CS4580 Advanced Data Structures

Project 1 (Individual Assignment)

1. General

1. Problem description

You are required to implement Dijikstra algorithm to find the length of a shortest path from a given source to each of the remaining vertices in an undirected graph, using leftist tree data structure as well as Fibonnaci heap, and measure the relative performance of the two implementations.

Both of the schemes MUST use the adjacency list representation for graphs.

In leftist tree you can assume decrease key operation as deleting the key and inserting new value to the tree.

2. Programming Environment

You may implement this assignment in Java or C++ or Python. You should be prepared to demonstrate your project.

2. Input Requirements

Running modes:

The name of your program should be dijikstra.cpp for C++ and dijikstra.java for Java. Your program MUST support all of the following modes.

(i) Random Mode:

Run with graphs generated by random number generator. The command line for this mode is:

$dijikstra -r n d x

// Run in a random connected graph with n vertices and d% of density. //the source node number is x.
// See Performance measurements section for details.

The following points should be noted in generating graphs in the random mode. Assume we have a function random (k) that returns a random integer in the range 0 to k-1. Also assume that the n vertices of a graph are labeled from 0 to n-1.

a. To generate an edge set i = random (n), j = random (n) and cost = random (1000) + 1, and add the edge into the graph when edge (i, j) is not in the graph.
b. After all edges are generated for a graph you need to make sure that the graph is connected.

You can do this by running a depth-first search on the graph. Repeat the process of generating graphs until the graph is connected.

(ii) User Input Mode:

user input mode- run with the operation sequence from user. In the user input mode, your program has to support redirected input from a file "file-name" which contains undirected graph representation. The command line for this mode is:

$dijikstra -l file-name // read the input from a file ‘file-name’ for leftist tree scheme

$dijikstra -f file-name // read the input from a file ‘file-name’ for Fibonacci heap scheme

In the user input mode, your program must get the following input format:

x //source node num

n m // The number of vertices and edges respectively in the first line

v1 v2 c1 // the edge (v1, v2) with cost c1

v2 v3 c2 // the edge (v2, v3) with cost c2

... // a total of m edges

Assume that vertices are labeled from 0 to n--1. An example input is shown below:

```
0
3 3
0 1 3
1 2 8
0 2 1
```

The graph consists of three vertices {0, 1, 2} and three edges (0,1), (1,2) and (0,2) with costs 3, 8 and 1 respectively.
In the user input mode, your program should display the cost of from each node to the other node in each line. Print the output to the standard output stream. The output for the example shown above is as follows:

```
0 //cost from node 0 to 0
3// cost from node 0 to 1
1 // cost from node 0 to 2
```

3. Performance Measurements

The performances will be measured only in the random mode. The experiment is to generate 6 connected undirected graphs with different edge densities (0.1%, 1%, 20%, 50%, 75%, and 100%) for each of the cases n= 1000, 3000, and 5000. Note that an undirected graph of n vertices can have at most n(n-1)/2 edges. For n = 1000, d = 0.1%, this may not form a connected

graph. Please skip this test case. For each graph assign random costs to the edges in the range between 1 and 1000. Measure the runtimes of the leftist tree scheme and the fibonacci-heap scheme on the same graph.

4. Submission

The following contents are required for submission:

1.Makefile: Your make file must be directly under the zip folder. No nested directories.

2. Source Program: Provide comments.

3. REPORT:
· The report should be in PDF format.
· The report should contain name and index number.
· Function prototypes showing the structure of your programs.
· A summary of result comparison: You should put first your expectation of the comparison before running your program: i.e. what you think about the relative performance of each scheme, and why.

4. Draw a plot or a table with execution time followed by comments whether it confirms your expectation. If different, why it is the case and state the factors that influence the results.

To submit, Please compress all your files together using a zip utility and submit to the Moodle. Your submission should be named IndexNo***.zip.***

5. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.
Correct implementation and execution: 60%
Comments and readability: 15%
Report: 25%
Note: ***If you do not follow the input/output or submission requirements above, 25% of your score will be deduced.***

6. Miscellaneous

- **Do not use complex data structures provided by programming languages**. You have to implement data structures using primitive data structures such as arrays and pointers. For example, linked list structure provided by libraries cannot be used. The reason is that most complex data

structures provided by libraries, e.g., C++ STL, give you flexibility with the cost of degrading performance in terms of running time.

- Your implementation should be your own. **You have to work by yourself for this assignment** (discussion is allowed). Program code will be checked and you may have negative result if it has reasonable doubt.

- Assume that all values are integers, i.e., key values are non-negative integers.

- **Measuring execution time (C++)**
  You can measure execution time like below:

```
#include <time.h>
clock_t Start, Time;
Start = clock();
.....  (your algorithm)
Time = clock() - Start;          // time in micro seconds
```

- **Measuring execution time (Java)**

```
long start = 0;
start = System.currentTimeMillis();\
.....  run your algorthim;
stop  = System.currentTimeMillis();
// execution time for your algorithm.
time = stop - start;
```