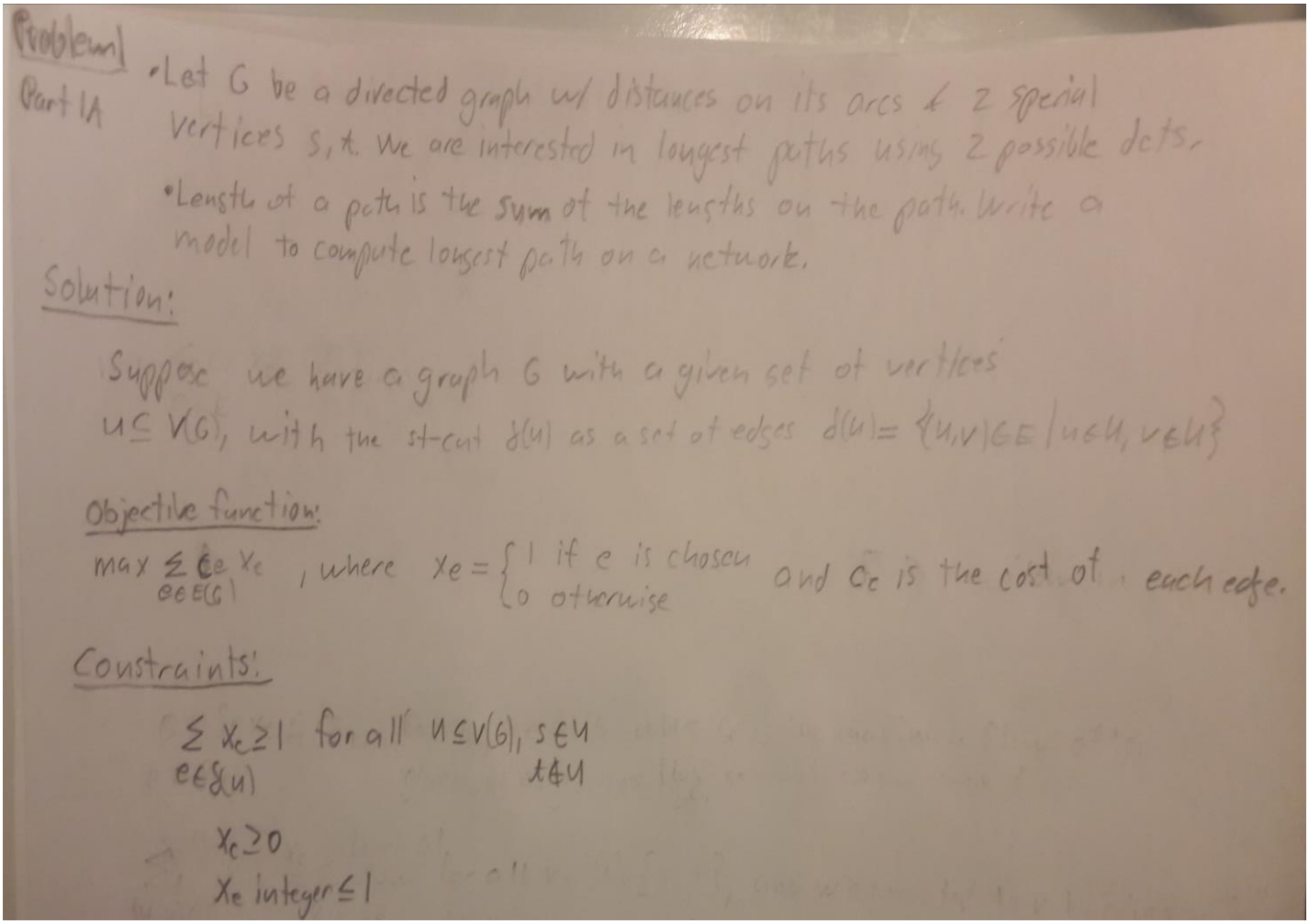
MAT 160 PROJECT 3

1. CHALLENGE: PROBLEMS ON GRAPHS & NETWORKS: We saw how the networks can be used to plan for building projects or to schedule the parts of a project. Here are some some more challenges:

   - Let G be a directed graph with distances or costs on its arcs and two special vertices s, t. We are interested on the "longest" paths using two possible definitions
     - If we call the length of a path the sum of the lengths on the path. Write a model to compute the longest path on a network.
     - If we instead call the length of a path the largest length among arcs the path, write another model to compute the longest path on a network under this definition.
   - Consider again, the TSP for n cities and cost of travel c_{ij}. Write a new model for the TSP, different from the one we saw in class. This time us the binary variables $x_{i,j,k}$, which are 1if the k-th leg of the trip, the salesman goes from city i to city j. Your formulation must have a cubic number of constraints.



Problem 1. part 2).

$d(i,j)$ distance between cti i and j

$X(i,j)$ binary specifying whether city i is traveled on k^{th}

$t(i,j,k)$ binary which shows if i and j are connected at k^{th} order

$o(i,j)$ binary representing i and j are conected

Minimize $z = \sum_{i=1}^{n} \sum_{j=1}^{n} o(i,j) * d(i,j)$

Constraints

$X(i,k) + X(j,k+1) \geq 2\,(t(i,j,k))$ $\quad k+1=1$ if $k=n$

$X(i,k) + X(j,k+1) \leq 2\,(t(i,j,k)) + 1$

$o(i,j) = \sum_{k=1}^{n} t(i,j,k)$

$\sum_{k=1}^{n} X(i,k) = 1 \qquad \sum_{k=1}^{n} X(j,k) = 1$

So the total number of constraints is $2n^3+n^2+2n+1$

Cubic number of constraints

2. CHALLENGE: FINDING A GOOD MATCH: Matching problems appear everywhere in applications. E.g., how do students get matched to medical schools? How do workers get assigned to jobs at work? How do organ donors get match to patients in need?

We can model matchings in graphs. A set M of the edges E is called a *matching* of G if and only if:

$$\forall u \in V,\ |\{e \in M \mid u \in e\}| \leq 1$$

In other words, a vertex is incident to at most one edge of M. A matching M is said to be *maximal* if there is no matching M' with $M \subset M'$. A maximal matching of the largest cardinality is a *maximum* matching. The size of a maximum matching is denoted by $\texttt{OPT}_G$.

In the following exercises we will explore several approaches to find a maximum matching in a graph:

a. Let M be a matching and let us denote by $V(M)$ the set of endpoints of edges in M:

$$V(M) := \{u \mid \exists e \in M,\ u \in e\}$$

Show that when M is a maximal matching of G, $V(M)$ is a vertex cover of G.

b. Let M be a maximal matching of G, show that $|V(M)| \leq 2\texttt{OPT}_G$.

c. Now consider the following *greedy* algorithm for finding a maximum matching: Start with an arbitrary edge as the very first matching. Find another edge that does not have a vertex in common with the current matching. If one exists add it to the current matching. Repeat until no more edges can be added.

d. What is the running time of this algorithm on a graph with n nodes and m edges?

e. Give an example where the algorithm fails to find a maximum matching. But show that the greedy method always yields a solution that has at least half as many edges as a true maximum matching.

f. Give now an integer-optimization model (i.e., now based on linear equations, inequalities) to construct a maximum matching of any graph. G.

2.) a.)

Suppose for graph $G(V, E)$
we have a maximal matching, M,
such that $\forall u \in V, |\{e \in M \mid u \in e\}| \leq 1$
and for every matching of G, M', $M \subseteq M'$
Let $V(M) = \{u \mid \exists e \in M, u \in e\}$ ie endpoints of M
Since the introduction of any edge
on M would imply $M \subset M'$
thus $\forall e \in E(G)\ \exists e' \in E(M)$ s.t. $ee' = \emptyset$
then let $V' \subseteq V(G)$ such that
$uv \in E(G) \Rightarrow u \in V'$ or $v \in V'$
then let $v = e' \in E(M)$
thus $V' = V(M)$ or vertex cover = Vertices of max matching

2.) b.)

Suppose graph, $G(V, E)$ is not trivial
then the max matching of G, $M \neq \emptyset$
Let $V(M)$ be the endpoints of M
then $\min\{|V(M)|\} = 2$ since for
any $uv \in E(M)$ there are 2 vertices.
thus by induction $|V(m)| \leq 2 \cdot |M|$

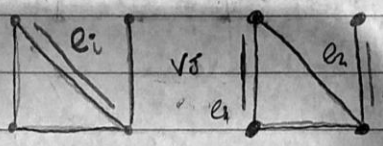
2.) c.d.)

Suppose a graph with n-nodes
and m-edges.
Using the greedy algorithm one has to check $\binom{m}{2}K$ match of G
since every edge is check for adjacency
thus the runny time $O \sim m!$

2.) e.)

Example: when max matching
isn't found

$|M'| = 1$ $|M| = 2$



Here M' matching of
G, $|M'| = 1$
but not max match
of G, which is $|M| = 2$

thus we see the greedy algorithm
success is dependent on choice of initial
edge e_i

2.) f.)

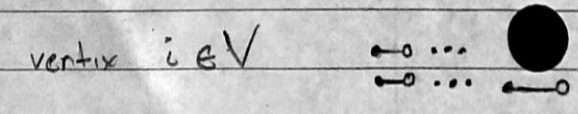
Suppose graph $G(V, E)$
Let $|M|$ be the cardinality of Max Match of G
then $x_{ik} = \begin{cases} 1 & \text{if } x \in M \quad (i,k) \in E(G) \\ 0 & \text{otherwise} \end{cases}$

Objective function:

$$\text{Min}\left\{\sum_{\{E\}} x_{ik}\right\} \Leftrightarrow \text{Max}\{|M|\}$$

Constraints:

- $x_{ik} \in \{0, 1\}$
- $\sum_{\{\alpha \mid \forall \alpha \text{ adjecent to } i\}} x_{i\alpha} \leq 1$ for every vertix $i \in V$



- $\sum_{\{\beta \mid \forall \beta \text{ adj. to } i\}} x_{i\beta} + \sum_{\{\beta \mid \forall \beta \text{ adj to } k\}} x_{\beta k} \leq 1 \quad \forall (i,k) \in E(G)$

3. Let G be a graph with two distinguished vertices s, t. An even st-path is a path from s to t with an even number of edges. Show that we can formulate the problem of finding an even st-path with as few edges as possible as a minimum cost perfect matching problem. HINT: Construct an auxiliary graph H from G, make a copy of the graph G, and remove vertices s, t. Call that new graph G'. Construct H starting with the union of G, G' and joining every vertex $v \in G$ different from s, t with its copy in G'. Use H.

Solution:

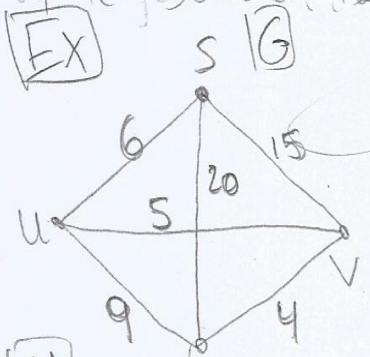
We construct G' as follows,

For each vertex $v \in V$, create two vertices V_0 and V_1 in G'. For each edge $(u,v) \in E$, create two edges (u_0, v_1) and (u_1, v_0) in G'.

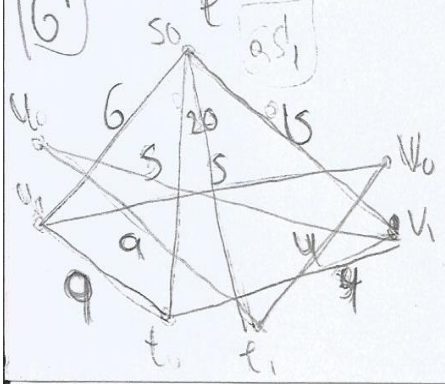
After that, we use Dijkstra algorithm on G', with start vertex S_0.

So, All paths vertex in G' ending at vertex V_0 have an even number of edges, and all paths ending at a vertex V_1 have an odd number of edges. Thus, the shortest even path (st-path) to a vertex e in G is the shortest path from S_0 to t_0 in G', without the subscripts which we called H.

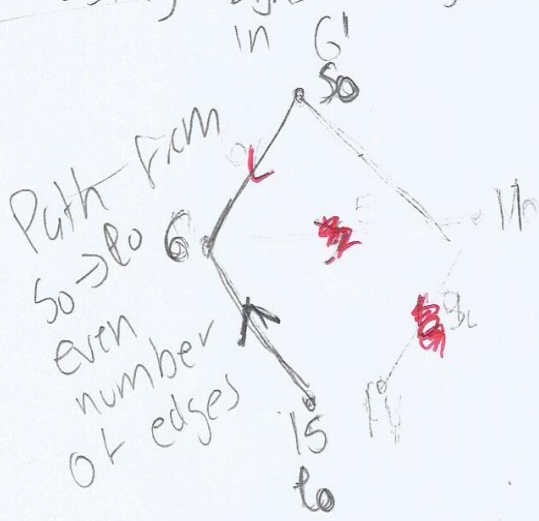
EX

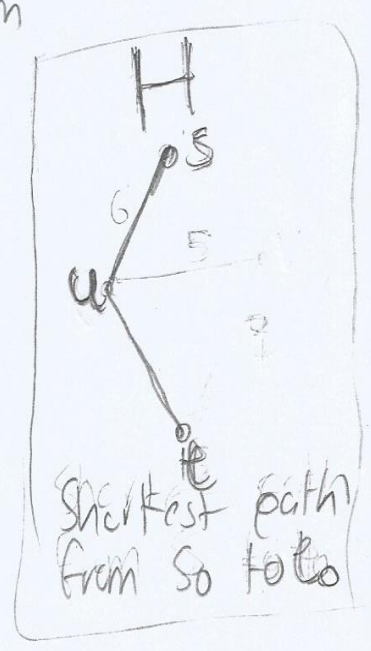


G'



Using Dijkstra Algorithm in G'





4. THEORY PROBLEMS: GEOMETRY OF INTEGER SOLUTIONS:

- MEDITATE ABOUT THIS: In the previous problems you dealt with several types of combinatorial optimization problems. Which ones have polynomial time solution? Which ones are NP-hard? Discuss whatever information you find about their complexity.
- Suppose $S = \{(y_1, y_2) \in Z^2 : y_1 - y_2 \leq 2,\ 3y_1 + y_2 \leq 21, y_1 + 5y_2 \leq 34\}$. Find (a) An inequality description of convex hull of S. (b) Find the extreme points of $conv(S)$.
- Use the branch-and-bound method to solve the following optimization problem. Show the solution graphically:

$$\begin{aligned} &\min\ y_1 + 3y_2 \\ &\textbf{subject to:} y_1 + 5y_2 \leq 12, \\ &y1 + 2y_2 \leq 8, \\ &y_1, y_2 \leq 0 \ \textbf{integer} \end{aligned}$$

.

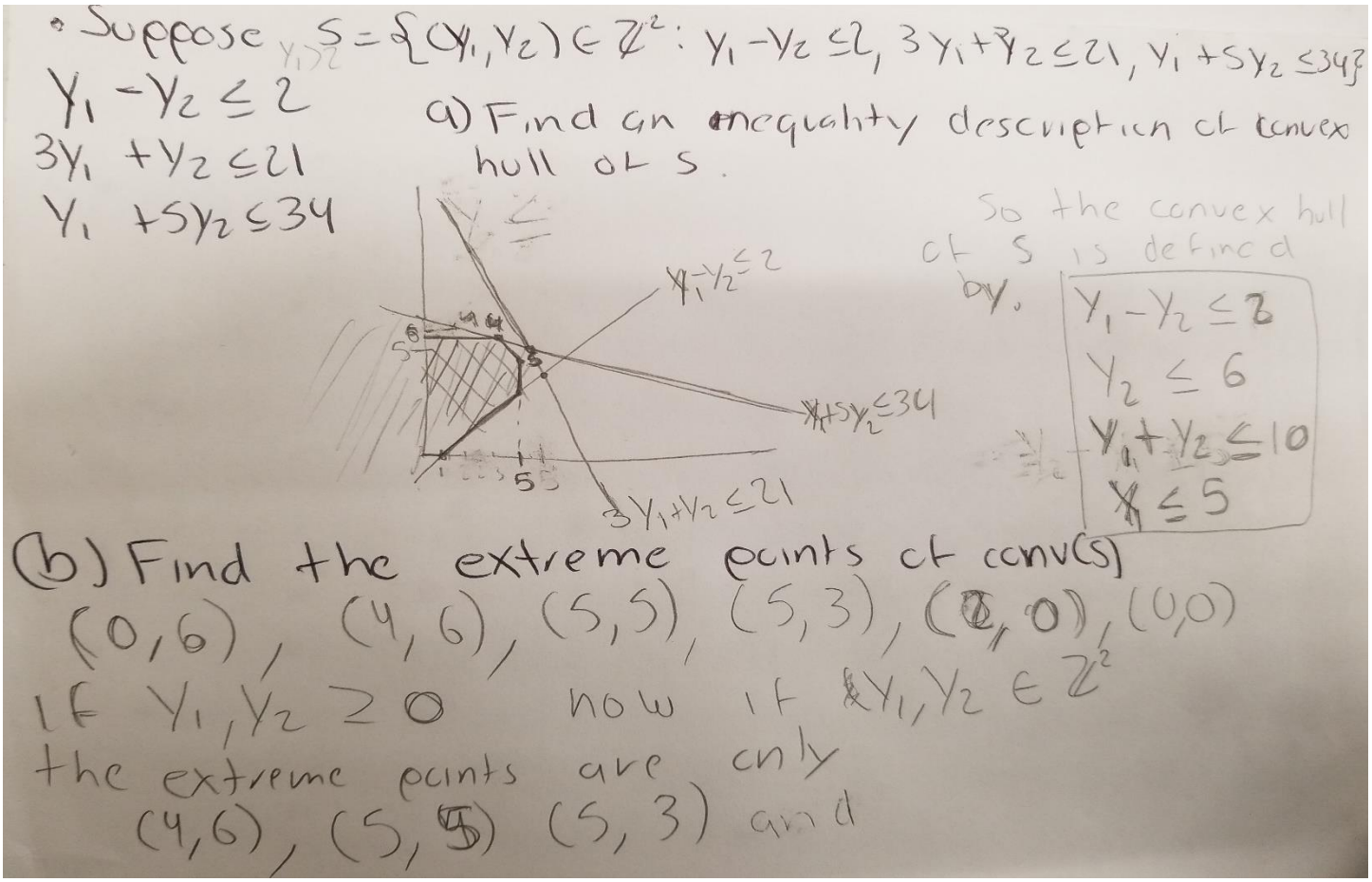
- Generate a valid inequality using Chvátal-Gomory cut procedure for the following problem:

$$\begin{aligned} &\min\ y_1 + y_2 + y_3 \\ &\textbf{subject to:} 3y_1 + 5y_2 - y_3 \leq 12, \\ &y1 + y_3 \leq 7, \\ &y1 - y_2 + 2y_3 \leq 9, \\ &y_1, y_2, y_3 \leq 0 \ \textbf{integer} \end{aligned}$$

In problem 1, the longest path is NP-hard, the TSP is also NP-hard as we already discussed.

In problem 2, problem 2f can be solved in polynomial time.

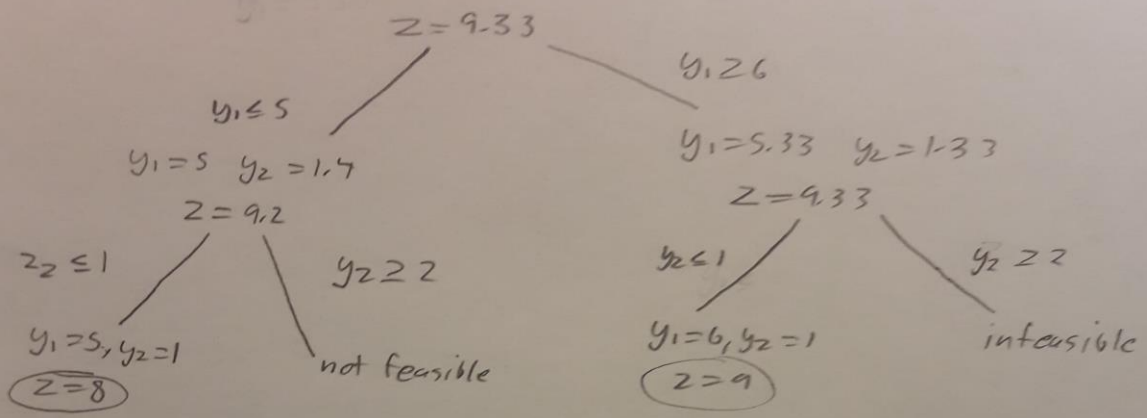
In problem 3, the even st-path can be also solved in polynomial time, since it is only a slightly variation of the regular st-path



Problem 4 Part 2

Use branch and bound method to solve, show solution graphically

We have optimal solution $y_1 = 5.33$ and $y_2 = 1.33$. using Branch and bound we can find optimal integer solution.



Hence $y_1 = 6$ and $y_2 = 1$ are the best solution, with maximum objective function $Z = 9$.

Generate a valid inequality using Chvatal-Gomory cut procedure for following problem:

$$\max \; y_1 + y_2 + y_3$$

$$\text{subject to } 3y_1 + 5y_2 - y_3 \leq 12$$

$$y_1 + y_3 \leq 7$$

$$y_1 - y_2 + 2y_3 \leq 4$$

$$y_1, y_2, y_3 \geq 0 \text{ integer}$$

After first iteration, we have

$$y_2 + 7/2\, y_4 + 1/7\, y_6 = 4\,5/7$$

$$\Rightarrow y_2 - 4 = 5/7 - 2/7\, y_4 - 1/7\, y_6 \Rightarrow 5/7 - 2/7\, y_4 - 1/7\, y_6 =$$

$$\Rightarrow -\frac{2}{7} y_4 - 1/7\, y_6 \leq -5/7$$

Hence the inequality we have to add is

$$\boxed{\frac{7}{2} y_4 + \frac{1}{7} y_6 \geq 5/7}$$

5. CHALLENGE: Applications to genetics, DNA SEQUENCE ALIGNMENT:

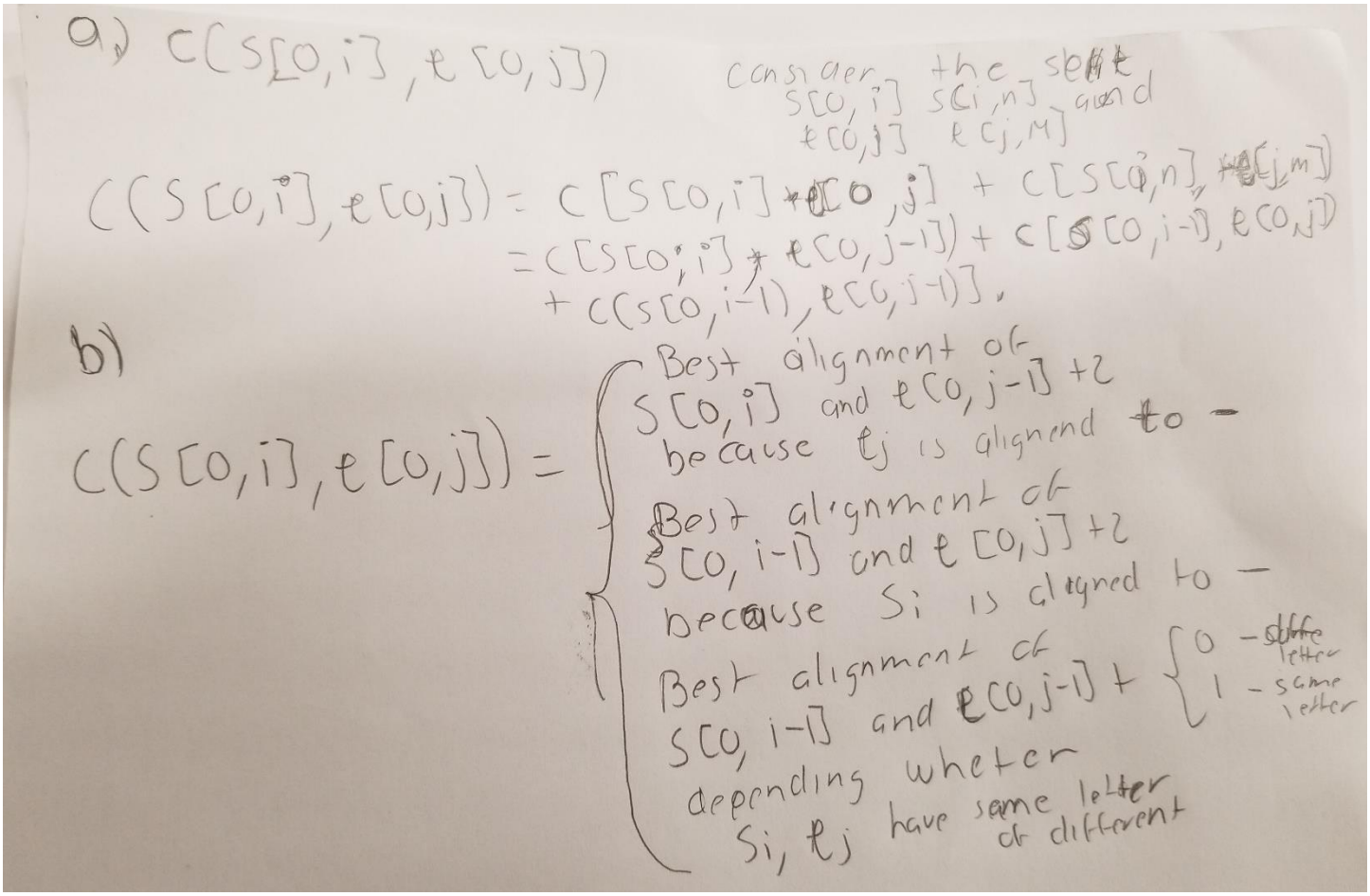
a. For a sequence s, $s[i:j]$ will denote the substring of s ranging from index i inclusive to j exclusive. Give a formula to compute $c(s[0:i], t[0:j])$ as a function of $c(s[0:i], t[0:j-1])$, $c(s[0:i-1], t[0:j])$ and $c(s[0:i-1], t[0:j-1])$. Think recursively.

b. Using part a. design and describe an algorithm to compute $c(s,t)$ [**hint:** think dynamically! It is very useful to think of this problem as a type of matching problem].

c. Implement the algorithm you designed above. A test dataset is available at `https://www.math.ucdavis.edu/~deloera/TEACHING/MATH160/dna.txt`. Each line of the datafile is the list of the first 10,000 base pairs of the genome of the well-known *Escherichia coli* bacteria (why is it famous? Do you know?). There are only two lines corresponding to two different species of this bacteria.

d. Write code (using MATLAB and/or SCIP) which reads the datafile and outputs the cost of the optimal alignment of the two DNA sequences. Do not forget to Submit the code you wrote. Using C or Python to help yourself is allowed.

e. Two DNA sequences are considered to have the same biological function if the cost of the optimal alignment, divided by the length of the sequence is smaller than 5%. Do the two DNA sequences in the datafile have the same biological function?



MATLAB CODE

```
function [TotalCost, Alignment_S, Alignment_T] = DNAsequence(seq, ~)
%tic;
[A, B] = textread(seq,'%s %s','delimiter','\n','bufsize',50000);
S = cell2mat(A); %sequence s
T = cell2mat(B); %sequence t

%gap penalty = -2
g = -2;
%match = 1, mismatch = -1
m = 1;
s = -1;

%gap | mismatch to compute total cost c := 2 * gap + m
gap = 2;
mistmatch = 1;

col = length(S);
row = length(T);
% rows|columns

%preallocating matrix F with the gap penalty as the top row and column
F = zeros(row+1,col+1);
F(2:end,1) = g * (1:row)';
F(1,2:end) = g * (1:col);

%variables
if (nargin == 3)
    matchS = 1;
    matchT = 1;
    %sets first match position
    counterS = matches(matchS);
    counterT = matches(matchT,3);
end

TotalCost = 0;
scores = zeros(row, col);
%Filling in the matrix
for i=2:row+1
    for j=2:col+1
        %wrapper function for input matches file
        if (nargin == 3)
           if (j == counterS && i == counterT)
               %assigns the position as a match
               F(i,j) = F(i-1,j-1) + scores(S(j-1),T(i-1));
               stop = matches(matchS,2);
               final = size(matches,1);
               %checks if end of the known match
               if (counterS == stop)
                   %checks if  end of the text file
                   if (matchS == final)
                       continue
                   else
                       matchS = matchS + 1;
                   end
               else
```

```
                    %counts known matched sequence
                    counterS = counterS + 1;
                    counterT = counterT + 1;
                end
                %skips to the next iteration
                continue
            end
        end
        %if the two positions match
        if (S(j-1) == T(i-1))
            scores(S(j-1),T(i-1)) = m;
        else
            scores(S(j-1),T(i-1)) = s;
        end
        %Filling-in partial alignments here
        Match     = F(i-1,j-1) + scores(S(j-1),T(i-1));
        MismatchS = F(i, j-1) + g;
        MismatchT = F(i-1, j) + g;
        %choose final score of the alignment and assigning it to F
        Temp = [Match MismatchS MismatchT];
        F(i,j) = max(Temp);
    end
end

Alignment_S = '';
Alignment_T = '';
i = length(T)+1; %row
j = length(S)+1; %col

while (i>1 && j>1)
   Score = F(i,j);
   DIAG = F(i-1,j-1);
   LEFT = F(i-1,j);
   UP   = F(i,j-1);

   %if scores are equal to the diagonal, No gap.
   if (Score == DIAG + scores(S(j-1),T(i-1)))
       Alignment_S = strcat(Alignment_S, S(j-1));
       Alignment_T = strcat(Alignment_T, T(i-1));
       %computes score, checks to see if the alignment are the same
       %characters
       if (S(j-1) ~= T(i-1))
           %mismatch
           TotalCost = TotalCost + mistmatch;
       end
       i = i-1;
       j = j-1;

   %gap in sequence T
   elseif (Score == UP + g)
       Alignment_S = strcat(Alignment_S, S(j-1));
       Alignment_T = strcat(Alignment_T, '-');
       j = j-1;
       %computes score by adding the gap penalty
       TotalCost = TotalCost + gap;
   %gap in sequence S
   else
       Alignment_S = strcat(Alignment_S, '-');
```

```
        Alignment_T = strcat(Alignment_T, T(i-1));
        i = i-1;
        TotalCost = TotalCost + gap;
    end
end
%Fill end of a sequence with gaps
while(j>1)
    Alignment_S = strcat(Alignment_S, S(j-1));
    Alignment_T = strcat(Alignment_T, '-');
    j = j-1;
    %2 for gaps
    TotalCost = TotalCost + gap;
end
while(i>1)
    Alignment_S = strcat(Alignment_S, '-');
    Alignment_T = strcat(Alignment_T, T(i-1));
    i = i-1;
    TotalCost = TotalCost + gap;
end
Alignment_S = fliplr(Alignment_S);
Alignment_T = fliplr(Alignment_T);
%displays the alignment score and alignments
disp('Total Aligment Cost =');
disp(TotalCost);
disp('Aligment S =');
disp(Alignment_S);
disp('Aligment T =');
disp(Alignment_T);
%toc;
```

PART e.

Since alignment cost = 223. The two sequences have same biological sequence because (223/10000 = 0.0223 < 0.05) the cost of the optimal alignment, divided by the length of the sequence is smaller than 5%.