# Creates the executable randGraph which
# uses the matrix representation to create a graph

randGraph: driver.o edge.o graphList.o
            gcc -g -o randGraph driver.o edge.o graphList.o

driver.o: driver.c graph.h edge.h
            gcc -g -c driver.c

edge.o: edge.c edge.h
            gcc -g -c edge.c

graphList.o: graphList.c graph.h edge.h misc.h
            gcc -g -c graphList.c

.PHONY: clean
clean:
        -/bin/rm *.o

.PHONY: tar
tar:
        tar -cvf - driver.c edge.c edge.h graph.h graphList.c makefile | gzip >
        src.tar.gz

.PHONY: a2ps
a2ps:
        a2ps -q -Avirtual -2 -o l15-code.ps makefile graphList.c graph.h driver.c

// file: graphList.c
// by: Sedgewick
// modified by: Dawn Lawrie
// from: Program 17.6
// Implements the adjacency list version of the graph

#include <stdlib.h>
#include <stdio.h>
#include "graph.h"
#include "misc.h"

typedef struct node Node;
struct node {
    int to;
    Node *next;};

struct graph {
    int vertices;
    int edges;
    Node **adj;};

Node *createLink(int toVertex, Node *next) {
    Node *link = (Node *) malloc(sizeof(Node));
    CHECKP(link);  link->to = toVertex;  link->next = next;
    return link;
}

// Purpose: Create a graph with V vertices and no edges
// input: number of vertices
// output: A graph
Graph *graphInit(int numVerts) {
    int i;
    Graph *theGraph = (Graph *) malloc(sizeof(Graph));
    CHECKP(theGraph);
    theGraph->vertices = numVerts;
    theGraph->edges = 0;
    theGraph->adj = (Node **) malloc(numVerts*sizeof(Node *));
    CHECKP(theGraph->adj);
    for (i = 0; i < numVerts; i++)
        theGraph->adj[i] = NULL;
    return theGraph;
}

// Purpose: Insert an edge into an undirected graph
// (allows self-loops and parallel edges)
// input: a graph and an edge
// output: none
void graphInsertEdge(Graph *theGraph, Edge theEdge) {
    int fromVertex = theEdge.from;
    int toVertex = theEdge.to;
    theGraph->adj[fromVertex] = createLink(toVertex, theGraph->adj[fromVertex]);
    theGraph->adj[toVertex] = createLink(fromVertex, theGraph->adj[toVertex]);
    theGraph->edges++;
}

// Purpose: Remove an edge from an undirected graph if it exists
// input: a graph and an edge
// output: none
void graphRemoveEdge(Graph *, Edge);
// Purpose: return an array of edges in the graph
// input: an empty array of edges (should be at least V^2) and the graph
// side-effect: writes to the Edge array (parallel edges inserted, but not
// self loops)
int graphEdges(Edge edgeArray[], Graph *theGraph) {
  int edgeCount = 0;
  Node *tmp;
  for (vertex = 0; vertex < theGraph->vertices; vertex++)
    for (tmp = theGraph->adj[vertex]; tmp != NULL; tmp = tmp->next)
      if (vertex < tmp->to) {
        edgeArray[edgeCount] = createEdge(vertex, tmp->to);
        edgeCount++;
      }
  return edgeCount;
}

// Purpose: create a deep copy of the Graph
// input: Graph *to be copied
// output: newly created graph
Graph *graphCopy(Graph *);

// Purpose: deallocates all the memory associated with the Graph
// input: Graph *to be destroyed
// output: none
void graphDestroy(Graph *);

// Purpose: Test for the existence of an Edge
// input: A graph and an Edge
// output: True if the edge exists and otherwise false
bool graphIsEdge(Graph *, Edge);

// Purpose: Output a graph to standard out
// input: A graph
// output: none
void graphShow(Graph *theGraph) {
  int vertex;
  Node *tmp;
  for (vertex = 0; vertex < theGraph->vertices; vertex++) {
    printf("%2d:
", vertex);
    for (tmp = theGraph->adj[vertex]; tmp != NULL; tmp = tmp->next)
      printf("%2d
", tmp->to);
    printf("\n\n");
  }
}

// Purpose: Return the number of edges in the graph
// input: A graph
// output: the number of edges
int graphNumEdges(Graph *theGraph) {
  return theGraph->edges;
}

// Purpose: Return the number of vertices in the graph
// input: A graph
// output: the number of vertices
int graphNumVertices(Graph *theGraph) {
  return theGraph->vertices;
}

// Purpose: Create a graph of V vertices and E random edges
// input: number of vertices and number of edges
Graph *graphRand(int vertices, int edges) {
  Graph *theGraph = graphInit(vertices);
  while (graphNumEdges(theGraph) < edges) {
    int from = (rand() / (RAND_MAX+1.0)) * vertices;
    int to = (rand() / (RAND_MAX+1.0)) * vertices;
    if (from < to) {  // NOTE: This use of random is better then rand() % vertices because
      graphInsertEdge(theGraph, createEdge(to, from));
    }
  }
  return theGraph;
}
// File: graph.h
// by: Sedgewick
// modified by: Dawn Lawrie
// from: Program 17.1

// This is a Graph interface.
// Either Graph representation can use this interface

#ifndef GRAPH_H
#define GRAPH_H

#include "edge.h"

typedef int bool;
#ifndef true
#define true 1
#endif
#ifndef false
#define false 0
#endif

typedef struct graph Graph;

// Purpose: Create a graph with V vertices and no edges
// input: number of vertices
// output: A graph
Graph *graphInit(int);

// Purpose: Insert an edge into an undirected graph (allows self-loops)
// input: a Graph and an Edge
// output: none
void graphInsertEdge(Graph *, Edge);

// Purpose: Remove an edge from an undirected graph if it exists
// input: a Graph and an Edge
// output: none
void graphRemoveEdge(Graph *, Edge);

// Purpose: return an array of edges in the graph
// input: an empty array of edges (should be V^2) and the graph
// output: number of edges in the array
// side-effect: writes to the Edge array
int graphEdges(Edge [], Graph *);

// Purpose: create a deep copy of the Graph
// input: Graph to be copied
// output: newly created graph
Graph *graphCopy(Graph *);

// Purpose: deallocates all the memory associated with the Graph
// input: Graph to be destroyed
// output: none
void graphDestroy(Graph *);

// Purpose: Test for the existence of an Edge
// input: A graph and an Edge
// output: True if the edge exists and otherwise false
bool graphIsEdge(Graph *, Edge);

// Purpose: Output a graph to standard out
// input: A graph
// output: none
void graphShow(Graph *);

// Purpose: Return the number of edges in the graph
// input: A graph
// output: the number of edges
int graphNumEdges(Graph *);
#endif
// Uses the graph ADT
// Program takes the number of vertices and edges as arguments, generates a random graph, and displays it
#include <stdlib.h>
#include <stdio.h>
#include "graph.h"

int main(int argc, char **argv) {
    if (argc < 3) {
        fprintf(stderr, "Usage: randGraph <# vertices> <# edges>\n");
        exit(1);
    }
    int vertices = atoi(argv[1]);
    int edges = atoi(argv[2]);
    Graph *theGraph = graphRand(vertices, edges);

    if (vertices < 20)
        graphShow(theGraph);
    else
        printf("%d vertices, %d edges\n", graphNumVertices(theGraph),
                graphNumEdges(theGraph));
}