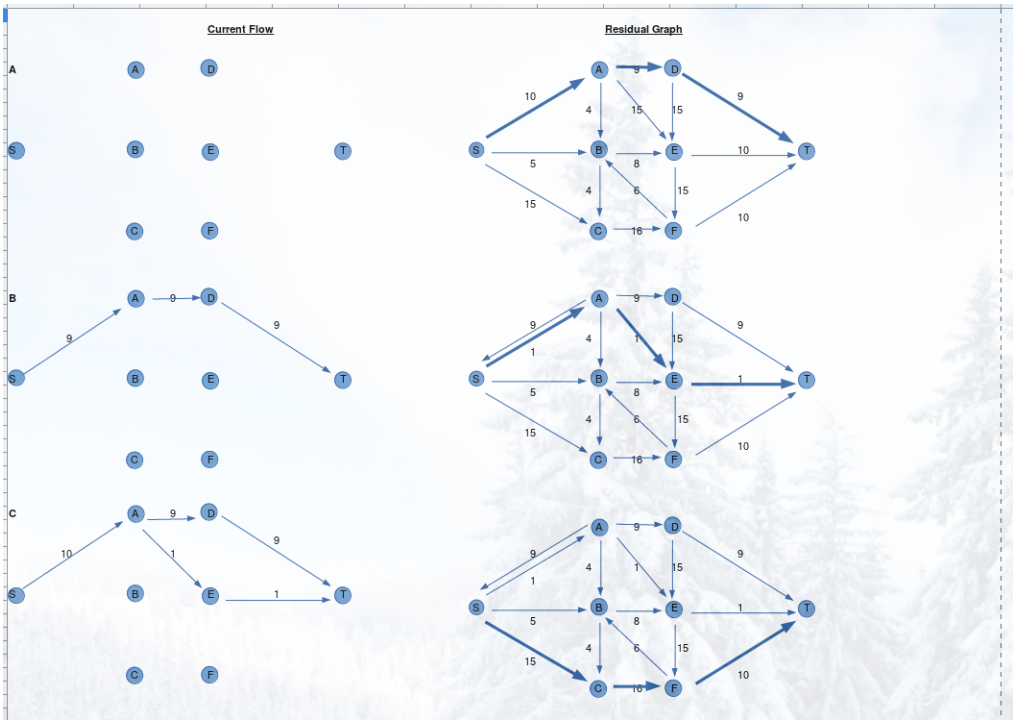
Homework 4

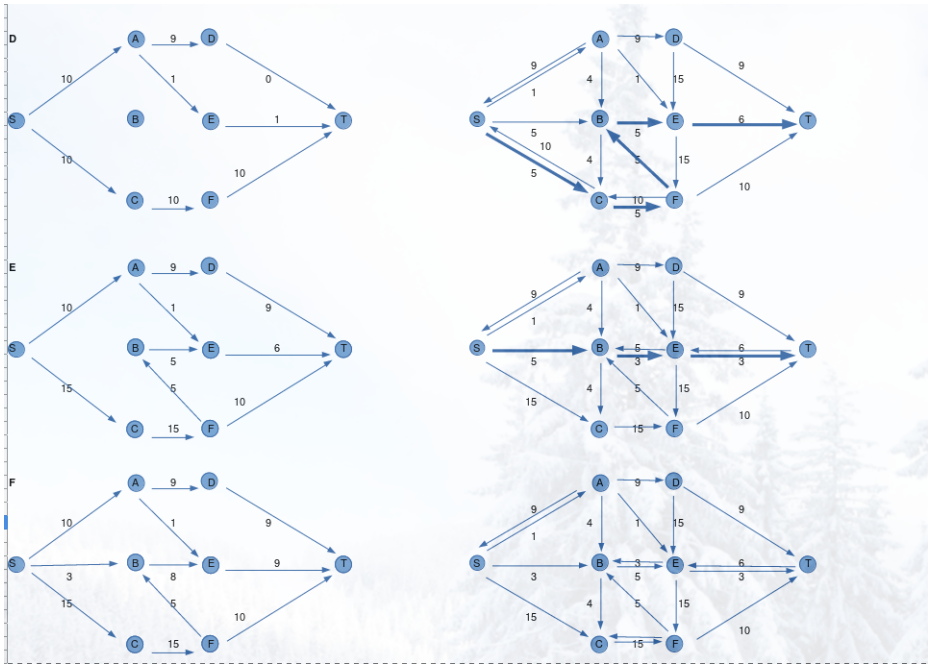
Kevin Martin
CIS675 - Syracuse University

February 19, 2020

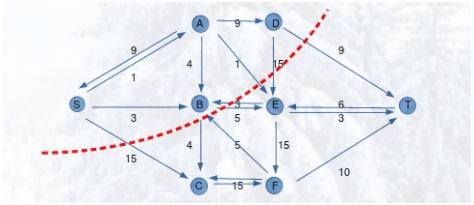
1. Question 1

   (a) The iterations of the maximum flow algorithm from node S to node T:





(b) The minimum cut:



2. Question 2 For an undirected connected graph, $G = (V, E)$:

   (a) Algorithm for maximum spanning tree, negate the weights and apply Kruskal's algorithm:

```
def maxSpan(G)
    sortDescending(e) // sort all edges e in descending order
    tree T = ∅
    for i = 1 to n do // for all edges n
        (u,v) = e_i
        if setCheck(u) != setCheck(v) // no cycles
            T = T ∪ e_i // merge sets containing u and v
            if count(T) == n - 1 then
                break
    return T
```

   (b) The runtime is $O(ElogV)$ where E is the number of edges and V is the number of vertices. We must use a comparision based algorithm to sort all the edges, which takes the majority of the running time. The rest of the operations can be done in constant $O(1)$ time., except for the find and union operations, witch take $O(LogV)$, which is still dominated by the sorting portion. The space complexity is $O(E + V)$ as we have to store all the edges and all the vertices to fully perform the algorithm.

3. Question 3 Given a convex polygon P with n vertices in the plane:

   (a) Pseudocode of a dynamic programming algorithm to find triangulation of minimum cost of polygon P:

```
def triangulate(V [], int n):
    if (n<3) then
        return 0 // not a true polygon
    matrix[n][n]
    for (check = 0 : n) do
        for (i = 1, j = check, j < n) do
            if (j < i + 2) then
                matrix[i][j] = 0
            else
                matrix[i][j] = ∞
                for (k = i + 1, k < j) do
                    temp = matrix[i][k] + matrix [k][j] + sum(V, i , j, k)
                    if (matrix[i][j] > temp) then
                    matrix[i][j]
    return matrix[0][n-1]
```

(b) The running time of this algorithm is an unfortunate (On^3). This is because we must perform n operations on a two dimensional matrix ($O(n^2)$). We must check the cost of each potential triangle, using the "sum" function, and store the results. Then we must compare those results to ensure our final result is the optimal (minimum) one. The space complexity for this algorithm is also large due to the matrix: $O(n^2)$ as we must complete a full n x n matrix.

4. Question 4 Matching two gene strings using a socring sequence, accomedating for gaps:

(a) Pseudocode of a dynamic programming algorithm to take in two strings and return highest scoring alignment of the two strings:

```
def scoreCalc(A [], B[] )
    score = 0
    matrix[length(A) + 1][length(B) + 1] // store results
    for i = 0 : length(A)
        matrix[i][0] = score * i
    for j = 0 : length(B)
        matrix[0][j] = score * j
    for i = 1 to length(A) + 1 do
        for j = 1 to length(B) + 1 do
            match = matrix[i - 1][j - 1] + similarity(A[i], B[i])
            delete = matrix[i - 1][j] + score
            insert = matrix[i][j - 1] + score
            matrix[i][j] = max(match, insert, delete)
    return matrix[i][j] // this gives the max score


def findAlignment(A [], B[])
    alignA = 0
    alignB = 0
    i = length(A)
    j = length(B)
    while ( i > 0 or j > 0) do
        if (i > 0 and j > 0 and matrix[i][j] ==
        matrix[i - 1][j - 1] + similarity(A[i], B[i])) do
            alignA = A[i] + alignA
            alignB = B[j] + alignB
            i -= 1
            j -= 1
        else if (i>0 and matrix[i][j] == (matrix[i - 1][j] + score)) do
            alignA = A[i] + alignA
```

```
        alignB = "-" + alignB // gap, insert in B
        i -= 1
else
        alignA = "-" + alignA // gap, insert in A
        alignB = B[j] + alignB
        j -= 1
```

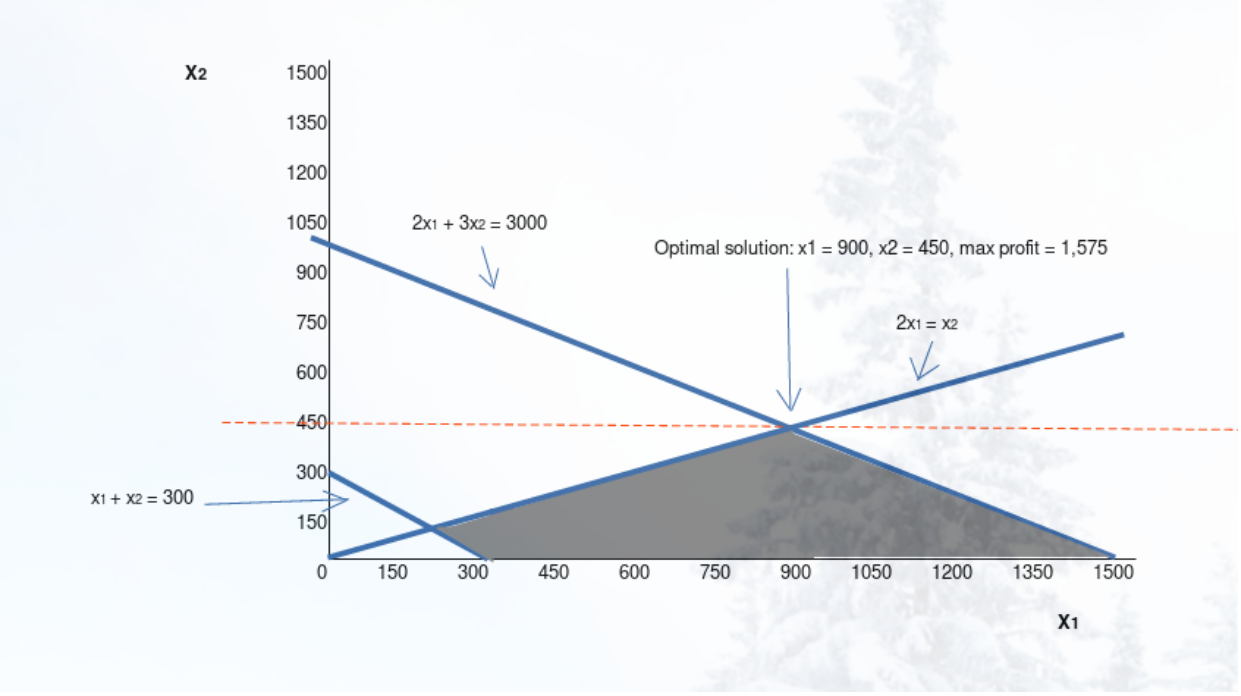
(b) The running time is $O(n^2)$, assuming the input strings A and B are of the same length. If they were of different lengths m and n, then the running time would be $O(mn)$. This is because each entry in the matrix must be calculated. The calculations are done in constant time but need to be applied to the entire set of possible combinations. Similarly, the sapce complexity is also (On^2), assuming the two input strings are equal (otherwise the space complexity would be $O(mn)$). This is because the algorithm fills the entire n x n matrix.

5. Question 5

(a) To represent the situation as a linear problem, we formulate as follows, letting x_1 be regular drinks and x_2 be strong drink:

Objective function	max $2x_1 + 3x_2$
Constraints	$x_1 + 1.5x_2 \leq 3000$
	$x_1 + x_2 \geq 300$
	$2x_1 \leq x_2$

(b) Graph of feasible region:



(c) The coordinates of all vertices of the feasible region are:
(300, 0), (1500, 0), (900, 450), (200, 100)

(d) The optimal product mix to maximize daily profit is:
900 regular drink at $ 2 each and 450 strong drink at $ 3 each, which gives a total profit of $1,575 per day. This is represented on the graph by the furthest out point on on the feasible region, where the tangential dotted line intercepts the point (900, 450).