Final Project Report

1. Approch to Problem 1

主要是使用 Prim's Algorithm 以 cost 為條件做 minimum spanning tree，一開始會將 remaining bandwidth 小於 packet size 的 edge 拿掉得到一個 modified_G，接著對 modified_G 做 Prim's，得到最後的 MTid，處理 modified_G 時間複雜度為 O(E)，Prim's 時間複雜度為 $O(V^2)$(我是用 adjacency matrix 去實作)，整體來說空間複雜度為 $O(E + V + V^2)$。

2. Class for Problem 1

std::map<int, bool> fullConnected:
用來存放該 id 的 destination 有沒有全部被接完。

std::map<int, int> packetSize:
用來存放每個 id 要求的 packet size。

std::map<int, Tree> connectedTrees:
用來存放正在運行的 multicast tree。

bool cmpEdge(int e1[], int e2[]):
用來比較邊是否相同的函式。

void insert(int id, int s, Set D, int t, Graph &G, Tree &MTid):
先將 remaining bandwidth 小於 packet size 的 edge 刪掉得到 modified_G，接著對 modified_G 用 Prim's 以 cost 為條件做 minimum spanning tree，會得到一顆 multicast tree，最後將他存入 MTid，並保存到 connectedTrees 裡面，若沒完全接好會將 fullConnected[id]設成 false，以便 stop 利用，並且將 packetSize[id]設為 t，最後將 G 與 modified_G 比較，將 modified_G 裡面有更動的 edge，更新至 G 裡面。

void stop(int id, Graph &G, Forest &MTidForest):
先將 G 裡面 connectedTrees[id]有用到的邊的 remaining bandwidth 加回去 packetSize[id]，接著從最小的 id 開始嘗試重連，一樣先將 remaining bandwidth 小於 packet size 的邊刪去，並將兩端點都屬於 multicast tree 的 edge 刪去得到 modified_G，接著尋找其中一端點已經 visited 但另一個點還沒被 visited 的 edge，從最小 cost 的邊當開始，做 Prim's 去嘗試把剩下的 destination 接完，最後把有動過 trees 放進 MTidForest。

void rearrange(Graph &G, Forest &MTidForest):
先把 G 還原，然後從 id 最小的 request 開始重新呼叫 insert，最後將所有 active 的 multicast tree 放進 MTidForest。

3. Approch to Problem 2

大致上做法跟第一題一樣，只是換成了以 bandwidth 當條件的 maximum spanning tree，期望能以這種方法，使盡可能多的邊仍有 remaining bandwidth 可供使用，時間複雜度與空間複雜度均與第一題一樣，分別為 $O(E + V^2)$ 和 $(E + V + V^2)$。

4. Class for Problem 2

std::map<int, bool> fullConnected:
用來存放該 id 的 destination 有沒有全部被接完。

std::map<int, int> packetSize:
用來存放每個 id 要求的 packet size。

std::map<int, std::set<int>> destination:
用來存放各個 request 的 destination。

std::map<int, Tree> connectedTrees:
用來存放正在運行的 multicast tree。

Graph OriginalG:
用來存放原始的 G，以便 rearrange 使用。

bool cmpEdge(int e1[], int e2[]):
用來比較邊是否相同的函式。

```
std::map<int, int> globalPacketSize;
std::map<int, int> globalDestNum;
bool cmpByPacketSize(int a, int b) {
        return (globalPacketSize.at(a) * globalDestNum.at(a)
        globalPacketSize.at(b) * globalDestNum.at(b));
}
```

用來排序重連順序 id 的 function，讓 rearrange 時能從總最低需求(packetSize * destinationNumber)的 bandwidth 最低的 id 開始重連。

bool insert(int id, int s, Set D, int t, Graph &G, Tree &MTid):
作法與 Problem1 基本一樣，差別在於這題是以 bandwidth 為條件做 maximum spanning tree，沒有辦法成功連結到所有 destination 的 tree 會直接被設為空的 MTid，並將它保存在 connectedTrees 裡面。

void stop(int id, Graph &G, Forest &MTidForest):
將 G 裡面 connectedTrees[id]有用到的 edge 還原，接著按照上面的 cmpByPacketSize 排序，按此順序將沒接好的 request 全部嘗試 insert 一次，並將有更動的 multicast tree 放入 MTidForest。

void rearrange(Graph &G, Forest &MTidForest):
先將 G 設成 Original_G，接著將所有 request 按照上面的 cmpByPacketSize 排序，全部重新 insert 一次，並將結果放入 MTidForest。

5. Test Case Design

Problem 1:
我使用 cpp 檔生成一個 100 個 vertices、4950 條 edges 的 Graph，edges 的 bandwidth 與 cost 使用 rand()生成，接著有 100000 行 request，其中每個 insert 的 packet size 都是 1，希望能以此方式增加需要接起來的樹，以此篩選掉執行效率不佳的 code。

Problem 2:
根據助教的算分規則，我的策略是盡可能地讓大家 return false 以增加我測資的難度，因此我將 graph 設成一直線，bandwidth 設成 10，而第一個 request 為 insert 一棵 packet size 為 10 的樹，然後 destination 為所有的點，這種方法基本上可以讓大家獲得 99999 筆 return false。

6. Reference

Minimum Spanning Tree：Prim's Algorithm (alrightchiu.github.io)

7. 心得

我覺得這次的 final project 的期末報告其實還挺有趣的，因為我這學期還有修計網概，能用 graph 模擬封包傳輸我覺得非常好玩，不過可能是因為這是第一次規劃 project，可能在設計上會有一些瑕疵，不過經過大家一起努力，這份 project 也越來越完整了，謝謝你們設計了這份 project，讓我獲益良多。