Flipped Lecture Concepts

Week 7

Ananda Guna

10.23.14

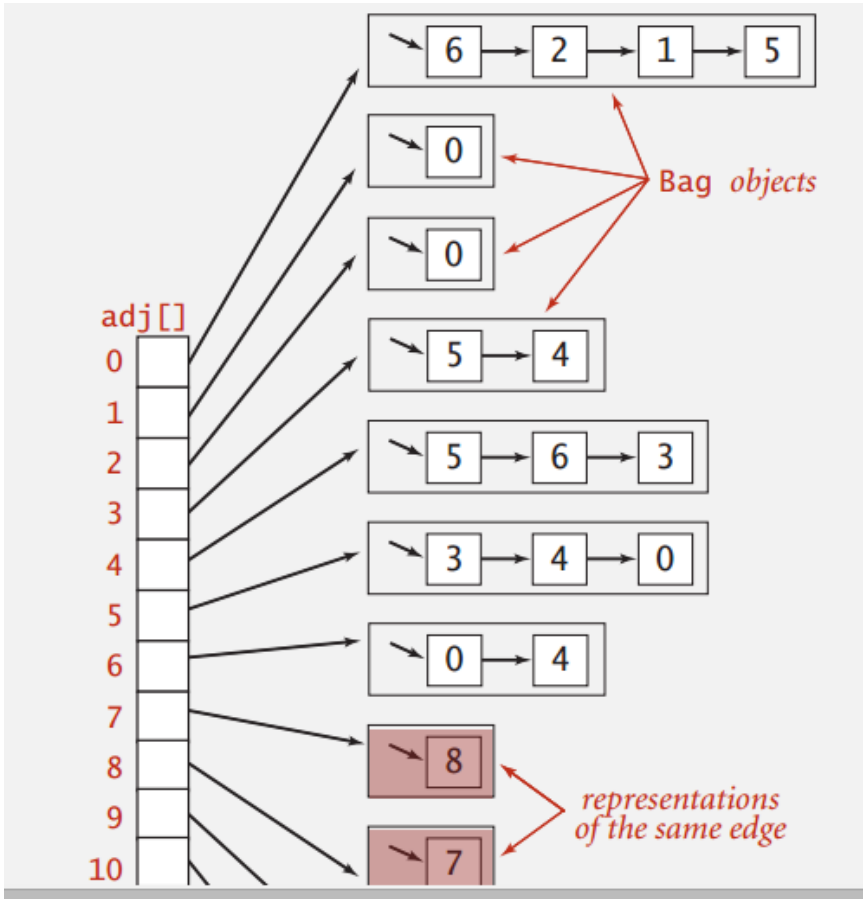
Topic this week

- Undirected graphs
  - Why graphs
  - API, representation
  - DFS and BFS
  - Connected components
  - challenges

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*

Graph representations



Adjacency-list

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	0	1	1	0	0	0
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0
5	1	0	0	1	1	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1

Adjacency-matrix

V={1,2,3,4}, E = {(2,3), (1,2), (3,3)}

List of edges

Representation criteria

- Sparse graphs
  - Adjacency list
- Dense graphs
  - Adjacency matrix

How to determine adjacency versus dense?

Complexity of operations

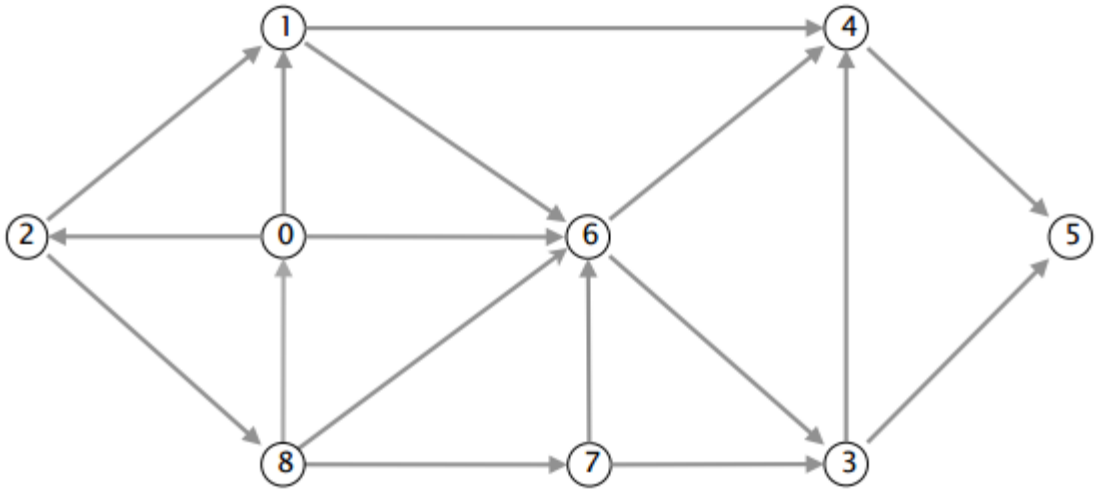
representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	$degree(v)$	$degree(v)$

DFS

DFS (to visit a vertex v)

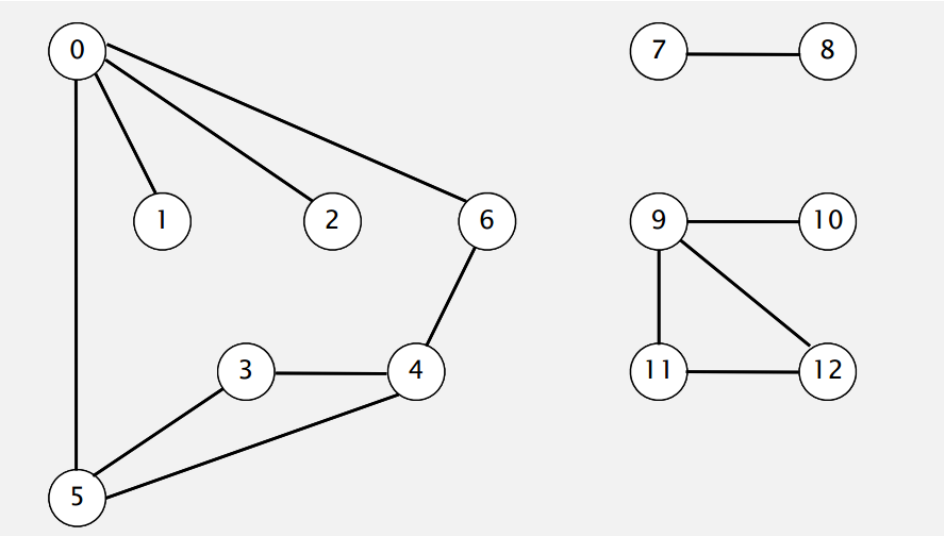
Mark v as visited.

Recursively visit all unmarked vertices w adjacent to v.



Run DFS

Exercise



v	marked[]	edgeTo[]

Recursively visit all nodes reachable from 0

Paths API

public class Paths		
	Paths(Graph G, int s)	*find paths in G from source s*
boolean	hasPathTo(int v)	*is there a path from s to v?*
Iterable<Integer>	pathTo(int v)	*path from s to v; null if no such path*

DFS code

```
private void dfs(Graph G, int v)
{
   marked[v] = true;
   for (int w : G.adj(v))
      if (!marked[w])
      {
         dfs(G, w);
         edgeTo[w] = v;
      }
}
```

Rewrite the recursive dfs iteratively (using a stack)

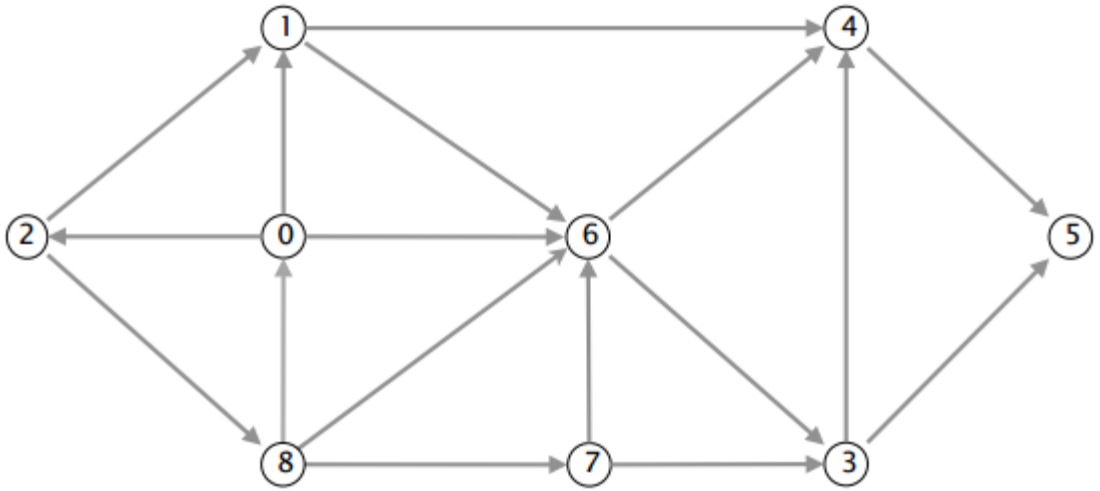
BFS

BFS (from source vertex s)

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

Repeat until the queue is empty:

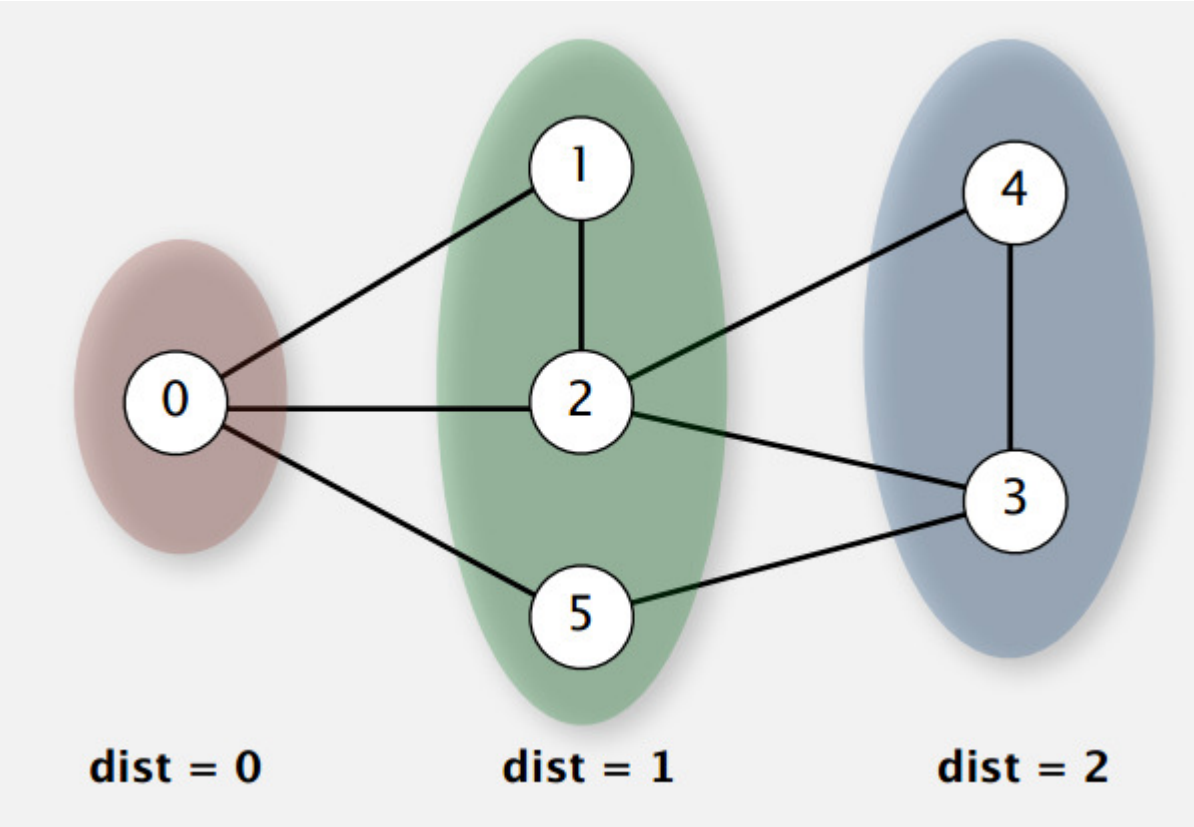
- remove the least recently added vertex v
- add each of v's unvisited neighbors to the queue, and mark them as visited.



Run BFS

Proposition

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



Connected components

```
public class CC
```

`CC(Graph G)`	*find connected components in G*
`boolean connected(int v, int w)`	*are v and w connected?*
`int count()`	*number of connected components*
`int id(int v)`	*component identifier for v (between* `0` *and* `count() - 1`*)*

algorithm

Connected components

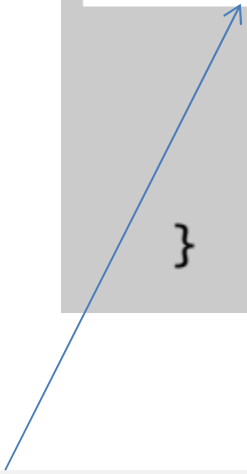
Initialize all vertices v as unmarked.

For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component.

Challenge: connected components are implemented with DFS. Implement CC class with a union-find data structure

Connected components

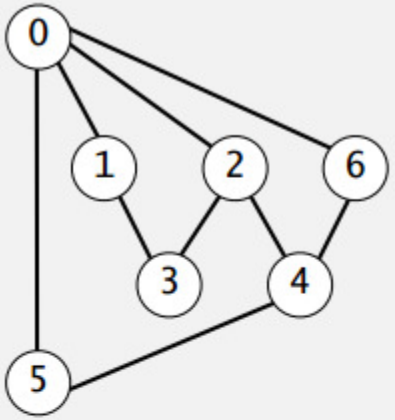
```
private void dfs(Graph G, int v)
{
   marked[v] = true;
   id[v] = count;
   for (int w : G.adj(v))
      if (!marked[w])
         dfs(G, w);
}
```



all vertices discovered in same call of dfs have same id

Challenges

Problem. Is a graph bipartite?

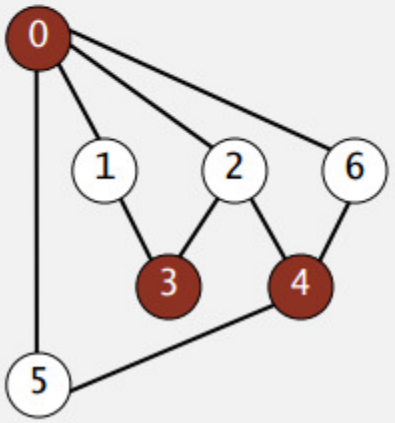


- Definition
  - Divide the graph into two disjoint sets, V and W
  - Each edge in the graph connects a vertex in V to a vertex in W

How difficult?

- Any programmer could do it.
- ✓ Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

simple DFS-based solution (see textbook)



Challenges

Problem. Find a cycle.

How difficult?

- Any programmer could do it.
- ✓ Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

simple DFS-based solution
(see textbook)

More Challenges

Euler cycle. Is there a (general) cycle that uses each edge exactly once?

Answer. A connected graph is Eulerian iff all vertices have even degree.

Problem. Find a (general) cycle that uses every edge exactly once.

✓ • Typical diligent algorithms student could do it.

Problem. Find a cycle that visits every vertex exactly once.

✓ • Intractable.

Hamilton cycle
(classical NP-complete problem)

Problem. Are two graphs identical except for vertex names?

✓ • No one knows.

More challenges

Problem. Lay out a graph in the plane without crossing edges?

linear-time DFS-based planarity algorithm
discovered by Tarjan in 1970s
(too complicated for most practitioners)

Summary

problem	BFS	DFS	time
path between s and t	✔	✔	$E + V$
shortest path between s and t	✔		$E + V$
connected components	✔	✔	$E + V$
biconnected components		✔	$E + V$
cycle	✔	✔	$E + V$
Euler cycle		✔	$E + V$
Hamilton cycle			$2^{1.657\,V}$
bipartiteness	✔	✔	$E + V$
planarity		✔	$E + V$
graph isomorphism			$2^{c\sqrt{V \log V}}$