - store a in Queue since in_deg becomes 0

**Sorted:** c
- span a; decrement in_deg of b, f
  - store b, f in Queue since ...

\[
\begin{array}{c}
\text{in\_deg}=0 \\
\text{in\_deg}=0 \\
\text{in\_deg}=0 \\
\text{in\_deg}=0 \\
\text{in\_deg}=2 \\
\text{in\_deg}=1 \\
\end{array}
\]

Sorted: c a
span b; store d in Queue

Sorted: c a b
- span f; decrement in_deg of e
  - no node with in_deg = 0 is found

Sorted: c a b f
span d; store e in Queue.

Sorted: c a b f d
span e; Queue is empty

Sorted: c a b f d e
Based on indegree of each vertex
- if it is 0, this node is the first one in the sorted list
- span this node
  - move this node from Queue to the sorted list
  - find nodes edged from this node
  - decrement indegrees of them

It is so similar to BFS
- can you do it like DFS?
```c
int indegree[MAXV];
queue zeroin;
for (i=1; i<=g->nvertices; i++) in[i] = 0;
for (j=0; j<g->degree[i]; j++) in[g->edges[i][j]] ++;

while (!empty(&zeroin)) {
    j = j+1;
    x = dequeue(&zeroin);
    sorted[j] = x;
    for (i=0; i<g->degree[x]; i++) {
        y = g->edges[x][i];
        indegree[y] --;
        if (indegree[y] == 0) enqueue(&zeroin,y);
    }
}
printf("Not a DAG -- only %d vertices found\n",j);
```
Problems 1

- Is a given undirected graph bicolorable?
Problem 2

- input: 4 digits number; $S_1 \ S_2 \ S_3 \ S_4$
- each digit can increment/decrement by one
- find minimal number of dec/inc operations to reach a target four digits number
- there are n forbidden digits where you should not reach at during operations

```
0 0 0 0 8 0 5 6
5 3 1 7 6 5 0 8
8 5
0 0 0 1 8 0 5 7
0 0 0 9 8 0 4 7
0 0 1 0 5 5 0 8
0 0 9 0 7 5 0 8
0 1 0 0 6 4 0 8
0 9 0 0 6 4 0 8
1 0 0 0 6 4 0 8
9 0 0 0
```