Problem 1

(a)

$$c[i,j,k] = \begin{cases} 0 & \text{if } i = 0 \vee j = 0 \vee k = 0 \\ c[i-1,j-1,k-1]+1 & \text{if } i,j,k > 0 \wedge x_i = y_j = z_k \\ \max(c[i-1,j,k], c[i,j-1,k], c[i,j,k-1]) & \text{if } i,j,k > 0 \wedge (x_i \neq y_j \vee x_i \neq z_k \vee y_j \neq z_k) \end{cases}$$

(b) For the base case, at least one string is empty ($i = 0 \vee j = 0 \vee k = 0$), so we set $c[i,j,k] = 0$, which is correct since the longest common subsequence for empty strings is 0.

The general case is divided into two further cases, similar to the LCS recurrence for two sequences. For $i,j,k > 0$, we consider the characters $W[i]$, $X[j]$, and $Y[k]$. If $W[i] = X[j] = Y[k]$, then we use the result at $c[i-1,j-1,k-1]$ and add 1 to indicate the matched character.

If at least one character of $W[i], X[j], Y[k]$ differs, then we take the maximum of $c[i-1,j,k], c[i,j-1,k], c[i,j,k-1]$, representing the lengths of the longest common subsequences of $\{W[1\ldots i-1], Y[1\ldots j], Z[1\ldots k]\}$, $\{W[1\ldots i], Y[1\ldots j-1], Z[1\ldots k]\}$, and $\{W[1\ldots i], Y[1\ldots j], Z[1\ldots k-1]\}$ (i.e. the maximum result of *not* advancing one of the characters). Since the characters differ, we consider $c[i,j,k]$ to be equivalent to the subcases, as if the current character $W[i]$, $X[j]$, or $Y[k]$ didn't exist.

In both cases, the recurrence yields the optimal solution for $c[i,j,k]$.

(c) Algorithm for finding the length of the longest common subsequence of three strings: (see next page).

(d) We assume the creation of the 3D array takes $O(1)$ time. Let k the time to execute lines 6 to 12. Then the running time is $1 + k(p+1)(m+1)(n+1) \leq ckpmn = O(pmn)$ for some constant c for large enough p,m,n.

Algorithm 1 Longest common subsequence of three sequences algorithm for Problem 1c.

```
1: function LCS3(W, X, Y, p, m, n)
2:     Create 3D array c[0...p, 0...m, 0...n].                    ▷ O(1)
3:     for i = 0 to p do                                          ▷ Fill c bottom-up.
4:         for j = 0 to m do
5:             for k = 0 to n do
6:                 if i = 0 or j = 0 or k = 0 then
7:                     c[i, j, k] ← 0
8:                 else if W[i] = X[j] and W[i] = Y[k] then       ▷ Found a matching character.
9:                     c[i, j, k] ← c[i − 1, j − 1, k − 1] + 1
10:                else
11:                    c[i, j, k] ← max(c[i − 1, j, k], c[i, j − 1, k], c[i, j, k − 1])
12:                end if
13:            end for
14:        end for
15:    end for
16:    return c[p, m, n]
17: end function
```

Problem 2

(a) We define a table $c[1 \ldots m, 1 \ldots n]$ with the following recurrence on each entry:

$$c[i,j] = \begin{cases} P(1,1) & \text{if } i = j = 1 \\ c[1, j-1] + P(1,j) & \text{if } i = 1, j > 1 \\ c[i-1, 1] + P(i,1) & \text{if } i > 1, j = 1 \\ \max\big(c[i,j-1] + P(i,j), c[i-1,j] + P(i,j), c[i-1,j-1] + \frac{3}{2}P(i,j)\big) & \text{if } i > 1, j > 1 \end{cases}$$

(b) After filling the table, the largest possible value walk will be stored in $c[m,n]$.

(c) The base case is the top left corner (1, 1). The only possible walk there is the value of the cell itself.

Along the upper and left edges, we only consider walks coming from the left and above respectively. (E.g. (1, 2) can't be approached from above, only from the left.) Thus in the recurrence, there is only one possible recurrence path, which is $c[1, j-1]$ for the upper edge, and $c[i-1, 1]$ for the left edge. As per the problem description, approaching a cell orthogonally only yields the value of said cell.

Finally, we consider cells not on the upper or left edge (i.e. $i > 1, j > 1$). These cells can be approached from all three directions (from the left, from above, and both). As per the problem description, approaching a cell diagonally yields the value of said cell with multiplier $\frac{3}{2}$. We gather all three cases and take the maximum to ensure that our memoised entry at $c[i,j]$ stores the value of the largest possible walk.

(d) Algorithm for finding the length of the longest common subsequence of three strings: (see next page).

(e) Let k be the time needed to executed lines 5 to 13. Since there are two loops iterating m and n times, so the running time is $kmn \leq cmn = O(mn)$ for some constant c for large enough m, n.

Algorithm 2 Prize collecting walk algorithm for Problem 2d.

function PRIZECOLLECTINGWALK(P, m, n)
    Create 2D array $c[1 \ldots m, 1 \ldots n]$. ▷ $O(1)$
    for $i = 1$ to m **do** ▷ Fill c bottom-up.
        for $j = 1$ to n **do**
            if $i = 1$ and $j = 1$ **then** ▷ Base case.
                $c[i, j] \leftarrow P[1, 1]$
            else if $i = 1$ and $j > 1$ **then** ▷ Upper edge.
                $c[i, j] \leftarrow c[i, j-1] + P(1, j)$
            else if $i > 1$ and $j = 1$ **then** ▷ Left edge.
                $c[i, j] \leftarrow c[i-1, j] + P(i, 1)$
            else ▷ General case, approach from three directions.
                $c[i, j] \leftarrow \max\big(c[i, j-1] + P[i, j], c[i-1, j] + P[i, j], c[i-1, j-1] + \frac{3}{2}P[i, j]\big)$
            end if
        end for
    end for
    return $c[m, n]$
end function

Problem 3

(a) (i) Here, $(i,j)-(k,l)$ indicates that an edge between the points (i,j) and (k,l).

$$\{(1,0)-(2,0)-\cdots-(n-1,0)\} \qquad \text{South wall.}$$

$$\cup\{(n-1,0)-(n-1,1)-(n-1,2)-\cdots-(n-1,n-1)\} \qquad \text{East wall.}$$

$$\cup\{(n-1,n-1)-(n-2,n-1)-(n-3,n-1)-\cdots-(0,n-1)\} \qquad \text{North wall.}$$

$$\cup\{(i-1,j)-(i,j) \mid 1\le i\le n-2, 1\le j\le n-2\} \qquad \text{H lines.}$$

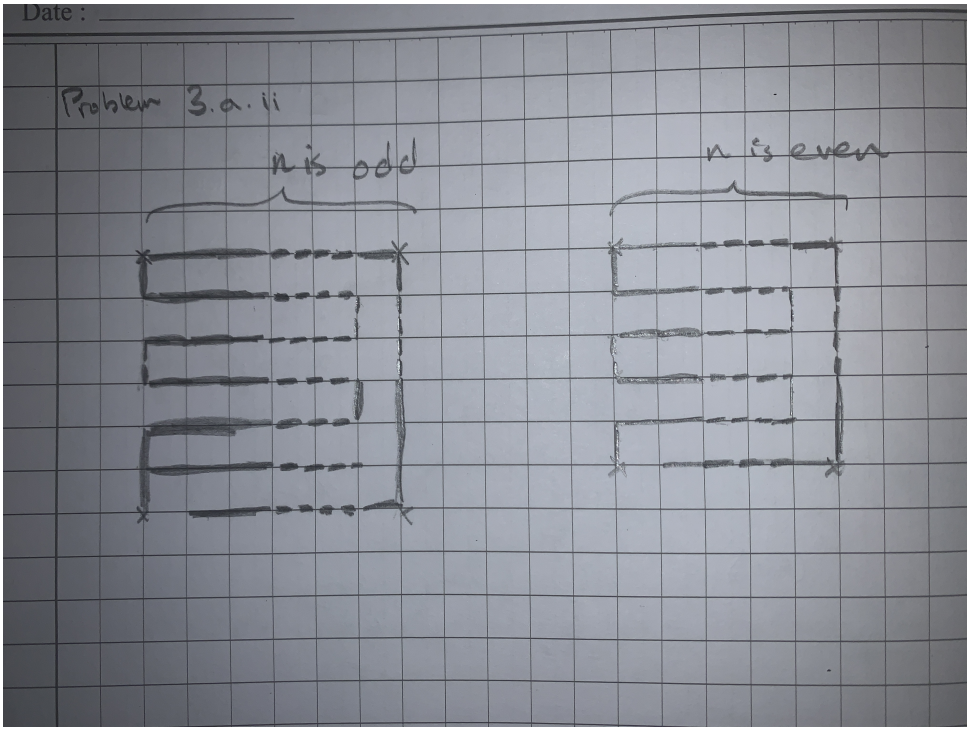
$$\cup\{(0,n-1-2k)-(0,n-2-2k) \mid 0\le k<\left\lfloor\frac{n}{2}\right\rfloor\} \qquad \text{V lines left.}$$

$$\cup\{(n-2,n-2-2k)-(n-2,n-3-2k) \mid 0\le k<\left\lfloor\frac{n}{2}\right\rfloor-1\} \qquad \text{V lines right.}$$

$$\cup R_n \qquad \text{The rest}$$

$$\text{where } R_n=\begin{cases}\{(0,1)-(0,0)\} & \text{if } n \text{ is odd}\\ \varnothing & \text{if } n \text{ is even}\end{cases}$$

(ii)



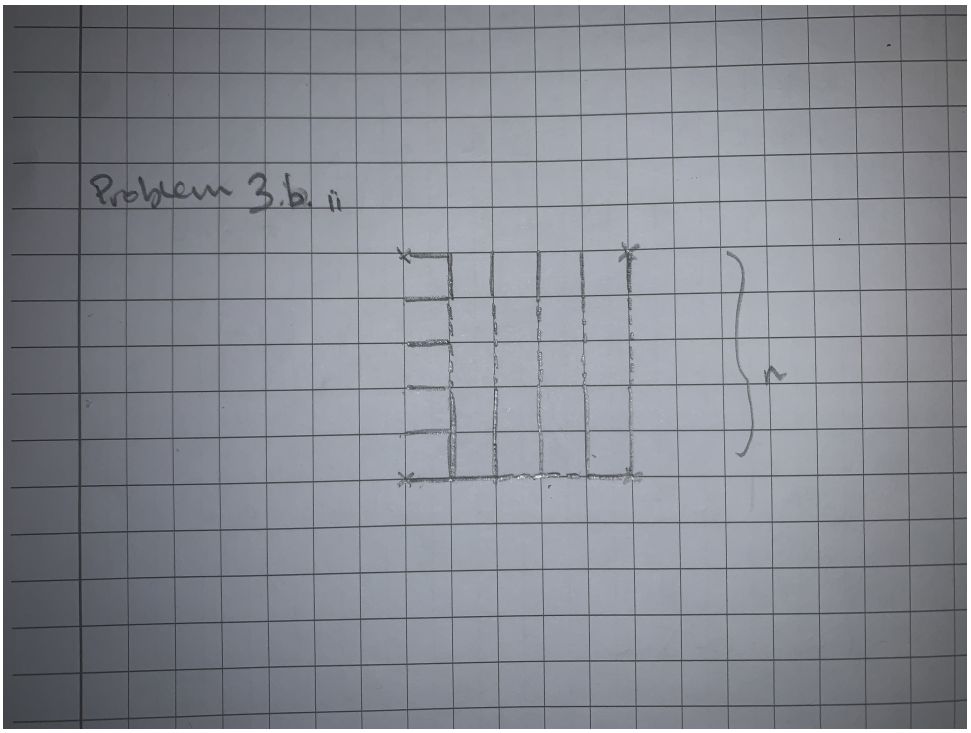
(b) (i)

$$\{(0,0)-(1,0)-\cdots-(n-1,0)\} \qquad \text{Base.}$$

$$\cup\{(i,0)-(i,1)-\cdots-(i,n-1) \mid 1 \le i \le n-1\} \qquad \text{V lines.}$$

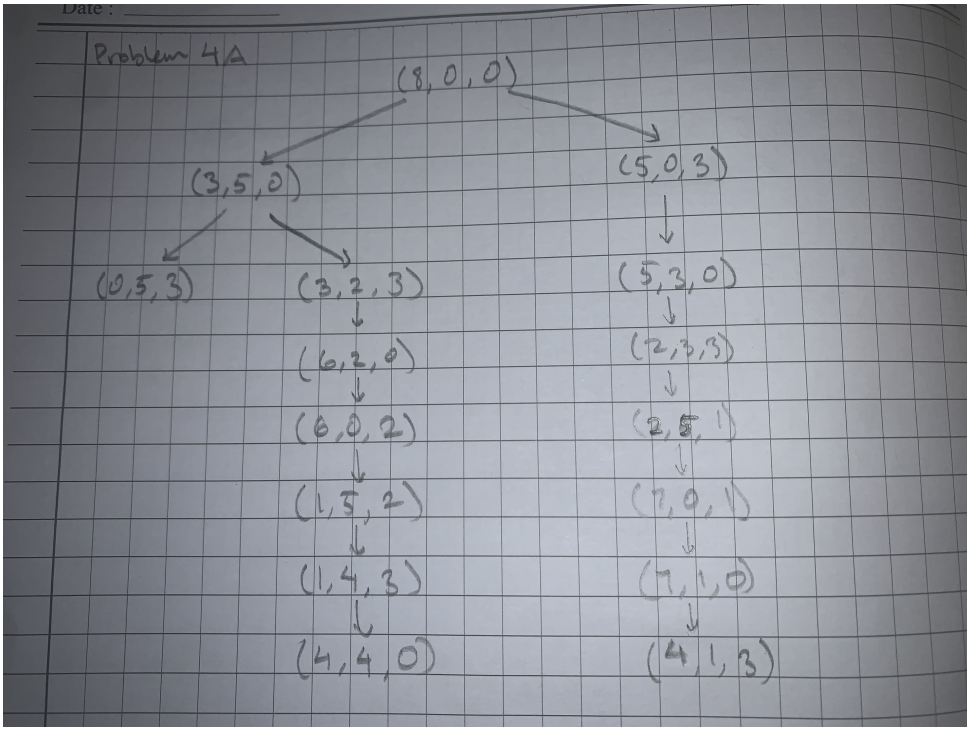
$$\cup\{(0,j)-(1,j) \mid 1 \le j \le n-1\} \qquad \text{H lines.}$$

(ii)

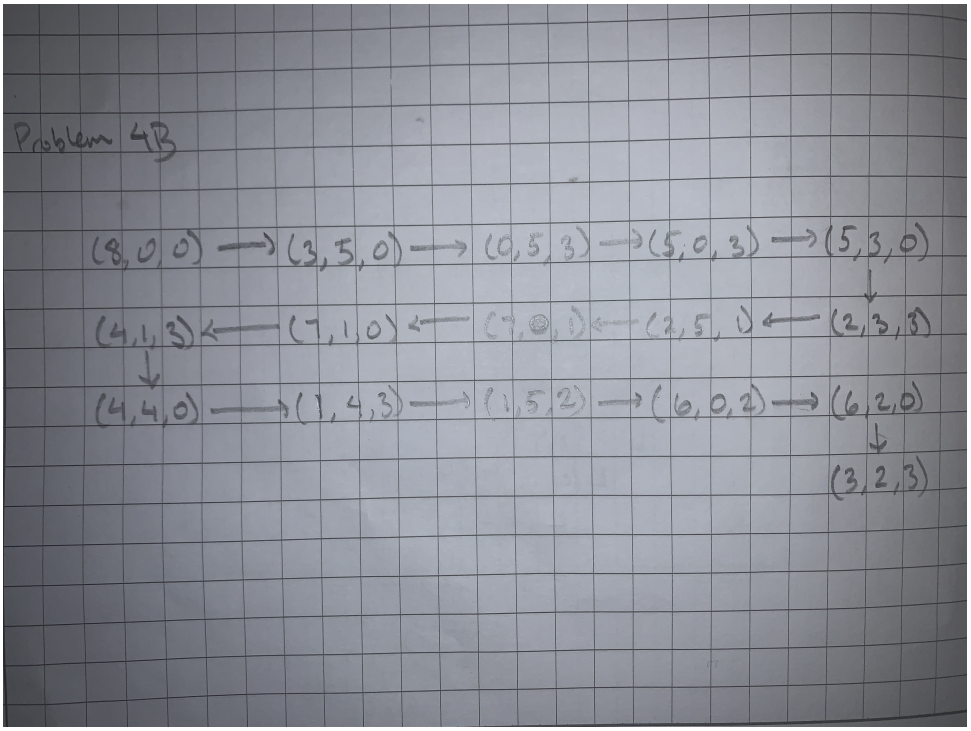


Problem 4

(A)


Problem 4A
(8,0,0)
(3,5,0)
(5,0,3)
(0,5,3)
(3,2,3)
(5,3,0)
(6,2,0)
(2,3,3)
(6,0,2)
(1,5,2)
(7,0,1)
(1,4,3)
(7,1,0)
(4,4,0)
(4,1,3)

(B)


Problem 4B
(8,0,0)
(3,5,0)
(0,5,3)
(5,0,3)
(5,3,0)
(2,3,3)
(4,1,3)
(7,1,0)
(4,4,0)
(1,4,3)
(1,5,2)
(6,0,2)
(6,2,0)
(3,2,3)

(C) Yes. BFS is a method to find a sequence of water pouring using the least number of pouring steps.

(D) Since the problem is modelled as an unweighted graph, BFS indiscriminately searches the graph in a radial search pattern (see (A)).

Further nodes are not visited until all closer nodes have been visited. BFS uses a queue to ensure that nodes are visited by distance.[1]

(E) No.

(F) No such sequence exists. This can be shown by performing BFS or DFS on the starting node (8, 0, 0) and exhausting all possible nodes (see (A) or (B)). No (6, 1, 1) node is connected to the graph of (8, 0, 0).

(G) For any $v \in V - \{v_0\}$, we can simply perform moves (iii) and (v) (i.e. moving water from b or c to a) to return to $v_0 = (8,0,0)$.

(H) A counterexample is sufficient to disprove the statement. v_0 is reachable from $(6,1,1)$ (use moves (iii) and (v)), but $(6,1,1)$ is not reachable from v_0.

[1] Distance meaning the length of the shortest path between two nodes.

Problem 5

(a)

i/j	1	2	3	4	5	6	7	8
1	5	25	35	50	85	110	160	175
2		15	25	40	70	95	145	160
3			5	15	35	55	105	115
4				5	20	40	85	95
5					10	30	70	80
6						10	40	50
7							20	30
8								5

i/j	1	2	3	4	5	6	7	8
1	1	2	2	2	2	2	5	5
2		2	2	2	2	5	5	5
3			3	3	4	5	5	7
4				4	5	5	6	7
5					5	5	6	7
6						6	7	7
7							7	7
8								8

(b) Cost: 175