Undirected Graphs

CS112 Recitation

Ashwin H

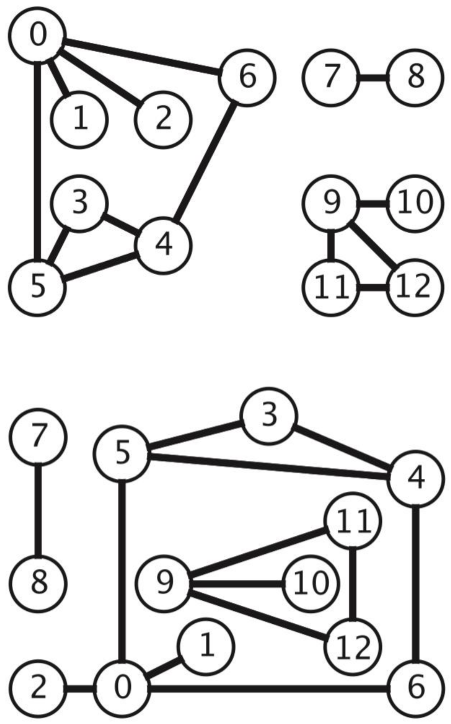
Original slides by: Hoai-An Nguyen
hnn14@scarletmail.rutgers.edu

Exam 3 coming up...

- Question 1: Old material
  - Review old exams
- Question 2: Priority Queues
  - Understand different implementations of priority queues
  - Insertion / deletion into min/max heap
  - Swim/sink operations
  - Visual representation (tree), actual implementation (array)
- Question 3: Hash Tables
  - Representation of hash table (array)
  - How to insert key, value pairs into hash table
    - Hash functions!
  - Collision resolution
    - Chaining
    - Linear Probing
- Review code from lectures
- Big-O!

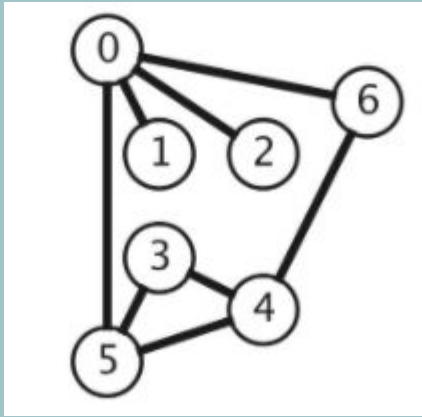
Let’s Review

- A graph is a set of vertices connected pairwise by edges
- You can store an undirected graph in an adjacency matrix or an adjacency list
  - AM: a 2D V-by-V boolean array
  - AL: a vertex-indexed array of lists
- The degree of a vertex is the number of edges incident to it
  - For instance, node 9 has a degree of 3

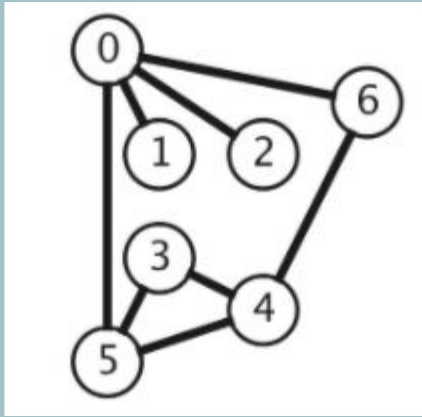


Two drawings of the same graph

Adjacency Matrix



Adjacency List



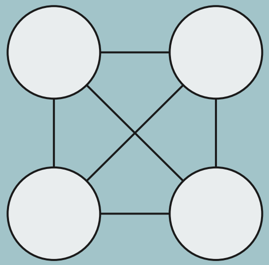
Warm-Up

1. What is the maximum number of edges in a undirected graph with V vertices and no parallel edges?
   a. Hint: (n choose k) = n! / k!(n-k)!

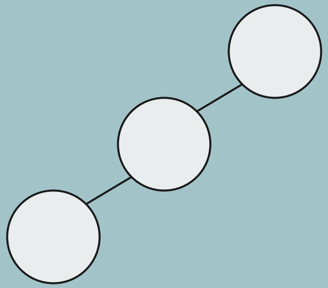
2. What is the minimum number of edges in a graph with V vertices, none of which is isolated (have degree 0)?

Warm-Up

1. What is the maximum number of edges in a undirected graph with V vertices and no parallel edges?
    a. The maximum number of edges in a graph with V vertices and no parallel edges is V * (V - 1) / 2. Since we do not have self-loops or parallel edges, each vertex can connect to V - 1 other vertices. In an undirected graph vertex v connected to vertex w is the same as vertex w connected to vertex v, so we divide the result by 2.
    b. Example: V = 4, E = 4 * (4 - 1) / 2 = 6



2. What is the minimum number of edges in a graph with V vertices, none of which is isolated (have degree 0)?
    a. The minimum number of edges in a graph with V vertices, none of which are isolated (have degree 0) is V - 1.
    b. Example: V = 3, E = 3-1 = 2

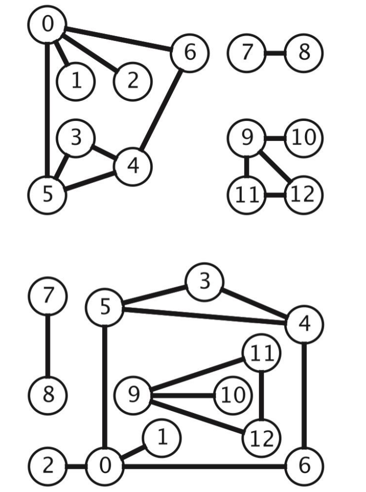


Question 1 - hasEdge()

Add a method hasEdge() to Graph (page 526) which takes two integer arguments v and w and returns true if the graph has an edge v-w, false otherwise.

Example: hasEdge(0, 1) outputs True
hasEdge(0, 4) outputs False

HINT: The method adjacency(vertex_num) returns a list of vertices connected to the vertex “vertex_num”



Two drawings of the same graph

Question 1 - hasEdge() Solution

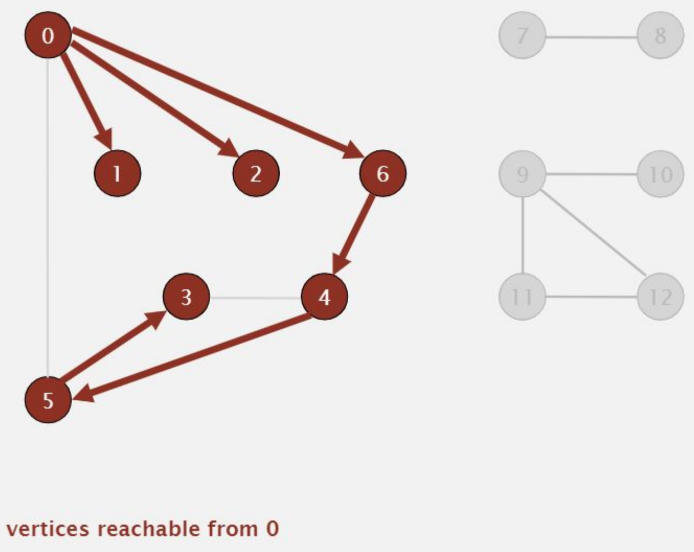
```
public boolean hasEdge(int vertex1, int vertex2) {
    for(int neighbor: adjacent(vertex1)) {
        if (neighbor == vertex2) {
            return true;
        }
    }

    return false;
}
```

Depth-first search demo

To visit a vertex v :

- Mark vertex v.
- 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	–


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

Question 2 - DFS

Show a detailed trace of the call dfs(0) for the following graph.

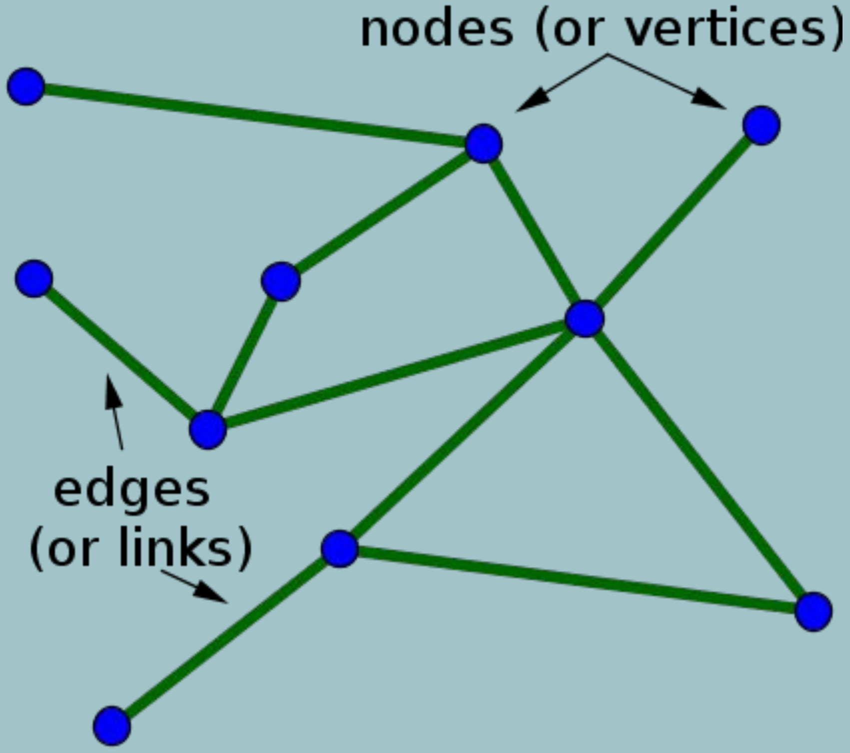
Also, draw the tree represented by the edgeTo[].

The first line corresponds to the number of vertices, the second line corresponds to the number of edges. The remaining lines represent edges between two vertices.

Adjacency list representation on next slide.

Question 2 - DFS Pt 2

12	
16	
8	4
2	3
1	11
0	6
3	6
10	3
7	11
7	8
11	8
2	0
6	2
5	2
5	10
5	0
8	1
4	1



Question 2: DFS

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

marked[]

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

edgeTo[]

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After dfs(0)...

marked[]

0	T
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After dfs(0)...

-look at marked[0], it is True so move on

-look at marked[5], it is false, so

edgeTo[5] = 0
Call dfs(5)

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After calling dfs(5)

-look at marked[5], it is True so move on

-look at marked[0], it is True so move on

-look at marked[10], it is False so

edgeTo[10]=5
Call dfs(10)

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After calling dfs(10),

-look at marked[10], it is True so move on

-look at marked[5], it is True so move on

-look at marked[3], it is False so

edgeTo[3]=10
Call dfs(3)

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After calling dfs(3)

-look at marked[3], it is True so move on

-look at marked[10], it is True so move on

-look at marked[6], it is False so

edgeTo[6]=3
Call dfs(6)

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After calling dfs(6)

-look at marked[6], it is True so move on

-look at marked[2], it is False so

edgeTo[2]=6
Call dfs(2)

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

After calling dfs(2)

-marked[2] = True so move on

-marked[5] = True so move on

-marked[6] = True so move on

-marked[0] = True so move on

-marked[3] = True so move on

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

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

(We are back in the call of dfs(6))

-marked[3] = True so move on
-marked[0] = True so move on

Backtrace to call of dfs(3)

-marked[2] = True so move on

Backtrace to call of dfs(5)

-marked[2] = True, terminate

marked[]	
0	T
1	
2	T
3	T
4	
5	T
6	T
7	
8	
9	
10	T
11	

edgeTo[]	
0	
1	
2	6
3	10
4	
5	0
6	3
7	
8	
9	
10	5
11	

Question 2 - Solution

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

edgeTo[]

0	
1	
2	6
3	10
4	
5	0
6	3
7	
8	
9	
10	5
11	

edgeTo[] tree

0
5
10
3
6
2

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code:

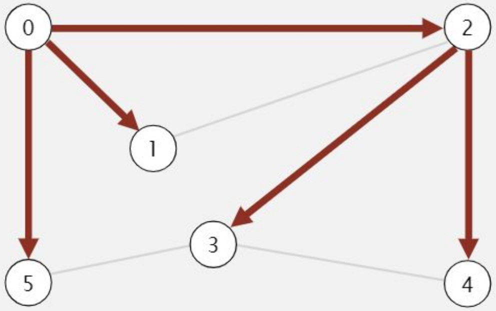
Recommendations:

1. Try tracing problem 1 but with BFS instead (the solution will be on Dynrec)

Breadth-first search demo

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

```
private void bfs(Graph G, int s) {
  Queue<Integer> q = new Queue<Integer>();
  q.enqueue(s);
  marked[s] = true;
  distTo[s] = 0;
  while (!q.isEmpty()) {
    int v = q.dequeue();
    for (int w : G.adj(v)) {
      if (!marked[w]) {
        q.enqueue(w);
        marked[w] = true;
        edgeTo[w] = v;
        distTo[w] = distTo[v] + 1;
      }
    }
  }
}
```

Optional - BFS

Show a detailed trace of the call dfs(0) for the following graph.

Also, draw the tree represented by the edgeTo[].

The first line corresponds to the number of vertices, the second line corresponds to the number of edges. The remaining lines represent edges between two vertices.

Adjacency list representation on next slide.

Question 2: BFS

Adjacency List:

0	0 5 2 6
1	1 4 8 11
2	2 5 6 0 3
3	3 10 6 2
4	4 1 8
5	5 0 10 2
6	6 2 3 0
7	7 8 11
8	8 1 11 7 4
9	9
10	10 5 3
11	11 8 7 1

edgeTo[]

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

distance[]

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

Solution

Tree represented by edgeTo[] after call to bfs(G, 0):

```
      0
   5  2  6
10    3
```