Com S 311 Spring 2021

Assignment 4: Single-Source Shortest Paths and NP-Completeness

Due: April 16, 11:59 pm
Early Submission: April 15, 11:59 p.m. (5% bonus)
Late Submission: April 17, 11:59 A.M. (25% penalty)

Haadi-Mohammad Majeed
Collaborated with Nathan Tucker and Matthew Hoskins

Guidelines

- For each problem, if you write the statement "I do not know how to solve this problem" (and nothing else), you will receive 20% credit for that problem. If you do write a solution, then your grade could be anywhere between 0% to 100%. To receive this 20% credit, you must explicitly state that you do not know how to solve the problem.
- You are allowed to discuss with classmates, but you must work on the homework problems on your own. You should write the final solutions alone, without consulting anyone. Your writing should demonstrate that you understand the proofs completely.
- When proofs are required, you should make them both clear and rigorous. Do not hand-waive.
- Please submit your assignment via Canvas.
  - You **must** type your solutions. Please submit a PDF version.
  - Please make sure that the file you submit is not corrupted and that its size is reasonable (e.g., roughly at most 10-11 MB).

  If we cannot open your file, your homework will not be graded.
- Any concerns about grading should be expressed within one week of returning the homework.

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 $(4, 5)$.

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

```
ShortestPathMostWeight
    Input: 2D Matrix $x$
    m = x.length
    n = x.height
    Table T is a new matrix T[m][n]
    for(i $\rightarrow$ m)//runtime loop of m
        for(j $\rightarrow$ n)//runtime loop of n
            T[i][j] = x[i][j]
            if(i == 0 and j > 0)
                T[0][j] += T[0][j-1]
            else if(j == 0 and i > 0)
                T[i][0] += T[i-1][0]
            else if(j > 0 and i > 0)
                T[i][j] += T[i-1][j-1]
            T[i][j] += max(T[i-1][j], T[i][j-1], T[i-1][j-1])
    return T[m - 1][n - 1];
```

This algorithm runs in the time of $O(M * N)$ as it iterates through the height (n) and width (m) of the input matrix to determine the best path to take.

Problem 2

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

P is a class of langauge in polynomial time such that $P = \bigcup_k TIME(n^k)$
To prove P is closed under the star operation, $L \epsilon P$. Now to decide L*

```
$L*$
    Input: y
    M = Assume input y = $y_1 y_2 ... y_n \epsilon \Sigma$
    if ($y = \varepsilon$)
        return ACCEPT
    table T[i,j] for i ≤ j
    for(i=1 → n)
        run M on $y_i$
        if($y_i \epsilon L$)
            T[i,i] = 1
    for(k=2 → n)
        for(i = 1 → n-k+1)
            j = i + k -1
            run m on $y_i ... y_j$
            if($y_i ... y_j \epsilon L$)
                T[i,j] = 1
            for(l = i → j-1)
                if(T[i,l] == 1 and T[l,j] == 1)
                    T[i,j] = 1
    if(T[1,n == 1])
        return ACCEPT
    else
        return REJECT
```

This algorithm is a polynomial time algo, and thus P is closed under 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 $3 - ANGLE \in P$.

```
Triangle
    Input: Adjacency Matrix $M$
    N = $M^2 \leftarrow O(V^3)$
    for(i → $N.width$)
        for(j → $N.height$)
            if(N[i][j] == 0 and M[i][j] == 0)
                for(k in vertices)
                    if(k is adjacent to N[i][j] and M[i][j])
                        return true
```

With this, we have created algorithm that can find a triangle within the polynomial time of $3 - ANGLE \epsilon P$ N is the matrix M times itself, and with that we iterate through N's height and width to determine if any of the verticies at the same point are zero, if they are then we check for neighbor verticies within M and N again

Problem 4

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

We define as the following:
L=$\langle n, a, b\rangle$ —n has a factor p in the range of $a \leq p \leq b$
L is within NP because the factor can be the certificate.
Since we assume P=NP, there is a polynomial algorithm
By repeating the algorithm, we can halve the search space each time by determining if there is a factor within the range of $(a, a + b/2)$. If there is not, that means there is a factor in the other range.
The number of times the algo has applied would equate to $log(n)$ or $O(k)$ where k is the number of bits of n. So a polynomial number of applications will isolate this down to one factor.
With a ceiling of O(k)factors, we can find all factors in polynomial time.