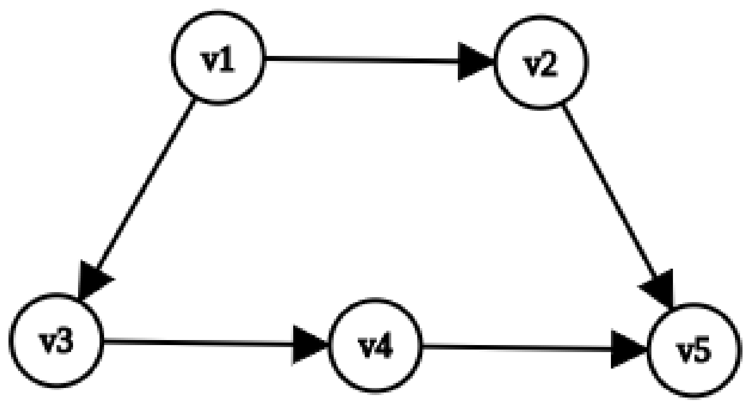
Homework 4

Amit Dharmadhikari (SBU id: 112044244)

7 November 2018

1 Question 1

1.1 (a)



The given algorithm does not give a correct answer for the above graph. The algorithm will return the 2 (edges (v_1, v_2) and (v_2, v_5)) whereas the correct answer is 3 (edges (v_1, v_3), (v_3, v_4) and (v_4, v_5)).

1.2 (b)

We will use dynamic programming. We will frame a sub-problem OPT(i) as the length of the longest path from v_1 to v_i. However, there may not be a path from v_1 to v_i for all v_i. For such cases, we will use a special value (like $-\infty$). OPT(1) will be 0 as the distance of v_1 from itself will obviously be 0.

algorithm longest path(n)
    Array D[1 ... n]
    D[1] = 0

```
For i = 2 to n
    dist = −∞
    For all edges (j, i)
        if D[j] ≠ −∞
            if dist < D[j] + 1
                dist = D[j] + 1
            endif
        endif
    endfor
    D[i] = dist
endfor
return D[n]
```

The runtime is $O(n^2)$.

2 Question 2

Lets say that $y_1 y_2 y_3 ... y_n$ is the optimal segmentation for string y. It follows that $y_1 y_2 y_3 ... y_{n-1}$ is an optimal segmentation for the prefix of y that excludes y_n. If it weren't, we could have used the optimal segmentation instead for computing the optimal segmentation of $y_1 y_2 y_3 ... y_n$. Thus, our optimal segmentation for $y_1 y_2 y_3 ... y_n$ would have been different, thus contradicting our initial assumption. We use this property for framing a sub-problem in order to write a dynamic programming algorithm.
For j = 0: OPT(0) = 0
For j > 0: $\max\limits_{1 \leq k \leq j} \{OPT(k-1) + quality(y_k ... y_j)\}$

```
algorithm find optimal segmentation(n):
  OPT[0] = 0
  for (j = 1 ... n)
    temp = −∞
    for (k = 1 ... j)
      if temp < OPT[k-1] + quality(y_k ... y_j)
        temp = OPT[k-1] + quality(y_k ... y_j)
      endif
    endfor
    OPT[j] = temp
  endfor
```

Run backwards through the OPT array and split the string wherever the previous value is higher than the current value.

Runtime of the algorithm is $O(n^2)$.

3 Question 3

This problem can be transformed into a problem of negative cycle detection, which can further be solved using the Bellman Ford algorithm. We will build a weighted directed graph G with a node for each stock and a directed edge between every two pair of successively traded stocks. Each edge will have a weight of $-log(r_{ij})$. Now, according to the problem, a trading cycle C in G is an opportunity cycle if and only if

$$\prod_{(i,j)\epsilon C} r_{ij} > 1 \tag{1}$$

Now, if we take logarithms of both sides, we get,

$$\sum_{(i,j)\epsilon C} \log r_{ij} > 0 \tag{2}$$

$$\sum_{(i,j)\epsilon C} -\log r_{ij} < 0 \tag{3}$$

Thus, a trading cycle C in G is an opportunity cycle if and only if it is a negative cycle. We can use the Bellman Ford algorithm on graph G in order to find the negative cycles. The time complexity will be $O(|V|.|E|)$, where $|V|$ is the number of vertices in G, i.e. the number of companies and $|E|$ will be the number of edges, i.e. the number of trades.

4 Question 4

Lets suppose s has n characters. For simplicity, we'll consider the repetition $x^{'}$ of x and $y^{'}$ of y consisting of n characters each. Let M be a boolean matrix. M[i,j] = true if S[1 : i+j] is an interleaving of $x^{'}$[1 : i] and $y^{'}$[1 : j]. Now, if s is an interleaving of $x^{'}$ and $y^{'}$, the last character comes from either $x^{'}$ or $y^{'}$. Removing this character, we get a smaller recursive problem. Thus, we can define a sub-problem for a dynamic programming algorithm as follows:

M[i, j] = true if and only if (M[i-1, j] = true and s[i+j] = $x^{'}$[i]) or (M[i, j-1] = true and s[i+j] = $y^{'}$[j])

We can implement this dynamic programming algorithm in a bottom up manner as follows:

algorithm check interleaving:
    M[0, 0] = true
    for k = 1 to n
        for all pairs (i,j) such that i + j = k
            if M[i-1, j] = true and s[i + j] = $x^{'}$[i]

M[i, j] = true
else if M[i, j-1] = true and s[i + j] = y'[j]
M[i, j] = true
else
M[i, j] = false
endfor
endfor
Return true if and only if there exists a pair (i,j) such that i+j = n and M[i, j] = true

The runtime is $O(n^2)$.

5 Question 5

Let M be a four dimensional boolean matrix. Let M[j, p, x, y] = true if it is possible to have at least x A-votes in district 1 and y A-votes in district 2, while p of the first j precincts are allocated to district 1. M[j, p, x, y] = false if it isn't possible. Now we will consider the next precinct, i.e. precinct j+1. Lets say it has w votes. To compute M[j+1, p, x, y], we either put precinct j+1 in district 1 or district 2. In the former case, we use the results of the subproblem M[j, p-1, x-w, y], while in the latter, we use the results of the subproblem M[j, p, x, y-w]. We need to fill in the entire 4 dimensional matrix M in a bottom-up manner. In the end, in order to find if a solution exists, we need to scan the entire matrix looking for an entry in the form of M[n, n/2, x, y] = true where x > mn/4 and y > mn/4. Since the number of sub-problems in this dynamic programming algorithm is n^2m^2, the time complexity is $O(n^2m^2)$.