HW4

P1

The problem ask us to find the minimum path of an undirected graph, however, we need to must go through some edges. In the beginning, I try to do it through dijkstra's method. The way is simple, list all the possible sequence of the order of must-go paths, then we must have one optimal solution, then, we will start at the starting point and moving to the following points one-by-one, i.e, we go from point a to the next must-go edge u-v, we compare the distance between au and av, then choose the path then go through uv or vu. Unfortunately, the problem has a serious problem: for the next must-go edges, it is not the case. Consider the case the distance between the second next must-go edges (3, 4) from the first must-go edges (1, 2) is (1, 3, 1), (2, 4, 999), (2, 3, 999), and for start point s, (s, 1, 1), (s, 2, 2), if we do it that way, we will go through path s, 1, 2, 4, however, it is absolutely not the shortest path, so the dijkstra's method failed. Then, I choose Floyd's method, finding out the distance between every two nodes respectively, though time complexity is n^3, the time limit is just fine. Then we again, try all the possible orders of the nodes and find the minimum one, but I do not have the correct answer in sone cases. It confuses me a lot, and I find a serious mistake, I

directly calculate the distance between must-go edges from the Floyd's method, which in some case will not go through the must-go edges! Problem solved, we obtain the correct program. In fact, despite out n^3 time complexity, if our q is super large in this case, we may be quicker than the other algorithm since we make a good use of the matrix. Besides, the thinking process of the program is truly easy.

P2

The question has 2 main points: i) the graph will change, ii) find the max_minimum_edges. For the first points, very easy, we can either use a queue or a list to update since there is no way to promote time efficiency for the changing start point and end point. For the second point, it's a little be difficult, so I decide to rewrite an algorithm by myself. First we have to find the possible paths we can go through. We choose the BFS method, since we have to find max minimum edges, and we have introduce new variables *mid* to store the value of current minimum value, and the process, using binary search, is similar to BFS rather than DFS (actually, I had trouble implement this using DFS). Then we can go to max_min_path() function. As we said previously, we use mid for the current path, is the path is available, it updates the value and move to the lower bounds, searching for a larger minimum

number. If not, it moves to the lower bound to find that. By repeating the search, we can find the results. And applying the update into the program, the problem is easily solved. I think the advantage of my problem is that the time complexity, that is ElogV, very efficient in comparison with the time limit.