Homework 6

Due date: 2/17/2021 right before class

Problem 1

Given a sequence $x_1, x_2, ...x_n$ of real numbers (positive and negative), design an algorithm that find a consecutive subsequence $x_i, x_{i+1}, ...x_j$ such that the sum of numbers in it is maximum over all consecutive subsequences. ($O(n)$ algorithm)

Solution 1

The idea of the algorithm is to store the current maximum sum and global max sum. We keep adding the numbers from left to right. If the sum is larger than the maximum sum we have seen so far, we save the beginning an ending index for current summation. If the current sum is negative, we just discard the sum and restart the summation from the next number. Note that it is possible we get a output $(n+1, n+1, 0)$ if all numbers are negative. If so, it means the subsequence we want is just empty. The time complexity of the algorithm is $O(n)$ since it use a linear scan of all numbers.

```
MAXSEQ({x1, x2, . . . , xn}):

    (start, end, sum) <- (0, 0, 0)
    (left, right, max) <- (0, 0, 0)
    for i <- i to n
        (start, end, sum) <- (start, i, sum+xi)
        if sum < 0
            (start, left, sum) <- (i + 1, i + 1, 0)
        if sum > global max
            (left, right, max) <- (start, end, sum)
    return (left, right, max)
```

Listing 1: algo

Problem 2

a.

Given a directed graph $G = (V, E)$ represented as by an incidence matrix, derive the incidence matrix of the transitive closure G^* of G. In G^* there is an edge from v_i to v_j if and only if in G there is a *directed path* from v_i to v_j.

In class we have seen the Floyd-Warshal algorithm to find shortest-paths from all to all. Show how to use that algorithm to produce the incidence matrix of G^* from G. The rational we have seen of the FW algorithm use the operation of plus, and min. How can you replace these operations to get the FW algorithm to produce the transitive closure.

b.

As in the obvious way we can also define transitive closure of undirected graph G (this of each edge as 2 antiparallel edges). Assume the graph is given as an adjacency list. Give $O(E)$ time algorithm to produce a *description* (your choice) of the transitive closure of G. A description should satisfy the the time to answer a query whether there is an edge between v_i and v_j should take a constant time.

Solution 2

a.

```
Algorithm FloydWarshall(G=(V,E)):
    Initialize A[n][n] to be the incidents matrix of graph G where
    n=|V|

    for i in range 0 to n:
        for j in range 0 to n:
            for w in range 0 to n:
                A[j][w] = A[i][w] or (A[j][i] and A[i][w])

    return A to represent G*
```

Listing 2: algo

Complexity is n^3 as we have to loop through the incidence representation using the FW algorithm. Instead of the FW plus and minus check, here we use logical OR and AND operations.

b.

One option we have here is to create a representation of all connected components in the graph followed by iterating each component and mapping its internal reachability. We can crate an updated transitive closure that will keep information of adjacent vertices.

```
Find components(G=(V,E)):
    M <- Adjacency list for G
    TC <- [|V|][|V|] init to 0s
    for i in range |V|:
        run BFS on i with the following changes:
            for every node y explored during the run fill TC.
            - TC[x][y], T[y][x] <- 1 if y directly reachable from x
            - TC[x][y], T[y][x] <- 2 otherwise (BFS level > 1)
        If unexplored nodes remain:
            i <- next unexplored node in the graph
    Return TC
```

Listing 3: algo

Answering the connectivity question will then take O(1). We simply query TC[i][j] and check if it equals 1. The runtime is that of BFS O(E) since we explore each edge exactly once.

Problem 3

Given a directed weighted graph $G = (V, E)$ with positive and negative weights. Give an algorithm that finds if a negative cycles exists in the graph. A negative weight cycle is a cycle whose edge weights sum to a negative value.

Solution 3

We claim that Bellman Ford finds these cycles for us. To prove the forward direction, let's assume that the graph has a negative cycle. If round n of Bellman-Ford algorithm doesn't change the shortest distance to any vertex, then the rounds $n+1, n+2, ...$ will not change them too. In other words once the shortest paths remain unchanged in one round, they will remain unchanged forever. Thus, the shortest path to any vertex is fixed at the end of round $n-1$ and will never change. This means that the shortest paths to every vertex is lower bounded by some finite number. But this is a contradiction because in a graph with a negative cycle, the shortest distance to some vertices in $-\infty$ by going around the negative cycle infinitely many times. This implies that round n of Bellman-Ford algorithm should change the shortest distance to some vertices. To prove the opposite direction. We first assume that round n of Bellman-Ford algorithm changes the shortest distance to some vertices, and show that the graph has a negative cycle. Assume for contradiction that the graph has no negative cycle. Then the shortest path to every vertex has at most n vertices and $n-1$ edges. Thus, at the end of round $n-1$ the shortest distances to all vertices are computed, and these distance won't change in future rounds because they cannot be improved. However, this is a contradiction to our assumption that the shortest distances to some vertices change in round n.