COT 5405: Programming Assignment 2 (Spring 2016)

Report by: Sansiri Tarnpradab

Purpose

The purpose of this assignment is to investigate the application of *Dynamic programming* on different kinds of problems which includes: RNA Secondary Structure, Sequence Alignment, and Max Flow (Ford-Fulkerson algorithm), and to determine an efficiency, running times, implementation problems, and potential improvements.

Introduction

Problem 1: This problem requires to find the secondary structure of the given RNA molecule: AUGGCUACCGGUCGAUUGAGCGCCAAUGUAAUCAUU. The secondary structure is created when pairs of bases at different positions in the RNA molecule join together given following the rules below:

1. Each base can join with at most one other base of the formats: A-U, U-A, C-G, or G-C.
2. No base can join with itself. In other words, no sharp turns allowed
3. No base can join with an adjacent base. In other words, the ends of each pair are separated by at least 4 intervening bases.

Base-pairs are nested and cannot cross, that is, there cannot be base-pairings of the positions (i,j) and (k,l) such that i less than k less than j less than l.

Out of all the secondary structures that are possible for a single RNA molecule, the ones to be looked for are likely to arise under physiological conditions, that is, the ones with the optimum total free energy. Assume that the free energy of a secondary structure is proportional to the number of base pairs that it contains. The Dynamic Algorithm, is thus applied to determine a secondary structure with the maximum possible number of base pairs.

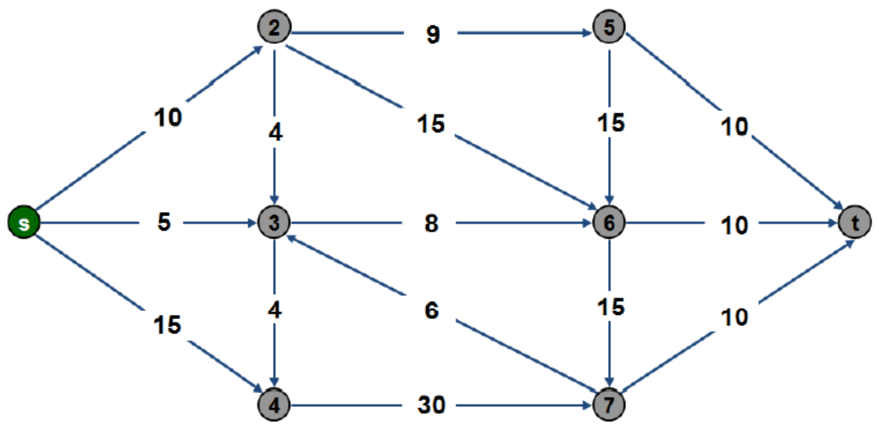
Problem 2: This problem requires to find the sequence of the two strings, which are:

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

Each mismatch has a penalty of 2, and each gap has a penalty of 1. The similarity between two strings is determined by computing the number of gaps and mismatches incurred when lining the two strings. The alignment which gives the least cost is the desired outcome since it makes both sequences most similar. Noted that, this particular problem is very important especially to the field of computational biology. The sequence of symbols in an organism's genome can be viewed as determining the properties of the organism. Suppose that a certain substring is determined in the DNA of X codes for a certain kind of toxin. If a very "similar" substring is discovered in the DNA of Y, we might be able to hypothesize, before performing any experiments at all, that this portion of the DNA in Y codes for a similar kind of toxin, and therefore we are able to take a proper action accordingly to prevent any potential problems.

<u>*Problem 3*</u>: This problem requires to find the max flow of the given directed graph, using Ford-Fulkerson algorithm.



Briefly, a flow network is a directed graph *G = (V,E)* with three features including: 1) Associated with each edge e is a capacity, which is a nonnegative number, 2) There is a single source node 's', and 3) There is a single sink node 't'. Additionally, a flow must satisfy two properties including: 1) *Capacity conditions*, that is, the flow on an edge cannot exceed the capacity of the edge, and 2) *Conservation conditions*, that is, for every node other than the source and the sink, the amount of flow entering must equal the amount of flow leaving. The source has no entering edges but it is allowed to have flow going out; whereas, the sink is allowed to have flow coming in, even though it has no edges leaving it. To compute the value of the flow *f*, denoted *v(f)*, is defined to be the amount of flow generated at the source: $v(f) = \Sigma f(e)$, in other words *v(f)* is the total capacities on the edge coming out of source s.

The idea behind the Ford-Fulkerson algorithm is: as long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, the flow can be sent along one of the paths, then find another path, and so on. A path with available capacity is called an augmenting path. The desired outcome is the maximum possible net flow values leaving the source.

Method

- ***Language used***:
  Java programming language

- ***Compilation instructions***:
  The code for each problem is contained in separate folders: Assignment2_prob1, Assignment2_prob2, and Assignment2_prob3. To compile and run, navigate to each folder and use the following commands.
  - To compile, use this command: **javac Runme.java**
  - To run, use this command: **java Runme**

- ***Format of input***: Not needed since all inputs are stored in the program.

- ***Platform specifics***:
  Windows 8

- ***Algorithms to solve the problems***:

  Dynamic Programming was applied to all three problems. However, there is a slight difference for each of the problems. More details for each one is the following.

RNA Secondary Structure

<u>Def:</u> OPT (i, j) = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

- *Case 1:* If $i \geq j - 4$, **OPT (i, j) = 0** by no-sharp turns condition.
- *Case 2:* If base b_j is not involved in a pair, **OPT(i, j) = OPT(i, j-1)**
- *Case 3:* Base b_j pairs with b_t for some $i \leq t < j - 4$.
  - non-crossing constraint decouples resulting sub-problems
  - **OPT(i, j) = 1 + max$_t$ { OPT(i, t-1) + OPT(t+1, j-1) }**

Noted that for this particular problem, the two dimensions are needed to depict the array: one for the left endpoint of the interval being considered, and one for the right endpoint.

Sequence Alignment

<u>Def:</u> OPT (i, j) = min cost of aligning strings $x_1\ x_2 \ldots x_i$ and $y_1\ y_2 \ldots y_j$.

- *Case 1:* OPT matches x_i-y_j, pay mismatch for x_i-y_j + min cost of aligning two strings $x_1\ x_2 \ldots x_{i-1}$ and $y_1\ y_2 \ldots y_{j-1}$
- *Case 2a:* OPT leaves x_i unmatched, pay gap for x_i and min cost of aligning $x_1\ x_2 \ldots x_{i-1}$ and $y_1\ y_2 \ldots y_j$
- *Case 2b:* OPT leaves y_j unmatched, pay gap for y_j and min cost of aligning $x_1\ x_2 \ldots x_i$ and $y_1\ y_2 \ldots y_{j-1}$

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Max flow

<u>Def:</u> Let *G(V,E)* be a graph, and for each edge from *u* to *v*, let *c(u,v)* be the capacity and *f(u,v)* be the flow. The maximum flow from the source *s* to the sink *t* is to be determined. The residual network $G_f(V,E_f)$ to be the network with capacity $c_f(u,v) = c(u,v) - f(u,v)$ and no flow.

<u>Algorithm:</u>

Inputs: Given a Network *G = (V,E)* with flow capacity *c*, a source node *s*, and a sink node *t*
Output: Compute a flow *f* from *s* to *t* of maximum value

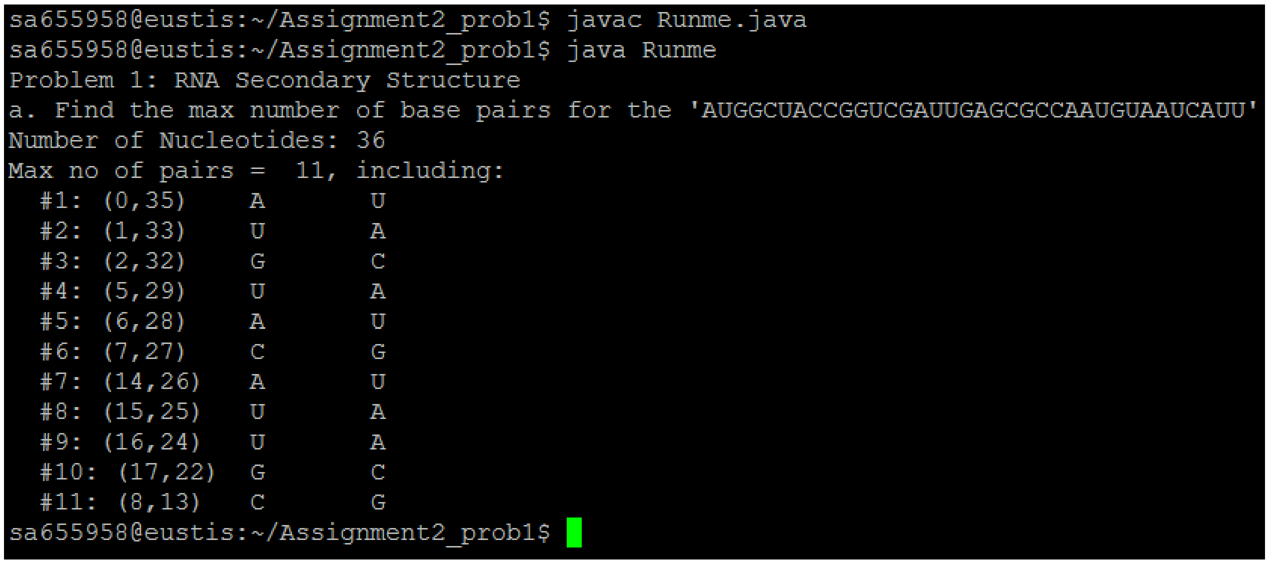
```
1. f(u,v) ← 0 for all edges (u,v)
2. While there is a path p from s to t in Gf, such that cf(u,v)>0 for all edges (u,v)ϵ p:
    1. Find cf(p) = min{cf(u,v) : (u,v)ϵ p}
    2. For each edge (u,v) p
        1.f(u,v) ← f(u,v) + cf(p) //send flow along the path
        2.f(u,v) ← f(u,v) - cf(p) //the flow can be undone or can be returned later
```

Outputs of test input data:

The output contains the following, (shown below are screenshots of output captured from the SSH.)

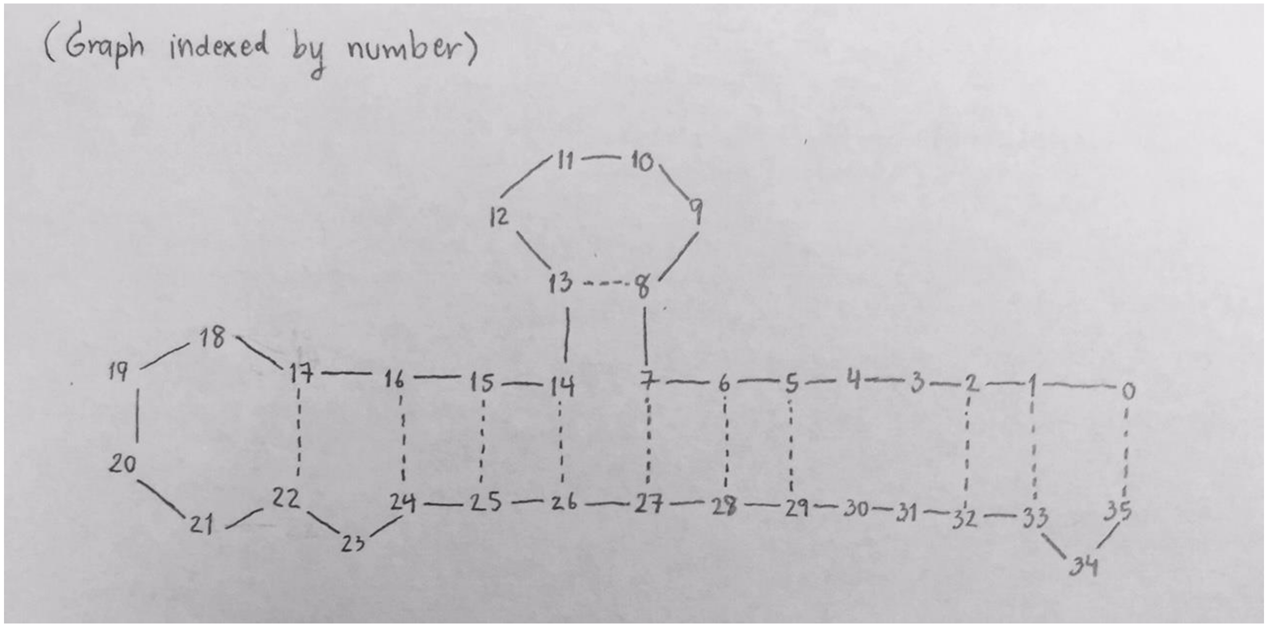
Problem 1:

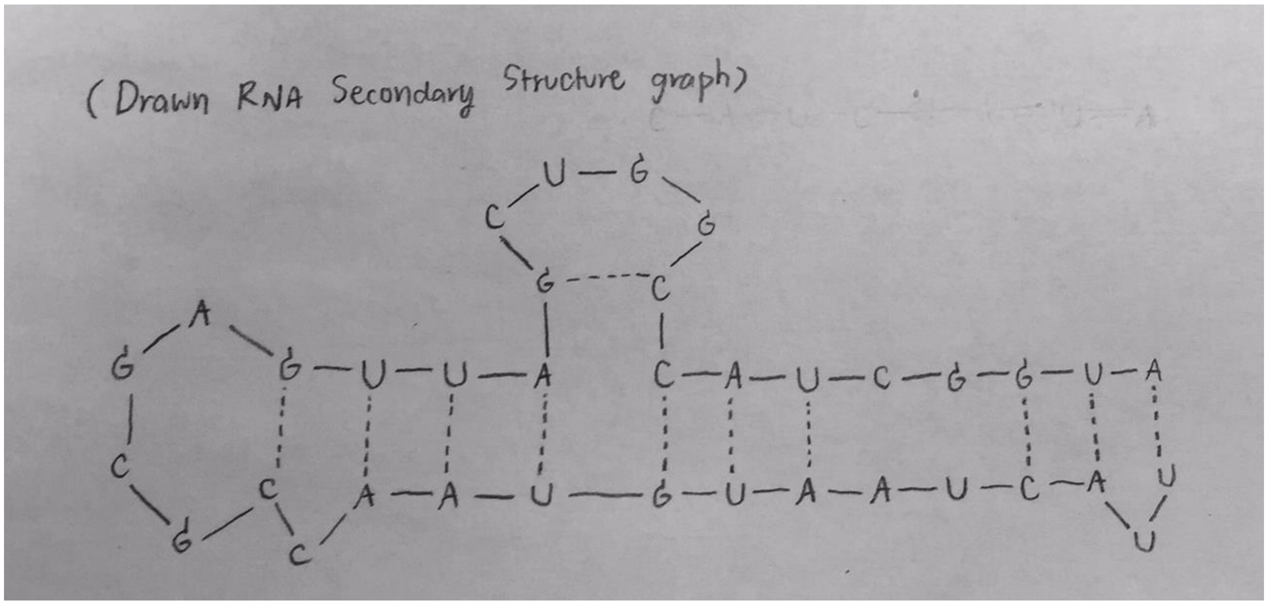
1. The number of base pairs of the given RNA molecule and how each pair looks like.



```
sa655958@eustis:~/Assignment2_prob1$ javac Runme.java
sa655958@eustis:~/Assignment2_prob1$ java Runme
Problem 1: RNA Secondary Structure
a. Find the max number of base pairs for the 'AUGGCUACCGGUCGAUUGAGCGCCAAUGUAAUCAUU'
Number of Nucleotides: 36
Max no of pairs =  11, including:
  #1: (0,35)    A       U
  #2: (1,33)    U       A
  #3: (2,32)    G       C
  #4: (5,29)    U       A
  #5: (6,28)    A       U
  #6: (7,27)    C       G
  #7: (14,26)   A       U
  #8: (15,25)   U       A
  #9: (16,24)   U       A
  #10: (17,22)  G       C
  #11: (8,13)   C       G
sa655958@eustis:~/Assignment2_prob1$
```

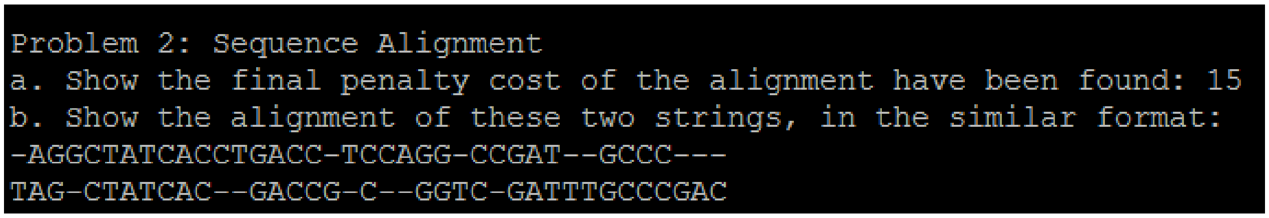
2. The drawn RNA secondary structure graph.





Problem 2:

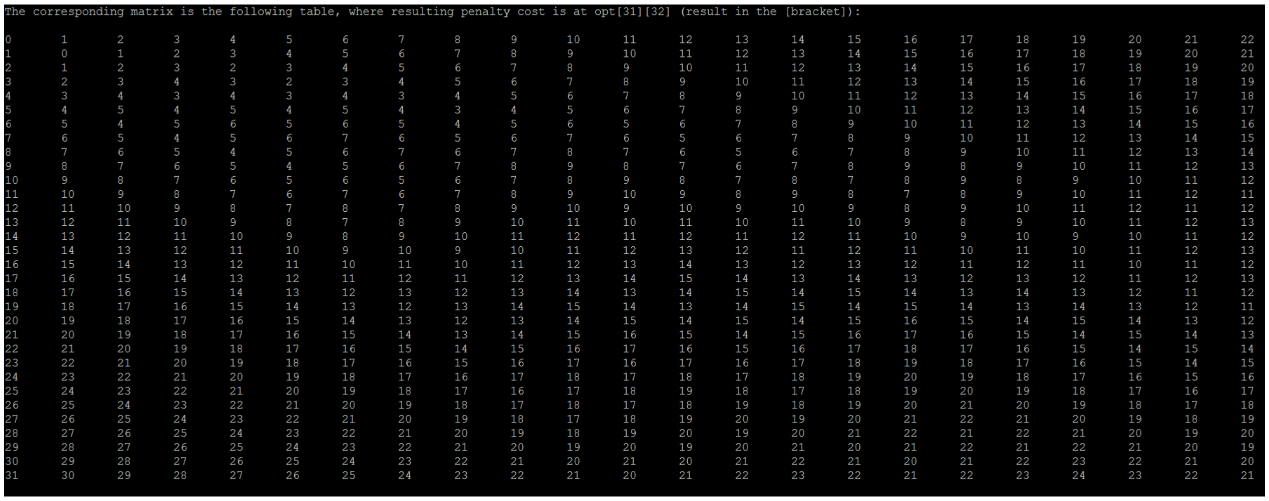
1. The final penalty cost of the alignment have been found.
2. The displayed alignment of these two strings.



```
Problem 2: Sequence Alignment
a. Show the final penalty cost of the alignment have been found: 15
b. Show the alignment of these two strings, in the similar format:
-AGGCTATCACCTGACC-TCCAGG-CCGAT--GCCC---
TAG-CTATCAC--GACCG-C--GGTC-GATTTGCCCGAC
```

Including the corresponding matrix where the resulting penalty cost is at the position: row = length of first string, column = length of second string.

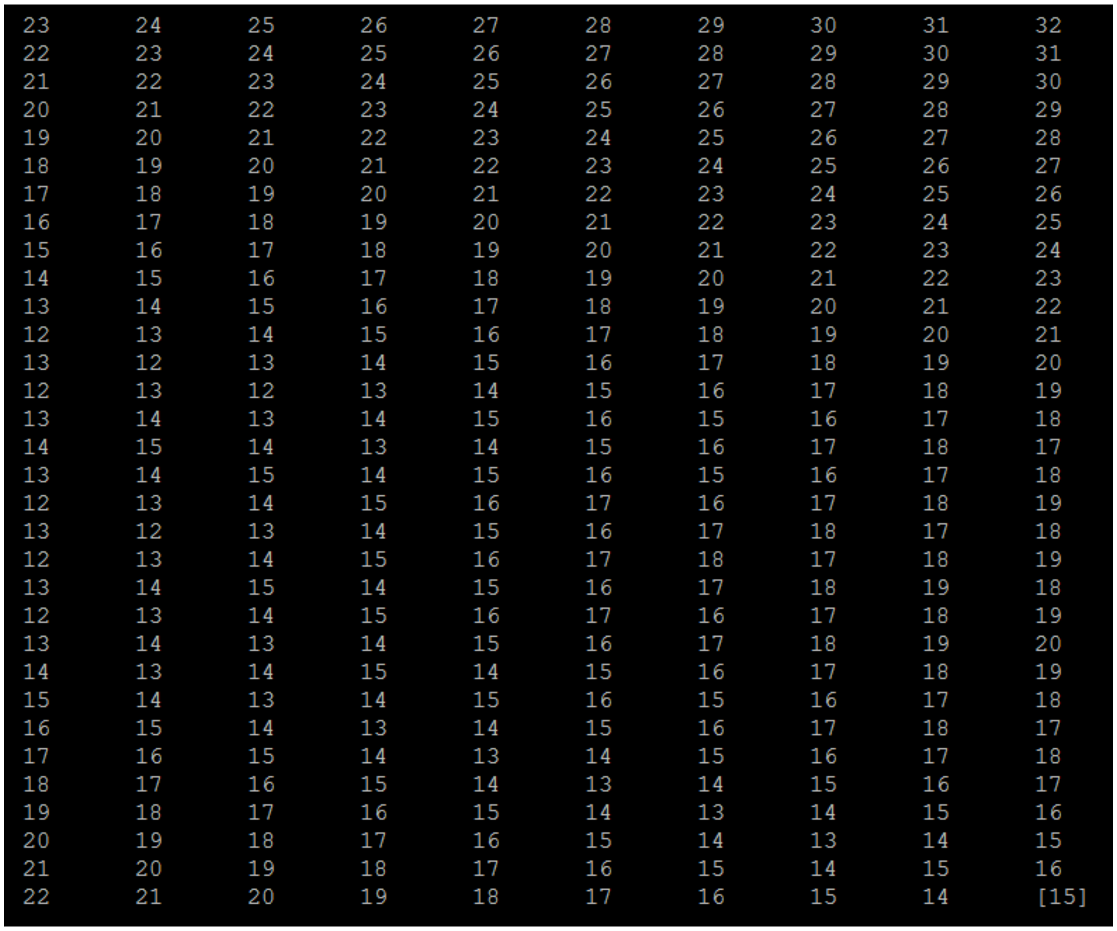
(Part1)



```
The corresponding matrix is the following table, where resulting penalty cost is at opt[31][32] (result in the [bracket]):

0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20      21      22
1       0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20      21
2       1       2       3       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19      20
3       2       3       4       3       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19
4       3       4       3       4       3       4       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
5       4       5       4       5       4       5       4       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17
6       5       4       5       6       5       6       5       4       5       6       5       6       7       8       9       10      11      12      13      14      15      16
7       6       5       4       5       6       7       6       5       6       7       6       5       6       7       8       9       10      11      12      13      14      15
8       7       6       5       4       5       6       7       6       7       8       7       6       5       6       7       8       9       10      11      12      13      14
9       8       7       6       5       4       5       6       7       8       9       8       7       6       7       8       9       8       9       10      11      12      13
10      9       8       7       6       5       6       5       6       7       8       9       8       7       8       7       8       9       8       9       10      11      12
11      10      9       8       7       6       7       6       7       8       9       10      9       8       9       8       7       8       9       10      11      12      11
12      11      10      9       8       7       8       7       8       9       10      9       10      9       10      9       8       9       10      11      12      11      12
13      12      11      10      9       8       7       8       9       10      11      10      11      10      11      10      9       8       9       10      11      12      13
14      13      12      11      10      9       8       9       10      11      12      11      12      11      12      11      10      9       10      9       10      11      12
15      14      13      12      11      10      9       10      9       10      11      12      13      12      11      12      11      10      11      10      11      12      13
16      15      14      13      12      11      10      11      10      11      12      13      14      13      12      13      12      11      12      11      10      11      12
17      16      15      14      13      12      11      12      11      12      13      14      15      14      13      14      13      12      13      12      11      12      13
18      17      16      15      14      13      12      13      12      13      14      13      14      15      14      15      14      13      14      13      12      11      12
19      18      17      16      15      14      13      12      13      14      15      14      13      14      15      14      15      14      13      14      13      12      11
20      19      18      17      16      15      14      13      12      13      14      15      14      15      14      15      16      15      14      15      14      13      12
21      20      19      18      17      16      15      14      13      14      15      16      15      14      15      16      17      16      15      14      15      14      13
22      21      20      19      18      17      16      15      14      15      16      17      16      15      16      17      18      17      16      15      14      15      14
23      22      21      20      19      18      17      16      15      16      17      16      17      16      17      18      19      18      17      16      15      14      15
24      23      22      21      20      19      18      17      16      17      18      17      18      17      18      19      20      19      18      17      16      15      16
25      24      23      22      21      20      19      18      17      16      17      18      19      18      17      18      19      20      19      18      17      16      17
26      25      24      23      22      21      20      19      18      17      18      17      18      19      18      19      20      21      20      19      18      17      18
27      26      25      24      23      22      21      20      19      18      17      18      19      20      19      20      21      22      21      20      19      18      19
28      27      26      25      24      23      22      21      20      19      18      19      20      19      20      21      22      21      22      21      20      19      20
29      28      27      26      25      24      23      22      21      20      19      20      19      20      21      20      21      22      21      22      21      20      19
30      29      28      27      26      25      24      23      22      21      20      21      20      21      22      21      20      21      22      23      22      21      20
31      30      29      28      27      26      25      24      23      22      21      20      21      22      23      22      21      22      23      24      23      22      21
```

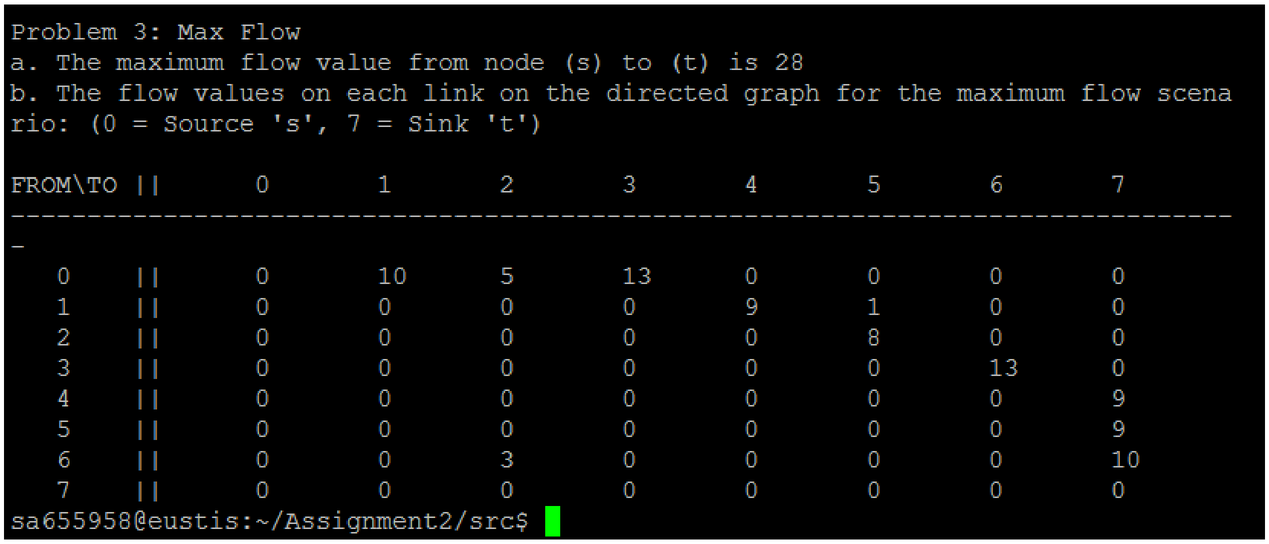
(Part2, the final outcome is in the bracket since it is where row = length of first string, column = length of second string)



```
23      24      25      26      27      28      29      30      31      32
22      23      24      25      26      27      28      29      30      31
21      22      23      24      25      26      27      28      29      30
20      21      22      23      24      25      26      27      28      29
19      20      21      22      23      24      25      26      27      28
18      19      20      21      22      23      24      25      26      27
17      18      19      20      21      22      23      24      25      26
16      17      18      19      20      21      22      23      24      25
15      16      17      18      19      20      21      22      23      24
14      15      16      17      18      19      20      21      22      23
13      14      15      16      17      18      19      20      21      22
12      13      14      15      16      17      18      19      20      21
13      12      13      14      15      16      17      18      19      20
12      13      12      13      14      15      16      17      18      19
13      14      13      14      15      16      15      16      17      18
14      15      14      13      14      15      16      17      18      17
13      14      15      14      15      16      15      16      17      18
12      13      14      15      16      17      16      17      18      19
13      12      13      14      15      16      17      18      17      18
12      13      14      15      16      17      18      17      18      19
13      14      15      14      15      16      17      18      19      18
12      13      14      15      16      17      16      17      18      19
13      14      13      14      15      16      17      18      19      20
14      13      14      15      14      15      16      17      18      19
15      14      13      14      15      16      15      16      17      18
16      15      14      13      14      15      16      17      18      17
17      16      15      14      13      14      15      16      17      18
18      17      16      15      14      13      14      15      16      17
19      18      17      16      15      14      13      14      15      16
20      19      18      17      16      15      14      13      14      15
21      20      19      18      17      16      15      14      15      16
22      21      20      19      18      17      16      15      14      [15]
```

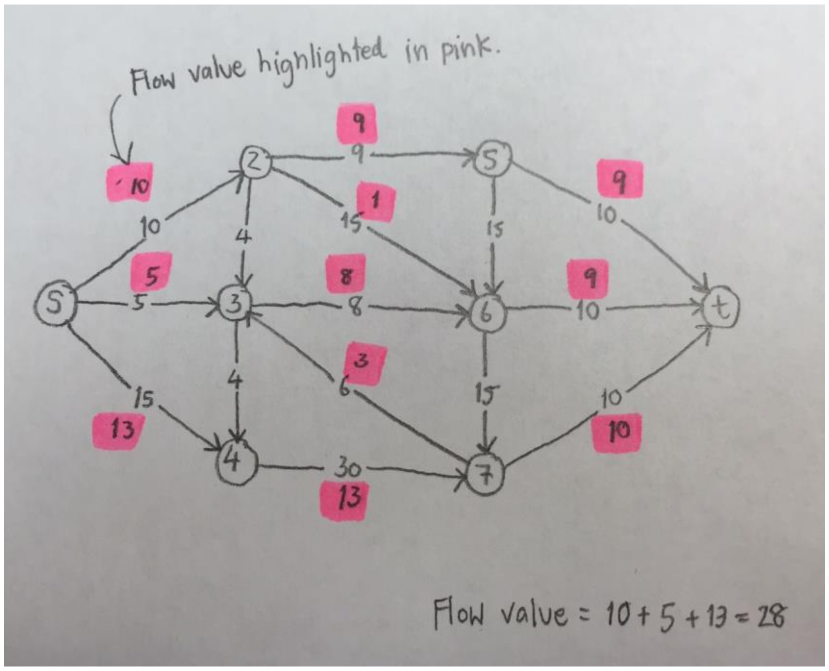
Problem 3:

1. The maximum flow value from node source (s) to sink (t).
2. The displayed flow values on each link on the directed graph for the maximum flow scenario.



```
Problem 3: Max Flow
a. The maximum flow value from node (s) to (t) is 28
b. The flow values on each link on the directed graph for the maximum flow scena
rio: (0 = Source 's', 7 = Sink 't')

FROM\TO ||      0       1       2       3       4       5       6       7
--------------------------------------------------------------------------------
-
   0    ||      0       10      5       13      0       0       0       0
   1    ||      0       0       0       0       9       1       0       0
   2    ||      0       0       0       0       0       8       0       0
   3    ||      0       0       0       0       0       0       13      0
   4    ||      0       0       0       0       0       0       0       9
   5    ||      0       0       0       0       0       0       0       9
   6    ||      0       0       3       0       0       0       0       10
   7    ||      0       0       0       0       0       0       0       0
sa655958@eustis:~/Assignment2/src$
```


Flow value highlighted in pink.
Flow value = 10 + 5 + 13 = 28

- ***Discussions***:

Each of the three problems can be discussed for the running time as follows.

RNA Secondary Structure

There are O(n^2) sub-problems to solve, and evaluating the recurrence of the given sequence B= AUGGCUACCGGUCGAUUGAGCGCCAAUGUAAUCAUU takes time O(n) for each. Thus the resulting running time is O(n^3).

Sequence Alignment

The running time is *O(mn)* for finding global and local alignments, since the array A has *O(mn)* entries, and at worst the program spends constant time on each.

Max Flow: Ford-Fulkerson Algorithm

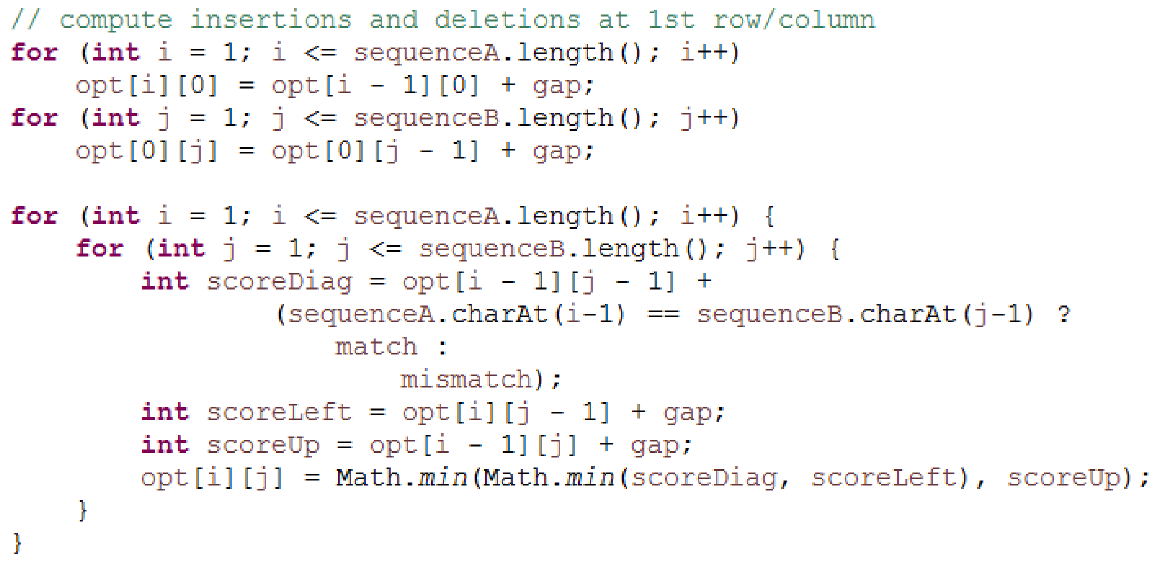
The flow augmenting path is being added to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found in the graph. The best way to guarantee that the answer will be correct if the algorithm terminates. The runtime of Ford-Fulkerson is bounded by *O(Ef),* where E is the number of edges in the graph and f is the maximum flow in the graph, because each augmenting path can be found (to increase the flow by an integer amount of at least 1) in O(E).

- ***Implementation problems and possible solutions/improvements***:

The interesting issues occurred in the implementation of Sequence Alignment, details as follows:

1. Initially, my implementation align values in the matrix from the bottom-right corner to the top-left corner, which causes the program to align the *reverse* of the two given sequences instead of the sequence itself. To solve this, I needed to flip the generated matrix table and re-compute the minimum cost, or simpler than that, I chose to the reverse both sequences before inputting them in the program. The result came out perfectly correct with the minimum cost possible.

2. The insertions and deletion at the first row procedure were missing at first, so I added the follow piece to take care of this issue:



```
// compute insertions and deletions at 1st row/column
for (int i = 1; i <= sequenceA.length(); i++)
    opt[i][0] = opt[i - 1][0] + gap;
for (int j = 1; j <= sequenceB.length(); j++)
    opt[0][j] = opt[0][j - 1] + gap;

for (int i = 1; i <= sequenceA.length(); i++) {
    for (int j = 1; j <= sequenceB.length(); j++) {
        int scoreDiag = opt[i - 1][j - 1] +
                (sequenceA.charAt(i-1) == sequenceB.charAt(j-1) ?
                    match :
                        mismatch);
        int scoreLeft = opt[i][j - 1] + gap;
        int scoreUp = opt[i - 1][j] + gap;
        opt[i][j] = Math.min(Math.min(scoreDiag, scoreLeft), scoreUp);
    }
}
```