CIS 675 (Spring 2019) Disclosure Sheet

Name: Venkata Sai Pawan Komaravolu

HW # 4

[✓ **Yes**]	**No**	Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?
Yes	[✓ **No**]	Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered **Yes** to one or more questions, please give the details here:

I've discussed solutions with Sanath Krishna

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

This disclosure sheet was based on one originally designed by Profs. Royer and Older.

DESIGN ANALYSIS OF ALGORITHMS

CIS 675

HOME WORK -4

VENKATA SAI PAWAN KOMARAVOLU

SUID:425210282

---

1. You are running an art museum. There is a long hallway with k paintings on the wall. The locations of the paintings are l1, ..., lk. These locations are real numbers, but not necessarily integers. You can place guards at locations in the hallway, and a guard can protect all paintings within 1 unit of distance from his location. The guards can be placed at any location, not just a location where there is a painting. Design a greedy algorithm to determine the minimum number of guards needed to protect all paintings.

Sol: We first sort the locations of the painting if not already sorted in the increasing order and we take first painting in the hall way and add a guard at l_i + 1 location(l_i is the location of first painting) as this is guard who is at l_i + 1 location covers painting at l_i and also furthest distance that guard can cover along with painting at l_i location. At this time the guard at l_i + 1 location covers painting at l_i location and any other painting at different location / locations within that distance of 1 unit. We also make a see how many paintings are not yet guarded. At every unprotected painting we place a guard at l_j + 1 location(l_j is the unprotected painting) and look for coverage of that particular guard who covers other paintings along with painting at location l_j . These steps are repeated until all the paintings are covered and in the final step we count the total number of guards .

2. Problem 2: Suppose you are given a set of jobs J1, ..., Jm to perform. Each job Ji will pay you some amount of money pi (not all jobs give the same payment). Each job Ji additionally has a deadline di , and if the job is not complete before its deadline, you will not receive any of the payment for that particular job. Assume that the timeline starts at time 0. Each job takes one unit of time to perform. For example, suppose that there are three jobs J1, J2, J3, with payments of 1, 2, 5 and deadlines of 3, 1, 1. Then the best schedule is to do job J3 from time 0 to time 1, and job J1 from time 1 to time 2 (or time 2 to time 3). Because the timeline starts at time 0 and each job takes 1 unit of time to complete, it is not possible to do both jobs J2 and J3 before their deadline at time 1. Then after time 1, J1 is the only job left, so we select it, and get a total payment of 6. Design a greedy algorithm to determine the schedule of jobs that will maximize the amount of payment that you receive. Your algorithm should output the actual schedule, not just the total payment (i.e., for each job, it should tell me when that job will start, or if it won't be scheduled at all, then it should say that).

Sol. We first sort the payment amount in the decreasing order . For the payment amount we see the corresponding job as it takes only 1 unit of space, we see the deadline corresponding to that job. We then schedule this job according to the deadline i.e, either right before the deadline or any other available time slot before the deadline. If the time slot is not available for that deadline then we leave this job and go on to the next highest paid job and there traverse all the jobs with payments and deadlines.

3. Problem 3: Suppose you have a large square lawn of grass. There are sprinklers at certain locations within the lawn. Each sprinkler i has coordinates (xi , yi). Each sprinkler covers a circle whose center is at the point the sprinkler is located. The sprinklers have different strengths, and each sprinkler i covers radius ri . In other words, sprinkler i covers the circle centered at (xi , yi) with radius ri . If a piece of the lawn is watered by more than one sprinkler, there is no additional benefit. You want to water as much of the lawn as possible, but your electrical system will only permit you to turn on k sprinklers. Design a

greedy algorithm to determine which k sprinklers to choose. Prove that the output of this algorithm is within a $(1 - 1/e)$ factor of optimal.

Sol: We first sort the radius in the decreasing order of their vules. The greater radius, greater will be the strength of the sprinkler. From the Decreasing sorted list of radius , we go through the list one by one . We first take first radius from the list and its corresponding points and sprinkler, we check if that sprinkler is sprinkling water which are with in the lawn dimensions and hence we pick that sprinkler.

We then go through the list and with the next biggest radius, we take corresponding points and sprinkler for that radius and check if the points for that particular radius is with in or on the circle of the circle formed by previous radius and points. If yes, then we ignore this points and move on to next radius from the list. If no, then we consider this sprinkler and go through the list .

We go on until all the sprinklers are visited.

If there are k sprinklers first k sprinklers are taken which are meeting the conditions/

This check is made and if the criteria is matched (the points i.e, the location of the sprinkler shouldn't be there in and on the circles of water formed by the previous selected sprinkler) that particular sprinkler is taken and we see all the locations and sprinklers are visited.

The proof that the solution is optimal is :

Claim: The above solution is optimal.

Proof: If the output of an algorithm is within a (1-1/e) factor of optimal, then the set cover function must be submodular and monotone.

We first prove the function is submodular. By definition, a function is submodular if it exhibits diminishing marginal returns.

Let $\Delta(x,S)$ be the marginal gain obtained by adding a new sprinkler x to solution set S.

Let f(S) be the solution set that contains all the sprinklers used to water the lawn,

f(S+x) be the solution set that contains all the sprinklers used to water the entire lawn along with the sprinkler x and

f(T) be another solution set that contains all the sprinklers used to water the entire lawn.

f is said to submodular if for all solution sets S and T, where $S \subseteq T$, and for all sprinklers x, $\Delta(x,S) \geq \Delta(x,T)$.

Here, S = One set of sprinklers placed at certain locations

T = Another set of sprinklers placed at other locations

Since S is a subset of T, by adding another sprinkler x to both S and T, the extra area of the lawn watered by T cannot be greater than the extra area watered by S.

Hence, $\Delta(x,S) \geq \Delta(x,T)$.

Hence the function is submodular.

Now, proving that the function is monotone.

A function is monotone if for all sets of solutions S and T where $S \subseteq T$, $f(S) \leq f(T)$.

Hence we add more number of sprinklers, we can water more area of lawn. It does not give us any additional benefit, but it does not make the result worse. So, we will not be losing any coverage by adding more sprinklers. Hence, the function is monotone.

Since the solution is both submodular and monotone, we can say that the output of this algorithm is within a (1-1/e) factor of optimal.

4. Problem 4: Suppose that you are traveling on a river by canoe. There are n canoe rental shops along the river. At each shop, you can pick up a canoe or drop off a canoe. The cost for renting a canoe from shop i to shop j is given by ci,j . There is no relationship between the different c values. You need to go from the first rental shop to the last rental shop. Design a dynamic programming algorithm to determine the minimum cost of such a trip. Hint: Let C[i] represent the cheapest way to get to rental shop i. How would your solution change if you also have to keep track of which shops you should rent from and drop off to in order to achieve the minimum cost trip? Hint: Define C[i] to be the cheapest way to get to rental shop i. If you stop at shop i, which rental shops could you have potentially stopped at immediately before you got to shop i?

Sol: Here we have to go from first rental shop to last rental shop. Here the problem is to determine the minimum cost of a trip to reach last shop i. For Dynamic Programming algorithm we have the subproblem as the final shop is i where we get the cheapest canoe price . We use DAG where each node is the shop at different locations in order and we draw an edge from each node I to each later node j and weights of the edges is the cost of renting from shop i to shop j which is $c_{i,j}$. Now we have nodes, weights of edges in a DAG. The solution to the problem is the shortest path from source to that node i in the DAG.

Without graph: We have shops and cost associated with each shop. We use cost(i) as the minimum cost required to get to the location i. We Initialize cost[1] = 0 , and from j = 2 to n in order , and then in each step

Cost[j] = min { cost[i] + $c_{i,j}$ i<j } , where min is taken only when i<j . Then we return cost[n] .

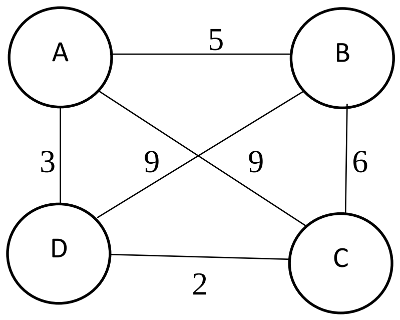
5. Problem 5: You have a set of coins. Each coin i has value vi . Your goal is to find a set of coins with total value exactly equal to V . You can use as many copies of each coin as you wish. Design a dynamic programming algorithm to solve this.

Sol: We can solve the above problem using knapsack problem . We first set each item I with value vi and weight as vi . The capacity of the knapsack is V , as the weight and value in the knapsack is same, we use , the maximum value which can get is V. Hence if run knapsack problem with repetitions , and see which items are selected to get to the total value V.

We cam solve it using knapsack , we can solve it using DAG , where each nodes takes values from 0 .. V and there will be an edge from a to b when there is a coin of value I such that a+ i = b and the edge value will be come i. Then we check if there is path from 0 to V , and returns the edges and its path.

Extra Credit: Suppose you are given an undirected, connected, weighted graph G. You want to find the heaviest weight set of edges that you can remove from G so that G is still connected. Propose an optimal greedy algorithm for this problem and prove that it is optimal. (Hint: use matroid theory!).

<u>Solution:</u> Let us consider a graph G to propose an optimal greedy algorithm.



The generic greedy algorithm is shown as follows. Let set E contains the elements/nodes of the graph G.

Let set I be the set of sets with the heaviest weight edges that can be removed and still keep the graph G connected. The algorithm is as follows: We Sort the elements of E in the descending order of their weights. Traverse the list in an order and add the elements as long as X is still in I. Hence, X = {{(A,C),(B,C)}, {(A,C),(B,D)}, {(B,D),(B,C)}, {A,C}, {B,D}, {B,C}, ϕ}.

Now showing that the algorithm is matroid.

To say that an algorithm is matroid, we must:

1. Define E and I.
   E here refers to the set of elements/nodes of graph G
   I here refers to the set of sets with the heaviest weight edges that can be removed and still keep G connected.
   I = {(A,C), (B,D), (B,C)}

2. Show that ϕ is in I.
   If we remove null element or no element, the graph G remains connected.

3. Show that I is closed under inclusion.
   Since the X is a subset of each element in I, then X is also in I.

4. Show that I satisfies exchange property.
   Exchange property states that if s and $s^{|}$ are elements of I, and s has fewer elements than $s^{|}$, then there is some element e in $s^{|}$ that is not in s such that s+e is in I.
   Let s = {(A,C),(B,D)} and $s^{|}$ = {(A,C), (B,D), (B,C)}
   Since s is a subset of $s^{|}$, adding the extra element (B,C) from $s^{|}$ to s, then s + (B,C) will still be in I.

Since the generic greedy algorithm is a matroid, we can say it is optimal using the matroid theory.