

COMP 250 - Homework #6

Due on April 11 2017, at 23:59.

Question 1. (25 points) Greedy and Dynamic Programming Algorithms

You enter a store with N dollars in your pockets. In the store there are k different types of objects for sale: let us call them O_0, O_1, \dots, O_{k-1} . Each type of object is available in infinite supply but is indivisible (i.e. you can't buy a fraction of an object). Each object O_i costs C_i dollars and has utility U_i . Both are integers. The utility of object O_i is the satisfaction you get by purchasing it. Your job is to design an algorithm to plan how to spend your money in order to maximize the total utility of the objects you purchase. Your algorithm should return an array $Q = Q_0 \dots Q_{k-1}$, where Q_i is a non-negative integer corresponding to the recommended number of objects of type i to purchase. The array Q must fit within your budget N , in other words

$$Q_0 * C_0 + Q_1 * C_1 + Q_2 * C_2 + \dots + Q_{k-1} * C_{k-1} \leq N$$

Your goal is to maximize the total utility of your purchase, where

$$\text{Utility}(Q_0 \dots Q_{k-1}) = Q_0 * U_0 + Q_1 * U_1 + Q_2 * U_2 + \dots + Q_{k-1} * U_{k-1}$$

Consider for example the situation where $k=4$, and

$C_0 = 2,$	$U_0 = 1$
$C_1 = 6,$	$U_1 = 5$
$C_2 = 8,$	$U_2 = 8$
$C_3 = 10,$	$U_3 = 9$

Then if $N=21$, an optimal choice is $Q_0=0, Q_1=0, Q_2=0, Q_3=2$, for a total cost of \$20 and a total utility of 18. Another optimal choice would be $Q_0=2, Q_1=0, Q_2=2, Q_3=0$.

If $N=26$, the optimal choice is $Q_0=1, Q_1=0, Q_2=3, Q_3=0$, for a total budget of \$26 and a total utility of 25.

- a) (8 points). Describe a greedy algorithm to try to solve the problem. Your algorithm will probably not be guaranteed to produce the optimal solution in all cases.

Algorithm GreedyChoice(int C[0...k-1], int U[0...k-1], int k, int N)

Input:

- An array C of costs for the k objects;
- An array U of utilities of the k objects;
- A total budget N.

Output: An array Q[0...k-1] of quantities to purchase.

- b) (4 points) Give an example of a situation (choice of C , U , and N) where your greedy algorithm fails to produce the optimal solution.
- c) (10 points) Write a dynamic programming algorithm to solve the problem. Your algorithm has to work for any choice of C , U , k , and N . First write the recursive formula for $Utility(N)$, the optimal utility that can be achieved within a maximum cost N . and then give the pseudocode for the dynamic programming algorithm.

Algorithm DynProgChoice(int $C[0\dots k-1]$, int $U[0\dots k-1]$, int k , int N)

Input:

- An array C of costs for the k objects;
- An array U of utilities of the k objects;
- A total budget N .

Output: An array $Q[0\dots k-1]$ of quantities to purchase.

- d) (3 points) Use your algorithm to calculate the optimal purchase for $N=38$

Question 2. (15 points) Sorting, one last time!

a) (10 points) Consider the following sorting problem: You are given an array of n integers. The integers are all between 1 and $2n$, but are not necessarily all distinct. Write an algorithm that sorts this array and that runs in time $O(n)$ in the worst case. (Hint: this is really easy once you see it. I could have said that the numbers are all between 1 and $10n$, or between 1 and $9394923n$ and it would make no difference, except that the constant hidden in the big-Oh notation would get larger.)

b) (5 points) Why can't your algorithm be used efficiently to sort an arbitrary set of integers?

Question 3. (20 points) Game trees

Consider the following two-player game that is a variant of the popular Nim game. The game starts with a stack of n matches. Players alternate in removing matches from the stack. Each player can remove from the stack either 2 or 3 matches (provided there are sufficiently many matches left). The player who removes the last match loses, except if there is only one match left, in which case the game is a draw. For example, if $n=4$, then the first player wins by removing two matches, leaving the opponent with no choice but to choose the two remaining matches. If $n=5$, then the first player wins by picking 3 matches. If $n=6$, then the game results in a draw.

a) (10 points) Draw the game tree for a game starting with $n=9$ matches. If both players play as well as possible, who will win?

b) (10 points) In general, if the game starts with n matches, who will win the game? Express your answer as a function of n .

Question 4. (40 points) Graph algorithms

For this question, assume a graph has n vertices numbered $0, 1, 2, \dots, n-1$.

In your algorithm, you can use the following graph ADT methods:

- *getNeighbors*(int i) returns the set of neighbors of vertex i . It is fine for you to write something like: for each vertex v in *getNeighbors*(17) do ...
- boolean *getVisited*(int i) returns TRUE if and only if vertex i has been marked as visited.
- *setVisited*(int i , boolean b) sets the visited status of vertex i to b .
- *getColor*(int i) returns the color of vertex i , either 0 or 1. If the color of vertex i has not been set previously, *getColor*(i) is undefined.
- *setColor*(int i , color c) sets the color of vertex i to c .

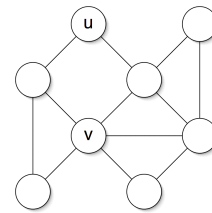
You may also want to associate to each vertex an integer called distance, which can be set and accessed through

- *setDistance*(vertex v , int d) sets the distance stored in v to d
- *getDistance*(vertex v) returns the distance stored in v . Undefined if that distance has not been set previously.

a) (20 points) In an undirected connected graph $G=(V,E)$, the distance $d(a,b)$ between vertices a and b is the number of edges in the shortest path between a and b . The eccentricity of a vertex a is defined as the largest shortest-path distance between vertex a and any other vertex:

$$\text{eccentricity}(a) = \max \{ d(a,b) : b \in V \}$$

For example, in the graph to the right, $\text{eccentricity}(u) = 3$ and $\text{eccentricity}(v) = 2$.



Problem: Write an algorithm to compute the eccentricity of a given vertex in a graph.

Algorithm *eccentricity*(vertex u)

Input: a vertex u from the graph

Output: the eccentricity of u

/* WRITE YOUR PSEUDOCODE HERE */

b) (20 points) Determining if an undirected graph can be colored with only three colors is an NP-complete problem. However, determining if it can be colored with only *two* colors is much easier. Write the pseudocode of an algorithm that determines if the vertices of a given connected undirected graph can be colored with only two colors (named 0 and 1)

so that no two adjacent vertices have the same color.

Algorithm is2colorable(vertex u)

Input: a graph vertex u

Output: true if the graph to which u belongs is 2-colorable, and false otherwise

/* WRITE YOUR PSEUDOCODE HERE */