2023 OOP&DS　Homework 2

111550129,林彥亨

Part. 1 (20%):

1. Implementation

   (1)

   在這步驟我做一個 creatGraph 的 struct。

   裡面定義以下三個函式

   - creatGraph(int)
   - void addEdge(int,int,int)
   - int kruskal()

```
 7  typedef pair<int, int>_pair;
 8
 9  struct creatGraph
10  {
11      int Vec;
12      int Edg;
13      vector<pair<int, _pair>> _edge;
14
15      creatGraph(int Vec){
16          this->Vec = Vec;
17      }
18      void addEdge(int u, int v, int w){
19          _edge.push_back({ w,{u,v} });
20      }
21      int kruskal();
22  };
```

(2)

在這段程式碼我定義一個 DisjointSets 的 struct

我用

*parent 存父節點的 data

*_rank 存每個 vertex 的 rank 值

n 代表頂點的數量

在這個 Disjointsets 的 struct 中包含了

- DisjointSets(int)
- Int searchVertex(int)
- Void merge(int,int)

```
24  struct DisjointSets
25  {
26      int* parent;
27      int* _rank;
28      int n;
29
30      DisjointSets(int n){
31          this->n=n;
32          parent=new int[n + 1];
33          _rank=new int[n + 1];
34
35          for (int i = 0; i <= n; i++){
36              _rank[i]=0;
37              parent[i]=i;
38          }
39      }
```

```
41      int searchVertex(int u){
42          if (u != parent[u]){
43              parent[u] = searchVertex(parent[u]);
44          }
45          return parent[u];
46      }
47
48      void merge(int x, int y){
49          x = searchVertex(x), y = searchVertex(y);
50          if (_rank[x] > _rank[y]) {
51              parent[y] = x;
52          }
53          else {
54              parent[x] = y;
55          }
56          if (_rank[x] == _rank[y]) {
57              _rank[y]++;
58          }
59      }
60  };
```

(3)

在這段程式碼中

- 主要是在寫 kruskal 演算法
- 利用 minmum spanning tree(mst)計算其權重最小且步行成迴路的路徑

```
62  int creatGraph::kruskal()
63  {
64      int pathWeight=0;
65      sort(_edge.begin(), _edge.end());
66
67      DisjointSets _disjointset(Vec);
68
69      vector<pair<int,_pair>>::iterator it;
70
71      for (it = _edge.begin(); it != _edge.end(); it++){
72          int u = it->second.first;
73          int v = it->second.second;
74
75          int set_u = _disjointset.searchVertex(u);
76          int set_v = _disjointset.searchVertex(v);
77
78          if (set_u != set_v){
79              pathWeight+= it->first;
80              _disjointset.merge(set_u, set_v);
81          }
82      }
83      return pathWeight;
84  }
```

(4)

此段程式碼

- 輸入城市數量
- 宣告一個 graph 去存取路徑的資料
- 輸入各個 edge 的資料
- 利用以上寫好的函式和 struct 去完成任務。

```
86  int main(){
87      int N;
88      int u, v, w;
89      cin >> N;
90      creatGraph g(N);
91      while (cin >> u >> v >> w) {
92          g.addEdge(u,v,w);
93      }
94      int pathWeight = g.kruskal();
95      cout <<pathWeight;
96      return 0;
97  }
```

2. Time complexity
   - creatGraph 的時間複雜度是 O(1)
   - Disjointsets 的時間複雜度是 O(N) (因為有一個 for 迴圈)
   - addEdge 的時間複雜度是 O(E)(因為輸入 E 個 edge 的數量)
   - kruskal 的時間複雜度是 O(ElogE)
   - MST 的時間複雜度是 O(1)

   因為我使用 kruskal 演算法所以時間複雜度主要會是 O(ElogE)，若計算總時間複雜度為 O(ElogE+N)，最後可以化簡為 O(ElogE)的時間複雜度。

3. Challenges/ discussion

   在研究這題的過程中，我發現可以利用老師上課時交的 Kruskal 演算法，因為此演算法較適合應用在以 edge 為主的情況之下。

   在時做此演算法的時候，我還學會了去使用 disjointset 的實作。在未來我希望，可以去嘗試使用不同的 sorting 方法去實作 kruskal 的演算法。

Part. 2 (20%):

1.Implementation

(1)

此段程式碼中，我定義了一個 struct edge 裡面包含

- dest---終點
- width----道路的寬度
- t_truck----卡車通過的時間
- t_car----車子通過的時間
- t_bike----腳踏車通過的時間

```
8  struct edge {
9      int dest;
10     int width;
11     int t_truck;
12     int t_car;
13     int t_bike;
14 };
```

(2)

此段程式碼，我定義 shortestTime 的函式，我利用 Dijkstra 演算法

- n 表示城市數量
- distance 表示城市間移動的最短時間
- visited 表示訪問過的 city
- 建立一個 min heap 去存城市到城市比較後得到的最短時間
- 進入 for 迴圈裏面，去訪問各個 city 然後比較從城市到城市間的 roadWidth 和各個交通工具的 size 是否可以通過。
- 最後會回傳從初始城市到終點城市的最短時間。

```
16  int shortestTime(vector<vector<edge>>& graph, int startCity, int endCity, int s_truck, int s_car, int s_bike) {
17      int n = graph.size();
18      vector<int> distance(n, INT_MAX);
19      vector<bool> visited(n, 0);
20      distance[startCity] = 0;
21
22      priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
23      pq.push({0, startCity});
24
25      while (!pq.empty()) {
26          int currentCity = pq.top().second;
27          pq.pop();
28
29          if (visited[currentCity]) {
30              continue;
31          }
32
33          visited[currentCity] = true;
34
35          for (const edge& _edge : graph[currentCity]) {
36              int neighborCity = _edge.dest;
37              int roadWidth = _edge.width;
38              int truckTime = _edge.t_truck;
39              int carTime = _edge.t_car;
40              int bikeTime = _edge.t_bike;
41
42              int costTime = 0;
43              int roadweight = 0;
44
45              if (roadWidth >= s_truck) {
46                  costTime = truckTime;
47                  roadweight = s_truck;
48              } else if (roadWidth >= s_car) {
49                  costTime = carTime;
50                  roadweight = s_car;
51              } else if (roadWidth >= s_bike) {
52                  costTime = bikeTime;
53                  roadweight = s_bike;
54              } else {
55                  continue;
56              }
57              int newDistance = distance[currentCity] + costTime;
58              if (newDistance < distance[neighborCity]) {
59                  distance[neighborCity] = newDistance;
60                  pq.push({newDistance, neighborCity});
61              }
62          }
63      }
64      return distance[endCity];
65  }
```

(3)

此段程式碼中

- 先宣告了一個二維的 vector 矩陣 graph
- 輸入有關起始城市、終點城市、路寬、個交通工具的時間，將其資料存在 graph 中。
- 輸入各種交通工具的 size。
- 輸入要輸入的測資數量
- 最後利用 shortesTime 函式中得到結果。

```
67  int main() {
68      int n,m;
69      cin >> n >> m;
70
71      vector<vector<edge>> graph(n);
72      for (int i = 0; i < m; i++) {
73          int startCity, endCity, width, t_truck, t_car, t_bike;
74          cin >> startCity >> endCity >> width >> t_truck >> t_car >> t_bike;
75          startCity--;
76          endCity--;
77          graph[startCity].push_back({endCity, width, t_truck, t_car, t_bike});
78          graph[endCity].push_back({startCity, width, t_truck, t_car, t_bike});
79      }
80
81      int s_truck, s_car, s_bike;
82      cin >> s_truck >> s_car >> s_bike;
83      int p;
84      cin >>p;
85
86      while (p--) {
87          int startCity, endCity;
88          cin >> startCity >> endCity;
89
90          startCity--;
91          endCity--;
92
93          int ans = shortestTime(graph, startCity, endCity, s_truck, s_car, s_bike);
94          cout << ans << endl
95      }
96
97      return 0;
98  }
```

2.Time complexity

- 讀取 input 的時間複雜度為 O(1)。
- 在 while 迴圈的執行次數取決於優先佇列 pq 的大小，而 pq 的大小最多是節點數 n，這個迴圈的時間複雜度是 O(n log n)。
- 此迴圈中迭代每個節點。在這個迴圈的時間複雜度是 O(n)。
- 主要的 while 迴圈執行 n 次，迴圈本身的時間複雜度是 O(n log n)，而迴圈主體的時間複雜度是 O(d)。因此，整段程式碼的時間複雜度可以表示為 O(n log n + m), 其中 m 是邊的總數。
- 總結而言，利用 Dijkstra 演算法，時間複雜度是 O(n log n + m)，其中 n 是 vertex 數量，m 是 edge 的總數。

3.Challenges/ discussion

在研究這個題目的時候，weight 大於 0，然後找最短路徑，我以為是用 prim 或是 kruskal 演算法去完成這個題目。

然而，和同學討論之後，發現應該要用 Dijkstra 演算法。在實作 Dijkstra 演算法的時候，老師在講解其步驟時候，我認為會蠻容易，但是實做面上有點困難。而針對 vertex 數量非常大的情況下，可能需要考慮使用其他的優化方法，min heap 來加速最短路徑的計算。另外，我在這寫這題的過程中，同學告訴我可以利用 priority queue 來做 min heap。