CMSC 641 Design & Analysis of Algorithms Spring 2023
Section 01 Homework 3

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

1 0-1 Knapsack Table

Fill in the following dynamic programming table for the 0-1 Knapsack Problem. Use the subproblem definition and recursion show during lecture:

OPT_KS(i, C) = the highest possible total value obtained by choosing items from item #i through item #n such that the weight of the chosen items does not exceed C.

$OPT\_KS(i, C) = \max\{ OPT\_KS(i + 1, C - w_i) + v_i , OPT\_KS(i + 1, C) \}$

C =Capacity

	w_i	v_i
1	1	2
2	4	6
3	3	5
4	1	3
5	2	4

item #i

	0	1	2	3	4	5	6	7	8
1									
2									
3									
4									
5									

Base Cases:

if C <0 , $OPT\_KS(i, C) = -\infty$

if C =0 , OPT_KS(i, C) = 0

if i=n+1 , OPT_KS(i, C) = 0

i=1 to 5 and C=8

$OPT\_KS(i, C) = \max\{ OPT\_KS(i+1, C-w_i) + v_i , OPT\_KS(i+1, C) \}$

$OPT\_KS(1, 8) = \max\{ OPT\_KS(2, 8-1) + 2 , OPT\_KS(2, 8) \} = \max\{13+2,14\}=15$

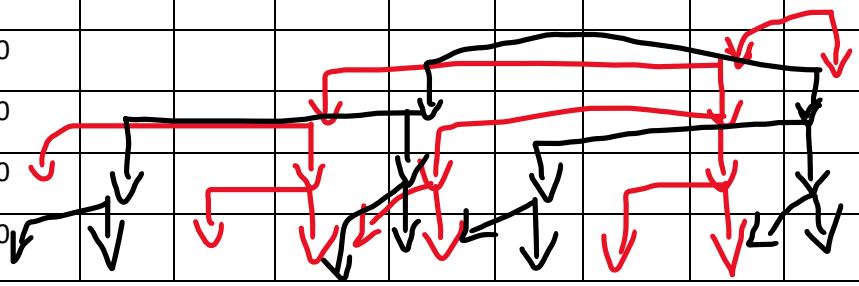
$OPT\_KS(2, 7) = \max\{ OPT\_KS(3, 7-4) + 6 , OPT\_KS(3, 7) \} = \max\{7+6,12\}=13$

$OPT\_KS(3, 3) = \max\{ OPT\_KS(4, 3-3) + 5 , OPT\_KS(4, 3) \} = \max\{0+5,7\}=7$

OPT_KS(4, 0) = 0

$OPT\_KS(4, 3) = \max\{ OPT\_KS(5, 3-1) + 3 , OPT\_KS(5, 3) \} = \max\{4+3,4\}=7$

	0	1	2	3	4	5	6	7	8
1	0								
2	0								
3	0								
4	0								
5	0								



OPT_KS(5, 2) = max{ OPT_KS(6, 2-2) + 4 , OPT_KS(6, 2) } = max{0+4,0}=4

OPT_KS(6, 0) = 0

OPT_KS(6, 2) = 0

OPT_KS(5, 3) = max{ OPT_KS(6, 3-2) + 4 , OPT_KS(6, 3) } = max{0+4,0}=4

OPT_KS(6, 1) = 0

OPT_KS(6, 3) = 0

OPT_KS(3, 7) = max{ OPT_KS(4, 7-3) + 5 , OPT_KS(4, 7) } = max{7+5,7}=12

OPT_KS(4, 4) = max{ OPT_KS(5, 4-1) + 3 , OPT_KS(5, 4) } = max{4+3,4}=7

OPT_KS(5, 4) = max{ OPT_KS(6, 4-2) + 4 , OPT_KS(6, 4) } = max{0+4,0}=4

OPT_KS(6, 2) = 0

OPT_KS(6, 4) = 0

OPT_KS(4, 7) = max{ OPT_KS(5, 7-1) + 3 , OPT_KS(5, 7) } = max{4+3,4}=7

OPT_KS(5, 6) = max{ OPT_KS(6, 6-2) + 4 , OPT_KS(6, 6) } = max{0+4,0}=4

OPT_KS(6, 6) = 0

OPT_KS(5, 7) = max{ OPT_KS(6, 7-2) + 4 , OPT_KS(6, 7) } = max{0+4,0}=4

OPT_KS(6, 5) = 0

OPT_KS(6, 7) = 0

OPT_KS(2, 8) = max{ OPT_KS(3, 8-4) + 6 , OPT_KS(3, 8) } = max{8+6,12}=14

OPT_KS(3, 4) = max{ OPT_KS(4, 4-3) + 5 , OPT_KS(4, 4) } = max{3+5,7}=8

OPT_KS(4, 1) = max{ OPT_KS(5, 1-1) + 3 , OPT_KS(5, 1) } = max{0+3,0}=3

OPT_KS(5, 0) = 0

OPT_KS(5, 1) = max{ OPT_KS(6, 1-2) + 4 , OPT_KS(6, 1) } = max{-∞ +4,0}=0

OPT_KS(6, -1) = -∞

OPT_KS(6, 1) = 0

OPT_KS(3, 8) = max{ OPT_KS(4, 8-3) + 5 , OPT_KS(4, 8) } = max{7+5,7}=12

OPT_KS(4, 5) = max{ OPT_KS(5, 5-1) + 3 , OPT_KS(5, 5) } = max{4+3,4}=7

OPT_KS(5, 5) = max{ OPT_KS(6, 5-2) + 4 , OPT_KS(6, 5) } = max{0+4,0}=4

OPT_KS(4, 8) = max{ OPT_KS(5, 8-1) + 3 , OPT_KS(5, 8) } = max{4+3,4}=7

OPT_KS(5, 8) = max{ OPT_KS(6, 8-2) + 4 , OPT_KS(6, 8) } = max{0+4,0}=4

	0	1	2	3	4	5	6	7	8
1	0								15 +2
2	0							13 +6	14
3	0			7	8			12	12
4	0	3		7 +3	7	7		7	7
5	0	0	4	4 +4	4	4	4	4	4

Filling the remaining data,

OPT_KS(1, 7) = max{ OPT_KS(2, 7-1) + 2 , OPT_KS(2, 7) } = max{12+2,12}=14

OPT_KS(4, 6) = max{ OPT_KS(5, 6-1) + 3 , OPT_KS(5, 6) } = max{4+3,4}=7

OPT_KS(3, 1) = max{ OPT_KS(4, 1-3) + 5 , OPT_KS(4, 1) } = max{-∞+5,3}=3

OPT_KS(4, -2) = -∞

OPT_KS(3, 2) = max{ OPT_KS(4, 2-3) + 5 , OPT_KS(4, 2) } = max{-∞+3,4}=4

OPT_KS(4, -1) = -∞

OPT_KS(4, 2) = max{ OPT_KS(5, 2-1) + 3 , OPT_KS(5, 2) } = max{0+3,4}=4

OPT_KS(3, 5) = max{ OPT_KS(4, 5-3) + 5 , OPT_KS(4, 5) } = max{4+5,7}=9

OPT_KS(3, 6) = max{ OPT_KS(4, 6-3) + 5 , OPT_KS(4, 6) } = max{7+5,7}=12

OPT_KS(4, -3) = -∞

OPT_KS(2, 1) = max{ OPT_KS(3, 1-4) + 6 , OPT_KS(3, 1) } = max{-∞+6,3}=3

OPT_KS(3, -3) = -∞

OPT_KS(2, 2) = max{ OPT_KS(3, 2-4) + 6 , OPT_KS(3, 2) } = max{-∞+6,4}=4

OPT_KS(3, -2) = -∞

OPT_KS(2, 3) = max{ OPT_KS(3, 3-4) + 6 , OPT_KS(3, 3) } = max{-∞+6,7}=7

OPT_KS(3, -1) = -∞

OPT_KS(2, 4) = max{ OPT_KS(3, 4-4) + 6 , OPT_KS(3, 4) } = max{0+6,8}=8

OPT_KS(3, 0) = 0

OPT_KS(2, 5) = max{ OPT_KS(3, 5-4) + 6 , OPT_KS(3, 5) } = max{3+6,9}=9

OPT_KS(2, 6) = max{ OPT_KS(3, 6-4) + 6 , OPT_KS(3, 6) } = max{4+6,12}=12

OPT_KS(1, 1) = max{ OPT_KS(2, 1-1) + 2 , OPT_KS(2, 1) } = max{0+2,3}=3

OPT_KS(2, 0) = 0

OPT_KS(1, 2) = max{ OPT_KS(2, 2-1) + 2 , OPT_KS(2, 2) } = max{3+2,3}=5

OPT_KS(1, 3) = max{ OPT_KS(2, 3-1) + 2 , OPT_KS(2, 3) } = max{4+2,7}=7

OPT_KS(1, 4) = max{ OPT_KS(2, 4-1) + 2 , OPT_KS(2, 4) } = max{7+2,8}=9

OPT_KS(1, 5) = max{ OPT_KS(2, 5-1) + 2 , OPT_KS(2, 5) } = max{8+2,9}=10

OPT_KS(1, 6) = max{ OPT_KS(2, 6-1) + 2 , OPT_KS(2, 6) } = max{9+2,12}=12

OPT_KS(1, 7) = max{ OPT_KS(2, 7-1) + 2 , OPT_KS(2, 7) } = max{12+2,13}=14

	0	1	2	3	4	5	6	7	8
1	0	3	5	7	9	10	12	14	15 +2
2	0	3	4	7	8	9	12	13 +6	14
3	0	3	4	7	8	9	12	12	12
4	0	3	4	7 +3	7	7	7	7	7
5	0	0	4	4 +4	4	4	4	4	4

2 History Final

You are taking a bring-your-own-notes final exam for a history class where the professor has allowed you to bring r index cards, each of which holds k lines of text. You have prepared n short summaries of events you have studied in this class. For example,

July 11, 1804: Weehawken, dawn. Guns drawn. A.Ham + A.Burr. (Everything was legal in New Jersey.)

Some of your summaries are longer than others. Let l_i denote the number of lines taken up by the i-th summary when written on an index card. Unfortunately you cannot fit all n summaries on the r index cards, so you have to pick and choose. There are some constraints:

• Each index card must have k or fewer lines of text.

• Every summary written on an index card must fit entirely on that index card. (I.e., summaries cannot start on one card and end on another card.)

• The summaries must be in chronological order. The summaries on each index card are sorted by date, and all of the summaries on card i predate the summaries on card i + 1.

Your objective is to include as many summaries as possible on the r index cards allotted to you.

Questions:

1. Define a function, MaxSumm, that can be used to solve this dynamic programming problem. To do this, you must describe the input parameters to MaxSumm and its "return value" using English sentences. (Note: you are specifying the input-output relation of the function. You should not describe how to compute the function.)

A. MaxSumm(i,r,k)= Maximum number of summaries included on r cards where we have n summaries and i goes from 1 to n, and each r card holds k lines of text

   i=1 to n
   n summaries, l_i denotes lines for i^{th} summary
   r cards which holds k lines of text
   k is used to keep track of lines left for the card

   The return value of the function MaxSumm(i,r,k) returns maximum number of summaries on r cards with the given constraints of the problem i.e

   each card must have <= k lines text
   summary should not start at one card and end at another
   Summaries must be in chronological order

2. Give a mathematical formula that shows how MaxSumm can be computed recursively. Then, explain all the major parts of the formula using English sentences. Remember to include the base cases.

A. if(l_i <= k)

   MaxSumm(i,r,k)=max(MaxSumm(i-1, r, k-l_i) + 1, MaxSumm(i-1,r,k))

   if(l_i > k)

   MaxSumm(i,r,k)=max(MaxSumm(i-1,r,k), MaxSumm(i,r+1,Original_k))

The mathematical function for l_i <=k computes for 2 different subproblems:

i. MaxSumm(i-1, r, k-l_i) + 1 : Add the i^{th} summary to the card and check for other summaries to be fit in k-l_i lines

ii. MaxSumm(i-1,r,k) : don't consider the current i^{th} summary and consider other possible summaries

For l_i >k,

MaxSumm(i-1,r,k) : the lines of text l_i >k, so don't consider that i^{th} summary and check if any other summaries will fit

MaxSumm(i,r+1,k) : no other summary would fit in this card, so consider another card and try considering i^{th} summary(i.e same summary for new card) and change the size of r card to its original value k

Original_k = size of card r i.e k, this would be the global value

Base Case:

if(i==0) then no summaries left. Therefore MaxSumm(0,r,k)=0
if(r==0) then no cards left. Therefore MaxSumm(i,0,k)=0

3. Describe how MaxSumm can be computed bottom up using a dynamic programming table. Be sure to include a description of the dimensions of the table and the order that the entries of the table are to be filled. Draw a diagram. Which entry has the solution to the original problem?

A. We need to create dynamic programming to compute MaxSumm(i,r,k) function bottom up. The table size will be n x r x k and its dimensions are n:1 to n, r:1 to r, k:1 to k
The table is filled with the base condition i.e

if(i==0) then no summaries left. Therefore MaxSumm(0,r,k)=0
if(r==0) then no cards left. Therefore MaxSumm(i,0,k)=0

The solution for the other subproblems can be computed using the base condition and this mathematical function:

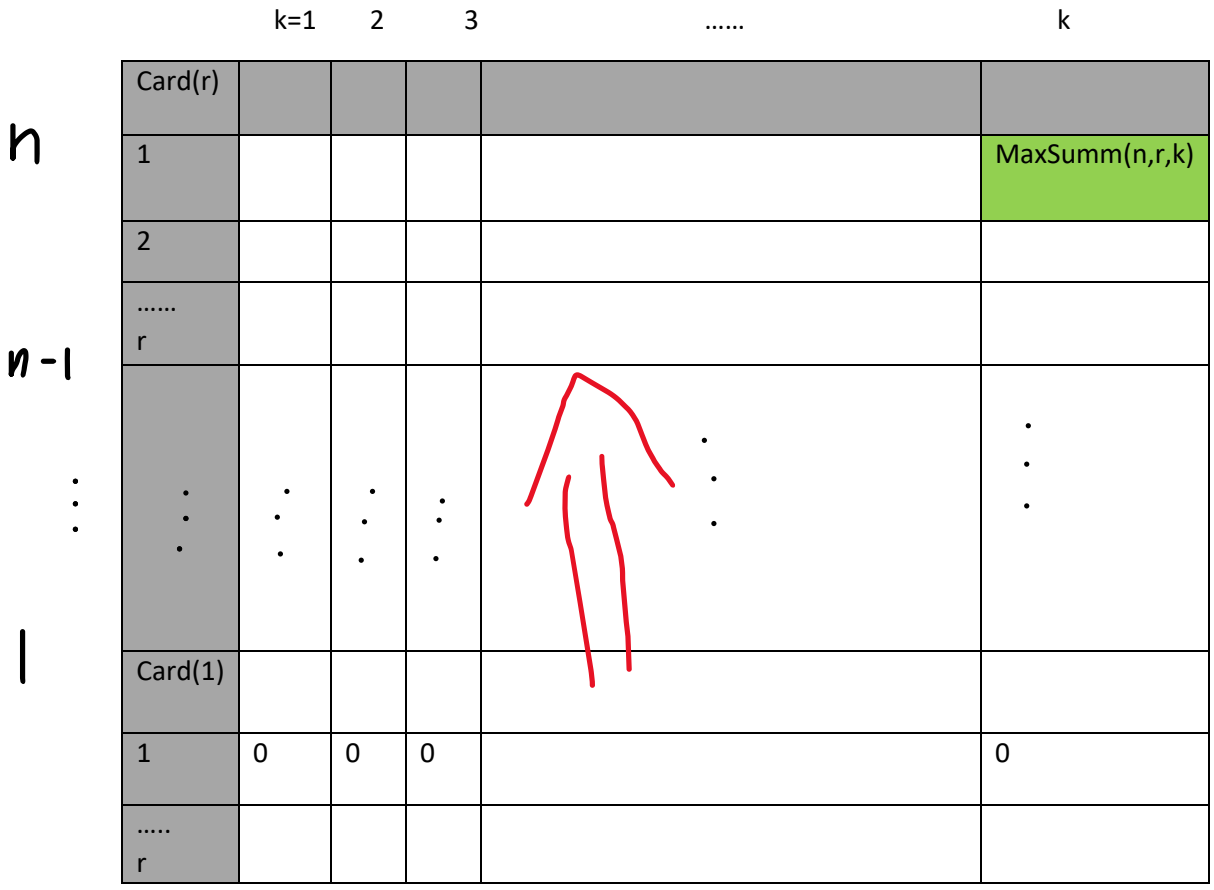
if(l_i <= k)

MaxSumm(i,r,k)=max(MaxSumm(i-1, r, k-l_i) + 1, MaxSumm(i-1,r,k))

if(l_i > k)

MaxSumm(i,r,k)=max(MaxSumm(i-1,r,k), MaxSumm(i,r+1,Original_k))

The solution to the problem is present at MaxSumm(n,r,k).



4. State and briefly justify the running time of your dynamic programming algorithm. Report your running time in terms of n, k and r.

A. For MaxSumm(i,r,k),
i = 1 to n
r cards, k = lines that each card can hold
The time complexity for a subproblem is O(r * k), each card has k lines and checks all summaries.
The total time complexity for n subproblems are r * n * k
The running time complexity = O (n * r * k)

3 Return to Oz

Upon returning to Kansas, Dorothy complains that she didn't get to see much of the Land of Oz. She just followed the Yellow Brick Road and missed all the lovely sights off that well-worn path. She makes a list of n destinations she would like to visit and asks you to take her. Why? Because you have a blimp (which is much better than a hot-air balloon).

So, here is the plan. You approximate Oz as a two-dimensional plane. Each of Dorothy's n destinations can be specified by its two-dimensional coordinates (xi , yi). Although, the blimp is powered and steerable, it is best to drift along with the prevailing winds. From the Munchkins, you find out that in the next month or so, the winds will blow east to west. After that the winds will switch directions and blow west to east. So, you can start in Munchkin Land on the eastern end of Oz and travel west visiting some of the n destinations. You'll end the first portion of your journey at the Wicked Witch's castle and hang out with the flying monkeys for a bit. After the winds start blowing eastward, you will travel east and visit the remaining destinations (the ones you didn't visit during your westward flights). You end your trip at Dorothy's house which had landed on the eastern edge of the map.

That's a good plan, but you are worried about the available fuel for your blimp and also your carbon foot print. (Let's keep Oz green!) Thus, you want to plan this trip so the total distance traveled is minimized. Here are some constraints:

• Both Munchkin Land and Dorothy's house are on the eastern edge of Oz. There are no destinations east of these points. Let's say Munchkin Land is at (x1, y1) and Dorothy's house is at (x2, y2) and x1 is a teeny bit larger than x2.

• The distance between two destinations is just the Euclidean distance between their coordinates. • During the first part of your journey, you must fly westward. The destinations that you visit must have decreasing x coordinate. (Let's just assume that none of the destinations have the same x coordinate.)

• Similarly, during the second part of your journey, you must fly eastward and visit destinations with increasing x coordinate.

• The Wicked Witch's castle is the most western destination and has the smallest x coordinate. For convenience, let's just assume that the destinations are sorted by decreasing x coordinate. That means the Wicked Witch's castle is located at (xn, yn). This all seems rather complicated, but when you meet up with the Scarecrow and he says, "You know, ever since the Wizard gave me that diploma, I've found dynamic programming to be much easier — even though I don't really have a brain!"

Questions:

1. Define a function, MinDist, that can be used to solve this dynamic programming problem. To do this, you must describe the input parameters to MinDist and its "return value" using English sentences. (Consider figures on the next page.) Note: describe the value that MinDist must return in relation to its parameters, not how to compute it, that's the next question.

A. MinDist(i,j)= Minimum distance taken to travel from initial position to the west i.e castle and from there to the destination which is in the east, where n destinations are given and each location is in the point (x,y)
i iterates from 1 to n is used to reach east destination

j is used to reach west destination 1 to n
The return value of the function MinDist(i,j) returns minimum distance required to travel from east to west and then to east.

2. Give a mathematical formula that shows how MinDist can be computed recursively. Then, explain the main parts of the formula in English. Remember to include the base cases.

A. MinDist(i,j)=(Euclidean(i,j)+min(MinDist(i-1,j) , MinDist(i,j+1))

Euclidean(i,j)=$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

The function MinDist(i,j) has 2 subproblems:

MinDist(i,j+1): It used to compute the distance to the westward direction from the source

MinDist(i-1,j): It is used to compute the for the east destination from westward point which is the minimum x

Base Condition:

if(j>i) then MinDist(i,j)= ∞

if(i=1) then MinDist(1,j)=Euclidean(1,j)

The base condition j>i ensures that the first destination is arrived which is in the west.

3. Describe how MinDist can be computed bottom up using a dynamic programming table. Be sure to include a description of the dimensions of the table and the order that the entries of the table are to be filled. Draw a diagram. Which entry has the solution to the original problem?

A. We need to create dynamic programming to compute MinDist(i,j) function bottom up.
The table size will be i x j and its dimensions are i:1 to n, j:1 to n,

The table is filled with the base condition i.e

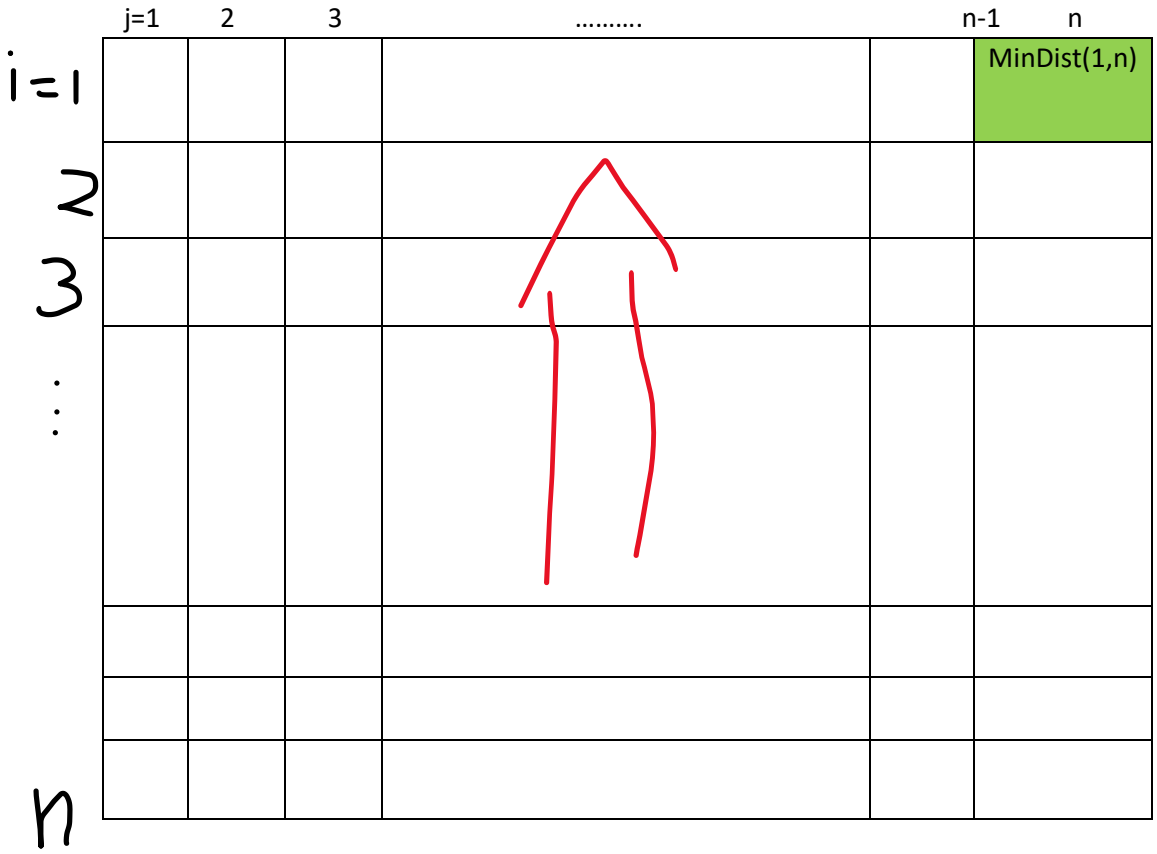
if(j>i) then MinDist(i,j)= ∞

if(i=1) then MinDist(1,j,)=Euclidean(1,j)

The solution for the other subproblems can be computed using the base condition and this mathematical function:

MinDist(i,j)=(Euclidean(i,j)+min(MinDist(i-1,j) , MinDist(i,j+1))

The solution to the problem is present at MinDist(1,n,).



4. State and briefly justify the running time of your dynamic programming algorithm.

A. The time complexity for a subproblem is O(n), to find the minimum cost from initial position to the westward position..
The total time complexity for n subproblems are n * n
The running time complexity = O (n^2)

References:

https://archive.org/details/introduction-to-algorithms-by-thomas-h.-cormen-charles-e.-leiserson-ronald.pdf/page/427/mode/2up

https://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF/page/n201/mode/2up

05-more-Dynamic-Programming-post.pdf - Google Drive

03-Dynamic-Programming-post.pdf - Google Drive

CMSC 641-01 Spring 2023 Shared Folder - Google Drive