Middle East Technical University
Department of Computer Engineering

CENG 140

Spring 2018

Take Home Exam 3

REGULATIONS

Due date: 31 May 2018, Thursday *(Not subject to postpone)*

Submission: Electronically. You will be submitting your program source code written in a file which you will name as `the3.c` through the cow web system. Resubmission is allowed (till the last moment of the due date), The last will replace the previous.

Team: There is **no** teaming up. The take home exam has to be done/turned in individually.

Cheating: All parts involved (source(s) and receiver(s)) get zero. You will face disciplinary charges.

INTRODUCTION

A *graph* is a set of *vertices* and a set of *edges* each of which run from a vertice to another. Vertices are labeled. It is also possible that vertices and/or edges have some data attached to them. Edges can be directed or not. If they have a direction then the graph is called a *directed graph*. Having a mathematical theory (Graph Theory), graphs play an enormous important role in Computer Science. Many problems are modeled by means of graphs. A branch of algorithms (Algorithmic Graph Theory) is entirely dedicated to deal with graph related problems.

PROBLEM

The problem we will consider is about merging two graphs. You will be supplied with two arbitrary, weakly connected, directed graphs. *(To enrich your terminology about graphs we strongly encourage you to consult the web. That is part of the education)*

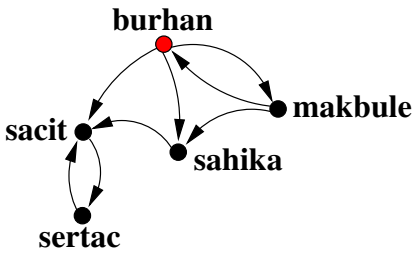
The graphs that we will be treating correspond to the 'friend' declarations in some social network (like facebook). In addition there is a *supernode* (or super vertex) in each of our graphs so that each other vertex is reachable by a (directed) path starting at this supernode. This node will be provided to you. Furthermore there will be no more then one edge (with the same direction) running among a pair of vertices. Also, for our edges the start and the end vertices are different, per se.

The job is simple, you are given a data representation for this kind of a graph so that the graph is uniquely represented (this representation will be explained below). You are expected to write a function which takes two arguments, each of which is a pointer to the supernode of a graph of the above defined nature. The data representation is so that when you have access to the supernode all graph is accessible.

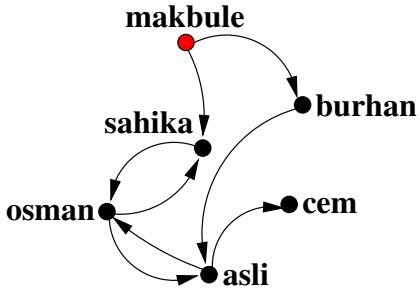
The label of the second argument supernode is also a label of a vertex of the graph reachable by the first argument supernode. Furthermore it is possible that the two graphs have other common labels in their vertices.

The task is to merge the two graphs together (over common labels) to form a single graph. You will do the merging by constructing a brand new data structure and returning the pointer to its supernode.

Now let us have an example *(supernodes are in red)* :

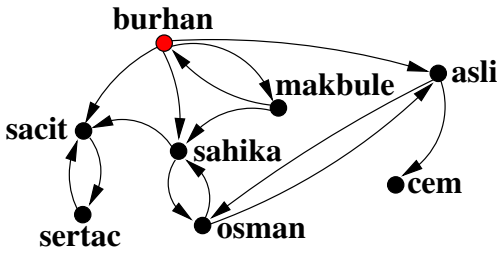


$GRAPH_1$



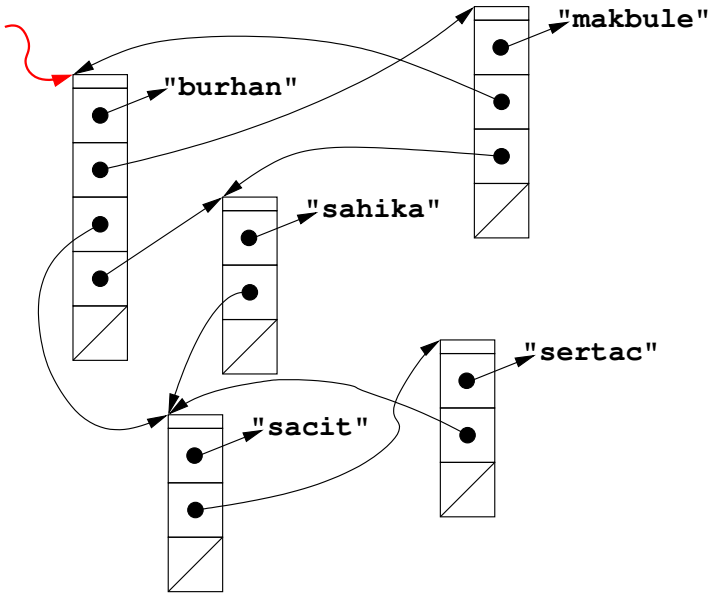
$GRAPH_2$

The merged graph is supposed to be:

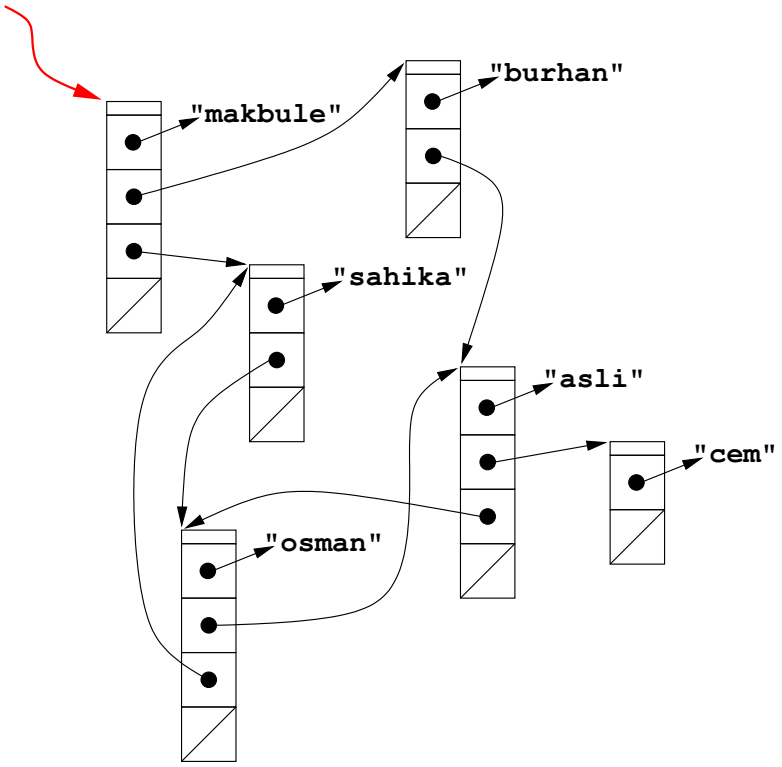


$GRAPH_{merged}$

The data structures that correspond to the example:

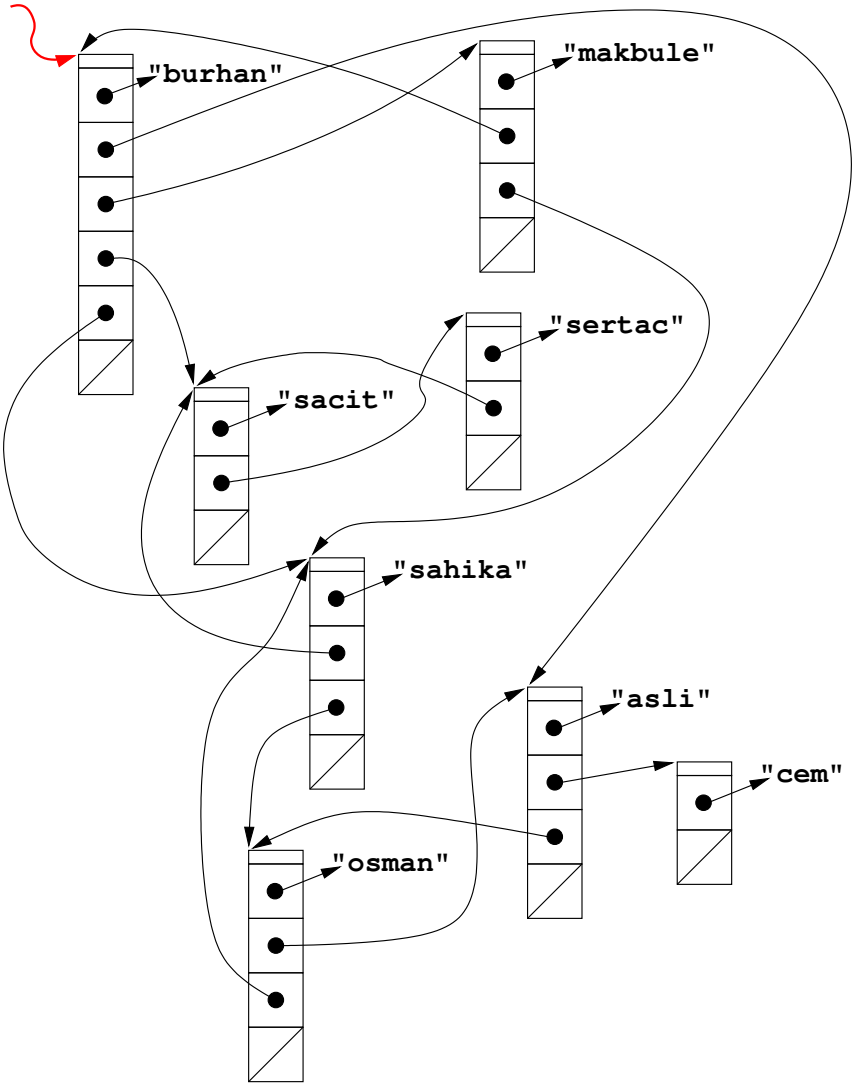


$GRAPH_1$



$GRAPH_2$

The merged graph is supposed to be:



$GRAPH_{merged}$

SPECIFICATIONS

The data structure that you are supposed to use is as follows:

- ```
  typedef
     struct vertex
        { char tag;
          char *label;
          struct vertex *edge[1]; }
     vertex, *vp;

  #define TAG(vp)   ((vp)->tag)
  #define LABEL(vp) ((vp)->label)
  #define EDGE(vp)  ((vp)->edge)
  ```

  The dynamic memory allocation technique that will be used to allocate space for a vertex in particular, has been explained in the lecture. You have to follow that technique.

- The function that you are going to define has a prototype of:

  ```
  vp merge(vp graph1, vp graph2);
  ```

  The function `merge` will take the pointers to supernodes of two graphs that conform with the descriptions/specifications and then return a pointer to the supernode of the newly created merged graph. This created data structure that corresponds to the merged graph has to comply with the descriptions/specifications.

  In `the3.c` you are expected to submit your definition of the `merge` function and your (helper) functions, if any, used by your `merge`.

- The supernode of the merged graph is always the supernode of the first argument.

- The newly created data structure will not use any pointer value from/to any input data structures. This includes the label strings as well. You have to duplicate them *(Aid:* `strdup`*)*. Even if you decide that a portion or all of the data structure of the input graph remains the same you are not allowed to use any pointer to these (similar) parts of the input.

- The labels are unique. In other words in the same graph no tow vertices will have the same label.

- Labels are strings of alphanumericals.

- No pointer that correspond to an edge of a vertex points to that vertex.

- There is a `tag` field of type `char` which you are free to use for any purpose. The checker program used for grading will not consider any data in this field.

- The labels of a vertex that are connected (are pointed) by an edge to a vertex are stored in an increasing lexicographical order. *(Observe the outgoing edges of* ***burhan*** *in the* $GRAPH_{merged}$ *for example. The first pointer points to the vertex of* ***asli****, the next is* ***makbule****, which is followed by* ***sacit****. The last one is pointing to the vertex of* ***sahika****).* The data structure that will be submitted as argument to `merge` will conform with this rule. The merged, fresh created data structure that will be returned by your implementation of merge has to conform with this rule as well. *(Aid:* `strcmp`*).*

- No array declaration is allowed. All such needs have to be fulfilled by dynamic memory allocations.

- You have to use the macros defined above. No subfield (dot) syntax, nor (`->`) is allowed in any function.
  ```

- Your code that you will turn in may include helper functions. But carefully **refrain** from defining the `main` function. If you do so the test program will not compile (because it is our test program that will provide the `main` function) and you will be graded zero.
- The lines that define the `structure` for the vertex and the `define` macros are provided as a header file `the3.h` obtainable from:
  http://ceng.metu.edu.tr/courses/ceng140_3/the3.h
- The use of the `TAG` and `LABEL` macros are straightforward. Note that `EDGE` can be used indexed. Here is an example: `EDGE(p)[2]` . *(This is just a possibility and not a necessity)*