Advanced Algorithm HW4

Seungho Jang

September 2020

1 Q1

The activity selection problem can be start with an situation where I own a bike, and I want to rent my bikes to my friends. Some of them are nice about hours, and some of them are not. Also, I want to be little bit nice while renting a bike because I wants many friends to rent the bike. The renting schedule is shown in the table below.

Activity Selection Problem											
i	1	2	3	4	5	6	7	8	9	10	11
s	1	2	0	4	6	7	7	11	10	12	14
f	3	5	5	8	8	10	11	13	14	15	16

Table 1: Table for Activity Selection Problem

Now, i represents the each person's schedule, s represents start time of each person's renting time, and f represents finish time of each person's renting time. As shown above, there might be some overlapping problem where my first friend want the bike from 1 to 3, but my second friend want 2 to 5, then there is overlapping 1 hours. Again, the goal here is to find the schedule time such that there are no overlapping, which is mutually compatible hours for my friends. There might be lots of ways to rent my bike to my friends, but I want the maximum friends. One possible solution is 1, 4, 8, and 11 or 1, 5, 8, and 11. Other solutions possibly could be 2, 6, 10 and 3, 6, and 10, but this solution is not optimal solution because it is only three schedules. In conclusion, the goal of activity selection problem is to find a maximum subsets that are mutually compatible activities, which they shouldn't be overlapping each other.

2 Q2

The code can be found in Q2-Various Algorithms for Activity Selection Problem

The activity selection problem exhibits optimal substructure. Let's assume that A_{ij} is the optimal solution includes a_k such that a_k is part of optimal solution. A_{ij} can be expressed $A_{ij} = Aik \cup a_k \cup A_{kj}$. Aik represents all the

activity finishes before A_k starts and A_{kj} represents after A_k finishes. Then the number of elements in $A_{i,j}$ can be sum of $A_{i,k}$, $A_{i,k}$, and 1. If $A_{i,k}$ is part of my optimal solution from $S_{i,k}$ and the length of $A'_{i,k}$ is larger than $A_{i,k}$ then $A_{i,k}$ will be bigger because $A_{ij} = Aik \cup a_k \cup A_{kj}$. Aik condition should be satisfied. Also, if we denote the length of A_{ij} as $c[i,j]$. Then, $c[i,j] = c[i,k]+c[k,j]+1$ can be defined. Then this could give us a recurrence algorithm such as $c[i,j] = 0$if $s[i,j] = 0, c[i,j] = max(c[i,k] + c[k,j] + 1)$ if $s[i,j] \neq 0$. As a result, this problem exhibits the optimal substructure because if we know a_k, we can create an optimal solution to $S_{i,j}$ by choosing a_k in company with all activities in optimal solution to the sub-problem $S_{i,k}$ and $S_{k,j}$

3 Q3

The code can be found in Q3-Compare and Check the iterative greedy algorithm running time

The method that I used in this code, is to find the running time given the activity selection table given in the problem in Q2. First, obtain the running time based on the table, which their finish time is all sorted. Then compare running time of iterative greedy algorithm with unsorted input by using a random library(basically shuffle the activity table). Then add the running time of algorithm with sorted input and the running time of sorted algorithm. As you can see the figure 1, it shows that the running time of iterative greedy algorithm + time it took for sorting the algorithm is equal to running time of iterative greedy algorithm with unsorted algorithm(which is green and orange is same line). There is a limitation on this method because some of the element doesn't shuffle all the way, some of them are still same as the original activity list. However, the conclusion is that average running time of any sorting algorithm + running time of iterative greedy algorithm is the running time of iterative greedy algorithm with unsorted activity. If the average sorting time is $n * log(n)$. Then the the total time of iterative greedy algorithm will be $n + n * log(n)$.

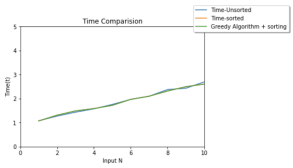

Figure 1: Running time comparison

4 Q4

Discuss the difference/s between the greedy algorithm and the dynamic programming method with the help of the two versions of the knapsack problem.

Include in your discussion how the greedy-choice property holds true only for one of the knapsack problems.

The main difference between the dynamic programming method and greedy approach is following: dynamic programming will solve the problem after observing all the substructure of solution, but greedy approach will find the greedy choice for a problem, then problem can be break down to smaller problem, and the solution will appear.

Both of problem of knapsack problem can be solved by using dynamic programming by observing all the possible from sub-structure. However, $0-1$ knapsack problem cannot be solved by using greedy algorithm. To solve this, the limitation of $0-1$ is required. The thief cannot break the gold down such that break the gold ingot to small piece to get the maximum outcome or the item is limited to single one entity. If the thief can have 50 kg knapsack and only take several gold bars weigh 10 kg, 20kg, and 30kg, and they are worth 60, 100, and 120. what greedy algorithm will do is to find the maximum value per pound. Then value per pound for this problem can be 6, 5, and 4. So the possible solution is 20 + 30, 10 + 30, and 20 + 10, which they are worth 220, 180, and 160. However, based on what greedy algorithm do, the output of greedy algorithm will be 20 + 10 because 10 kg worth 6 dollar per pound and 20 kg worth 5 dollar per pound, which they are ascending order of maximum weight per pound. If this problem were fractional knapsack problem, then the problem can be solved by partitioning the gold bar into gold dust or piece of gold bar. Then they can be achieved by observing the weights per pound, and find the maximum profit that thief can steal.

5 Q5

Picking the interval with the lowest overlap won't solve the activity selection problem. As shown in the example figure below, "x" indicates that maximum overlapping occurs and blue marker indicates that the solution of this activity selection problem. Even though the optimal solution have to be 4 lines straight, but after getting rid of 2 in the middle, it's not going to be the optimal solution, which means that this method won't work for solving activity selection problem.

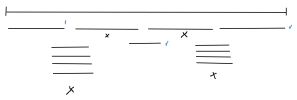

Figure 2: Example : The Lowest Overlap

6 Q6

Discuss how the activity selection problem exhibits the optimal substructure property. It did mentioned on the question 2.