

CS xyz3-001 Foundations of Programming and Data Structures

Instructor [Dr. Turgay Korkmaz](#)

Homework 12

Due date: check BB

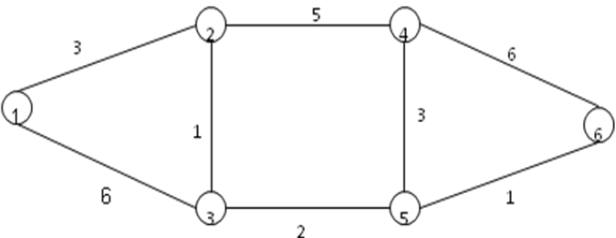
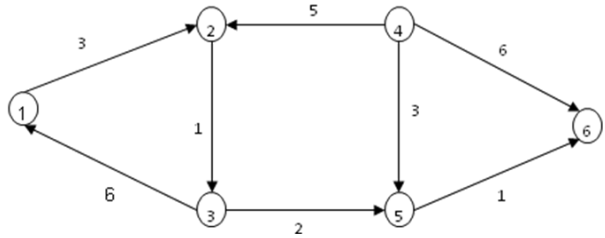
!!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!

(Graphs – graph functions)

At the end of this document, you are given the basic code as **graph.c** which includes the structures and functions that we implemented in the slides to create/read/print graphs. First get **hw12-graph.zip** which includes **graph.c** and sample input files for the below graphs.

Then, compile graph.c and run it for the given input files as follow:

- **gcc graph.c -o graph**
- **./graph undirectedgraph1.txt**
- **./graph directedgraph1.txt**

	
undirectedgraph1.txt 6 8 0 1 2 3 1 3 6 2 3 1 2 4 5 3 5 2 4 5 3 4 6 6 5 6 1	directedgraph1.txt 6 8 1 1 2 3 2 3 1 3 1 6 3 5 2 4 2 5 4 5 3 4 6 6 5 6 1

After studying and understanding the given code, **first modify** `insert_edge()` function so that it can keep the link list sorted with respect to (w.r.t.) neighbor IDs. **Second implement** `graph_copy()` to create a copy of the given graph. User will call the original graph as **myg1** and the copy as **myg2**, for which we use the same pointer names in the program. Now extend **the main function** so that it can asks user to enter various commands in a loop and performs these commands on the related graphs. Accordingly, you also need to **implement those functions** and call them. Finally, when ending the main function, make sure you **free the graphs**

----- see more explanations given at the end of this handout-----

Specifically, your program will ask user to enter a command and related parameters (if any) in a loop, and then perform the given commands. Here is the list of commands that your program must implement: [Your command names should be **the same** as written below so the TA can copy paste his/her test cases... for your own testing you can use initials etc, but your code should be able to handle the full command names below]

```
* insert          [myg1 | myg2] x y w
* delete          [myg1 | myg2] x y

* printgraph      [myg1 | myg2]
* printdegree     [myg1 | myg2] // if directed, print both in- and out-degree
* printcomplement [myg1 | myg2]

* eliminatelinks  [myg1 | myg2] minW maxW
* differentlinks  [myg1 | myg2] [myg1 | myg2]
* commonlinks     [myg1 | myg2] [myg1 | myg2]

* dfs             [myg1 | myg2] x
* bfs             [myg1 | myg2] x
* isconnected     [myg1 | myg2]
* numofconncomp   [myg1 | myg2]

* quit
```

----- see more explanations given at the end of this handout-----

Please make sure your program processes the above comments as is so that TA can copy/paste his test cases.

As always, make sure you release (free) the dynamically allocated memories if you allocate any memory in your programs. So, before submitting your program, run it with `valgrind` to see if there is any memory leakage...

Also if you need to debug your program, compile your programs with `-g` option and then run it with `gdb` and/or `ddd`.

What to do and return: !!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!

1. Create a directory `abc123-hw12`, using your own `abc123`. Do all your work under that directory. You can implement everything in one `.c` file. Or as we did before, you can implement graph related things as a library and then use it with a simple driver program. Either way is fine. But it may be easier to implement all in one `.c` file.

2. To easily compile the library and driver program, you must have a Makefile and use “make” to compile your code.

3. Follow the problem solving methodology, and solve the problem(s). Then convert your solution(s) to a C program. You can name your program here as

hw12.c

```
/*
 * Don't forget to include comments about the
 * problem, yourself and each major step in your
 * program! so that we can understand your
 * solution(s).
 */
```

4. Compile your program using Makefile. Then run it with a few times with different input values and copy/paste the results in an output file, which you can name as hw12-out.txt. Also make sure you get hw12-valgrind.txt, as described in previous assignments.

5. Zip the whole directory abc123-hw12 as abc123-hw12.zip

6. Go to BB Learn (<http://learn.utsa.edu/>) , login using your abc123

7. Submit your abc123-hw12.zip for hw12 under Assignments

You must submit your work using Blackboard Learn and respect the following rules:

- 1) All assignments must be submitted as either a zip or tar archive file unless it is a single pdf file.
 - 2) Assignments must include all source code.
 - 3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
 - 4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.
-

graph.c

```
#include <stdio.h>
#include <stdlib.h>
typedef enum {FALSE, TRUE} bool;
#define MAXV 100

typedef struct edgenode {
    int y;
    int weight;
    struct edgenode *next;
} edgenodeT;

typedef struct {
    edgenodeT *edges[MAXV+1];
    int degree[MAXV+1];
    int nvertices;
    int nedges; // number of directed edges....
    bool directed;
} graphT;

void initialize_graph(graphT *g, bool directed);
void read_graph(graphT *g, char *filename);
void insert_edge(graphT *g, int x, int y, int w);
void print_graph(graphT *g, char *name);
void free_graph(graphT *g);
graphT *copy_graph(graphT *g);
// put prototypes for the other functions you will implement....

int main(int argc, char *argv[])
{
    graphT *myg1=NULL, *myg2=NULL;

    if(argc < 2){
        fprintf(stderr, "Usage: %s graph_filename", argv[0]);
        exit(-1);
    }
    myg1 = (graphT *) malloc(sizeof(graphT));
    if (myg1==NULL) {
        fprintf(stderr, "Cannot allocate memory for the graph");
        exit(-1);
    }
    initialize_graph(myg1, FALSE);
    read_graph(myg1, argv[1]);
    print_graph(myg1, "myg1");

    // first implement copy_graph function and call it here
    myg2 = copy_graph(myg1);
    print_graph(myg2, "myg2");

    // NOW in a loop get commands and
    // call related functions to perform them...
    free_graph(myg1);
}
```

```

void initialize_graph(graphT *g, bool directed)
{
    int i;
    g->nvertices = 0;
    g->nedges = 0;
    g->directed = directed;

    for (i=1; i<=MAXV; i++)
        g->edges[i] = NULL;

    for (i=1; i<=MAXV; i++)
        g->degree[i] = 0;
}

void read_graph(graphT *g, char *filename)
{
    int i;
    int n, m, dir;
    int x, y, w;
    FILE *fp;
    if((fp=fopen(filename,"r"))==NULL){
        fprintf(stderr, "Cannot open the graph file");
        exit(-1);
    }
    fscanf(fp,"%d %d %d", &n, &m, &dir);
    g->nvertices = n;
    g->nedges = 0; // insert function will increase it;
    g->directed = dir;
    for (i=1; i<=m; i++) {
        fscanf(fp,"%d %d %d", &x, &y, &w);
        insert_edge(g, x, y, w);
        if(dir==FALSE)
            insert_edge(g, y, x, w);
    }
    fclose(fp);
}

void insert_edge(graphT *g, int x, int y, int w)
{
    edgenodeT *pe;
    pe = malloc(sizeof(edgenodeT)); // check if NULL
    pe->weight = w;
    pe->y = y;

    // YOU MUST MODIFY THIS FUNCTION SO IT WILL KEEP LINK LIST SORTED
    // W.R.T. NEIGHBOR IDs.

    pe->next = g->edges[x];
    g->edges[x] = pe;

    g->degree[x]++;
    g->nedges++;
}

```

```

void print_graph(graphT *g, char *name)
{
    edgenodeT *pe;
    int i;
    if(!g) return;
    printf("Graph Name: %s\n", name);
    for(i=1; i<=g->nvertices; i++) {
        printf("Node %d: ", i);
        pe = g->edges[i];
        while(pe){
            //          printf(" %d", pe->y);
            printf(" %d(w=%d)", pe->y, pe->weight);
            pe = pe->next;
        }
        printf("\n");
    }
}

void free_graph(graphT *g)
{
    edgenodeT *pe, *olde;
    int i;
    for(i=1; i<=g->nvertices; i++) {
        pe = g->edges[i];
        while(pe){
            olde = pe;
            pe = pe->next;
            free(olde);
        }
    }
    free(g);
}

graphT *copy_graph(graphT *g)
{
    graphT *newg;

    // I simply return the same graph as a copy
    // but you really need to dynamically create
    // another copy of the given graph
    newg = g;

    return newg;
}

// your other functions will go here.....

```

HERE ARE SOME CLARIFICATIONS

- * `insert [myg1 | myg2] x y w`
`[myg1 | myg2]` means user will type either myg1 or myg2
`x y w` means user will enter an integer for each
- * `insert myg1 3 4 20`
insert a new edge 3-4 into myg1 graph with weight of 20.
If this is an undirected graph also insert edge 4-3 with weight of 20. If that edge is already in the graph, don't insert a new edge but update the weight based on the newly given weight... Remember you need to insert edges in a sorted manner!
- * `delete myg1 2 4`
delete edge 2-4 from myg1. If this is an undirected graph also delete edge 4-2. If that edge is not in the graph, don't delete anything...
- * `printgraph myg1`
print graph using the code given...
- * `printdegree myg1`
if myg1 is undirected, then simply count the number of neighbors in the adjacency list for each node and print that number as the degree of each node.

if the graph is directed, then again you can simply count the number of neighbors in the adjacency list for each node and print that number as the out-degree of each node... BUT you also need to find in-degree. For this, you can check every node (say node i) and count how many times node i appears in the all adjacency lists. Then print that count as the in-degree for node i.
- * `printcomplement myg2`
First create the complement graph of myg2 as cg, and call `printgraph(cg)` then free complement graph cg. Don't worry about weight in the new graph cg. for example, you can set all weights to 1 in cg
- * `eliminatelinks myg1 minW maxW`
minW and maxW will be two numbers
check each edge pe
if (pe->w < minW || pe->w > maxW) delete that edge
- * `differentlinks myg1 myg2`
print edges that are in myg1 but not in myg2

```

* commonlinks      myg1 myg2
  print edges that are both in myg1 and in myg2

* dfs      myg1 x
  print in which order nodes are visited according to dfs
  then for each node i print the path from x to node i

* bfs      myg2 x
  print in which order nodes are visited according to bfs
  then for each node i print the path from x to node i

* isconnected      myg1
* numofconncomp    myg2

```

last two commands isconnected numofconncomp will be
 performed if the graph is UNdirected ... and print
 appropriate messages.
 if the graph is directed don't do anything or just print
 "Purchase the next version of this program :)"