Project: Maximum Bandwidth Path

TING-WEI SU (UIN:231003210)
willytwsu@tamu.edu
CSCE 629 Analysis of Algorithm

1. Introduction

In our CSCE 629 course, professor has introduced a problem on finding the largest path from source to destination in a given undirect weighted graph. The bandwidth of the path could be define by the smallest edge of all the path between the source and destination. This algorithm is well-known in all kind of network communciation system. Two main famous algorithm has been discussed along with the suitable data structure. Using Dijkstra without heap data structure, Dijkstra with heap data structure and Kruskal. In the following section, I will address the implementation details and the result.

2. Design and Implementation

The benefit of this structure, is to seperate the graph generation from the max-bw algorithm. Thus once, we generate the graph, we could pass the graph's address to the Graph object pointer inside the algorithm class, which is the MaxBWDijkstra or MaxBWKruskal class. Then authorize the class to read data from the graph and apply any algorithm on the graph to achieve our goal.

Class structure

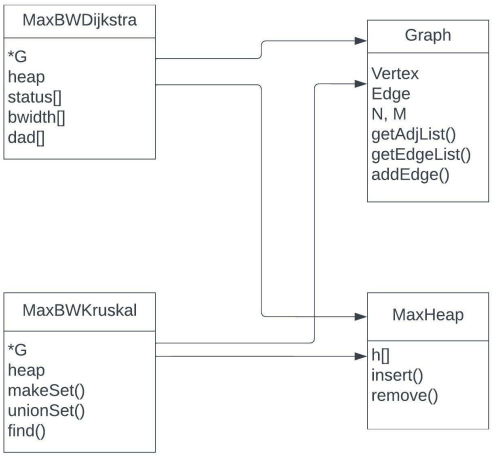

Figure 1: Class hierarchy

Dijkstra algorithm without heap

Dijkstra algorithm is based on greedy approach to find the minimal path. In this implementation, I simply use three array to keep track of the vertex status, the parent of the vertex and bwidth of the vertex. The status is a enum type of three state, unseen, fringer and intree. Whenever a source vertex is select, we will marked the vertex as intree and find its neighbor vertex and marked those as fringer. Then use a routine to continue check the vertex bwidth value to be smallest of all.

Dijkstra algorithm with heap

The heap data structure can be used to store the vertex in non-increasing order. However, we have to implement insertion and deletion by ourself. The insert operation could be divided into two action. The first is swap that will exchange the position of the last element in the heap with the element intended for deletion. The second is max_heapify_down that will reconstruct the elements position in order to maintain heap property after the swap operation. For insertion, we attach the new element at the end of the heap, then do the max_heapify_down to maintain the heap property.

```
class MaxHeap
{
private:
...
    void swap(struct node &a, struct node &b);
    void max_heapify_up(int i);
    void max_heapify_down(int i);
public:
...
    // Insert new element to heap
    void insertNode(int idx, int bw);
    // Remove an element from heap
    void removeMax();
};
```

Kruskal algorithm

Kruskal approach required us to first construct the maximum spannign tree in the graph first then do a DFS explore on the MST to find the maximum bandwidth path. Therefore, in order to construct the MST, a Union-Find-Makeset model is require to handle the set operation for we view the each vertex as independent set at the beginning. Constantly find the vertex from the largest bwidth, and do the union with neighbor vertex if not already added into the set. When all vertex united in a single set, the exploration has done, we have successfully construct the maximum spanning tree.

```
    // Disjoint set operation:
    // Create a set whose only member is x
    void makeSet(int x);
    // Unites the two set that contain x and y
    // Say S = Sx U Sy
void unionSet(int x, int y);
    // Find the root of x
int find(int x);
```

Graph generation

How to generate a randomly enough graph has composed a huge effort in this project. We have two type of graph to generate, with the first one comes with less edge and the later one being denser graph. I calculate the total number of graph edge needed at first, then

3. Test and Result

In the result table, we see that Dijkstra with heap has a absolute strength on finding the max-bw path with a better time complexity. However. Kruskal can do even better in some of the case. While in the denser graph, Kruskal did not perfom so well.

Graph type & Algorithm	Dijkstra w/o heap	Dijkstra w/ heap	Kruskal
Type 1	21217.1	291.745	30920
Type 2	14731.5	14353.1	685546

```
Dijkstra w/o heap on type 1 graph (s, t) = (901, 1317)
Bandwidth: 81
Time: 21217.1 us.

Dijkstra w/ heap on type 1 graph (s, t) = (901, 1317)
Bandwidth: 12
Time: 291.745 us.

Kruskal on type 1 graph (s, t) = (901, 1317)
Bandwidth: 1
Time: 30920 us.

Dijkstra w/o heap on type 2 graph (s, t) = (901, 1317)
Bandwidth: 98
Time: 14731.5 us.

Dijkstra w/ heap on type 2 graph (s, t) = (901, 1317)
Bandwidth: 98
Time: 14353.1 us.

Kruskal on type 2 graph (s, t) = (901, 1317)
Bandwidth: 1
Time: 685546 us.
```

4. Future Optimization

Graph randomness

Graph generation speedup

Better data structure

A 2-3 tree might be a good idea on optimizing the algorithm.