Homework 4

Learning Outcomes

After completion of this assignment, you should be able to:

- Understand the concept of structures.
- Implement graph algorithms.
- Write Functions and manage the call stack.

Getting Started

To complete this homework assignment, you will need the MARS simulator. Download it from Blackboard. You can write your programs in the MARS editor itself. You can choose to use other text editors if you are not comfortable with the MARS editor. At any point, if you need to refer to instructions click on the Help tab in the MARS simulator.

Read the rest of the document carefully. This document describes everything that you will need to correctly implement the homework and submit the code for testing.

You should have already setup Git and configured it to work with SSH. If you haven't then do Homework 0 first!

The first thing you need to do is download or clone this repository to your local system. Use the following command:

```
$ git clone <ssh-link>
```

After you clone, you will see a directory of the form *cse220-hw4-<...>*, where username is your GitHub username.

In this directory, you will find *hw4.asm*. This file has function stubs that you will need to fill up. You can define your own helper functions if required as well. At the top of the file you will find hints to fill your full name, NetID, and SBU ID. Please fill them up accurately. This information will be used to collect your scores from GitHub. If you do not provide this information, your submission may not be graded. The directory also has a template test file ending with *hw4_test.asm*. Use the file for writing your test cases. You can change the data section or the text section in this file to test different cases for each part (described later). You may also create your own *_test.asm* files if necessary. Don't push these additional *_test.asm* files to the repository.

Note the hw4.asm file doesn't have a .data section. Do not add a .data section.

Don't forget to add your name and IDs at the top of hw4.asm. Follow the exact format, i.e, replace the hints with the correct information. Remember to submit your username to Blackboard. You will be penalized if you do not follow the instructions.

Assembling and Running Your Program in MARS

To execute your MIPS programs in MARS, you will first have to assemble the program. Click on the *assemble option* in the *Run* tab at the top of the editor. If the instructions in your program are correctly specified, the MARS assembler will load the program into memory. You can then run the program by selecting the *Go* option in the same *Run* tab. To debug your program, add breakpoints. This is done after assembling the program. Select the execute tab, you will see the instructions in your program. Each instruction will have a checkbox associated with it. Clicking on the checkbox will add a breakpoint, that is, when the program is run, control will stop at that instruction allowing you to inspect the registers and memory up to that point. The execute tab will show you the memory layout in the bottom pane. The right-hand side pane shows the list of registers and their values.

Always assume that memory and registers will have garbage data. When using memory or registers, it is your responsibility to initialize it correctly before using it. You can enable the *Garbage Data option under Settings* in MARS to run your programs with garbage data in memory.

Test Cases

In this homework assignment, you will not be provided with executable test cases. It is your job to come up with test cases to verify the correctness of your implementation.

Problem Specification

In this assignment, we will learn how to build and manipulate a custom data structure using MIPS.

Mark Jobs has a brilliant idea to create a network of people in which nodes represent people and edges between nodes represent relationships between people. Mark's goal, through this network, is to capture various properties about people and their relationships. Further, he wants to be able to query the network

- for persons using their names and
- for persons related to each other as friends.

We need to help Mark build and manage such a network. To this end, we will construct and maintain a graph-based data structure called Network.

The Network Data Structure

The Network data structure uses two other structures called Node and Edge. We first define the structure of Node and Edge. The Node data structure represents a person with a name and is defined as follows:

```
struct Node {
  byte[N] name // string of N characters
}
```

The Edge data structure represents a relationship between two people. It is defined as follows:

```
struct Edge {
  Node e1          // reference to a person
  Node e2          // reference to a person
  int relation  // integer to indicate type of relationship
}
```

Notice that Edge does not contain the actual person nodes in a relationship, but a reference to the nodes that are related. The relation attribute/property is 1 if the relationship is a friendship. For other types of relationships (e.g., siblings) the property will be greater than 1. If the property is 0, it implies that the relationship type is not known.

We define *Network* as follows:

```
struct Network {
  int total_num_nodes     # max no. of nodes in the network
  int total_num_edges     # max no. of edges in the network
  int num_of_nodes        # The size of a node
  int size_of_edge        # The size of an edge
  int curr_num_nodes      # No. of nodes currently in Network
  int curr_num_edges      # No. of edges currently in Network
  Node[n] nodes           # n = total_num_nodes
  Edge[e] edges           # e = total_num_edges
}
```

Here is a brief description of the elements in Network:

1. *total_num_nodes* is a 4-byte nonnegative integer that represents the maximum no. of nodes that the network can hold. If this limit is reached, the network should not accept anymore people.
2. *total_num_edges* is a 4-byte nonnegative integer that indicates the maximum no. of edges that the network can hold. If the limit is reached, then the network should not accept anymore relationships.
3. *size_of_node* is a 4-byte nonnegative integer that indicates the size of a node. Essentially, this property represents the maximum length of a person's name. A person's name may not be null-terminated.
4. *size_of_edge* is a 4-byte nonnegative integer that denotes the size of an edge (i.e., a relationship between two people). Assume this size is always 12 as an edge is made of 3 entities each of size 4 — two person references and one friend attribute.
5. *curr_num_nodes* is a 4-byte nonnegative integer that represents the no. of nodes (people) currently in the network. Assume this number is never greater than *total_num_nodes*.
6. *curr_num_edges* is a 4-byte nonnegative integer that represents the no. of edges currently in the network. This number is never greater than total_num_edges.
7. *nodes* is a set of nodes (or people) currently in the Network. It is bound by *total_num_nodes*. A node will typically hold a person's name. But some people may like to stay anonymous. In the absence of a name, the node will contain 0s.
8. *edges* is a set of relationships between people currently in the Network. It is bound by *total_num_edges*.

Assume the *Network* structure will always begin at a word-addressable address. Further, assume that the edges set will begin at a word-addressable address. this implies that *total_nodes * size_of_node)* will always be a multiple of 4.

Assume nothing else about the *Network* data structure.

As an example, consider the following instantiation of Network in MIPS:

```
Network:
  .word 3   #total_nodes (bytes 0 - 3)
  .word 6   #total_edges (bytes 4- 7)
  .word 12  #size_of_node (bytes 8 - 11)
  .word 12  #size_of_edge (bytes 12 - 15)
  .word 0   #curr_num_of_nodes (bytes 16 - 19)
  .word 0   #curr_num_of_edges (bytes 20 - 23)
  # set of nodes (bytes 24 - 59)
  .byte 1 2 3 4 5 6 7 8 9 10 11 12 13 ...
  # set of edges (bytes 60 - 131)
  .word 1 2 3 4 5 6 7 8 9 10 11 12 13 ...
```

In this example, *Network* can have a maximum of 3 nodes and 6 edges. Further, each node can hold exactly 12 characters. This implies that every person in the network can have a name with a maximum length of 12. Every element in the set of edges is a collection of 3 properties -- the first two are node (or person) addresses and the third one is the relationship indicator. The total no. of bytes occupied by Network in this example is 4 + 4 + 4 + 4 + 4 + 4 + 36 + 72 = 132 bytes.

Notice how the set of nodes is really an array of strings, where the strings are names of people in the Network. Similarly, the set of edges is an array of references, where the references are addresses of person nodes in the network. Since both these sets are arrays, we can access every element in them by calculating an offset from the base address of the set (array indices start at 0). The base address of nodes and edges will be at a fixed location relative to the base address of Network.

We will use these data structures to define operations/functions that will help us implement the social network envisioned by Mark Jobs. Your task in this assignment will be to implement each of the defined functions.

Part 1: Create A Person

Node* create_person(Network* ntwrk)

The function *create_person* takes the address of *Network* (as defined above), creates a person node, adds it to the network, and returns the person node's address. Creating a person involves obtaining a reference (address) to the first free node in the Network's nodes set. Once we have the address of such a node, we initialize the node with 0s, increment the current no. of nodes in the Network by 1, and return the node address.

It is possible that the Network is at capacity, that is, no free nodes are available. In that case, the function should return -1.

In summary, the function takes the base address of Network as an argument and returns the address of a node in the network or -1 in register $v0.

For example, suppose we call *create_person(network)*. Also, suppose at the time of calling the Network structure is as shown below:

```
Network:
  .word 3   #total_nodes (bytes 0 - 3)
  .word 6   #total_edges (bytes 4- 7)
  .word 12  #size_of_node (bytes 8 - 11)
  .word 12  #size_of_edge (bytes 12 - 15)
  .word 1   #curr_num_of_nodes (bytes 16 - 19)
  .word 0   #curr_num_of_edges (bytes 20 - 23)
  # set of nodes (bytes 24 - 59)
  .byte 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 . . .
  # set of edges (bytes 60 - 131)
  .word 1 2 3 4 5 6 7 8 9 10 11 12 . . .
```

After the call, the function should return address(*Network*) + *36* in $v0 and the network should be as follows:

```
Network:
  .word 3   #total_nodes (bytes 0 - 3)
  .word 6   #total_edges (bytes 4- 7)
  .word 12  #size_of_node (bytes 8 - 11)
  .word 12  #size_of_edge (bytes 12 - 15)
  .word 2   #curr_num_of_nodes (bytes 16 - 19)
  .word 0   #curr_num_of_edges (bytes 20 - 23)
  # set of nodes (bytes 24 - 59)
  .byte 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 11 19 . . .
  # set of edges (bytes 60 - 131)
  .word 1 2 3 4 5 6 7 8 9 10 11 12 . . .
```

Note how the node at address(*Network*) + *36* is initialized with 0s and the current no. of nodes in 2.

As another example, suppose we call this function when the network structure is as shown below:

```
Network:
  .word 3   #total_nodes (bytes 0 - 3)
  .word 6   #total_edges (bytes 4- 7)
  .word 12  #size_of_node (bytes 8 - 11)
  .word 12  #size_of_edge (bytes 12 - 15)
  .word 3   #curr_num_of_nodes (bytes 16 - 19)
  .word 0   #curr_num_of_edges (bytes 20 - 23)
  # set of nodes (bytes 24 - 59)
  .byte 0 0 0 0 . . .
  # set of edges (bytes 60 - 131)
  .word 1 2 3 4 5 6 7 8 9 10 11 12 . . .
```

This time the function should return -1 in $v0 since the network is at capacity based on fact that the current no. of nodes in the network is the same as the total no. of nodes.

Part 2: Name a person

int add_person_property(Network* ntwrk, Node* person, char* prop_name, char* prop_val)

A person can have numerous properties. One such property is the person's name. This function *add_person_property* sets the name property of an existing person in the Network. It takes as arguments a reference to the network in $a0, a reference to a person node in $a1, the property name (a null-terminated string) in $a2, and the property value (a null-terminated string) in $a3. In this function we are only interested in the "NAME" property. Hence, we should ignore all other properties. To this end, this function should set the name of an existing person in the Network to the string *prop_val* if and only if:

1. *prop_name* is equal to the string constant "NAME", and
2. person exists in Network, and
3. The length of *prop_val* (excluding null character) <= *Network.size_of_node*, and
4. *prop_val* is unique in the Network.

The function should return 1 if the name property is added successfully to the person. It should return 0, otherwise. The return value should be in register $v0.

Part 3: Query Network

Node* get_person(Network* network, char* name)

The function *get_person* takes two arguments --a reference to Network in *$a0* and a string name in *$a1*. The string name is null-terminated. The function should return a reference (or address) to the person node in Network that has its name property set to the input argument name. If no such person is found, then the function should return 0. The return value should be in register *$v0*.

Part 4: Add Relationship

int add_relation(Network* ntwrk, char* name1, char* name2)

The function *add_relation* takes a reference to Network in *$a0* and two strings, *name1* and *name2*, in *$a1* and *$a2*. Both strings are null-terminated. If both *name1* and *name2* exist in the Network, then an edge between person1, with name *name1*, and person2, with name *name2*, should be added to the edges structure in network. The relation property of the edge must be 0. Recall an edge is defined in struct *Edge* outlined above. The function must return 1 if the relation was added successfully. It fails to add the relation if:

1. If no person with *name1* exists in the network, or
2. If no person with *name2* exists in the network, or
3. The network is at capacity, that is, it already contains the maximum no. of edges possible, or
4. A relation between a person with *name1* and a person with *name2* already exists in the network. Relations must be unique, or
5. *name1* == *name2*. A person cannot be related to themselves.

The function must return 0 if the relation could not be added to the network. The function must return values in register *$v0*.

Part 5: Add Friendship

int add_relation_property(Network* ntwrk, char* name1, Node* name2, char* prop_name, int prop_val)

The function *add_relation_property* takes as arguments a reference to Network in *$a0*, strings *name1* and *name2* in *$a1* and *$a2*. The strings are null-terminated. If the length of both strings is greater than the node size then ignore the additional characters. The null-terminated string (*prop_name*) is an argument in *$a3*. The function also takes a fifth argument, which indicates the value of the property being set.

It sets the relation property of an **existing** relation in the network to *prop_val*. The function returns 1 if the property value is added successfully and 0 if the property value was not added. The property value will be added successfully if and only if:

1. A relation between a person with *name1* and person2 with *name2* exists in the network, and
2. The argument *prop_name* == "FRIEND", and
3. The argument *prop_val* == 1.

The function should return 0 in register *$v0* if any of the above conditions are violated. The function must return values in register *$v0*.

Part 6: Distant Friends

int is_a_distant_friend(Network* ntwrk, char* name1, char* name2)

A person *p1* is a distant friend of *p2* if *p1* and *p2* are not immediate friends, that is, there is no edge between *p1* and *p2* but there is a path in the network connecting *p1* and *p2* and the path contains only one edge. For example, consider a network of 6 persons *p1*, *p2*, *p3*, *p4*, *p5*, *p6*. Suppose the friendships in the network are captured as follows:

```
    p1   p6
    /\   /
   /  \ /
  p2  p3
  /    \
 p5    p4
```

In this network, *p1* is distant friends with *p4*, *p5*, and *p6* since there is a path with more than one edge from *p1* to *p4*,*p5*, and *p6*. However, *p1* is not distant friends with *p2* and *p3* as there is at most one edge between *p1*,*p2* and *p1*,*p3*.

Note two people in the network are related to each other but may not be friends. For instance, in the network above, suppose *p1* and *p3* are siblings instead of friends then *p1* and *p4* are not distant friends even if there is a path connecting *p1* and *p4* via *p3*.

Assume the network passed as argument to the function is a connected graph.

The function *is_a_distant_friend* takes a reference to the network in *$a0* and two null-terminated strings indicating person names in *$a1* and *$a2*. The function returns 1 if a person with *name1* is a distant friend of a person with *name2* or vice-versa. It returns 0 if a person with *name1* is not a distant friend of a person with *name2*. Further, it returns -1 if *name1* or *name2* does not exist in Network. The return value must be in register *$v0*.

Submitting Code to GitHub

You can submit code to your GitHub repository as many times as you want till the deadline. After the deadline, any code you try to submit will be rejected. To submit a file to the remote repository, you first need to add it to the local git repository in your system, that is, directory where you cloned the remote repository initially. Use following commands from your terminal:

```
$ cd /path/to/cse220-hw4-<username>  (skip if you are already in this directory)

$ git add hw4.asm
```

To submit your work to the remote GitHub repository, you will need to commit the file (with a message) and push the file to the repository. Use the following commands:

```
$ git commit -m "<your-custom-message>"

$ git push
```

After you submit your code on GitHub, Enter your GitHub username in the Blackboard homework assignment and click on Submit. This will help us find your submission on GitHub.