COM S 311 HOMEWORK 4

NATHAN TUCKER (NJTUCKER@IASTATE.EDU)

This assignment represents my own work in accordance with University regulations.
Collaboration done with Haadi Majeed (hmajeed@iastate.edu) and Matthew Hoskins (mgh@iastate.edu)

- Nathan Tucker

Problem 1

Consider a virtual directed acyclic grid graph with its set of vertices and set of edges defined as follows. Let m and n be two positive integers. Each vertex in the graph is of the form (i, j), with $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$, where vertex (i, j) is in row i and column j. The number of vertices in the graph is mn. For each $i = 1, 2, ..., m$, row i consists of vertices (i, j) with $j = 1, 2, ..., n$. For each $j = 1, 2, ..., n$, column j consists of vertices (i, j) with $i = 1, 2, ..., m$. The graph contains three types of directed edges: horizontal edges, vertical edges and diagonal edges. For each $i = 1, 2, ..., m$ and each $j = 2, 3, ..., n$, there is a directed horizontal edge from vertex $(i, j - 1)$ to vertex (i, j) with a weight of $w(1, j)$. For each $i = 2, 3, ..., m$ and each $j = 1, 2, ..., n$, there is a directed vertical edge from vertex $(i - 1, j)$ to vertex (i, j) with a weight of $w(i, 1)$. For each $i = 2, 3, ..., m$ and each $j = 2, 3, ..., n$, there is a directed diagonal edge from vertex $(i - 1, j - 1)$ to vertex (i, j) with a weight of $w(i, j)$. Note that some weights in the weight matrix w are negative, while other weights are non-negative.

For each $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$, define $S(i, j)$ to be the maximum weight of all paths from the source vertex $(1, 1)$ to a target vertex (i, j), where the weight of a path from $(1, 1)$ to (i, j) is the sum of weights of each edge in the path. Develop and justify a formula for computing the matrix S. Design and analyze an algorithm with a running time of $O(mn)$ for computing a maximum-weight path from $(1, 1)$ to (m, n). The input to the algorithm is a weight matrix w of m rows and n columns. You need to express the algorithm in pseudocode and to determine its running time. Note that there is no need to build this grid graph. But it is helpful to carry out the computation in order of rows or in order of columns of this virtual grid graph. Below an example weight matrix w of 4 rows and 5 columns. A maximum-weight path is $< (1, 1), (1, 2), (2, 3), (3, 3), (4, 4), (4, 5) >$ with a weight of $(-2) + 8 + (-1) + 9 + (-3) = 11$. Note that $w(3, 1) = -1$ is the weight of the edge from vertex $(2, 3)$ to vertex $(3, 3)$ and that $w(1, 5) = -3$ is the weight of the edge from vertex $(4, 4)$ to vertex $(4, 5)$.

$$\begin{bmatrix} 0 & -2 & -4 & -8 & -3 \\ -2 & -5 & 8 & -5 & 2 \\ -1 & 1 & 2 & 2 & 3 \\ -4 & 2 & 3 & 9 & 1 \end{bmatrix}$$

```
MaxWeightPath
    Input: A cost matrix $M$ with rows $i$ and columns $j$
    Let $T[i,j]$ be a new matrix to store the DP values of calculating cost.
    for $n = 0$ to $i$ do
        for $m = 0$ to $j$ do
            $T[n,m] = M[i,j]$
        if $n == 0$ and $m \geq 0$ then
            $T[0,m] = T[0,m] + T[0,m-1]$
        else if $m == 0$ and $n \geq 0$ then
            $T[n,0] = T[n,0] + T[n-1,0]$
        else
            $T[n,m] = T[n,m] + max\{T[n-1,j], T[n,m-1], T[n-1,m-1]\}$
            Output $max\{T[n-1,j], T[n,m-1], T[n-1,m-1]\}$
```

This algorithm will compute the weight as well as the path of the greatest weight path of the weight matrix. It will search the neighbors of the current cell and find the largest of the neighbors and store the value in the matrix T. It pre-computes the possible paths to take (in case the greatest path is not the path to be taken by the greatest adjacent neighbor). This is done over each index of an n by m matrix, making it an $O(n * m)$ runtime algorithm, and outputting the path in that same runtime.

Problem 2

Prove that P is closed under the star operation by dynamic programming.

Consider the language $L \in P$. The following procedure decides $L*$.

```
$L*$
    **Input:** y
    $M =$ On input, assume $y = y_1, y_2, ...y_n \in \sum$
    **if** $y = \epsilon$ **then**
        return true
    **for** $i = 1$ *to* $n$ **do**
        **if** $y_i \in L$ **then**
            Run $M$ on $y_i$
            $T[i, i] = 1$
    Let T[i, j] be a 2d array for $i \leq j$
    **for** $k = 2$ *to* $n$ **do**
        Assume $j = i + k - 1$
        **if** $y_i...y_j \in L$ **then**
            Run $M$ on $y_i...y_j$
            $T[i, j] = 1$
        **for** $l = i$ *to* $j - 1$ **do**
            **if** $T[i, l] = 1$ *and* $T[l, j] = 1$ **then**
                $T[i, j] = 1$
    **if** $T[1, n] = 1$ **then**
        return true
    **else**
        return false
```

Because the above algorithm is a polynomial time algorithm, we can conclude that P is closed under the star operation.

Problem 3

A *triangle* in an undirected graph G is defined to be a 3-clique (i.e., three vertices in G that are pairwise connected by edges) and $3 - ANGLE := \{\langle G\rangle \mid G \text{ contains a triangle } \}$. Prove that $3 - ANGLE \in P$.

Let the graph be $G = (V, E)$ where V denotes the set of verticies and E denotes the set of edges. We need an algorithm that will enumerate all triplets of the graph given as encoding of graph G as an adjacency matrix $\langle G\rangle$

```
1 ContainsTriangle(M)
    Input: G = (V, E) represented as adjacency matrix M
2   Let $N$ be a matrix representation of $M^2$ with width $x$ and height $y$ for $i$ to $x$ do
3       for $j$ to $y$ do
4           if $N[i, j] \geq 0$ and $M[i, j] \geq 0$ then
5               for $k$ in $V$ of matrix $M$ do
6                   if $k$ adjacent to $i$ and $k$ adjacent to $j$ then
7                       return true
8   return false
```

We can use some properties of an adjacency matrix to find if there is a triangle in a graph G. First, if we square the original matrix, we can look for indexes on the original and squared matrix that both have a non-zero value at the same index. Once we have two of the vertexes in a potential triangle, we can enumerate the rest of the vertexes to determine if it is adjacent to both of the vertexes found. If this is the case, then we know we have a triangle.

The runtime of this algorithm is $O(V^3)$, as the time to compute matrix multiplication, if done naively, takes $O(V^3)$ time, where V is the vertexes of the graph or, in our case, the height of a single matrix. Then, it takes $O(V^2)$ time to enumerate all the shared points in each matrix where they are both non-zero, and then another $O(V)$ search of the vertexes that are adjacent to the first two. Admiteddly this could be done outside with more dynamic programming and only take $O(V^2 + V)$ time but it will still be polynomial in nature.

$\therefore 3 - ANGLE \in P$.

Problem 4

Prove that if $P = NP$, then we can factor integers in polynomial time.

Consider the language $L = \{\langle n, a, b\rangle \mid n$ has a factor p in the range $a \leq p \leq b\}$. L is in NP, since the factor can serve as the certificate. Because we are assuing $P = NP$ above, we know there is a polynomial time algorithm that decides the above language. Repeated applications of this algorithm allow us to divide our search in half each iteration by looking at the range $(a, a + \frac{b}{2})$. If there is not, we know there is a factor in the other range.

The total number of times we have to iterate through this algorithm is $logn$, or $O(k)$ where k is the number of bits in n. A polynomial number of iterations of this algorithm allows us to isolate a single factor. Because there are at most $O(k)$ factors as well, we can find all factors in polynomial time.

$\therefore$ if $P = NP$, then we can factor integers in polynomial time.