RUTGERS

CS112 Data Structures

Recitation 12

Yu Yang
yu.yang@rutgers.edu
Office: CoRE 331
Office Hour: 3-5pm, Wed.

1. Suppose a weighted undirected graph has n vertices and e edges. The weights are all integers. Assume that the space needed to store an integer is the same as the space needed to store an object reference, both equal to one unit. *What is the minimum value of e* for which the adjacency matrix representation would require less space than the adjacency linked lists representation? Ignore the space needed to store vertex labels.

SOLUTION

Space for adjacency matrix (AMAT) is n^2. Space for adjacency linked lists (ALL) is $n + 3*2e = n + 6e$. (Each node needs 3 units of space: 1 for the neighbor number, 1 for the edge weight, and 1 for the next node reference. And there are $2e$ nodes.) The space required by AMAT and ALL is the same when $n^2 = n + 6e$, i.e. when $e = (n^2 - n)/6$.

The minimum value of e for which the adjacency matrix representation would require less space than the adjacency linked lists representation is one more than the e above, which would be $(n^2 - n)/6+1$.

2. The complement of an **undirected** graph, **G**, is a graph **GC** such that:
   - **GC** has the same set of vertices as **G**
   - For every edge *(i,j)* in **G**, there is no edge *(i,j)* in **GC**
   - For every pair of vertices *p* and *q* in **G** for which there is no edge *(p,q)*, there is an edge *(p,q)* in **GC**.

   Implement a method that would return the complement of the **undirected** graph on which this method is applied.

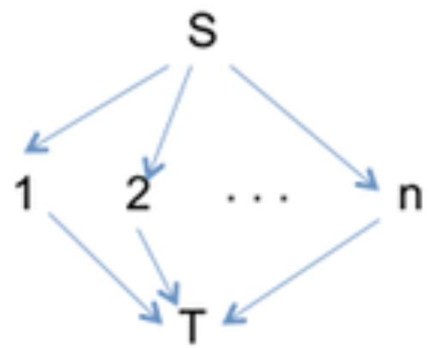
```
class Edge {
   int vnum;
   Edge next;
}

public class Graph {
   Edge[] adjlists;  // adjacency linked lists
   ...
   public Graph complement() {
      // FILL IN THIS METHOD
      ...
   }
}
```

   What would be the worst case running time (big O) of an implementation for a graph with *n* vertices and *e* edges?

SOLUTION

```
public Graph complement() {
   boolean[][] temp = new boolean[adjlists.length][adjlists.length];
   // in temp, fill in trues for the edges
   for (int v=0;v < adjlists.length;v++) {
      for (Edge e=adjlists[v];e != null;e = e.next) {
         temp[v][e.vnum] = true;
      }
   }
   // complement temp
   for (int i=0; i < adjlists.length; i++) {
       for (int j=0; j < adjlists.length; j++) {
           if (i != j) { // leave out the diagonal
              temp[i][j] = !temp[i][j];
           }
       }
   }
   // now create the adjacency linked lists for the complement graph
   Edge[] compall = new Edge[adjlists.length];
   for (int v=0;v < compall.length;v++) {
      for (int j=0; j < adjlists.length; j++) {
         if (temp[v][j]) {
            Edge e = new Edge();
            e.vnum = j;
            e.next = compall[v];
            compall[v] = e;
         }
      }
   }
   // create new Graph and return
   Graph comp = new Graph();
   comp.adjlists = compall;
   return comp;
}
```

3. Consider this graph:



This graph has $n+2$ vertices and $2n$ edges. For every vertex labeled i, $1 <= i <= n$, there is an edge from S to i, and an edge from i to T.

1. How many different depth-first search sequences are possible if the start vertex is S?
2. How many different breadth-first search sequences are possible if the start vertex is S?

RUTGERS

SOLUTION

1. *n!*, for the different permutations of the vertices `1` through `n`. (Note: If a vertex `v` in this set is visited immediately after `S`, then `T` would be immediately visited after `v`.)

   For instance, say *n* = *3*. Here are all possible DFS sequences (3! = 6):

```
S 1 T 2 3
S 1 T 3 2
S 2 T 1 3
S 2 T 3 1
S 3 T 1 2
S 3 T 2 1
```

2. *n!*, similar to DFS. The only difference is that `T` will be the last vertex to be visited. So, if *n* = *3*, the possible BFS sequences are:

```
S 1 2 3 T
S 1 3 2 T
S 2 1 3 T
S 2 3 1 T
S 3 1 2 T
S 3 2 1 T
```