|  |
| --- |
| CSCI204 |
| Space Exploration |
| Report |
|  |
| **Leon Tan** |
| **10/15/2011** |

|  |
| --- |
| (Interpreted) requirements of the program, Diagram illustrations of program design, Summary of implementation of each module in the program, Reflections on program development. |

(Interpreted) requirements of the program

1. Read in statistical data (via manual input)
2. Compute the likelihood of life / civilization evolving in each location
3. Prioritize a list of top 5 locations for flat-space exploration
4. Compute total distance travelled (based on priority in c)
5. Develop LocationData class
   1. store statistical data
   2. implements a formula to compute a location’s civilization index
6. Develop PointTwoD class
   1. store (x, y) coordinate info
   2. encapsulate a LocationData object
   3. Information stored within LocationData objects are classified secret and should not be stored / access elsewhere, other than via PointTwoD object
7. Develop MissionPlan class
   1. main driver class whose methods are called to start the program
   2. When started, it should print a menu providing the following functionalities
      1. Allow user to input statistical data
      2. Compute the civilization index (based on all statistical data entered so far)
      3. Print top 5 exploration destination (based on all location data available)
      4. Total travel distance (to explore all recommended destinations)

Diagram / Illustrations of program design

Additional objects (other than LocationData, PointTwoD and MissionPlan):

LinkedList



A linked list consist of a head node that points to a node.

Each node of the linked list contains an object and and pointer to a node.

The pointer of the last node has a null value.

Binary Search Trees

root

PointTwoD Object

left

right

PointTwoD Object

left

right

PointTwoD Object

left

right

A Binary Search Tree consist of a root node that points to a node.

A node contains a PointTwoD object and two pointers (to nodes).

Summary of implementation of each module in my program

Input statistical data

* Prompts the user to input the data
* After entering data, each entry is checked for errors.
  + If there are errors, program tells the user where the error lies
  + If there are no errors, the information is stored in a PointTwoD object and added into a new node in a linked list named “entry”

Compute civilization index (for all records)

* Computes the civilization index in each PointTwoD object in each node of the linked list
* Transferring all data from linked list to the binary search tree while at the same time deleting the linked list
* The objects are inserted in to the tree according to their civilization index (lower civIndex to the left and higher civIndex to the right)
* Displays the number of nodes updated / transferred to tree

Printing top 5 exploration destinations

* Displays the total number of records available, that is the total number of records updated or the number of nodes in the tree (not the number of nodes in the linked list)
* Displays the top 5 nodes in the tree that contains the PointTwoD objects with the highest civilization index in descending order

Printing total travel distance

* Computes and displays the total travel distance to explore the top 5 exploration destination in the tree

Display civIndex and travel distance to and back from a location

* Prompts the user to enter the x and y coordinates of the location
* Checks if the location exist in the records
  + If it exist but record not updated, prompts the user to update all records first
  + If it exist and is updated, displays civIndex and travel distance to and back from the location
  + If it does not exist, display error message

Reflections on program development

**Difficulties faced**

Had some problems on deciding the most efficient and effective algorithm for the program. Wanted to use arrays initially but I feel that there could be a lot of redundancy in the memory usage as the memory allocation is not dynamic enough. For example, we do not know beforehand how many times the user wants to enter data and therefore could not set an optimum array size. Hence we would have to allocate a certain size larger than what we would expect the number of times the user enters correct data. This results in redundancy in memory usage. I preferred to use a more dynamic approach to store the data, therefore I chose to use linked list and binary search trees.

The assignment requires us to print the top 5 exploration destination sorted in descending order by Civ Idx value. This means that some sorting algorithm must be involved. If using arrays, I would visualize each PointTwoD object as a heavy chunk of memory (as it contains memory for more than one variable as well as methods) and it would take some energy for the computer to swap or move each chunk of memory. This is why I chose to insert the objects directly in a binary search tree where it is directly placed in order.

Initially I wanted to compute the civilization index directly after input so that I can store the input object directly into the binary search tree, however the assignment requires us to have an option to “Compute civ. index value (for all records)”. This means that we must have a separate function from the input code to specifically compute the civilization index for all entered records. This is the reason I created a linked list where each node stores a PointTwoD object that the user has entered. Upon updating, the civilization index in computed for each object in the node of the linked list, then the object is extracted from each node and placed in the binary search tree according to its civilization index, deleting the node in the process, hence the list would be empty after each update so it can be said to be like a temporary storing place.

The downside to this algorithm is that binary search trees make use of many recursive methods which may require more memory to process as compared to using a loop. There are instances using linked list where I can use a loop for the methods, however I chose to use recursion as it seemed more elegant to me.

**What could have been done better / possible enhancements in future**

Linked Lists and Binary Search Trees are only some of the few abstract data structure that I have learnt so far. There may be other kinds of ADT that I have yet to learn that could make the program more efficient.

**What I have learnt**

Using binary search trees, I have explored and learnt a few ways of using recursive methods to accomplish certain tasks, such as counting the nodes and printing the top 5 exploration destination.