TCSS 543 Homework #2

1. *Design an efficient (polynomial time) algorithm that, given n, p, and q as inputs, determines the minimum cost among all possible traversal strategies. Your algorithm must make no assumption about n, p, and q except that they are all positive.*

   Let n be the total number of leaves in the tree and k be the number of leaves in the left sub tree. Then $n - k$ be the number of leaves in the right sub tree.

   Thus, to travel from root to the left subroot will cost $(n-k)p$ and to travel from root to the right subroot will cost $k \cdot q$.

   Let $f(n)$ be the total cost of traversal to each of the n leaves in the tree.

   If there is only $n = 1$ node: since the only node is both the root and the leaf, $f(1) = 0$.

   If $n = 2$ nodes: since there will be two sub tree with one leaf each,

   The total cost is $f(2) = p + f(1) + q + f(1) = p + q$.

   If $n = 3$ nodes: $f(3) = min\Big(\big(2p + f(1) + q + f(2)\big), (p + f(2) + 2q + f(1))\Big)$

   If $n = 4$ nodes: the total cost is

   $$f(4) = min\Big(\big(3p + f(1) + q + f(3)\big), \big(2p + f(2) + 2q + f(2)\big), (p + f(3) + 3q + f(1))\Big)$$

   If $n = 5$ nodes: the total cost is

   $$f(5) = min\Big(\big(4p + f(1) + q + f(4)\big), \big(3p + f(2) + 2q + f(3)\big), \big(2p + f(3) + 3q + f(2)\big), (p + f(4) + 4q + f(1))\Big)$$

   To summarize: $f(n) = \begin{cases} 0 & , \; n = 1 \\ min\big((n-k)p + f(k) + kq + f(n-k)\big) & , \; n > 2, 1 \le k \le n-1 \end{cases}$

   Pseudo code: min_traversal_cost (n, p, q)

```
A = [∞] // keeping track of the minimum cost
B = [0] // keeping track of subroot locations
For i in range of 2 to n:
    A.append(∞)
    B.append(0)
    For k in range of 1 to i:
        t = (i-k)p + A(k) + kq + A(i-k)
        If t < A(i):
            A(i) = t
            B(i) = k
Return A(n)
```

2. *Consider a cubic $n \times n \times n$ asteroid field where some positions are occupied by asteroids while all others are empty space. Let A be an $n \times n \times n$ table such that $A[i][j][k] = 1$ if there is an asteroid at coordinates (i, j, k), otherwise $A[i][j][k] = 0$.*

   Devise an algorithm that runs in $O(n^3)$ time and finds the position (specified by the top-left-front corner) and the size of the largest empty cubic region (i.e. containing no asteroids) in this field.

   Let $A[i][j][k]$ be the asteroid cube and $S[i][j][k]$ be a 3D array to store edge sizes of the largest cube that could be formed.

   Pseudo code: find_largest_cube ($A[i][j][k]$)

```
max = 0
S[i][j][k] be an empty 3D array
// fill in S[i][j][k] with the size of largest cube on the j-k surface.
i = n
For j in range of n to 0:
    For k in range of n to 0:
        If A[i][j][k] = 1:
            S[i][j][k] = 0
        Else:
            S[i][j][k] = 1
        If S[i][j][k] > max:
            max = S[i][j][k]

// fill in S[i][j][k] with the size of largest cube on the i-k surface.
j = n
For i in range of n to 0:
    For k in range of n to 0:
        If A[i][j][k]  = 1:
            S[i][j][k] = 0
        Else:
            S[i][j][k] = 1
        If S[i][j][k] > max:
            max = S[i][j][k]

// fill in S[i][j][k] with the size of largest cube on the i-j surface.
k = n
For i in range of n to 0:
    For j in range of n to 0:
        If A[i][j][k]  = 1:
            S[i][j][k]  = 0
        Else:
            S[i][j][k] = 1
        If S[i][j][k] > max:
            max = S[i][j][k]
```

```
// fill in the 3D matrix S[i][j][k]  with size of largest cube
For i in range of n − 1 to 0:
    For j in range of n − 1 to 0:
        For k in range of n − 1 to 0:
            If A[i][j][k] = 1:
                S[i][j][k]  = 0
            Else:
                S[i][j][k]  = min(S[i + 1][j][k], S[i][j + 1][k],
                                  S[i][j][k + 1], S[i + 1][j + 1][k],
                                  S[i + 1][j][k + 1], S[i][j + 1][k +
                                  1], S[i + 1][j + 1][k + 1]) +1
            If S[i][j][k] > max:
                max = S[i][j][k]
Return max
```

3. *A network of sensors will be installed in a city to monitor vehicular CO2 emissions. The city consists of n main regions, which are connected by a total of m two-way roads. To reduce financial costs, the sensors must only be deployed along roads with high traffic flows. However, to enable the estimation of CO2 emissions between any two regions, the subset of roads with sensors must span all regions, even though the connections can be indirect in the sense of crossing intervening regions.*

   Let $f(r, s)$ be the observed traffic flow (in vehicles per minute) along the road connecting regions r and s. Design an efficient algorithm that, given the value of $f(r, s)$ for all regions, finds a road subnetwork (i.e. determines which roads will receive sensors) that spans all regions and maximizes the total traffic flow to reduce the sensor deployment costs.

   We can solve this problem by alternating the original maximum spanning tree algorithm:

   Let M be the set of main regions and R denote the set of roads connecting the regions.

   Pseudo code: Subnetwork G = (M, R, $f(r, s)$)

```
T = empty tree
For each road r from f(r, s) in descending order of traffic flow:
    Add r and its end points into T if it does not create a cycle in T
Return T
```

4. (bonus 5 points) Consider again the problem in question 1. Design a $traversal$ algorithm (i.e. one that actually traverses a given graph rather than just computing the minimum cost) that, starting at the top node, visits all n bottom nodes in left-to-right order, and attains the minimum traversal cost, given the number n of nodes and the (respectively left and right) edge costs p and q as inputs.

This question narrows down from Question 1 to finding the minimum cost traversal from left to right.

Same as before, let n be the total number of leaves in the tree and k be the number of leaves in the left sub tree. Then $n - k$ be the number of leaves in the right sub tree.

Pseudo code:

```
min_traversal_left (n, p, q)
    B = [0, …, 0] // B is a size n array of 0's to keep track of subroot locations.
    If n=1:
        min_cost = 0
    Else:
        k = B[n]
        left = Traversal(current node-k, p, q) // travel left n-k steps from root
        right = Traversal (k, p, q) // travel right k steps from root
        min_cost = left + right
    Return min_cost
```