CS 3358
Data Structures and Algorithms, Fall 2021

Project 5

Due: Thursday, Nov 29 at 11:59 PM
Goal: implement and study a SSSP and MST Algorithms

All projects files in CS 3358 have to be submitted using Canvas. Note that files are only submitted if Canvas indicates a successful submission. This project has to be done individually. If the code is written poorly and not well commented, you will lose points. See the end of this handout for Code Requirements.

Support files are available on Canvas

5.1 SSSP Algorithms [50 points]

Write a C++ function called “BellmanFord” in pr5.cpp file (provided in Canavs) with the following prototype that implements BellmanFord algorithm for weighed edges using adjacency lists (i.e., for directed graphs).

```
static void BellmanFord(const int src, const ECLgraph& g, int* const
dist,int* const parent)
```

The vertices are numbered 0 through `size`-1. The `ECLgraph` has `nlist` an array that contains all the adjacency lists concatenated together. `eweight` contains the edge weights in the same format. Finally, `nindex` contains the starting indices of the individual adjacency and edge-weight lists. This is called the compressed sparse-row (CSR) format. For any vertex `v`, the indices `nindex [v]` ... `nindex [v+1]-1` refer to the entries in `nlist` and `eweight` that belong to `v`. Note that `nindex` has `size`+1 entries.

Do not implement Dijkstra‟s algorithm it’s provided for you but instead study the code and understand the format of ECLgraph. For BellmanFord, set all distances to infinity (INT_MAX) except the distance of the source node, which should be initialized to zero. Set all parent to null. Then run the following operations in a loop. For each vertex whose distance is not infinity, compute the distance to all adjacent nodes and update their distance if a new lower distance was found. Repeat these computations until a fixed point is reached, i.e., no distance was updated. Use the test drive in the provided pr5.cpp file to test your algorithm with the given graphs.

5.2 Prim Algorithm [50 points]

Write a C++ function called "MST_prim" in pr5.cpp file (provided in Canavs) with the following prototype that implements an

```
static void MST_prim(const int src, const ECLgraph& g, int* const
dist, int* const prev, int* const color)
```

The vertices are numbered 0 through size-1, and you will use the same ECLgraph format.

Steps for Prim:

1) Set all distances to infinity (`INT_MAX`) except the distance of the source node, which should be initialized to zero.
2) Create an empty priority_queue pq. Every item of pq is a pair (weight, vertex). Weight is used as first item of pair as first item is by default used to compare two pairs. Use Dijkstra algorithm as a reference.
3) Insert source vertex into pq and make its key as 0.
4) Then run the following operations in a loop
    a. Extract minimum key vertex v from pq
    b. Set v color to 1(i.e. in the MST)
    c. Loop through all the neighbors of v and do following
        i. If weight of edge (v,u) is smaller than dist of u and u is not already in MST (use color array to help you with this step)
            1. Update dist of u, i.e dist[u] = weight(v, u)
            2. Insert u into the pq
            3. Set parent[u] = v

Use the test drive in the provided pr5.cpp file to test your algorithm with the graph7.egr, your output should look like.

```
(1 ,  0) and dist 1

(2 ,  1) and dist 4

(3 ,  2) and dist 5

(4 ,  3) and dist 6

(5 ,  4) and dist 3

(6 ,  5) and dist 1

(7 ,  6) and dist 4

(8 ,  7) and dist 5
```

For this project you should submit 1 files (pr5.cpp)

Code Requirements

- Make sure your code compiles with g++ and runs on zeus.cs.txstate.edu.
- Make sure your code is well commented.
- Make sure your code does not produce unwanted output such as debugging messages.
- Make sure your code's runtime is not excessive.
- Make sure your code is correctly and consistently indented.
- Make sure you use a consistent coding style.
- Make sure your code does not exceed array bounds.
- Make sure your code does not include unused variables, unreachable code, etc.

Code Submission

- Make sure your code complies with the above code requirements before you submit it.
- Do not submit any unnecessary files (e.g., provided or generated files).
- Any special instructions or comments to the grader should be included in a "README" file.

You can submit your code as many times as you want before the deadline. Each new submission will erase any previous submission of the same project. Be sure to submit at least once well before the deadline.