Adam Paz
1363522

Homework 2

1 Walk Ways

1. Let OPT_r be the optimal time to get to the right walkway, and OPT_l be the optimal time to get the the left walkway at the current position j, where $1 < j < n$ n being the total amount of walkways on each side. And let l_j and r_j be the time it takes to take the left and right walk ways at walk way j respectively.

   i. $OPT_r(j)=\begin{cases} j = 1 : OPT(j) = \mathrm{r}_1 \\ else : min\{\mathrm{OPT}_r(j-1) + r_j, OPT_l(j-1) + r_j + k\} \end{cases}$

   ii. $OPT_l(j)=\begin{cases} j = 1 : OPT(j) = \mathrm{l}_1 \\ else : min\{\mathrm{OPT}_l(j-1) + l_j, OPT_r(j-1) + l_j + k\} \end{cases}$

   iii. Additionally we will need **OPT(j) = min{**$OPT_r(j), OPT_l(j)$ **}** to determine whether we end on the left or right walkway. And return the minimum time possible to get to the nth pair of walkways

2. The reason that this OPT holds is because we have OPT_r and OPT_l such that each tells us the optimal to get the the right or left side respectively. We reach our base case through recursion because j-1 calls j-2 which calls j-3 and so on. Once we reach the base case, it builds up from there. The initial problem of taking the first walk way is trivial, you take the faster of the left and right side. The second round through the n round of computations will compute the time it would take to get from both previous walkways to the left and right current ones. And for both previous walkways we compute both previous walkways to that, and so on and so forth, therefore any optimal time will be computed because all possible combinations are looked at and the minimum path is chosen due to the min function. Our overall OPT(j) which just chooses the minimum between the right and left side OPTs will decide whether the left or right side is optimal to take at the current step. Therefore it will return the overall minimum once we reach the nth pair of walkways.

3. For this OPT we can store our results in an array that keeps track of the minimum time taken up until this point, which is returned by iii throughout each step of the recursion. So once it gets to the base case it will store the shorter time of the two pathways and then it will build upward from there filling out the array with cumulative times until it gets to the last index. The last index will hold the minimum total value for getting to the last walkway whether it ends on the left or on the right. There is no direct need to store results however, because the OPT itself will return the value desired.

4. No additional information is needed to get the solution, the OPT returns the value needed.

5. This algorithm runs in O(n) time. There are 2n walksways which are compared simultaneously so it can be thought of as n entries with constant time decision making between the two walkways, therefore it is O(n).

2 Taxi or Limo

1. $OPT(j)$=$\begin{cases} j < 5 : OPT(j) = L[j] * r \\ else : min\{\text{OPT(j-1) + L[j]*r, OPT(j-5)+ b}\} \end{cases}$

2. The reason that this OPT function works recursively is because we know that for getting to any of the first 4 points we need to use a taxi. We can use these as our base. From there we can trust our recursion will work from the end back to the base case. We compare the difference between taking a limo for the last 5 stops with taking a taxi from the last stop to the current one, and since OPT is a cumulative value when OPT(j-5)+b is compared with OPT(j-1)+l[i]*r we can determine which of the two transportation methods is more valuable at the current time. At city 5 the first real decision is made to whether we should take the limo for the last 5 stops or should we take a taxi from the last stop the current stop. Since the algorithm looks back 5 stops to choose to use the limo rather than forward we do not need to worry about it taking a limo for less than 5 stops at the end.

3. The data structure we will use to implement this algorithm is an array. It will store a character of either an L for the start of a limo trip and x for the following 4 or a T for taxi. These will keep track of what mode of transportation is used at each stop. It will fill the first 4 with T's until it reaches the computation for the 5th stop, in which case there are two possibilities. Either it fills the 5th slot with another T or it replaces all 5 slots with an L and 4 Xs because the limo takes up the first 5 spaces. It continues to the 6th slot and does the same thing and so on and so forth.

4. Once we have reached the end of the array there can be conflicting limo times so we move backwards through the array. Moving backwards if we see an L within 5 spaces of another L we rid of the L of lower index and the X's following it replacing them with Ts.

5. The OPT runs in O(n) time because there are n stops and it is constant time for each computation. It is the O(n) to go back through the array and replace conflicting limos. Therefore it is O(n).

Adam Paz
1363522

Homework 2

3 Museum

1. Let M[n,m] be the maximum total facts learned by the problem. Lets define n as the number of exhibits where i is 0¡i¡n. m as the max amount of time at the museum and j is 0¡j¡m. x is our test variable for how many minutes are spent at the exhibit where $0 < x < j$.
   OPT(j)=$\begin{cases} i = 0 : OPT(0,j) = 0 \\ else : max\{\text{OPT(i,j), OPT(i-1,j-x) + f}_i(x))\} \end{cases}$

2. The reason this OPT works is because it covers every possible combination of times and exhibits. It starts at the end then recursively because of i-1 moves all the way back to the base case, which is starting with no exhibits and spending every amount of time, filling with 0s. Then continues working down row wise. So 1 exhibit then 2 and so on and so forth until every possibility is covered, covering every possibility of what x is for every i and every j. It only needs things that have already been solved based on the base case and working out row-wise, then width wise recursively until everything is solved. Deciding at any given (i,j) pair what it is x should equal. When x = 0, it means no time is spent at the exhibit. It compares the optimal of what we have thus far with the optimal of i-1 including the newest facts, and uses these to choose how much time to spend at the current exhibit.

3. The information is stored in a 3-dimensional array. It starts by filling the top row of and the width and moves down row wise then width wise until it gets to the bottom. The three dimensions are m,n, and our variable x which has max j. So it would be m,n,j. It fills in every possible x for every given i and j. So if it were in loops it would be three nested for loops where x is the innermost loop.

4. You look at the last index of the array in order to get the maximal amount of exhibits visited, and trace back through the entries using the recorded values in the x dimension to find the best or optimal distribution of our time to learn the facts.

5. Since there is n exhibits and there is m amount of possible time and x counts up to m, therefore we have m*m for the total possible times to calculate multiplied by the total number of exhibits. Therefore it is O(n*m*m), which is also supported by the three nested for loops.

Adam Paz
1363522

Homework 2

4 Restaurants

1. Let P(j) = OPT(J-d-1)

   $$\text{OPT(j)=}\begin{cases} j = 0 : OPT(j) = 0 \\ else : max\{\text{V}_j + OPT(P(j)),\, Opt(J-1)\} \end{cases}$$

2. This recursion holds true because you are computing the total possible value of the previous item that is compatible with the current and adding it to the current item, then we compare this value with the item right before the current one. If you know the value of OPT(0) then finding OPT(1) is trivial, and from there OPT(2) is as well and so on and so forth, until you get to J-1. You always take the largest of the two values so when the algorithm terminates it will choose either

   (a) to take OPT(J) and accept this as the largest number, meaning that it is the largest possible profit due to the fact that all of the numbers compatible to it that came previously are larger than all the compatible options previous to OPT(J-1). Therefore you end up with the largest possible total profit for every location possible for the stores.

   (b) or you take OPT(J-1) deeming it as the largest possible total profit up until that point which includes in it's analysis either (a) it is the largest profit until this point, or (b) which includes the element below it, which in turn takes into consideration the element below that and so on.

   Therefore every element possible is analyzed and compared to every other viable solution up until the point that it is tested.

3. To store the items we will use an array where j will determine the index of the array. The way to store the data is to choose whether the algorithm chooses (a) or (b). If (a) is chosen then we can put a 1 in that place in the array, if (b) is chosen we can put 0. The array will fill up from the smallest index up until the largest index because of the way recursion works in this circumstance, where the sub problems for a smaller input size are solved before moving on. Then once we have the largest index filled we can find the solution from there.

4. We will end up with an array of size n with binary code telling whether we took the element in this index of the array or not. We then look for the last 1 in this array and take all other compatible elements with this element. Do this by taking P(j) recursively, and this will give all compatible elements. This will produce the optimal solution that we seek.

5. This algorithm runs in O(n) assuming that the possible locations of the restaurants are already sorted by location. It takes O(1) per index in the array, solves for the max and stores a digit in an array, and there are n indices. To move back through the array also is an O(n) operation and therefore the algorithm runs in O(n).