Brandon Vo

Homework 11

1. When computing the optimal binary search tree, we could have also created another table called root[1..n][1..n] which holds which key which is the root of the optimal binary search tree for keys i through j. Fill in the tables w, e and root for the optimal binary search tree algorithm presented in class[1] where n = 7 and the word frequencies are:

begin	6%
end	6%
while	22%
if	15%
else	10%
return	30%
for	24%

Begin	End	While	If	Else	Return	For
0.06	0.06	0.22	0.15	0.1	0.3	0.24

$$w(i,j) = \sum_{l=i}^{j} p_l$$

w	0	1 (Begin)	2 (End)	3 (While)	4 (If)	5 (Else)	6 (Return)	7 (For)
1(Begin)	0	0.06	0.12	0.34	0.49	0.59	0.89	1.13
2 (End)		0	0.06	0.28	0.43	0.53	0.83	1.07
3 (While)			0	0.22	0.37	0.47	0.77	1.01
4 (If)				0	0.15	0.25	0.55	0.79
5 (Else)					0	0.1	0.4	0.64
6 (Return)						0	0.3	0.54
7 (for)							0	0.24

$$e[i,j] = \begin{cases} 0 \; if \; j = i-1 \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j)\} \; if \; i \le j \end{cases}$$

$$e[1,1] = e[1,0] + e[2,1] + w(1,1)$$

$$e[1,1] = 0 + 0 + 0.06 = 0.06$$

- E[1,2] = MIN{
  - E[1, 0] + E[2, 2] + w(1, 2) = 0 + 0.06 + 0.12 = 0.18
  - **E[1, 1] + E[3, 2] + w(1, 2) = 0.06 + 0 + 0.12 = 0.18**
- E[2, 3] = MIN{
  - E[2, 1] + E[3, 3] + W(2, 3) = 0 + 0.22 + 0.28 = 0.50
  - **E[2, 2] + E[4, 3] + W(2, 3) = 0.06 + 0 + 0.28 = 0.34**
- E[3, 4] = MIN{
  - E[3, 2] + E[4, 4] + W(3, 4) = 0 + 0.15 + 0.37 = 0.52

  - E[3, 3] + E[5, 4] + W(3, 4) = 0 + 0.34 + 0.37 = 0.71
- E[4, 5] = MIN{
  - **E[4, 3] + E[5, 5] + W(4, 5) = 0 + 0.1 + 0.25 = 0.35**
  - E[4, 4] + E[6, 5] + W(4, 5) = 0.15 + 0 + 0.25 = 0.40
- E[5, 6] = MIN{
  - E[5, 4] + E[6, 6] + W(5, 6) = 0 + 0.4 + 0.4 = 0.8
  - **E[5, 5] + E[7, 6] + W(5, 6) = 0.1 + 0 + 0.4 = 0.5**
- E[6, 7] = MIN{
  - **E[6, 5] + E[7, 7] + W(6, 7) = 0 + 0.24 + 0.54 = 0.78**
  - E[6, 6] + E[8, 7] + W(6, 7) = 0.3 + 0 + 0.54 = 0.84
- The rest were done in Microsoft Excel

e	0	1 (Begin)	2 (End)	3 (While)	4 (If)	5 (Else)	6 (Return)	7 (For)
1(Begin)	0	0.12	0.18	0.52	0.82	1.12	1.91	2.49
2 (End)		0	0.06	0.34	0.64	0.94	1.67	2.25
3 (While)			0	0.22	0.52	0.79	1.49	2.04
4 (If)				0	0.15	0.35	0.9	1.38
5 (Else)					0	0.1	0.5	0.88
6 (Return)						0	0.3	0.78
7 (for)							0	0.24
N+1								0

Root	Begin	End	While	If	Else	Return	For
Begin	1	1	3	3	3	4	6
End		2	3	3	3	4	6
While			3	3	4	4	6
If				4	4	6	6
Else					5	6	6
Return						6	6
for							7

Brandon Vo

2. Show the longest common subsequence table, L, for the following two strings: X = “notice”, Y = “encircle”. What is a longest common subsequence between these strings?

	0	e	n	c	i	r	c	l	e
0	0	0	0	0	0	0	0	0	0
N	↑0	↑0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1
O	↑0	↑0	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1	← 1	← 1
T	↑0	↑0	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1	↑ 1
I	↑0	↑0	↑ 1	↑ 1	↖ 2	← 2	← 2	← 2	← 2
C	↑0	↑0	↑ 1	↖ 2	↑ 2	↑ 2	↖ 3	← 3	← 3
E	↑0	↖ 1	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↖ 4

The longest subsequence would be 4 letters: NICE

Brandon Vo

3. Question 15-11 on page 411 of CLRS with the following modification. Instead of providing an algorithm:
    - provide the recurrence formula
    - create an example to show how your algorithm works

In your answer define what your variables mean.

15-11 Inventory planning

The Rinky Dink Company makes machines that resurface ice rinks. The demand for such products varies from month to month, and so the company needs to develop a strategy to plan its manufacturing given the fluctuating, but predictable, demand. The company wishes to design a plan for the next n months. For each month i, the company knows the demand d_i, that is, the number of machines that it will sell. Let D = $\sum_{i=1}^{n} d_i$ be the total demand over the next n months. The company keeps a full-time staff who provide labor to manufacture up to m machines per month. If the company needs to make more than m machines in a given month, it can hire additional, part-time labor, at a cost that works out to c dollars per machine. Furthermore, if, at the end of a month, the company is holding any unsold machines, it must pay inventory costs. The cost for holding j machines is given as a function h(j) for j = 1, 2, …, D, where h(j) ≥ 0 for 1 ≤ j ≤ D and h(j) ≤ h(j + 1) for 1 ≤ j ≤ D – 1.

Give an algorithm that calculates a plan for the company that minimizes its costs while fulfilling all the demand. The running time should be polynomial in n and D.

- We can create a matrix that represents the cost for each month based off the number of machines being sold.
- Revenue would be considered 0 if the number of machines being manufactured doesn't meet demand due to failing the requirements.
- We can use a matrix Cost[i][j] to the cost of building a total of machines for month j if we had leftovers from month i.
    - We can then calculate the cost based on how many extra machines we could make, how much we could save based on our stockpile and how many we manufactured for the month.
- We have 3 cases based on the number of machines we could make
    - We have manufactured more than the demand in which case we pay extra for inventory of h(m – d_i)
    - We manufactured less than the demand in which case we pay c * (the amount of extra machines we need): c*(d_i – s[k][j] - m)
        - Array s[k][j] refers to the leftover machines used for a previous month that manufactured k machines
    - If we match the exact amount of the demand, we pay only the cost of manufacturing.
    - We must also count machines that could be used as leftovers from the previous month. The leftover machines will be stored in a separate array labelled as s[1…n] for each month.
    - We can also choose to manufacture more machines than d_i to save for the next month. A will represent the number of additional machines we can manufacture.

Brandon Vo

$$Cost[i,j] = \begin{cases} 0 \; if \; m = d_i \; and \; A == 0 \\ MIN\{Cost[m - d_i][j] + h(m - s[k][j] - d_i) + A * c\} \; if \; m + A - s[k][j] > d_i \\ \text{MIN}\{Cost[d_i - m][j] + c(m - A - s[k][j])\} \; if \; m + A - s[k][j] < d_i \end{cases}$$

- $MIN\{Cost[m - d_i][j] + h(m + d_k - s[k-1] - d_i) + c * d_k\}$ refers to the cost for manufacturing an additional d_k machines even if we have met the month's quota of d_i machines.
    - It would cost an additional $h(m + A - s[k-1] - d_i)$ for inventory cost
    - It would cost an additional c*d_k to manufacture the extra materials
- $MIN\{Cost[m - d_i][j] + h(m + A - s[k-1] - d_i)\}$ refers to the cost to store the extra number of (m – d_i) machines because we manufactured more machines than d_i's quota.
- $\text{MIN}\{Cost[d_i - m][j] + c * (A)$ refers to the additional cost to manufacture extra machines such that we atleast meet the quota of d_i machines.
    - It would cost an additional c*A to manufacture an extra number of A machines

If we're planning for 4 months where people are ordering 4, 3, 9, and 6 machines.

- If m machines = 4 machines
- C cost would be an additional 3 dollars per machine
- The cost of inventory h(j) is an additional $5 for each machine.
- The demand has been predicted to be for 4, 3, 9, and 6 machines for 4 months with the total demand D = 32
- Where staff can provide maintenance for up to 6 machines

- Cost[0, M1] = 0
    - Cost[5 - 4, M1] + c*(1) + h(1) = 8
    - Cost[6 - 4, M1] + c* 2 + h(2) = 16
    - Cost[3, M1] + c*3 + h(3) = 24
- Cost[M1, M2] = MIN{
    - Cost[, M2] + h(4 – 0 – 3) = Cost[M1, 5] + h(1) = 0 + 5 = 5
    - Cost[1, M2] + h(5 – 0 – 3) =
- Cost[M2, M3] = MIN{
    - Cost[9 – 5 - 1, M3] + c * (9 – 5 – 1) = Cost[3, M3] + c(3) = 5 + 3(3) = 14
- Cost[M3, M4] = MIN{
    - Cost[6 – 5 - 0, M4] + c * (6 – 5 – 0) = Cost[1, 1] + c(1) = 14 + 1 * 3 = 17

s	D_1 - A	D_2 - A	D_3 - A	D_4 - A
0	0	1	-4	-1
1	1	2	-3	0
2	2			

Brandon Vo

4. (Optional - do not turn in) Question 15-12 on page 411 of CLRS .
5. You are taking a road trip to visit Aunt Mary (Yet again)!

Since you are taking Design and Analysis of Algorithms, you are going to optimize your route.

You will be traveling with your cousin. Since your cousin owns the car, he has decide which route you will take. You have, however, convinced your cousin you are the one to decide what hotels to stop at on your way (they all are mysteriously on the side of the road). You are going to optimize your route so it costs the least (you are a graduate student after all and need to save money).

On your route are a series of hotels $h_1, h_2, \ldots, h_n$. The hotel h_i costs c_i.

If you travel more than 300 miles per day, you realize you will spend more on coffee, specifically you will spend (n/2) for n miles over 500 (your cousin crashes hard, and hiring someone to help move him into the hotel room is reasonably expensive)

You would like to minimize your expenses. For this problem:

• provide the recursive solution

• prove that the problem has optimal substructure

• use your recurrence formula to find the minimum cost to get to California which is 2,700 miles away for the case where the hotels are at mile markers: 400, 850, 926, 1420, 1540, 1950, 2100, 2400 and the hotels cost $56, $75, $46, $76, $49, $60, $55, $87.

In your answer define what your variables mean.

- Cost[i, j] will represent the amount of money based on which hotel we enter from which starting point
- We still also have an m table which represents the amount of miles we travel based on our starting location.
- We must include mile 0 as our starting point and choose the hotels from there.
- Will make an assumption that we must always move from one hotel to another for the sake of efficiency. We will not stay in one place for any longer than one night.

$$Cost[h_i] = \begin{cases} MIN\{Cost[h_k] + P(i)\}\ if\ M(k) - M(i) < 500 \\ MIN\left\{Cost[h_k] + P(i) + \frac{\big(M(i) - M(k)\big) - 500}{2}\right\}\ if\ M(i) - M(k) > 500 \end{cases}$$

- Where $0 \le k \le i$
- P(i) means the cost to stay at hotel i
- Cost[h_k] represents the cost of reaching hotel k
- M(i)-M(k) represents the cost to travel from hotel k to hotel I in terms of miles.

- Cost[H1] = Cost[H1, 0] + P1 = 56 + 0 = 56
- Cost[H2] = MIN{

    - Cost[0, H2] + Cost[H2, H2] + [W(H2) – W(0)-500]/2 +P2 = 75 + 0 + (850-500)/2 = 250
    - **Cost[0, H1] + Cost[H1, H2] + P2= 56 + 75 + 0 = 131**
- Cost[H3] = MIN{
    - **Cost[H1] + Cost[H1, H3] + P3 = 56 + 46 + 0 = 102**
    - Cost[H2] + Cost[H2, H3] + P3 = 131 + 46 = 177
    - Cost[H0, H3] + W[(H3-H0) – 500]/2 + P3= 46 + (926 – 500)/2 = 259
- Cost[H4] = MIN{
    - Cost[0, H4] + Cost[H4, H4] + P4 = (1420 – 500)/2 + 76 = 536
    - **Cost[H3] + Cost[H3, H4] + P4 = 102 + 76 = 178**
    - Cost[H2] + Cost[H2, H4] + P4 = 131 + 76 + (1420 - 850 - 500)/2= 242
    - Cost[H1] + Cost[H1, H4] + P4 = 56 + (1420-400-500)/2 + 76= 392
- Cost[H5] = MIN{
    - Cost[0, H5] + Cost[H5, H5] + P5 = 49 + 0 + 520 = 569
    - Cost[H4] + Cost[H4, H5] + P5 = 178 + 49 = 227
    - **Cost[H3] + Cost[H3, H5] + P5 = 102 + 49 + (1540 – 926 - 500)/2= 208**
    - Cost[H2] + Cost[H2, H5] + P5 = 131 + (1540 – 850 - 500)/2 + 76 = 302
    - Cost[H1] + Cost[H1, H5] + P5 = 56 + (1540 – 400 - 500)/2 + 49 = 425
- Cost[H6] = MIN{
    - Cost[0, H6] + Cost[H6, H6] + P6 = (1950 – 0 - 500)/2 + 60 = 785
    - Cost[H5] + Cost[H5, H6] + P6 = 208 + 0 + 60 = 268
    - **Cost[H4] + Cost[H4, H6] + P6 = 178 + (1950 – 1420 - 500)/2 + 60 = 253**
    - Cost[H3] + Cost[H3, H6] + P6 = 102 + (1950 – 926 - 500)/2 + 60 = 424
    - Cost[H2] + Cost[H2, H6] + P6 = 131 + (1950 - 850 – 500)/2 + 60 = 491
    - Cost[H1] + Cost[H1, H6] + P6 = 56 + (1950 – 400 – 500)/2 + 60 = 641
- Cost[H7] = MIN{
    - Cost[0, H7] + Cost[H7, H7] + P7 = (2100 – 500)/2 + 55 = 855
    - Cost[H6] + Cost[H6, H7] + P7 = 253 + 0 + 55 = 306
    - **Cost[H5] + Cost[H5, H7] + P7 = 208 + (2100 – 1540 - 500)/2 + 55 = 293**
    - Cost[H4] + Cost[H4, H7] + P7 = 178 + (2100 – 1420 – 500)/2 + 55 = 323
    - Cost[H3] + Cost[H3, H7] + P7 = 102 + (2100 – 926 – 500)/2 + 55 = 494
    - Cost[H2] + Cost[H2, H7] + P7 = 131 + (2100 – 850 – 500)/2 + 55 = 561
    - Cost[H1] + Cost[H1, H7] + P7 = 56 + (2100 – 400 – 500)/2 + 55 = 711
- Cost[H8] = MIN{
    - Cost[0, H8] + Cost[H8, H8] + P8 = (2400 – 500)/2 + 0 + 87 = 1037
    - Cost[H7] + Cost[H7, H8] + P8 = 293 + 0 + 87 = 380
    - **Cost[H6] + Cost[H6, H8] + P8 = 253 + 0 + 87 = 340**
    - Cost[H5] + Cost[H5, H8] + P8 = 208 + (2400 - 1540 – 500)/2 + 87 = 475
    - Cost[H4] + Cost[H4, H8] + P8 = 178 + (2400 - 1420 – 500)/2 + 0 + 87 = 505
    - Cost[H3] + Cost[H3, H8] + P8 = 102 + (2400 - 926 – 500)/2 + 0 + 87 = 676
    - Cost[H2] + Cost[H2, H8] + P8 = 131 + (2400 - 850 – 500)/2 + 0 + 87 = 743
    - Cost[H1] + Cost[H1, H8] + P8 = 56 + (2400 - 400 – 500)/2 + 0 + 87 = 893
- Cost[H9] = MIN{
    - Cost[0, H9] + Cost[H9, H9] = (2700 - 500)/2 + 0 + 0 = 1100
    - Cost[H8] + Cost[H8, H9] = 340 + 0 = 340

- **Cost[H7] + Cost[H7, H9] = 293 + (2700 – 2100 – 500)/2 = 340**
- Cost[H6] + Cost[H6, H9] = 253 + 0 = 253 + (2700 – 1950 – 500)/2 = 378
- Cost[H5] + Cost[H5, H9] = 208 + (2700 – 1540 – 500)/2 = 538
- Cost[H4] + Cost[H4, H9] = 178 + (2700 – 1420 – 500)/2 = 568
- Cost[H3] + Cost[H3, H9] = 102 + (2700 – 926 – 500)/2 = 739
- Cost[H2] + Cost[H2, H9] = 131 + (2700 – 850 – 500)/2 = 806
- Cost[H1] + Cost[H1, H9] = 56 + (2700 – 400 – 500)/2 = 956

- The optimal route would be H1, H3, H5, H7 to the final destination with the cost of $340

Cost[I, j]	0	H1	H2	H3	H4	H5	H6	H7	H8	H9
0	0	56	131							
H1		0	75	121						
H2			0	46	122					
H3				0	76	125				
H4					0	49	109			
H5						0	60	115		
H6							0	55	142	
H7								0	87	340
H8									0	0
N+1										0

MILE[I, j]	0	H1	H2	H3	H4	H5	H6	H7	H8	H9
0	0	400	850	926	1420	1540	1950	2100	2400	2700
H1		0	450	526	1020	1140	1450	1700	2000	2300
H2			0							1850
H3				0						1774
H4					0					1280
H5						0				1160
H6							0		450	750
H7								0	300	600
H8									0	300

	H1	H2	H3	H4	H5	H6	H7	H8	H9
m	400	850	926	1420	1540	1950	2100	2400	2700

Brandon Vo

6. You have your first retail job. You earn commissions on how much you sell. Fortunately, during the black Friday sales, you don't have to work the floor - you only have to ring up customers. You have worked long enough to know just by looking at a customer the approximate value of merchandise they will purchase (ready for you to make a commission on). It is Friday night on Black Friday, and only you and a single co-worker are ringing up sales. The announcement system chimes - the store is closing and the customers now should go to the cash registers to pay for their items and exit the store. The store has (for some reason best understood by middle management) two lines to get to the two registers: you and your co-worker take turns signaling the next customer to service. Design an algorithm to maximize the profit you will make - given that your coworker has also decided to maximize their profit (assume your co-worker will also act optimally). In addition to providing an algorithm for this problem, you need to:

- Provide the recurrence relation. [2]
- Create a small example to show how your algorithm works by showing how the values are added to the subproblem table your algorithm uses.

- Two lines line i and line j. Make the correct choice of who to put in the line choose the max of line i or line j.
- Must optimize for one line as much as possible while leaving the minimum profits for the other line. Must aim for the optimal solution which may not necessarily be a greedy approach.
- Whenever you choose a customer, the other person will select one as well
- Assuming that you have to choose a person for the optimal solution, the solution will be determined by n/2 choices
    - Whenever we choose from one line, the revenue will have to count based either the line we chose from or the line that we didn't chose from
    - The line that we chose from will have i+1 or j+1 because the customer is removed after that point not including the other person's turn
    - This leaves the other person to choose form line I which means i+2 left or j which leaves j+1 left.
    - The matrix rev will count the profit based on the possibilities of choosing either line as Rev[I, j]
- We begin by choosing line I or line j as REV[i] or REV[j]

$$\mathrm{COM}[i,j] = \begin{cases} 0 \; if \; i < 0 \; or \; j < 0 \; or \; (i+j) > n \\ MAX(REV[i], REV[j]) \; when \; i+1 = j \\ V_i + MAX\{LINE\_I[i-1] + LINE\_J[\,j-1] + W(i), LINE\_I[i-2, j] + \; LINE\_J[j] + W(i)\} \end{cases}$$

For an example, if we were left with 6 customers remaining who had the following total price for the merchandise:

Line I: $25, $35, $50

Line J: $65, $15, $20

Brandon Vo

- We start by choosing line I or line j
- REV[1, 1] = MAX{LINE I[1] + W(i), LINE J[1] + W(j)}
    - **WI(2) + Rev[0, 1] = $35 + 65 = 100**
    - WI(2) + Rev[1, 0] = $35 + 25 = 60
- REV[1, 2] = MAX{
    - **WI(2) + Rev[0, 1] = 35 + 65 = 95**
    - WI(2) + Rev[1, 0] = 35 + 25 = 60
    - WJ(3) + Rev[0, 1] = 20 + 65 = 85
    - WJ(3) + Rev [1, 0] = 20 + 25 = 45
- REV[1, 3] = MAX{
    - WI(2) + Rev[0, 2] = 25 + 15 = 40
    - **WI(2) + Rev[1, 1] = 25 + 100 = 125**
- REV[2, 1] = MAX{
    - **WI(3) + Rev[0, 1] = 50 + 65 = 115**
    - WI(3) + Rev[1, 0] = 50 + 25 = 75
    - WJ(2) + Rev[1, 0] = 15 + 65 = 80
    - WJ(2) + Rev[0, 1] = 15 + 25 = 40
- COM[2, 2] = MAX{
    - WI(3) + REV[0, 2] = 50 + 15 = 65
    - **WI(3) + REV[1, 1] = 50 + 100 = 150**
    - WI(3) + REV[2, 0] = 50 + 35 = 85
    - WJ(3) + REV[1, 1] = 20 + 100 = 120
    - WJ(3) + Rev[2, 0] = 20 + 35 = 55
    - WJ(3) + Rev[0, 2] = 20 + 15 = 35
- COM[2, 3] = MAX}
    - WI(3) + REV[0, 2] = 50 + 15 = 65
    - WI(3) + REV[1, 2] = 50 + 95 = 145
    - **WI(3) + REV[2, 1] = 50 + 115 = 165**
- COM[3, 1] = MAX{
    - WJ(2) + REV[2, 1] = 15 +
    - WJ(2) + REV[1, 1] = 15 + 100
- COM[3, 2] = MAX{
    - WI(3) + REV[3, 0] = 50 + 50 = 100
    - WJ(3) + REV[2, 2] = 20 + 89 = 20 + 89 = 109
    - WJ(3) + REV[3, 0] = 20 + 50 = 70
- COM[3, 3] = MAX{
    - WI(3) + REV[1, 3] = 50 + 85 = 135
    - **WI(3) + REV[2, 2] = 50 + 89 = 139**
    - WJ(3) + REV[2, 2] = 20 + 89 = 109

w	0	1	2	3
0	0	65	15	20
1	25	100	115	125
2	35	0	150	165

Brandon Vo

3	50		0	139

	1	2	3
i	$25	$35	$50

	1	2	3
j	$65	$15	$20

COM(WI, Wj)

Let rev[1…n, 1…n] be a new table

For I = 1 to WI.length

REV[I, 0] = 0

For j = 1 to WJ.length

REV[0, j] = 0

For I = 1 to m

For j = 2 to n

Q = -∞

If (I – 1 ≥ 0)

If (Q > REV[i-1, j-1] + WI(i)) Q = REV[I-1,j-1]+WI(i)

If (Q > REV[i-1, j-1] + WJ(j)) Q = REV[I-1,j-1]+WJ(J)

If (I – 2 ≥ 0)

If (Q > REV[I – 2, j] + WI(I)) Q = REV[I – 2, j] + WI(I))

If (Q > REV[I - 2, j] + WJ(J)) Q = REV[I – 2, j] + WJ(J))

If (j – 2 ≥ 0)

If (Q > REV[I, j-2]+WJ(J)) Q = REV[I, j - 2] + WJ(J)

If (Q > REV[I, j-2]+WI(I)) Q = REV[I, j - 2] + WI(I)

If (I – 1, j – 1 ≥ 0)

If (Q > REV[I – 1, j – 1] + WJ(J)) Q = REV[I – 1, j – 1] + WJ(J)

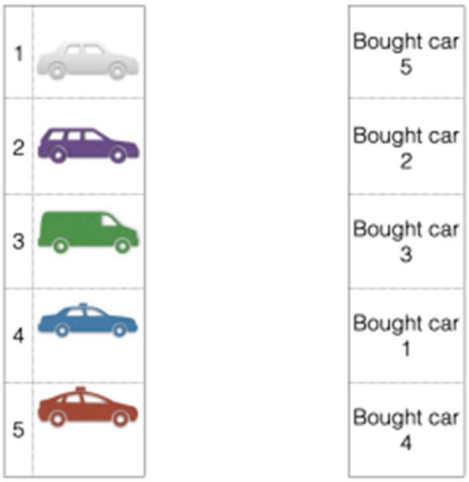
If (Q > REV[I – 1, j – 1] + WI(I)) Q = REV[I – 1, j – 1] + WI(I)

Brandon Vo

REV[I, j] = Q

Brandon Vo

7. Your heart stops. You see the car of your dreams. You bid and you win your bid at the car auction (perhaps $10,000 for the rusty mustang was a bit too much to bid - or perhaps you just scored. You hope the car actually runs...) At the auction house, the requirement is for everyone to socially distant themselves, due to the design of the building (it's a very pretty modern design constructed well before 2020) there is not that much space on the paths to collect the vehicles. Cars are put on one side of the row, and people start on the other side. What is the maximum number of people who can walk to their car where without crossing paths with each other. For example, in the graph below person buying car 1 and person buying car 4 could both walk to their newly bought cars at the same time without crossing each other's path. Similarly the person buying car 2, the person buying car 3 and the person buying car 4 could walk to their newly bought cars at the same time. However, it would not be allowed for the person who bought car 3 and the person who bought car 1 to walk to their newly bought cars at the same time since



their paths intersect.

What is the maximum number of car that can be delivered to their new owners at the same time (i.e. no car can cross the path of another car to make sure no collisions happen).

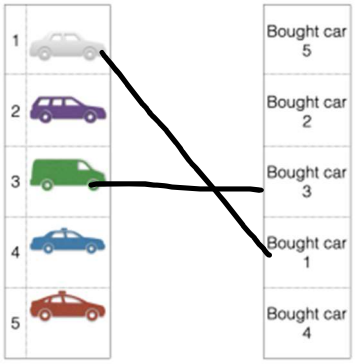
For this problem you only need to provide the following:

Start with one person, one car and build from that. Build a graph through DP.

The format will be built similarly to an LCS graph where everybody starts from one side and will move towards their car until they reach a path that's being crossed by another person.

- The location of the people, the car, and the paths to reach their cars can be represented as a matrix.
    - Each person must travel across the matrix to reach their destination.

- It will be assumed that nobody will go around each other or behind each other for this algorithm as everybody starts on one end of the matrix and will not leave the matrix to reach their car.
- It will also be assumed that everybody travels at about the same pace and start at the same time so that an intersection is determined if their paths intersect.
- It will also be assumed that nobody will stop and wait for the other person if they intersect, so the only optimal solution is to determine who gets to move at what time.



The figure above means that if customer 1 goes to car 1. The only way customer 3 can go to van 3 is if he waits for customer 1 to leave first. He cannot go outside of his given area. He cannot go faster than customer 1.

- ROAD[1…X,1…Y] represents the area containing the customers and cars. This will determine where within the bounds of ROAD that the customers may move in to reach their car.
- CAR[1…n] will represent the list of cars on ROAD and their location based on ROAD.
- CUST[1…n] will represent the list of customers who need to reach their car.
  - Each element contains the [X, Y] coordinates of where that customer is on the matrix.
- Start with one person and have them move towards their car. That path will be recorded after as the person moves along the road.
- The next person would then have his/her path used to move along the road.
  - If this person reaches point that would intersect with the other person's movements, then that means that these two people will have intersecting paths

MAN-CAR(CUST, CAR, m, n)

//CUST is a stack containing all of the people and where they start at

//CAR is the location of the cars for each respective customer

Let ROAD[1…m,1…n] be a new table

Brandon Vo

```
TIME[1…n]        //Record who gets to move at what time
For I = 1 to m
   C[I,0] = 0
For j = 0 to n
   C[0, j] = 0
A = 0
While(CUST is not empty)
   (X, Y) = POP(CAR)        //Show where customer i's car is located
   (i, j) = POP(CUST)       //Show where customer i is located
   TIME[a] = 0              //Record this customer's time
   A++
      While ((X, Y) != (i, j))
         If (ROAD[i, j] > 0 and TIME[a] == ROAD[I, j])        //If an intersection is found
            TIME[a]] = ROAD[I, j] + 1                         //This person will wait after the previous person
         If (X < i and Y > j)
           ROAD[I, j]++
           ROAD[I, j] = ↖ //Car is in the top left area
         Else If (X < M and Y == M)          //Car is in the left area
            ROAD[I, j]++
            ROAD[I, j] = ←
         Else If (X < M and Y < M)           //Car is in the bottom-left area
            ROAD[I, j]++
            ROAD[I, j] = ↙
         Else //Car is below the customer
            ROAD[I, j]++
            ROAD[I, j] = ↓
Return largest value in TIME[n]
```

Brandon Vo

8. (3 bonus points) Think of a good exam or homework question for dynamic programming.

Dave and Goliath are competing to take down a tree, but they are too afraid to get near it. Instead of using an axe or a saw, they instead want to throw spears at the tree until it falls down. Fortunately, both Dave and Goliath have impressive accuracy and can strike any part of the tree to do any number of damage based on where he hit the tree. Dave and Goliath have 3 spears each and they will not pick up their spears until the tree falls down. In addition, Once a spear hits a section of the tree, no other spears can hit that tree and do matching damage. The weak points of the tree are represented below:

-5 hp	-10 hp	-5 hp
-3 hp	-100 hp	-60 hp
-10 hp	-20 hp	-30 hp

If the tree had to take approximately 200 hp worth of damage to fall down, what would be an optimal solution for Dave to do the finishing blow to the tree? Assume that Goliath and Dave are both playing optimally.