HW1 ASSIGNMENT

DHAYALINI NAGARAJ
A20359686

1. A pirate has hidden gold on all of the integer points in the positive orthant of the plane $g_{i,j}$ ounces of gold at point (i,j). You must start at point $(1,1)$ and moving only either up or right one unit, collect as much gold as possible on your way to point (n,n)

Let the function be $G(i,j)$

Recurrence Relation:

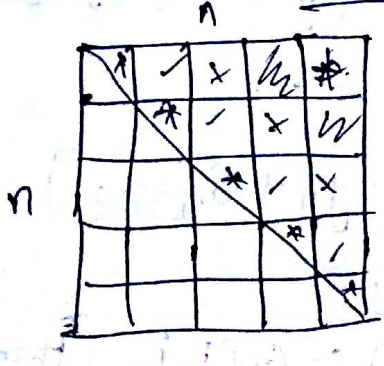
The principle of Optimality tells us

$$G[i,j] = \begin{cases} g_{1,1} & \text{if } i=j=1, \\ g_{1,j} + G[1,j-1] & \text{if } i=1, j>1, \\ g_{i,1} + G[i-1,1] & \text{if } i>1, j=1, \\ g_{i,j} + \max(G[i-1,j], G[i,j-1]) & \text{otherwise.} \end{cases}$$

MEMOIZE:

~~Memoize~~ Iterative code: MEMOIZED ITERATIVECODE

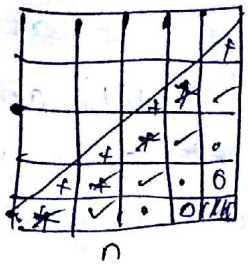
Eg:



The array should be filled in either row by row or column by column in constant time per entry.

To fill G[1,j] we need to fill the squares above below and left.

(ie)



G(n, n)

Thus the following algorithm takes time $\Theta(n^2)$:

```
for j=1 to n do
  for i=1 to n do
    if i=j=1 then
       G[i,j] ← g_ij
    else if i=1, j>1 then
       G[i,j] ← g_ij + G[1,j-1]
    else if i>1, j=1 then
       G[i,j] ← g_ij + G[i-1,1]
    else // i>1, j>1
       if G[i-1,j] > G[i,j-1] then
          G[i,j] ← g_ij + G[i-1,j]
       else
          G[i,j] ← g_ij + G[i,j-1]
       end if
    end if
  end for
end for
```

MEMOIZE MAXIMUM GOLD

In order to collect as much gold as possible on the way to point (n, n') we need to keep track of the cells.

So track the cell from which we reach the cell i,j, the one below or the one to the left.

Let's take another array F[i,j] in which each entry can be noted down.

Lets denote '↓' for below, '←' for left and 's' for start.

```
for j = 1 to n do
    for i = 1 to n do
        if i = j = 1 then
            G[i,j] ← gij
// Let's take the array and start to fill the array.
            F[i,j] ← "s"
        else if i = 1, j > 1 then
            G[i,j] ← gij + G[1, j-1]
            F[i,j] ← "↓"   // (since it moves up the below cell is tracked)
        else if i > 1, j = 1 then
            G[i,j] ← gi1 + G[i-1, 1]
// Now it is moved right so consider left cell.
            F[i,j] ← "←"
```

```
else
        // when i > 1, j > 1
    if G[i-1, j] > G[i, j-1] then
        G[i, j] ← g_ij + G[i-1, j]
        F[i, j] ← "←"
    else
        G[i, j] ← g_ij + G[i, j-1]
        F[i, j] ← "↓"
    end if
  end if
 end for
end for.
```

Unmemoized Algorithm : (Unmemoized Recursive)

According to the problem, the time taken to for a recursive algorithm will be

(i) Time taken to solve $G(i-1, j)$ and $G(i, j-1)$

(ii) $\text{Time}(G(i-1, j)) + \text{Time}(G(i, j-1))$

ii) If $i = j = 1$, then the time taken is

1

Let time be Time(i,j) and function to fill the squares be G(i,j)

then

$$Time(i,j) = \begin{cases} 1 & \text{if } i=j=1 \\ Time(i-1,j) + Time(i,j-1) & \text{otherwise} \end{cases}$$

The recurrence for Time(i,j) is recurrence for binomial co.efficient.

Total time to fill G(n,n) is central binomial co.efficient $\binom{2n-2}{n-1}$

Time: ∴ By starling Formula $4^n/\sqrt{n}$ where $n \to \infty$.

Algorithm:

```
function G[i,j]
for j=1 to n do
  for i=1 to n do
    if i=j=1 then
      return 1
    else if
      return G[i-1,j] + G[i,j-1]
    end if
  end for
end for.
```

In this algorithm the calculated values are not saved and it is calculated recursively each time.

2) - Given a string of letters, you must split it into as few strings as possible such that each string is its own reversal.

For example.

MADAMIMADAM

Recurrence Relation.

Let the function for spliting a string and it should its own reversal be Palinsplit[i, j]

Then the Principle of Optimality is

$$\text{Palinsplit}[i,j] = \begin{cases} 0 & \text{if } i=j=1, \\ 0 & \text{if str}[i \ldots j] \text{ is a palindrom} \\ & \text{of} \\ \text{Min}(\text{Palinsplit}(str,i,k)+1 + & \\ \quad \text{Palinsplit}(str,k+1,j)\} & \text{otherwise} \end{cases}$$

where K varies from i to j-1

Memoized Iterative code.

If the string is a palindrome, then we simply return 0.

Lets try to split the string at all possible places and calculate the cost of each and return minimum value..

/

Lets take two arrays to store substrings and to save the previous values

~~function Palinsplit[i,j]~~
// Let n be length of the string
~~n ← stringlength(str)~~
// Let mincut[i] be minimal cut and Palin[i,j] be true if the string is palindrome.

```
function Palinsplit[i,j]
    n ← stringlength(str)
for i=0 to n do
    Palin[i,j] ← true
end for
for l=2 to n do      // Let l be the substring length
    for i=0 to n-l+1 do
        j ← i+l-1
    if l=2
        Palin[i][j] ← (str[i]==str[j])
    else
        Palin[i][j] ← (str[i]==str[j])
                      && Palin[i+1][j-1]
    end if
    end for
end for
```

```
for i=0 to n do
    if Palin[0][i] = true
        mincut[i] ← 0
    else
        mincut[i] ← INT_MAX
        for j=0 to i do
            if Palin[j+1][i] = true and
                1+mincut[j] < mincut[i]
                    mincut[i] = 1+mincut[j]
            end if
        end for
    end if
end for
return Palinsplit[i,j] ← mincut[n-1]
```

Time complexity: $O(n^2)$

This algorithm is done by calculating all palindrom substring from given string. Once all palindrom substring is found, the minimal cut is calculated.

The palindrom substring are saved in Palin array and value of minimal cut is also saved in mincut array.

Unmemoized Recursive Code:

The maximum number of splits that can be done for a given string is length of the string. Each character itself is a palindrome.

```
function Palinsplit [string s]
    l = string length of s
    if s = "" or palindrome(s)
      return 0
    else
      min cut = INT_MAX
      for i=1 to l do
        mincut= min (1 + Palinsplit ( s.substring(0,i))
                    +palinsplit (s.substring(i, l)) )
      end for
      return min cut
    end if

function Palindrome [string s]
    l = string length of s
    for i=0 to n/2+1
      if string char(i) not equal to
                  stringchar (n-i-1))
          return false
      end if
    end for
    return true
```

In this algorithm the given string is split and then it is checked whether it is a palindrome or not. So it repeated splits the string and checks for palindrome. Recursion will stop only if it finds a substring that is palindrome.

Time Complexity:

If there are k characters in a string then k-1 cuts are possible.

eg: MADAM - minimum 4 cuts.

∴ Time complexity is $2^{(k-1)}$

Ans: Time Complexity $2^{(k-1)}$