Student Name: Yinsheng Dong
Student Number: 11148648
NSID: yid164
CMPT360 Assignment 4

Question 1.

Reduce the maximum flow problem.

In this question, the flow network N which all of the edges have capacity equal to one is given. The integer k is given. We want to find the maximum flow in N which is reduced by deleting k of the edges from N.

Solution:

To solve this question, we need to declare:

Assume that N (V, E) which V represents the vertices (nodes) and E represents the edges in flow network N.

Let n be the value of the maximum flow of network N.

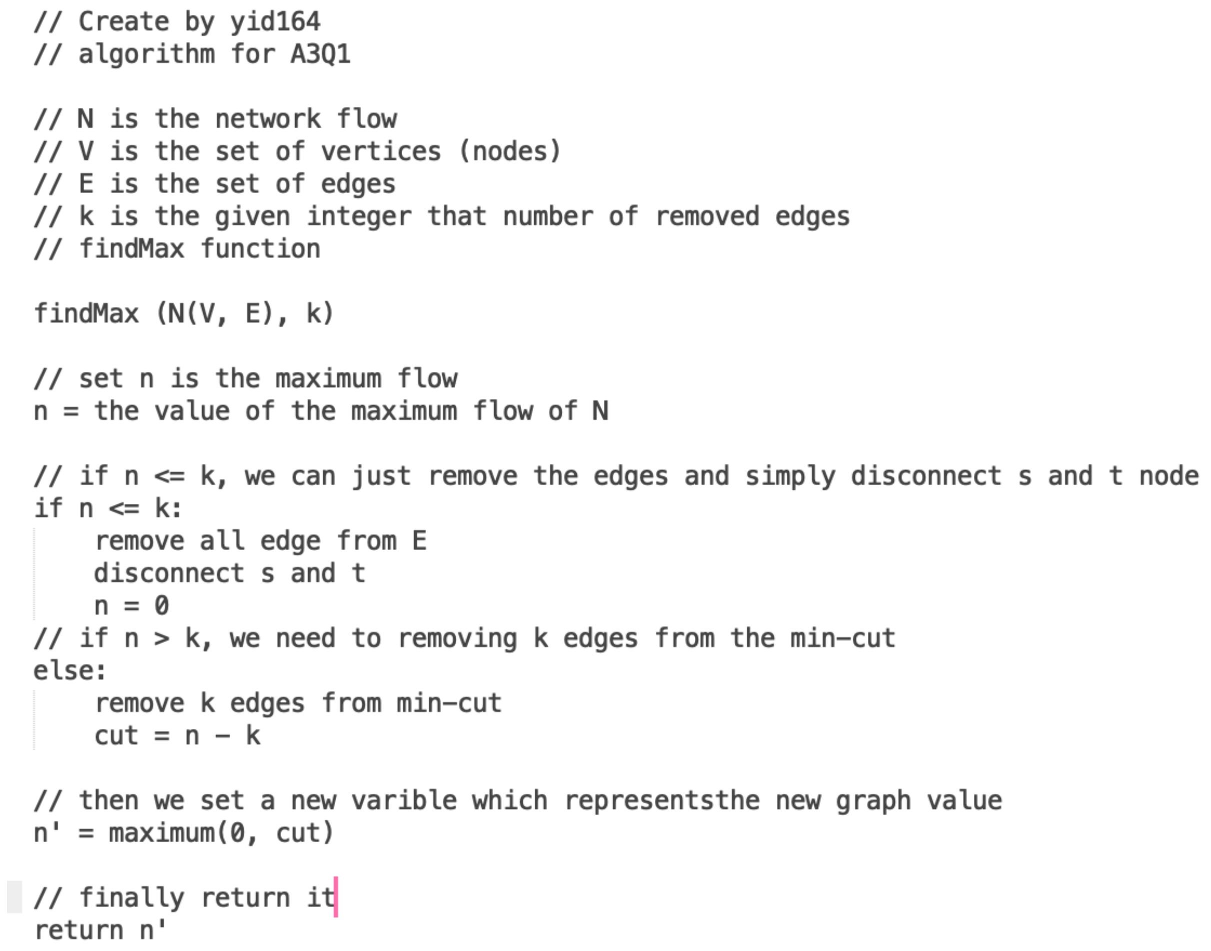
We know that the result cannot be less than n – k because the edges capacity equal to one. According to the Max-Flow Min-Cut Theorem. Then we got 2 conditions:

1. If n <= k, then we can just remove all edges in that cut, and disconnecting s and t in N, then the maximum flow will be 0.

2. If n > k, then we need to remove k edges from the min-cut, we create a cut with value n-k

According the statement above, we can claim that the min-cut in the new graph has the value of n' = max (0, n-k). Then based on Max-Flow Min-Cut Theorem, the max-flow of new graph is n'.

The Algorithm Code: (in next page)



```
// Create by yid164
// algorithm for A3Q1

// N is the network flow
// V is the set of vertices (nodes)
// E is the set of edges
// k is the given integer that number of removed edges
// findMax function

findMax (N(V, E), k)

// set n is the maximum flow
n = the value of the maximum flow of N

// if n <= k, we can just remove the edges and simply disconnect s and t node
if n <= k:
    remove all edge from E
    disconnect s and t
    n = 0
// if n > k, we need to removing k edges from the min-cut
else:
    remove k edges from min-cut
    cut = n - k

// then we set a new varible which representsthe new graph value
n' = maximum(0, cut)

// finally return it
return n'
```

Time Complexity:

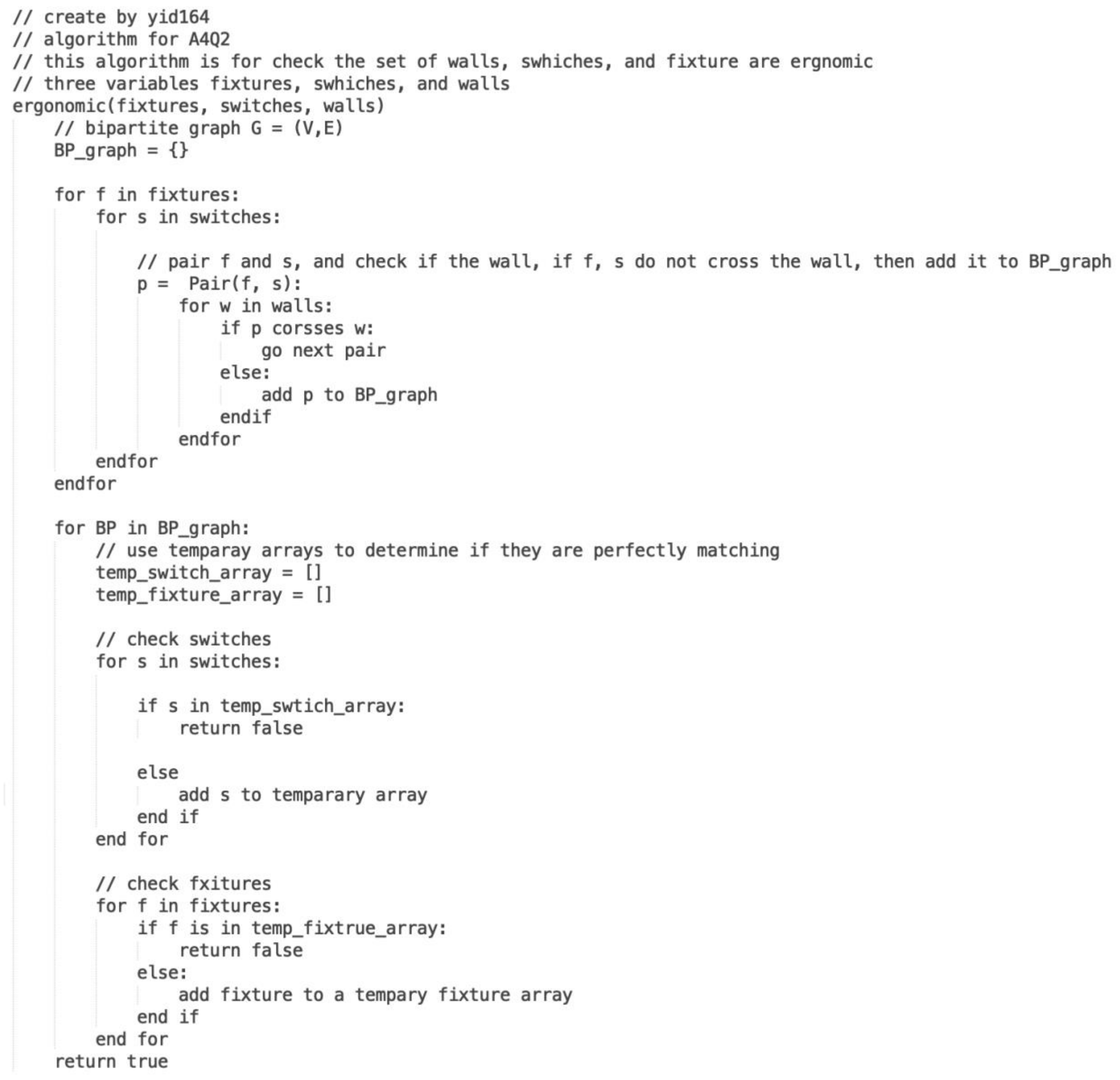
This algorithm's running time is polynomial because the minimal s – t cut represents polynomial time, and to remove the k edge needs linear in k time.

Question 2.

Kleinberg and Tardos page 416 #6

The Ergonomic Architecture Commission problem.
The question has given multiple variables which are fixtures, switches and walls. We want to use these variables to determine if they are ergonomic or not.

Algorithm design:



```
// create by yid164
// algorithm for A4Q2
// this algorithm is for check the set of walls, swhiches, and fixture are ergnomic
// three variables fixtures, swhiches, and walls
ergonomic(fixtures, switches, walls)
    // bipartite graph G = (V,E)
    BP_graph = {}

    for f in fixtures:
        for s in switches:

            // pair f and s, and check if the wall, if f, s do not cross the wall, then add it to BP_graph
            p =  Pair(f, s):
                for w in walls:
                    if p corsses w:
                        go next pair
                    else:
                        add p to BP_graph
                    endif
                endfor
        endfor
    endfor

    for BP in BP_graph:
        // use temparay arrays to determine if they are perfectly matching
        temp_switch_array = []
        temp_fixture_array = []

        // check switches
        for s in switches:

            if s in temp_swtich_array:
                return false

            else
                add s to temparary array
            end if
        end for

        // check fxitures
        for f in fixtures:
            if f is in temp_fixtrue_array:
                return false
            else:
                add fixture to a tempary fixture array
            end if
        end for
    return true
```

In this algorithm, I used 2 loops. The first one is for creating the bipartite graph which contains pairs of switches and fixture, and I used the walls to determine if the pair should add to the bipartite graph or not. The second loop is for checking the floor plan by looping the bipartite graph.

The complexity of this algorithm:

The first loop will cost O(n^2 * m) which n is the number of switches and fixtures, and m is the number of walls.

The second loop will cost O(n) because it runs sequentially.

Then the totally time complexity is O(n^2 * m)

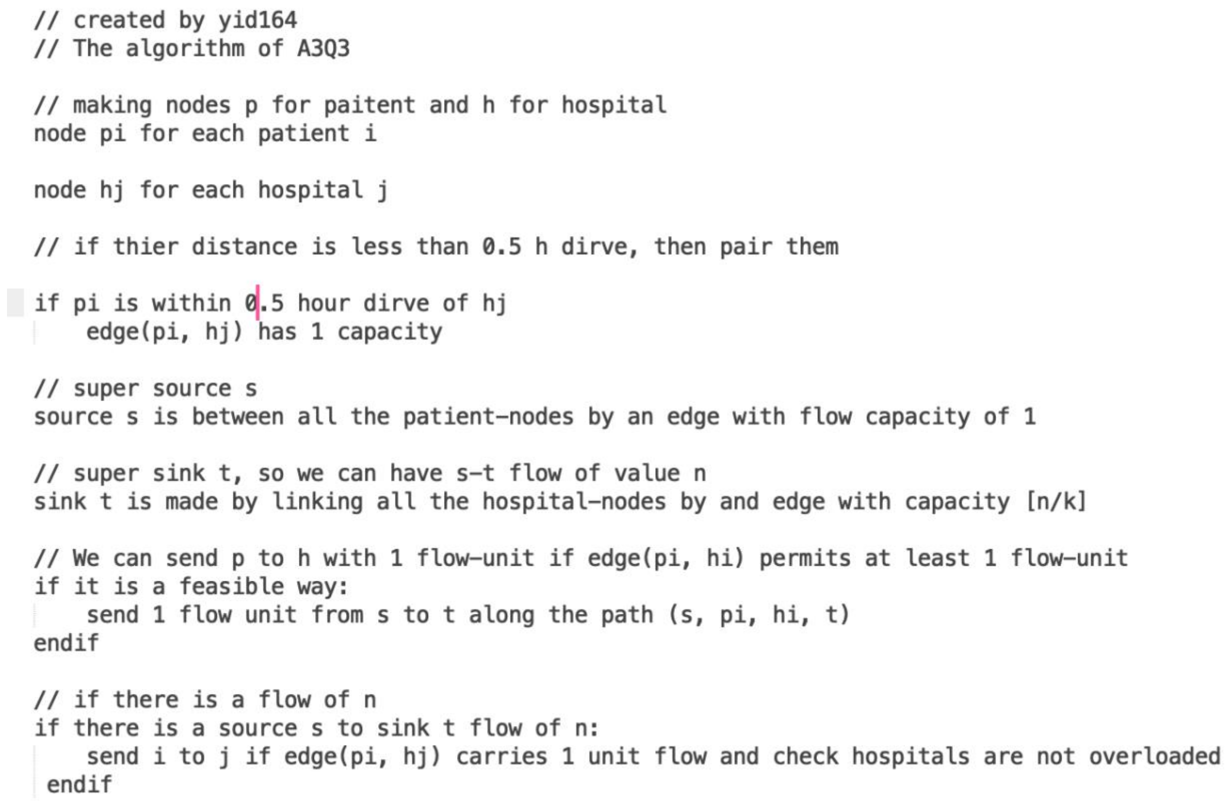
Question 3.

Kleinberg and Tardo page 419 #9

The problem for dealing with natural disasters and other crises.

To solve this question, we could build a flow network

Solution algorithm:



```
// created by yid164
// The algorithm of A3Q3

// making nodes p for paitent and h for hospital
node pi for each patient i

node hj for each hospital j

// if thier distance is less than 0.5 h dirve, then pair them

if pi is within 0.5 hour dirve of hj
    edge(pi, hj) has 1 capacity

// super source s
source s is between all the patient-nodes by an edge with flow capacity of 1

// super sink t, so we can have s-t flow of value n
sink t is made by linking all the hospital-nodes by and edge with capacity [n/k]

// We can send p to h with 1 flow-unit if edge(pi, hi) permits at least 1 flow-unit
if it is a feasible way:
    send 1 flow unit from s to t along the path (s, pi, hi, t)
endif

// if there is a flow of n
if there is a source s to sink t flow of n:
    send i to j if edge(pi, hj) carries 1 unit flow and check hospitals are not overloaded
 endif
```

About this algorithm, this approach of sending patients to hospitals will not break the capacity limitation of edges.
Time complexity of this graph algorithm of max-flow problem:

Nodes (patients and hospitals): it will take O (n + k)

Edges: it will take O (nk)