University of Toronto

CSCB70S—Data Structures and Techniques, Spring 1999

Assignment 2: Graphs

Due: Friday February 26 at 4:00 pm.
Worth: 10%.

The problem

The purpose of this assignment is to teach you how to represent and traverse a graph in C. It will also help you to get comfortable with pointers, dynamic allocation and modules in C.
Suppose that you are writing part of the code for a program that solves mazes (as part of a game). It is possible to represent mazes by directed graphs: vertices represent locations in the maze, and edges represent passageways from one location to the next. With this representation, given a starting point in a maze (i.e., an initial node in the graph), we can look for an exit by executing a search algorithm on the graph. For this assignment, you will write implementations for two different search algorithms: "Breath-First Search" (BFS) and "Depth-First Search" (DFS).

Programming

Make a directory in your account:

```
% cd
% mkdir hwk2
% cd hwk2
```

and copy the skeleton file into it from:

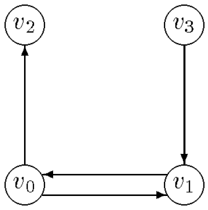
```
http://www.cs.toronto.edu/~daniel/teaching/B70
```

You have to write code to implement the functions `bfs`, `dfs`, `insert_edge`, and `create_graph`. You should also add other functions to the program as needed to structure your code properly. Pay special attention to the comments in the code, which will tell you what parts of the program you need to add code to. In any case, **you should not change the input/output**. Give good comments in your code, including a comment above each of the functions you write that describes the purpose of the function, the meaning of its parameters and its output, and any relevant pre- and post-conditions.
The function `create_graph` will read information from an input file in the following format: first, the number of vertices in the graph; then, a list of directed edges given as pairs of vertex numbers. The graph should be stored using adjacency lists, in the following way: `create_graph` should create a dynamically allocated array for the vertices in the graph, and each entry in the array should be a pointer to a linked list of integers representing the edges adjacent to that vertex. For example, the input below represents the given graph and should be represented by the given data structure:

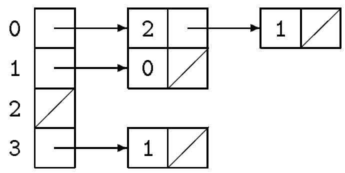
Input

```
4
0 1
3 1
0 2
1 0
```

Graph



Data Structure



Edges should be added to the graph structure using a second function `insert_edge` that will be called from `create_graph` for each edge to be inserted. These two functions should be put in the file "`graph.c`", together with any other related functions you may want to add (make sure to modify the header file "`graph.h`" if you add new functions that are meant to be used in other parts of the program).
After having done this, write code for the functions `bfs` and `dfs`, both in the file "`search.c`", to implement breath-first search and depth-first search, respectively. These functions should take as parameters a graph and the numbers of the initial and final vertices, and they should create and return a linked list containing the vertices on a path from the initial vertex to the final vertex, if such a path exists (they should return `NULL` otherwise). As before, you may add other related functions to the file, and modify the header file "`search.h`" if you add new functions that are meant to be used in other parts of the program.
Once you are done, compile and test your program as follows:

```
% make
% search inputfile
```

Testing and Report

Test your program with the `inputfile` provided on the Web page. In addition, test your program with another input file that you create. Choose tests that make a thorough checking of your program. Capture the screen output by using input and output redirection, as follows: create a file containing start and finish vertex numbers (two per line), whose last line is "`-1 -1`", and run the program using the command-line

```
% search my-input < searches > my-output
```

(assuming that your input file is called "`my-input`" and the file that contains the search numbers is called "`searches`").
Also write a short report where you will explain the data structures and the relationship between the functions in your program, as well as discuss your test results.

How to hand in your assignment

Hand in the following:

- the cover page (included below); a hardcopy printout of all your programs;
- a hardcopy printout of the output of your program; and the report.

Put this into an ENVELOPE. Write the following on the outside of the envelope:

- your name (with surname underlined) and student number;
- your account name; and your tutorial section.

Your programs must also be submitted electronically by the deadline. You must submit all your files for assignment 2 by executing `exactly the following commands` from the directory that contains the source files on your account:

```
% make clean
% submit -N a2 cscb70s *
```

You will lose marks if you don't follow these submission instructions exactly.

COVER SHEET FOR ASSIGNMENT 2

Account name:

Surname:

Given Name(s):

Student Number:

Tutorial:

I declare that this assignment is my own work and is submitted in accordance with the University of Toronto Code of Behavior on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism described in the course syllabus.

I discussed this assignment with the following people:

Signature

GRADING

Programs		
insert_edge	______	/ 10
create_graph	______	/ 10
bfs	______	/ 15
dfs	______	/ 15
Style (including comments)	______	/ 15
Thoroughness of your own tests	______	/ 10
Report		
Content	______	/ 15
Style and readability	______	/ 10
Total	______	/ 100