Programming Assignment #3 Report

B11901100 電機二 王竣平

There are three kinds of graph instances for the cycle breaking problem in the assignment:

1. unweighted undirected graph
2. weighted undirected graph
3. weighted directed graph

Since the input file of unweighted undirected graph assigns the edges to have weight = 1, we can simply see it as the same kind of graph instance as weighted undirected graph, but with all edges having weight = 1.

Overall, we use two classes to implement our idea, class Edge and class UD_node. (names are just names, UD_node functions properly in directed edges too.) In class Edge, we store its weight, its starting node and ending node, and whether it has been chosen. In class UD_node, we store its index, rank and parent (for union and findset operations), and its adjacency list (using vector of class Edge to realize it). Findset operation is a member function of UD_node. Here, we look into two different cases:

Case A: undirected graph

For undirected graphs, to break edges so the graph can be acyclic, we use the idea of minimum spanning tree, which has no cycle in the results. However, it is required that we yield the answer of the minimum total weight of cut edges. Hence, we do the same thing as MST except we want the edges of more cost to stay. By implementing Kruskal's algorithm, but adding edges in decreasing order of weight, we can achieve our goal.

Using the pseudocode in lecture slide to help present the idea:

```
KRUSKAL_MODIFIED(G,w)
1   A=0
2   for each vertex v ∈ G.V
3       MAKE-SET(v)
4   sort the edges of G.E into nondecreasing order by weight w
5   for each edge (u,v) ∈ G.E, taken in decreasing order by weight
6       if FIND-SET(u)≠FIND-SET(v)
7           A=A∪{(u,v)}
8           UNION (u,v)
9   return A
```

Note that sorting G.E into nondecreasing order is fine, since we take them in decreasing order by weight in line 5.

Here, we use the sort function in the header <algorithm> to do the sorting job for us. Moreover, we implement path compression at line 6 and union by rank at line 8. The ideas are almost the same as the slide, so no pseudocode presented here.

In the submitted code, we initialize the graph then call the Kruskal function to obtain the minimum cut weight and check_and_print function to print out all cut edges. The time complexity is O(ElgE), dominated by the sorting part.

Case B: directed graph

To achieve a less total cost of cut edges, we want: as many negative edges cut as possible, and as less positive edges cut as possible. Note that this is only a try to get a small total cut weight, the result may not be the best, but it is correct in terms of acyclic and connected requirement. The idea is, to see the graph as an undirected one first and do Kruskal on it (the modified kind mentioned in case A). Then we add each edge to see if it causes a cycle (edges with positive weight only). If it does, then remove it; keep it otherwise. We check if a cycle exists by depth first search.

The whole process is similar to that of case A but adding the “put back positive edges” part. The undirected Kruskal takes O(ElgE), and each DFS takes O(V+E). There are O(E) edges to check, yielding the total time of O(ElgE) + O(E(V+E)) = O(E(V+E)). Since usually V∈O(E), so the total time complexity is approximately O(E^2).

The data structure used in the assignments are array and disjoint set. Arrays are used to store the nodes (vertices), edges, and elements in a vertex’s adjacency list. The adjacency list is implemented by vector since it supports more convenient and useful operations. In disjoint set, a tree-like structure is implemented to support the Find-set operation with path compression, and Union by rank.