Question 1

You are travelling with your friends along the Geraldton coast with cities $c_0, c_1, c_2, \ldots, c_n$ on the shore. You are starting in city c_0 where a famous spa is, and need to reach the airport situated in city c_n, so you will visit each city $c_0, c_1, c_2, \ldots, c_n$ in that order.

In each city you may swap the animal you are riding on and the choices are a giraffe, a mammoth, an ant, an iguana, and a lemur, denoted G, M, A, I, L respectively. However, each city has its own rules what kind of animal exchanges are allowed. For example, in some of the cities you can swap a giraffe only for a lemur or an ant (and you **cannot** remain on your giraffe), in others you can swap a mammoth only for a giraffe or decide to remain on your mammoth, and so on. You know all the rules of all the cities $c_1, \ldots, c_n$, expressed by a function $R(i, a, b)$ where $R(i, a, b) = 1$ if in city c_i one can swap animal a for animal b, and zero otherwise (a and b belong to the set $\{G, M, A, I, L\}$, and $1 \le i < n$). You also know the speed $v(a)$ in km/h ($a \in \{G, M, A, I, L\}$) of each of the five animals, as well as the distances $d(i)$ in km between cities c_{i-1} and c_i for all $i = 1, 2, \ldots, n$. Calculating a given value of R, v, or d can be done in $O(1)$ time.

You may begin your journey from c_0 on any of the five animals. You need to choose which animals to use for each of the n trips between cities, such that your travel time is minimised and every animal swap is valid.

1.1 [2 marks] Consider the case of $n = 2$, with $d(1) = 1$, $d(2) = 3$, and v defined as

$$v(G) = 7, \qquad v(M) = 3, \qquad v(A) = 2, \qquad v(I) = 1, \qquad v(L) = 4.$$

Define R so that

$$R(1, G, M) = 1, \quad R(1, A, M) = 1, \quad R(1, A, A) = 1, \quad R(1, I, G) = 1, \quad R(1, L, I) = 1,$$

and $R(1, a, b) = 0$ for all other values of a, b.

Determine which animals you should choose on each trip in order to minimise the total time taken to travel from c_0 to c_2. You *must* justify your selection.

1.2 [12 marks] Design an algorithm which determines the minimal amount of time (in hours) it will take to get from c_0 to c_n without making invalid swaps, as well as the animals required on each trip to achieve this time. Your algorithm must run in $O(n)$ time.

1.3 [6 marks] While planning your trip, your friend Mae points out that animals are living creatures, and do, in fact, need to rest. To take this into consideration, you estimate the effect of consecutive trips on the animals, and come up with a magical constant, $0 < \varepsilon < 1$. For each consecutive trip an animal makes, the speed of the animal is multiplied by this constant.

For example, if you decide to travel on a mammoth from c_0 to c_1 to c_2 to c_3 without ever changing animals, then the mammoth's speed will be $v(M)$ from c_0 to c_1, $\varepsilon v(M)$ from c_1 to c_2, and $\varepsilon^2 v(M)$ from c_2 to c_3. Of course, if you then swap animals, and later decide to travel by mammoth again, the new mammoths are not tired, and thus have speed $v(M)$. You may not "swap" an animal for "new" animals of the same type.

Design an algorithm which determines the minimal amount of time (in hours) it will take to get from c_0 to c_n without making invalid swaps. Your friends are judging you for not considering this in the first place, so your algorithm must run in $O(n^2)$ time before things become more awkward.

The fastest way to reach city i may involve taking a slower than optimal sequence of animals to city $i - 1$. However, if we consider only the routes that end in a sequence of the same animal a certain number of times, we can optimally solve a more restricted subproblem.

Question 2

You are given a string $w = w_1 w_2 \ldots w_n$ of n letters that come from a fixed alphabet. A *palindrome* is a substring w' such that w' can be read the same forwards and backwards. For example, $w' = kayak$ forms a palindrome while $w'' = kayaak$ does not form a palindrome. A *palindromic substring* is a contiguous subsequence of w that forms a palindrome.

2.1 [**8 marks**] We want to eventually find a minimum cost decomposition of w into palindromes. However, to do this, we need to first find *where* the palindromes are. Given a string $w = w_1 \ldots w_n$, describe an $O(n^2)$ algorithm to identify all of the palindromic substrings of w.

Hint. How many subproblems can you have at most?

- There are faster algorithms such as a modification to *Manacher's algorithm* but if you use the algorithm, you will have to state the entire algorithm, and prove its correctness and running time.

2.2 [**12 marks**] A *palindromic decomposition* of a string w is a partition of w into substrings $u_1, \ldots, u_m$ such that $w = u_1 \ldots u_m$ and each part u_i forms a palindrome. For example, if $w =$ `abcbabaab`, then $u_1 = \texttt{a}, u_2 = \texttt{bcb}, u_3 = \texttt{a}, u_4 = \texttt{baab}$ is a palindromic decomposition of w.

We assign each substring u_i a cost denoted by $\mathsf{cost}(|u_i|)$, where $|u_i|$ is the length of u_i. Finally, we define the cost of a decomposition $u_1, \ldots, u_m$ to be the sum of the costs of the individual substrings u_i; that is, if $w = u_1 \ldots u_m$, then the cost of the decomposition is

$$\mathsf{cost}(|u_1|) + \cdots + \mathsf{cost}(|u_m|).$$

We want to find the minimum cost of a *palindromic decomposition* of w. Given a string $w = w_1 \ldots w_n$ and an array of all costs, design an $O(n^2)$ algorithm to compute the minimum cost of a palindromic decomposition of w.

Hint. Perform a pre-processing step using **2.1**.

Question 3

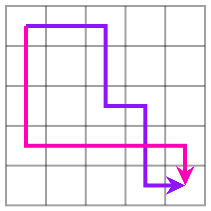
You are participating in a robotics competition, and need to program a robot to move through an m by n grid. The robot starts at cell $(1, 1)$, and must reach cell (m, n) to finish.

Your robot accepts two types of instructions:

- Start moving down at cell (i, j),
- Start moving right at cell (i, j).

An instruction is *not* required to tell the robot which way to move at the start, as there is only one possibility based on the first turning instruction.

For example, in this grid with $m = n = 5$, the purple path requires four instructions (down at $(1, 3)$, right at $(3, 3)$, down at $(3, 4)$, right at $(4, 4)$) while the pink path requires two (right at $(4, 1)$, down at $(4, 5)$).



3.1 **[6 marks]** Design an algorithm that runs in $O(mn)$ time and determines the number of distinct paths through the grid that the robot can traverse.

Hint. A closed form solution is possible, but we naturally encourage the use of Dynamic Programming which will assist with the rest of the question. A similar problem will also be discussed in the Week 8 Tutorial. Consider the 4 ways to reach a cell (i, j).

3.2 **[8 marks]** Unfortunately due to rising inflation your robot now only has a limited amount of memory, and can only store r instructions. Design an algorithm that runs in $O(mnr)$ time and determines the number of distinct paths through the grid that the robot can traverse using at most r instructions.

There are four ways to reach a cell (i, j) when you consider the second last and third last cells in the path. Two of these ways use an instruction, while the other two don't.

3.3 **[6 marks]** To make the competition more interesting, Blake has placed obstacles in some of the cells, which the robot is unable to pass through. They have given you an array $O[1 \ldots m][1 \ldots n]$, where $O[i][j]$ is True if cell (i, j) has an obstacle and False otherwise. Of course, Blake has made sure that a robot can travel from $(1, 1)$ to (m, n) by moving only right and down.

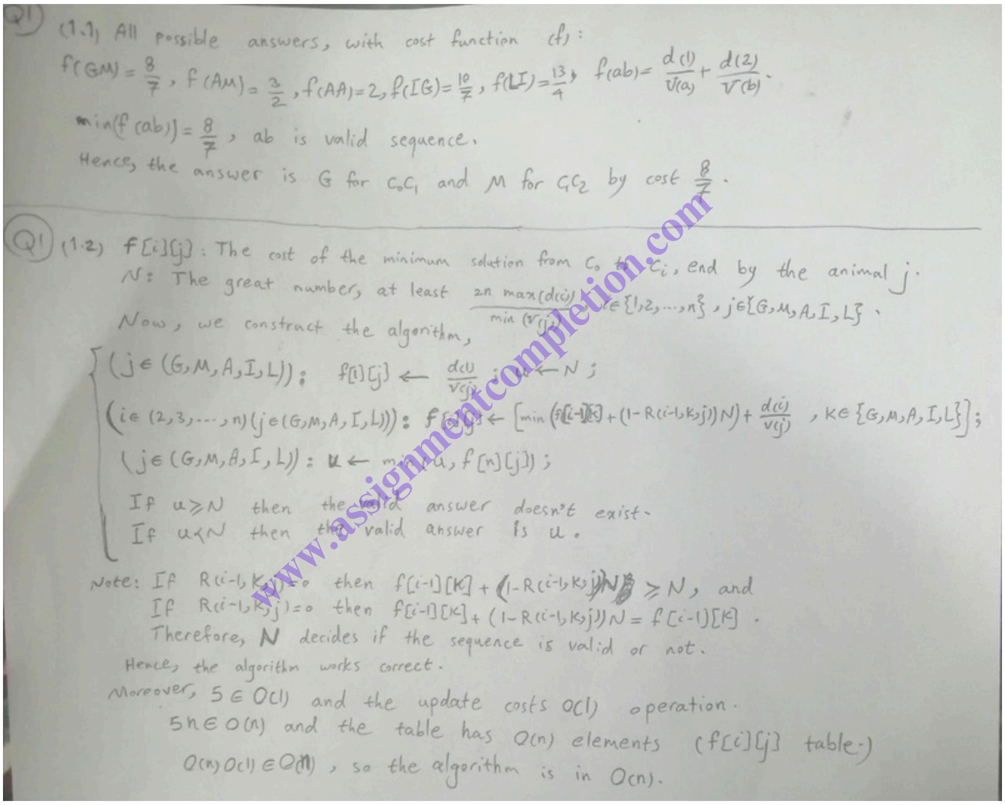
You wish to find the smallest amount of memory r that your robot needs to be able to traverse the grid.

Design an algorithm that runs in $O(mnr)$ time and finds the smallest number of instructions needed to traverse the grid.

Note that the parameter r in the time complexity is also the answer your algorithm is trying to find. This is similar to the Ford-Fulkerson algorithm for max flow, where the time complexity is dependent on the max flow for the given graph.

An algorithm that runs in $O(mn(m+n))$ will be eligible for at most 4 marks for this part.

Algorithms and Programming Techniques



Q1) (1.1) All possible answers, with cost function (f):

$f(GM) = \frac{8}{7}$, $f(AM) = \frac{3}{2}$, $f(AA) = 2$, $f(IG) = \frac{10}{7}$, $f(LI) = \frac{13}{4}$, $f(ab) = \frac{d(1)}{V(a)} + \frac{d(2)}{V(b)}$.

$\min(f(ab)) = \frac{8}{7}$, ab is valid sequence.

Hence, the answer is G for C_0C_1 and M for C_1C_2 by cost $\frac{8}{7}$.

Q1) (1.2) $f[i][j]$: The cost of the minimum solution from C_0 to C_i, end by the animal j.

N: The great number, at least $\frac{2n\max(d(i))}{\min(V(j))}$, $i \in \{1,2,\dots,n\}$, $j \in \{G,M,A,I,L\}$.

Now, we construct the algorithm,

```
{ (j ∈ (G,M,A,I,L)): $f[1][j] \leftarrow \frac{d(1)}{V(j)}$ ; $u \leftarrow N$ ;
  (i ∈ (2,3,...,n))(j ∈ (G,M,A,I,L)): $f[i][j] \leftarrow \left[\min\left(f[i-1][k] + (1-R(i-1,k,j))N\right) + \frac{d(i)}{V(j)}, k \in \{G,M,A,I,L\}\right]$;
  (j ∈ (G,M,A,I,L)): $u \leftarrow \min(u, f[n][j])$ ;
    If $u \geq N$ then the valid answer doesn't exist.
    If $u < N$ then the valid answer is $u$.
```

Note: If $R(i-1,k,j) = 0$ then $f[i-1][k] + (1-R(i-1,k,j))N \geq N$, and

If $R(i-1,k,j) = 0$ then $f[i-1][k] + (1-R(i-1,k,j))N = f[i-1][k]$.

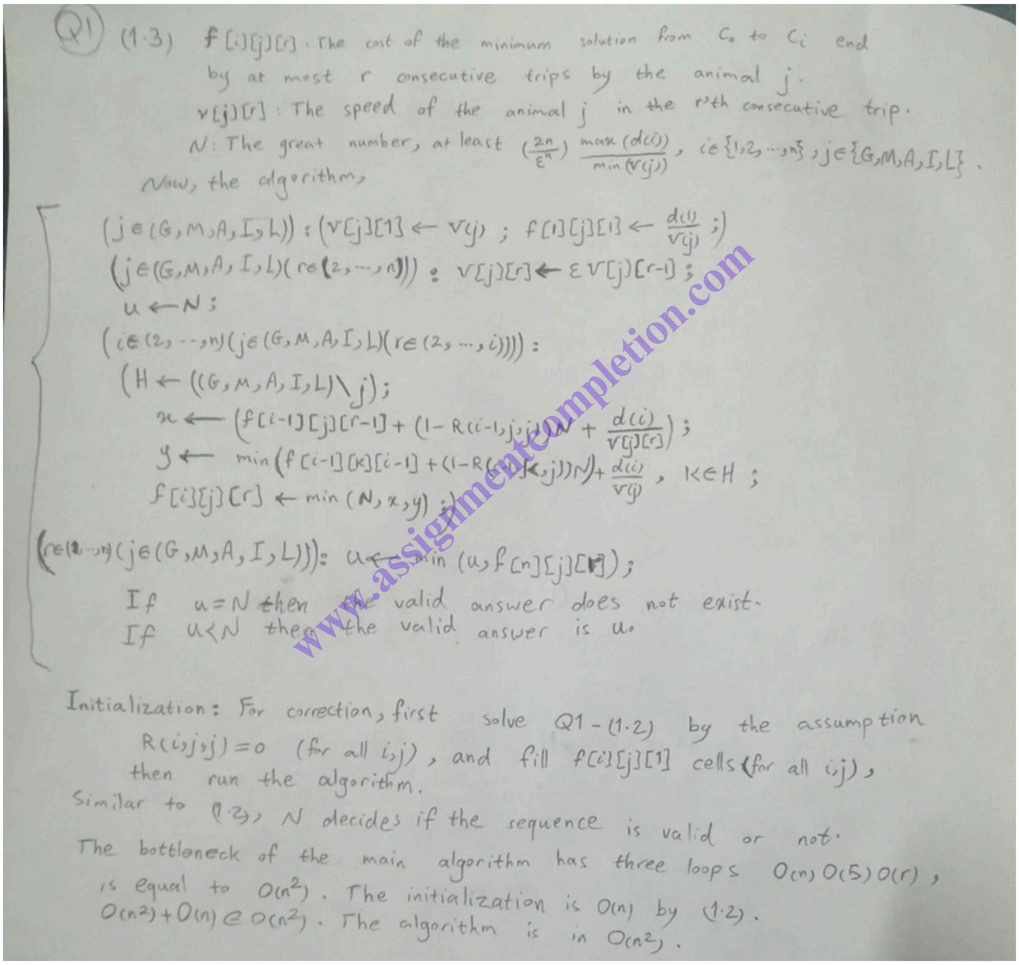
Therefore, N decides if the sequence is valid or not.

Hence, the algorithm works correct.

Moreover, $5 \in O(1)$ and the update costs $O(1)$ operation.

$5n \in O(n)$ and the table has $O(n)$ elements ($f[i][j]$ table.)

$O(n)O(1) \in O(n)$, so the algorithm is in $O(n)$.



Q1 (1.3) $f[i][j][r]$: The cost of the minimum solution from C_0 to C_i end by at most r consecutive trips by the animal j.

$v[j][r]$: The speed of the animal j in the r'th consecutive trip.

N: The great number, at least $(\frac{2n}{\varepsilon^n}) \frac{\max(d(i))}{\min(v(j))}$, $i \in \{1,2,\cdots,n\}$, $j \in \{G,M,A,I,L\}$.

Now, the algorithm,

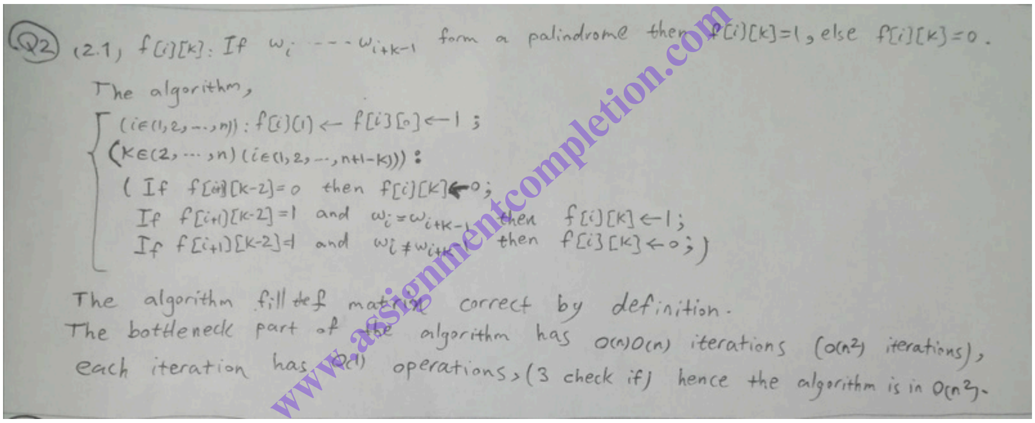
```
(j ∈ (G,M,A,I,L)) : ($v[j][1] \leftarrow v(j)$ ; $f[1][j][1] \leftarrow \frac{d(1)}{v(j)}$ ;)
(j ∈ (G,M,A,I,L)(r ∈ (2,...,n))) : $v[j][r] \leftarrow \varepsilon v[j][r-1]$ ;
u ← N ;
(i ∈ (2,...,n)(j ∈ (G,M,A,I,L)(r ∈ (2,...,i)))) :
  (H ← ((G,M,A,I,L)\j) ;
     $x \leftarrow (f[i-1][j][r-1] + (1-R(i-1,j,j))N + \frac{d(i)}{v[j][r]})$ ;
     $y \leftarrow \min(f[i-1][k][i-1] + (1-R(i-1,k,j))N) + \frac{d(i)}{v(j)}$ , $k \in H$ ;
     $f[i][j][r] \leftarrow \min(N, x, y)$ ;)
(r ∈ (1,...,n)(j ∈ (G,M,A,I,L))) : $u \leftarrow \min(u, f[n][j][r])$ ;
   If u = N then the valid answer does not exist.
   If u < N then the valid answer is u.
```

Initialization: For correction, first solve Q1 - (1.2) by the assumption $R(i,j,j) = 0$ (for all i,j), and fill $f[i][j][1]$ cells (for all i,j), then run the algorithm.

Similar to (1.2), N decides if the sequence is valid or not.

The bottleneck of the main algorithm has three loops $O(n)\,O(5)\,O(r)$, is equal to $O(n^2)$. The initialization is $O(n)$ by (1.2).

$O(n^2) + O(n) \in O(n^2)$. The algorithm is in $O(n^2)$.

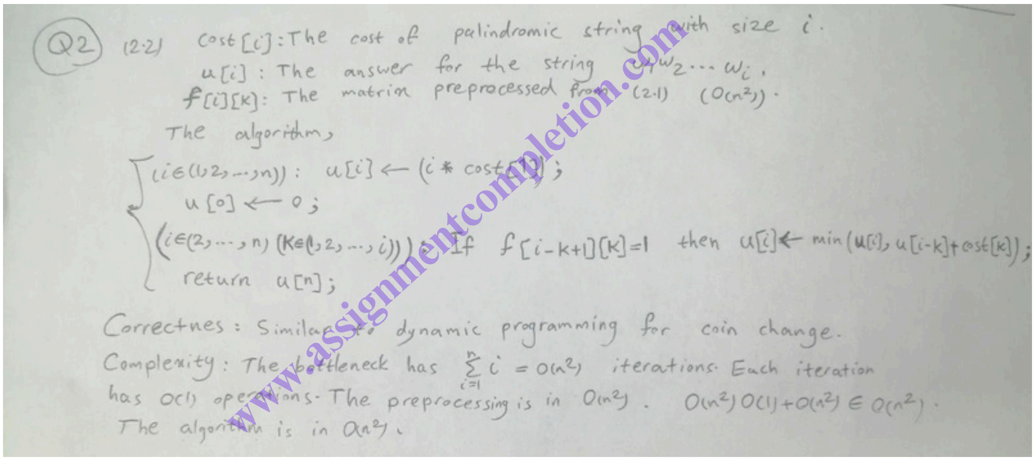


Q2 (2.1) $f[i][k]$: If $w_i \cdots w_{i+k-1}$ form a palindrome then $f[i][k]=1$, else $f[i][k]=0$.

The algorithm,

```
(i ∈ (1,2,...,n)) : $f[i][1] \leftarrow f[i][0] \leftarrow 1$ ;
(k ∈ (2,...,n) (i ∈ (1,2,...,n+1-k))) :
    ( If $f[i+1][k-2]=0$ then $f[i][k] \leftarrow 0$;
      If $f[i+1][k-2]=1$ and $w_i = w_{i+k-1}$ then $f[i][k] \leftarrow 1$;
      If $f[i+1][k-2]=1$ and $w_i \neq w_{i+k-1}$ then $f[i][k] \leftarrow 0$; )
```

The algorithm fill ~~the~~ f matrix correct by definition.
The bottleneck part of the algorithm has $O(n)O(n)$ iterations ($O(n^2)$ iterations), each iteration has $O(1)$ operations, (3 check if) hence the algorithm is in $O(n^2)$.

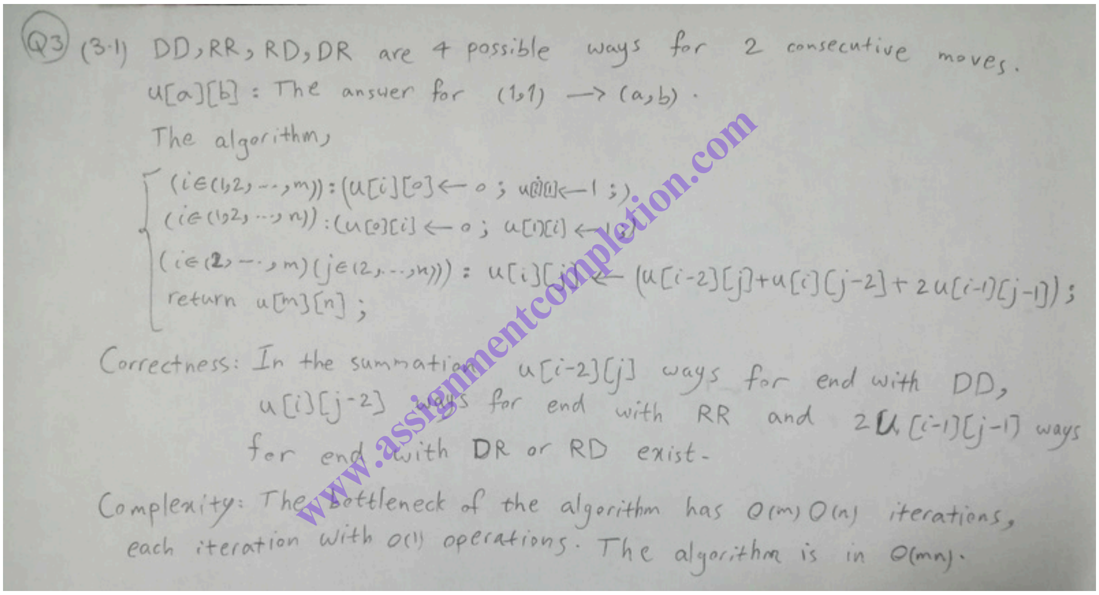


Q2 (2.2) $cost[i]$: The cost of palindromic string with size i.
$u[i]$: The answer for the string $w_1 w_2 \cdots w_i$.
$f[i][k]$: The matrix preprocessed from (2.1) ($O(n^2)$).

The algorithm,

```
(i ∈ (1,2,...,n)) : $u[i] \leftarrow (i * cost[1])$;
    $u[0] \leftarrow 0$;
(i ∈ (2,...,n) (k ∈ (1,2,...,i))) : If $f[i-k+1][k]=1$ then $u[i] \leftarrow \min(u[i], u[i-k]+cost[k])$;
    return $u[n]$;
```

Correctnes : Similar to dynamic programming for coin change.
Complexity : The bottleneck has $\sum_{i=1}^{n} i = O(n^2)$ iterations. Each iteration has $O(1)$ operations. The preprocessing is in $O(n^2)$. $O(n^2)O(1)+O(n^2) \in O(n^2)$.
The algorithm is in $O(n^2)$.



Q3 (3.1) DD, RR, RD, DR are 4 possible ways for 2 consecutive moves.

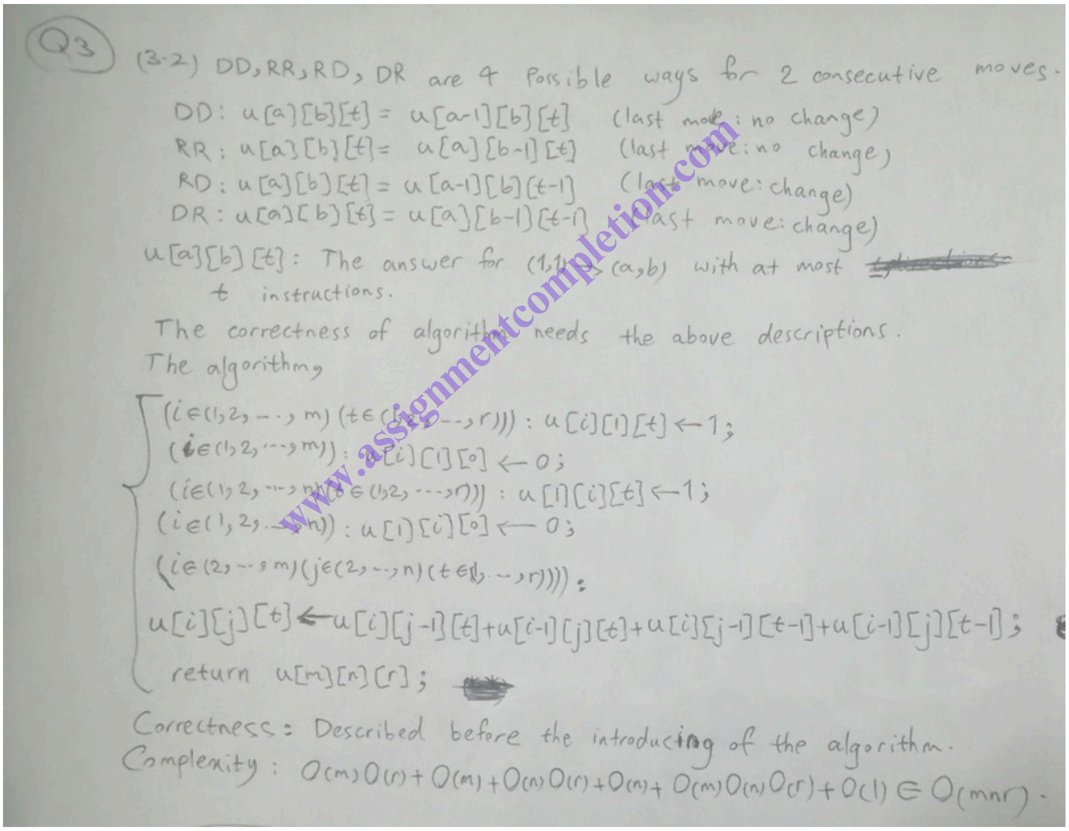
u[a][b] : The answer for $(1,1) \longrightarrow (a,b)$.

The algorithm,

```
(i∈(1,2,...,m)) : (u[i][0] ← 0 ; u[i][1] ← 1 ;)
(i∈(1,2,...,n)) : (u[0][i] ← 0 ; u[1][i] ← 1 ;)
(i∈(2,...,m)(j∈(2,...,n))) : u[i][j] ← (u[i-2][j]+u[i][j-2]+2u[i-1][j-1]) ;
return u[m][n] ;
```

Correctness: In the summation u[i-2][j] ways for end with DD, u[i][j-2] ways for end with RR and 2u[i-1][j-1] ways for end with DR or RD exist.

Complexity: The bottleneck of the algorithm has $O(m)\,O(n)$ iterations, each iteration with $O(1)$ operations. The algorithm is in $O(mn)$.



Q3

(3.2) DD, RR, RD, DR are 4 Possible ways for 2 consecutive moves.

DD: $u[a][b][t] = u[a-1][b][t]$ (last move: no change)

RR: $u[a][b][t] = u[a][b-1][t]$ (last move: no change)

RD: $u[a][b][t] = u[a-1][b][t-1]$ (last move: change)

DR: $u[a][b][t] = u[a][b-1][t-1]$ (last move: change)

$u[a][b][t]$: The answer for $(1,1) \to (a,b)$ with at most [illegible] t instructions.

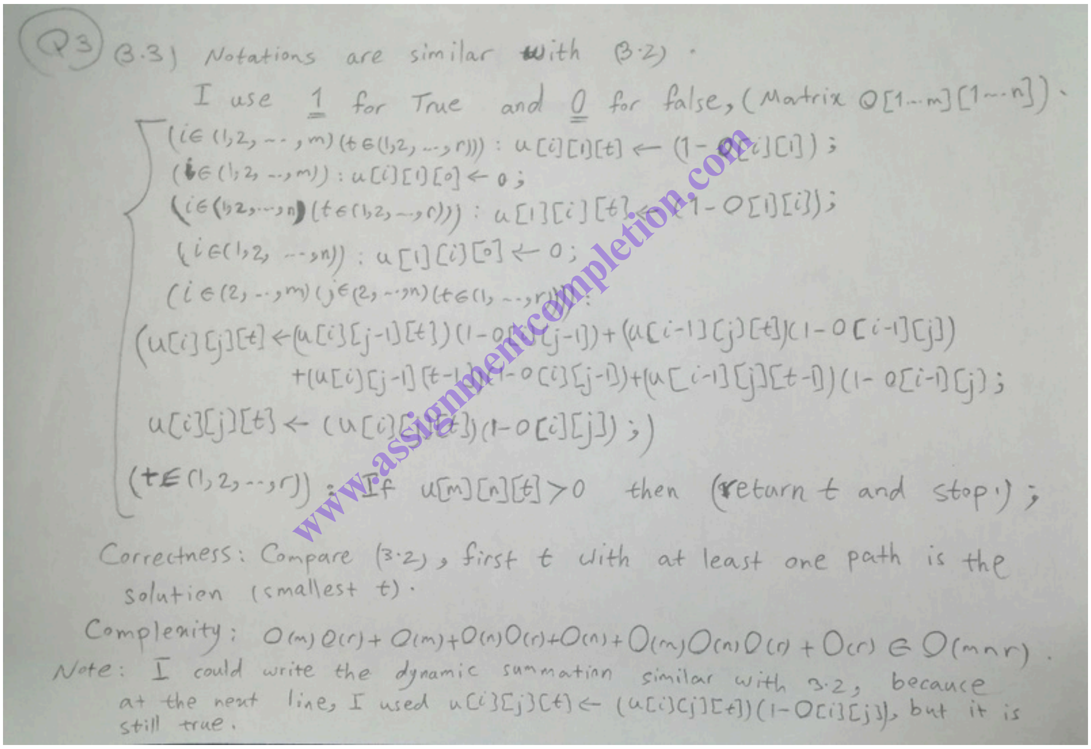
The correctness of algorithm needs the above descriptions.

The algorithm,

```
$(i \in (1,2,\dots,m)(t \in (1,2,\dots,r)))$ : $u[i][1][t] \leftarrow 1$;
$(i \in (1,2,\dots,m))$ : $u[i][1][0] \leftarrow 0$;
$(i \in (1,2,\dots,n)(t \in (1,2,\dots,r)))$ : $u[1][i][t] \leftarrow 1$;
$(i \in (1,2,\dots,n))$ : $u[1][i][0] \leftarrow 0$;
$(i \in (2,\dots,m)(j \in (2,\dots,n)(t \in (1,\dots,r))))$ :
$u[i][j][t] \leftarrow u[i][j-1][t] + u[i-1][j][t] + u[i][j-1][t-1] + u[i-1][j][t-1]$;
  return $u[m][n][r]$;
```

Correctness: Described before the introducting of the algorithm.

Complexity: $O(m)O(r) + O(m) + O(n)O(r) + O(n) + O(m)O(n)O(r) + O(1) \in O(mnr)$.



Q3 (3.3) Notations are similar with (3.2).

I use $\underline{1}$ for True and $\underline{0}$ for false, (Matrix $O[1 \ldots m][1 \ldots n]$).

```
$(i \in (1, 2, \ldots, m)(t \in (1, 2, \ldots, r)))$ : $u[i][1][t] \leftarrow (1 - O[i][1])$;
$(i \in (1, 2, \ldots, m))$ : $u[i][1][0] \leftarrow 0$;
$(i \in (1, 2, \ldots, n)(t \in (1, 2, \ldots, r)))$ : $u[1][i][t] \leftarrow (1 - O[1][i])$;
$(i \in (1, 2, \ldots, n))$ : $u[1][i][0] \leftarrow 0$;
$(i \in (2, \ldots, m)(j \in (2, \ldots, n)(t \in (1, \ldots, r))))$ :
$(u[i][j][t] \leftarrow (u[i][j-1][t])(1 - O[i][j-1]) + (u[i-1][j][t])(1 - O[i-1][j])$
        $+ (u[i][j-1][t-1])(1 - O[i][j-1]) + (u[i-1][j][t-1])(1 - O[i-1][j])$;
$u[i][j][t] \leftarrow (u[i][j][t])(1 - O[i][j])$;)
$(t \in (1, 2, \ldots, r))$ : If $u[m][n][t] > 0$ then (return $t$ and stop!);
```

Correctness: Compare (3.2), first t with at least one path is the solution (smallest t).

Complexity: $O(m)O(r) + O(m) + O(n)O(r) + O(n) + O(m)O(n)O(r) + O(r) \in O(mnr)$.

Note: I could write the dynamic summation similar with 3.2, because at the next line, I used $u[i][j][t] \leftarrow (u[i][j][t])(1 - O[i][j])$, but it is still true.