Cycle Breaking

Author: 陳郁安
Student ID: B10703104

A. Cycle Breaking for Undirected Graph.

a. Idea

Let us denote the undirected graph as G = (V, E).
If all the edge is positively weighted, then the minimum cycle breaking (MCB) problem is equivalent to finding maximum spanning tree.
If there's any edge negatively weighted, we first find maximum spanning tree, then those edges which are excluded from the MSP or which are negatively weighted (we can take as many negatively-weighted edges as possible to minimize to total weight) form MCB set.

b. Brief & Ideal Pseudo Code.

```
MCB_UG(G)
1.  initialize an empty queue --- solution
2.  MSP = Prim(G)
3.  for each edge e in G.E
4.    if (e not in MSP) or e.weight < 0
5.      solution.enqueue(e)
6.  return solution
```

c. Data structure used

Here I use 3 self-design classes: UD_vertex, UD_edge and UD_Graph (UD means undirected), all of which contains pretty much every attribute we need to solve the problem faster and make the code look cleaner.

Another data structure I use here is priority_queue of C++ standard library, in order to do Prim algorithm.
This data type provides some useful function, such as
a. Using push() to insert a queue
b. Using pop() to pop out the first-priority element, and use top() to read it
c. I can design an operator myself, to define how the priority is managed.
But priority_queue has one downside; that is, when we change the key value of the element inside, the queue does not update itself. Therefore, there is no way to do the

“Increase Key” (“Increase because we are finding maximum spanning tree”) operation for the heap used in Prim with this data type.
However I figure out another way. That is, we don’t care about this flaw. Instead, we just push the vertex into the queue again whenever its key value has changed. Since the new key value must be larger than the old key value, the updated vertex must be popped out earlier than the old one. So, we just need to record that which of the vertices has been popped out once (bool visited), if it has been popped out before, then we do nothing when it’s popped out again, since we have dealt with it already.

B. Cycle Breaking for Directed Graph.

a. idea

Let us denote the directed graph as G = (V, E).
To find the MCB set of G, we can first extract all negative weight edge from G into MCB set, since there can’t be too many negative weight edge in a MCB set.
Next step, we decompose G into several SCCs. We can ignore the edge connecting between SCCs, since it must not be included in a cycle, and it must be positively-weighted (negative weight edges are all extracted). Now, we just need to find the MCB set in each SCC.
In each SCC, we find MCB set by trying every permutation of vertex order, and the edge (u, v) should be removed if v is previous to u. Lastly, we find the permutation in which the removals are least costly in total.

b. Data Structure Used

```
MCB_DG(G)
1.  initialize an empty queue --- solution
2.  for each edge e in G.E
3.    if e.weight < 0
4.      extract e from G
5.      solution.enqueue(e)
6.  SCC_Set(G) = Decompose(G)
7.  for each SCC in SCC_Set(G)
8.    initialize vector opt_perm = {0, 1, 2 … SCC.V.size()}
      // opt_perm[vertex_index] = order
9.    opt_weight = INF;
10.   for each permutation perm from opt_perm
11.     weight = 0
12.     for each edge (u,v) in SCC.E
```

```
13.       if (perm[u.index] > perm[v.index])
14.         weight += w(u, v)
15.     if (weight < opt_weight)
16.       opt_weight = weight
17.       opt_perm = perm
18.   for each edge (u,v) in SCC.E
19.     if (perm[u.index] > perm[v.index])
20.       solution.enqueue((u, v))
21. return solution
```

c. Data Structure Used

Again, here I also use 3 self-design classes: D_vertex, D_edge and D_Graph (D means directed), all of which contains pretty much every attribute we need to solve the problem quicker.