Graph

## 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
- $\mathrm{E}=(\mathrm{x}, \mathrm{y})$ where $\mathrm{x}, \mathrm{y} \in \mathrm{V}$
- 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 vertice
- the next line to be executed is connected thru edge


## Graphs



- 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

- $G=(V, E), i V i=n$ and $|E|=m$
- adjacency-matrix
- $\mathrm{n} \times \mathrm{n}$ matrix M

$$
\begin{array}{r}
M[i, j]= \\
0, \text { if }(i, j) \in E \\
0, \text { if }(i, j) \notin E
\end{array}
$$

- good
- easy to check if an edge ( $i, j$ ) is in $E$
- easy to add/remove edges
- bad
- space overhead if $n \gg m$


## Examples



|  | 1 | 2 | 3 | 4 | 5 |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |


|  | 1 | 2 | 3 | 4 | 5 |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 |

- 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


$5 \longrightarrow 4$

- 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 grapg
- 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

```
#define MAXV 100 /* maximum number of vertices */
#define MAXDEGREE 50 /* maximum vertex outdegree */
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;
```

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


## adding an edge - insert_edge(g, 1, 2, false)

```
insert_edge(graph *g, int x, int y, bool directed)
{
```

```
    if (g->degree[x] > MAXDEGREE)
```

    if (g->degree[x] > MAXDEGREE)
    printf("Warning: insertion(\%d, \%d) exceeds max degree\n",x,y);
    printf("Warning: insertion(\%d, \%d) exceeds max degree\n",x,y);
    g->edges[x] [g->degree[x]] = y;
    g->edges[x] [g->degree[x]] = y;
    g->degree[x] ++;
    g->degree[x] ++;
    if (directed == FALSE)
        insert_edge(g,y,x,TRUE);
    else
        g->nedges ++;
    ```


edges[101][51]

degree[51]

\section*{Graph Traversal}
- visit vertices and edges
- all of them for completeness
- exactly once for efficiency
- Breadth First Search (BFS)
- Depth First Search (DFS)

\section*{BFS}



Undiscovered
Discovered
Queue: s 2


Undiscovered
Discovered
Queue: s 23


Undiscovered
Discovered
Queue: s 235


Undiscovered
Discovered
Queue: 2354


Undiscovered
Discovered
Queue: 2354


Undiscovered
Discovered
Queue: 2354


Undiscovered
Discovered
Queue: 354


Undiscovered
Discovered
Queue: 3546


Undiscovered
Discovered
Queue: 546


Undiscovered
Discovered
Queue: 46


Undiscovered
Discovered
Queue: 468


Undiscovered
Discovered
Queue: 687


Undiscovered
Discovered
Queue: 6879


Undiscovered
Discovered
Queue: 879


Undiscovered
Discovered
Queue: 79


Undiscovered
Discovered
Queue: 79


Undiscovered
Discovered
Queue: 79


Undiscovered
Discovered
Queue: 9


Undiscovered
Discovered
Queue: 9


Undiscovered
Discovered
Top of queue
Queue:
\(\Rightarrow\) Since Queue is empty, STOP!

\section*{BFS Algorithm}
```

BFS(G, s)
for each vertex u \inV - {s}
do color[u] \leftarrow GRAY
d[u]}\leftarrow
\pi[u]}\leftarrow\textrm{NIL
color[s]}\leftarrow BLU
d[s]}\leftarrow
\pi[s]}\leftarrow\mathrm{ NIL
ENQUEUE(Q,s)
while (Q \# ¢)
do u \leftarrow DEQUEUE(Q)
for each v \in Adj[u]
do if color[v] \leftarrow GRAY
then color[v]}\leftarrow\mathrm{ BLUE
d[v]}\leftarrow\textrm{d}[\textrm{v}]+
\pi[v]}\leftarrow\mathbf{u
ENQUEUE(Q,v)
color[u]}\leftarrow GREE

```
- similar to Backtracking
- go as deep as you can
- next one is your siblings
- stack is an ideal candidate
\begin{tabular}{|c|c|}
\hline DFS(G, v) & \\
\hline for all edges \(e\) & e incident on \(v\) \\
\hline do if & f edge \(e\) is unexplored then \\
\hline distant to v & \(\omega \leftarrow\) opposite \((v, e) / /\) return the end point of e \\
\hline & if vertex \(w\) is unexplored then \\
\hline & marke as a discovered edge \\
\hline & recursively call DFS(G, w) \\
\hline & else \\
\hline & mark e as a back edge \\
\hline
\end{tabular}


Adjacency Lists
A: \(\quad F G\)
B: A I
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

\section*{assume "left child first"}

Function call stack:

\section*{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


\section*{Algorithm Example}
- find source nodes (indegree \(=0\) )
- if there is no such node, the graph is NOT DAG

- 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 ...


Sorted: ca
- span b; store d in Queue


Sorted: cab
- span f; decrement in_deg of e
- no node with in_deg \(=0\) is found


Sorted: cabit
- span d; store e in Queue.


Sorted: cabid
- span e; Queue is empty


Sorted: cable

\section*{Example Algorithm Summary}
- 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?
```

topsort(graph *g, int sorted[])
int indegree[MAXV];
queue zeroin;
int x, y;
int i, j;
compute_indegrees(g,indegree);
init_queue(\&zeroin);
for (i=1; i<=g->nvertices; i++)
if (indegree[i] == 0) enqueue(\&zeroin,i); 이ᄇ려ᄀ차수 '0'이ᄂ 노드에서 시자ᄀ!
j=0;
while (empty(\&zeroin) == FALSE) { 큐가 비워지ᄅ 때 까지 루프내의 도ᄋ자ᄀ으ᄅ 수해ᄋ!
j = j+1;
x = dequeue(\&zeroin);
sorted[j] = x;
for (i=0; i<g->degree[x]; i++) {
y = g->edges[x][i];
indegree[y] --; 노드 y와 여ᄂ겨ᄅ되ᄂ 노드의 이ᄇ려ᄀ차수르ᄅ 하나씨ᄀ 가ᄆ소!
if (indegree[y] == 0) enqueue(\&zeroin,y);
}
}
if (j != g->nvertices)
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 $\mathbf{n}$ 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 |  |  |  |  |  |
| 1 | 0 | 0 | 0 |  |  |  |  |  |
| 9 | 0 | 0 | 0 |  |  |  |  |  |

