ITS202: Algorithms and Data Structure

Graphs

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

December 2, 2020

Graph

A graph is a set of vertices and a collection of edges that each connect a pair of vertices.

Graphs

Example of Graph: 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	intersection	street
internet	class C network	connection
game	board position	legal move
social relationship	person	friendship
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.

Graph Terminology

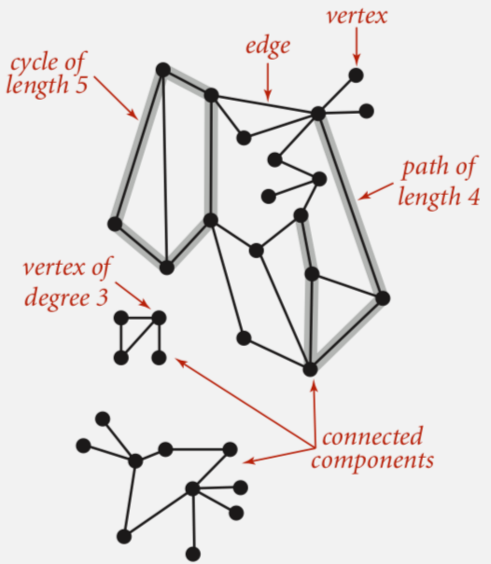


Figure 1

Some graph-processing problems

problem	description
s–t path	*Is there a path between s and t ?*
shortest s–t path	*What is the shortest path between s and t ?*
cycle	*Is there a cycle in the graph ?*
Euler cycle	*Is there a cycle 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 ?*
biconnectivity	*Is there a vertex whose removal disconnects the graph ?*
planarity	*Can the graph be drawn in the plane with no crossing edges ?*
graph isomorphism	*Do two adjacency lists represent the same graph ?*

Undirected Graph: Graph API

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*

Undirected Graph: Graph API

```
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)

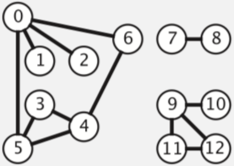
Graph API: Sample Client

Graph input format.

tinyG.txt

V → 13
E → 13

```
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
```




```
% java Test tinyG.txt
0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
⋮
12-11
12-9
```

Graph API: Sample Client

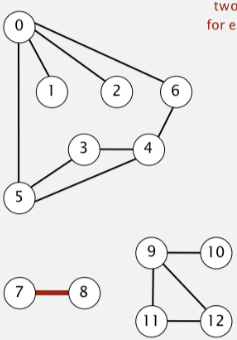
```
// degree of vertex v in graph G
public static int degree(Graph G, int v)
{
    int degree = 0;
    for (int w : G.adj(v))
       degree++;
    return degree;
}
```

Graph Representation

1. Adjacency-matrix graph representation
2. Adjacency-list graph representation

Graph Representation: Adjacency-matrix graph representation

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

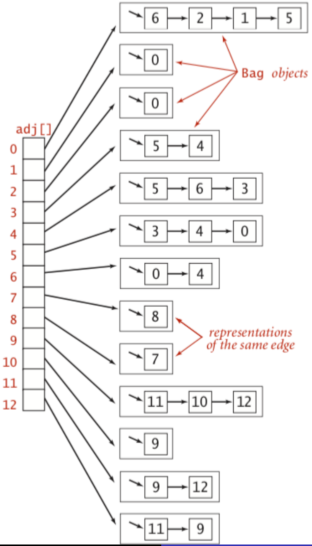


two entries
for each edge

	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 Representation: Adjacency-list graph representation

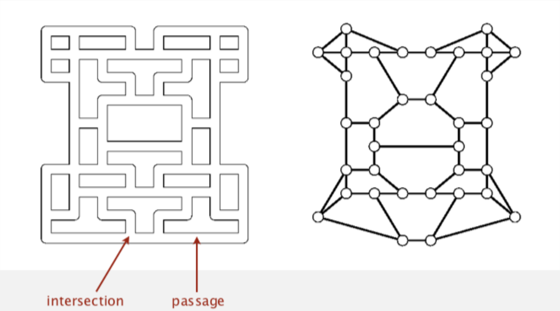
Maintain vertex-indexed array of lists.



Maze Exploration

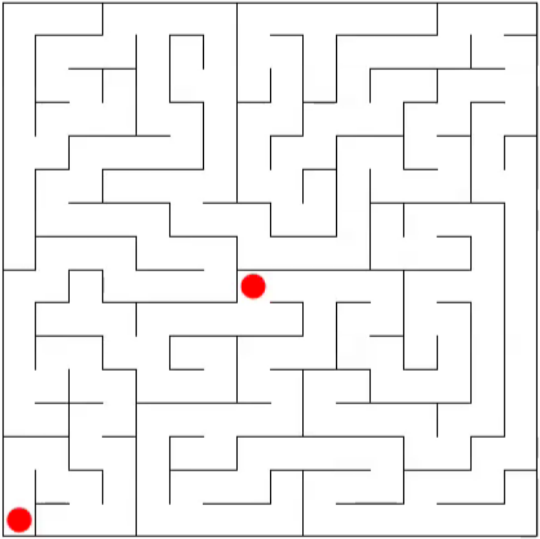
Maze graph.

- Vertex = intersection.
- Edge = passage.



Goal. Explore every intersection in the maze.

Graph Traversal: Depth-first Search



Graph Traversal: Depth-first Search

Goal. Systematically traverse a graph.

DFS (to visit a vertex v)

Mark v as visited.

Recursively visit all unmarked vertices w adjacent to v.

Typical applications

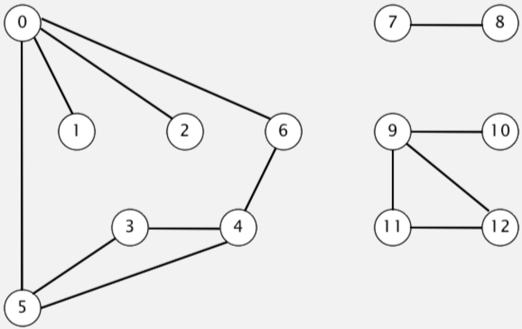
- Find all vertices connected to a given source vertex.
- Find a path between two vertices.

 17 / 29

Depth-first Search Demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



tinyG.txt

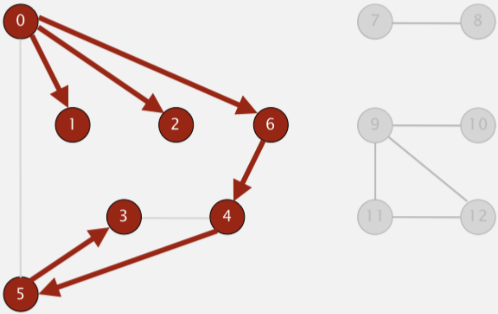
$V \rightarrow$ 13
13 $\leftarrow E$
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3

graph G

Depth-first Search Demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



v	marked[]	edgeTo[]
0	T	–
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

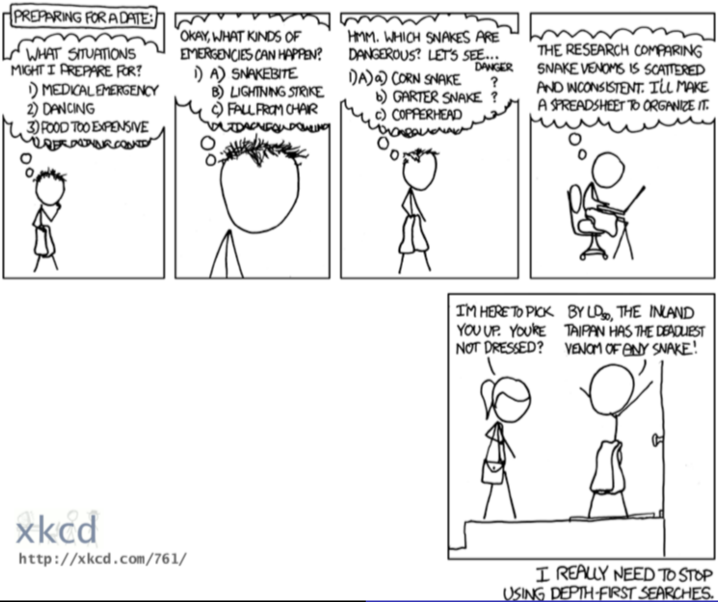
vertices reachable from 0

Depth-first Search: Data Structure

Data structures.

- Boolean array marked[] to mark visited vertices.
- Integer array edgeTo[] to keep track of paths. (edgeTo[w] == v) means that edge v-w taken to visit w for first time .

Depth-first search application: preparing for a date



http://xkcd.com/761/

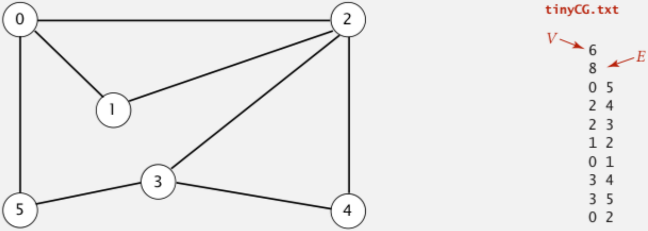
Depth-first search application: preparing for a date

Maze Application that we discussed

Graph Traversal: Breadth-first Search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.

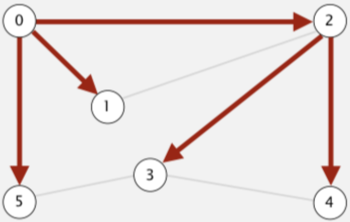


graph G

Graph Traversal: Breadth-first Search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



v	edgeTo[]	distTo[]
0	–	0
1	0	1
2	0	1
3	2	2
4	2	2
5	0	1

done

Algorithm

BFS (from source vertex s)

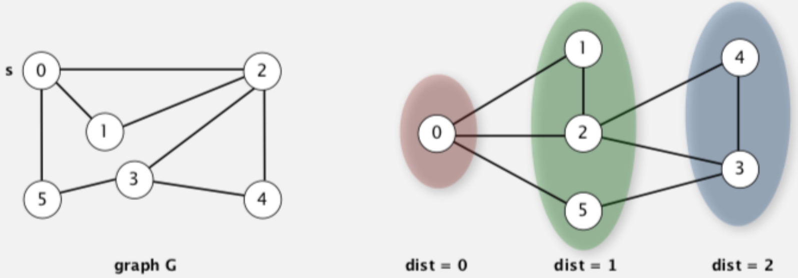
Put s onto a FIFO queue, and mark s as visited.

Repeat until the queue is empty:

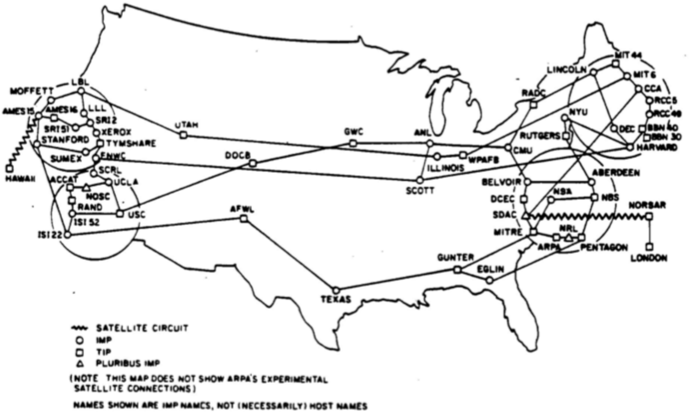
- remove the least recently added vertex v
- add each of v's unvisited neighbors to the queue, and mark them as visited.

Graph Traversal: Breadth-first Search

Proposition. In any connected graph G, BFS computes shortest paths from s to all other vertices in time proportional to $E + V$.



Breadth-first search application: routing

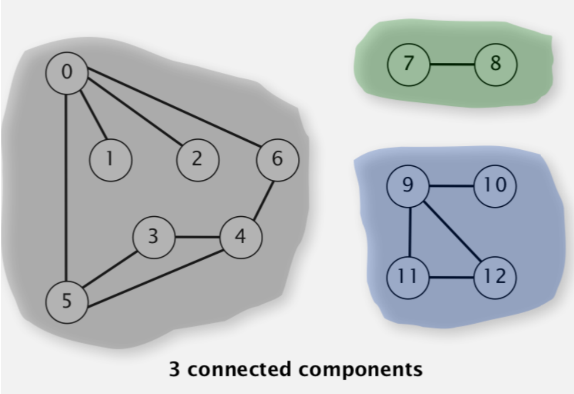


ARPANET, July 1977

Connected Componets

Def. Vertices v and w are connected if there is a path between them.

Def. A connected component is a maximal set of connected vertices.



Graph Traversal Summary

problem	BFS	DFS	time
path between s and t	✔	✔	$E + V$
shortest path between s and t	✔		$E + V$
connected components	✔	✔	$E + V$