CS 341, Winter 2015
Andrew Arnold, Timothy Chan, Doug Stinson

Assignment 3 (due Tuesday, March 3, 4pm)

Please read `https://www.student.cs.uwaterloo.ca/~cs341/policies.html` for general instructions. As well, please read the programming guidelines `https://www.student.cs.uwaterloo.ca/~cs341/prog.html`

Make sure that your section number and the first two letters of your surname appear on the cover page.

1. [*25 marks*] **Programming question.** Implement the divide-and-conquer algorithm from Assignment 2, Question 3 which runs in time $O(n \log n)$, as described in the model solutions. The input to your program is given in the following format:

$$n \; x_0 \; y_0 \; c_0 \; x_1 \; y_1 \; c_1 \; \ldots \; x_{n-1} \; y_{n-1} \; c_{n-1}$$

where the x_i, y_i are integers and $c_i = 0$ or 1 for $0 \leq i \leq n-1$. $c_i = 0$ indicates that the point (x_i, y_i) is red and $c_i = 1$ indicates that the point (x_i, y_i) is blue.

You may assume that no two x-coordinates are the same, and no two y-coordinates are the same. **You can also assume that the points are already pre-sorted and are given in increasing order of their x-coordinates.**

To simplify the output format, your program only needs to return the total number of pairs (r, b) such that r is a red point, b is a blue point, and r dominates b. The output to your program should thus be a single number (in a single line): k. Do not include any other information or text in the output.

   (a) [*20 marks*] Hand in a printed copy of your program and electronically submit the source file, named a3.cpp or a3.java.

   (b) [*5 marks*] Compare the performance (i.e., actual runtimes) of your divide-and-conquer algorithm with a naive $O(n^2)$ brute-force algorithm (which just computes k by checking all pairs (r, b)) on randomly generated input of various sizes. (Use the UNIX time method to measure actual runtimes; e.g., type /usr/bin/time -p your-command and watch for user time.) Describe (on paper) how your input is generated, present the results in a table or graph, and briefly comment on whether the experimental results agree with the theoretical analysis. (You do not need to submit the brute-force program or your input-generating program.)

2. [*21 marks*] The post office wants to locate mailboxes along a road. Suppose that houses exist at *kilometres* $x_1, x_2, \ldots, x_n$ on this road (these values can be assumed to be distinct integers that record the distance in metres from a specified origin). The objective is to minimize the number of mailboxes while ensuring that every house is no further than D kilometres from the nearest mailbox.

(a) [*6 marks*] Design a greedy algorithm that always finds an optimal solution for this problem. The algorithm should look at the houses one at a time, in some order, and choose the locations of the mailboxes in an intelligent (greedy) way. Give a clear pseudocode description of your algorithm and analyze its complexity.

(b) [*10 marks*] Prove that your algorithm is *correct*, i.e., that it always finds an optimal solution.

(c) [*5 marks*] Illustrate the execution of your algorithm (showing all your work) on the following problem instance:

house locations: $1, 2, 4, -1, -2, -5, 11, 5, 8, -8, -10, 14, 12, 19, 16$
maximum mailbox distance: $D = 3$.

Answer:

(a) Our idea is to to sort the houses from left to right, and add postboxes at the rightmost location y_i which covers the leftmost uncovered mailbox.

Greedy algorithm:

0. Sort and relabel x_i such that $x_1 < x_2 < \cdots < x_n$
1. $y_0 = -\infty$
2. $m = 0$
3. for $i = 1$ to n do
4.   if $x_i > y_m + D$ then
5. $y_{m+1} = x_i + D$
6. $m = m + 1$
7. Return $y_1, \ldots, y_m$

Analysis: Line 0 costs $\theta(n \log n)$ using mergesort. The remaining cost is dominiated by the for loop with n iterations each with cost $\theta(1)$. Thus we obtain a total cost $\theta(n \log n)$.

(b) We first argue the following claim.

Claim: Suppose the x_i are re-labelled such that $x_1 < \cdots < x_n$. Let $y_1^* < \cdots < y_\ell^*$ be an optimal solution and $y_1, \ldots, y_m$ be the greedy solution (note that $\ell \leq m$). Then $y_i^* \leq y_i$ for $1 \leq i \leq \ell$.

Proof. As x_1 is within D of some y_i^*, we must have $y_1^* \leq x_1 + D = y_1$.
Now suppose $y_i^* \leq y_i$, for some $i < \ell$. Let x_{j^*} be the least index such that $x_{j^*} > y_i^* + D$. j^* exists, or else $y_1^*, \ldots, y_i^*$ would cover all the mailboxes, contradicting the optimality of ℓ. We thus must have that $x_{j^*} + D \geq y_{i+1}^*$, or else we would have

$$y_i^* + D < x_{j^*} < y_{i+1}^* - D,$$

and x_{j^*} would not be within D of any mailbox. If we let j be the least index such that $x_j > y_i + D$, our greedy approach chooses $y_{i+1} = x_j + D$. As $y_i \geq y_i^*$, we must have that $x_j \geq x_{j^*}$, and $y_{i+1}^* \leq x_{j^*} + D \leq x_j + D = y_{i+1}$, proving the claim. □

Suppose that $\ell < m$, and that there exists a least index j such that x_j is not covered by $y_1, \ldots, y_\ell$. Then $x_{j+1} > y_\ell + D$. We then have by our claim that

$$y_\ell^* + D \leq y_\ell + D < x_{j+1},$$

contradicting the feasibility of y_ℓ^*. Thus we must have $\ell = m$, and $y_1, \ldots, y_m$ is optimal.

Alternative solution:

Again suppose the greedy solution has m mailboxes.

For $1 \leq i \leq m$, let z_i be the location of the first (i.e., leftmost) house that is distance $\leq D$ from the i-th mailbox. Define $Z = \{z_1, \ldots, z_m\}$. Observe that the distance between consecutive z_i's is greater than $2D$. Therefore we need at least m mailboxes so that every z_i is no more than D away from a mailbox.

(c) We sort the points to get $-10, -8, -5, -2, -1, 1, 2, 4, 5, 8, 11, 12, 14, 16, 19$. We first put a mailbox at $y_1 = -10 + 3 = -7$. The next uncovered house is at -2, so we put the mailbox at 1. Continuing in this fashion, we get a solution $y = (-7, 1, 8, 15, 22)$.

3. [*12 marks*] Consider the following version of the coin-changing problem: We have a fixed set of *coin denominations*, $d_1, d_2, \ldots, d_n$, such that

- $d_1 > \cdots > d_n = 1$, and
- d_{j-1} is divisible by d_j, for $2 \leq j \leq n$.

Now we are given a positive integer T, which is called the *target sum.* We want to find an n-tuple of non-negative integers, say $[x_1, \ldots, x_n]$, such that $T = \sum_{i=1}^n x_i d_i$ and such that $N = \sum_{i=1}^n x_i$ is minimized. (That is we want to make exact change for T cents, using the smallest number of coins, where the coins have the specified denominations. Here x_i denotes the number of coins of denomination d_i, $1 \leq i \leq n$.)

We analyze the Greedy Algorithm which looks at the coins in decreasing order, an always chooses the maximum possible number of coins of each value. This algorithm can be described in pseudocode as follows:

$N \leftarrow 0$
for $i \leftarrow 1$ *to* n *do*
    $x_i \leftarrow \left\lfloor \frac{T}{d_i} \right\rfloor$
    $T \leftarrow T - x_i d_i$
    $N \leftarrow N + x_i$
return$([x_1, \ldots, x_n], N)$

The goal of this question is to prove that this Greedy Algorithm always find the optimal solution for any target sum T. Suppose the greedy solution is $X = [x_1, \ldots, x_n]$ and $X^* = [x_1^*, \ldots, x_n^*]$ is any optimal solution.

(a) [*3 marks*] For $2 \leq i \leq n$, prove that $0 \leq x_i \leq d_{i-1}/d_i - 1$.

(b) [*3 marks*] For $2 \leq i \leq n$, prove that $0 \leq x_i^* \leq d_{i-1}/d_i - 1$.

(c) [*6 marks*] Suppose that $X \neq X^*$. Let i be the last (i.e., highest) index such that $x_i^* \neq x_i$. Using the results proven in (a) and (b), derive a contradiction, thus proving that $X = X^*$.

Answer:

(a) Let T_i be the value of T at the start of the i-th iteration. Observe that, for $i > 1$, that $T_i = T_{i-1} - d_{i-1}x_{i-1}$. Now, as $x_{i-1} = \lfloor T_{i-1}/d_{i-1} \rfloor > T_{i-1}/d_{i-1} - 1$, it follows that $d_{i-1}x_{i-1} > T_{i-1} - d_{i-1}$, such that $T_i < d_{i-1}$. We thus have that $x_i = \lfloor T_i/d_i \rfloor < d_{i-1}/d_i$. As d_i divides d_{i-1}, we have $x_i \leq d_{i-1}/d_i - 1$.

(b) Towards a contradiction, suppose $x_i^* \geq d_{i-1}/d_i$. Then modify the solution by reducing x_i^* by d_{i-1}/d_i, and incrementing x_{i-1}^* by 1. This still gives a solution as the net change in total value is $d_{i-1} - (d_{i-1}/d_i)d_i = 0$, but the number of coins is reduced by $1 - d_{i-1}/d_i > 0$. This contradicts the optimality of X^*.

(c) Note: Essentially we are arguing using modular arithmetic. We did not use "mod" in case any students are unfamiliar with that notation.

Suppose first that i exists. i cannot be 1, else

$$\left(\sum_{j=1}^{n} x_j^* d_j\right) - \left(\sum_{j=1}^{n} x_j d_j\right) = (x_1^* - x_1)d_1 \neq 0.$$

Suppose then that $i > 1$. We then have that $0 \leq x_i^* \neq x_i < d_{i-1}/d_i$. Let $\delta = (x_i^* - x_i)d_i$. Then we have $0 < |\delta| < d_{i-1}$. Note that

$$\sum_{j=1}^{i-1} (x_j^* - x_j)d_j,$$

is a multiple of d_{i-1}, i.e. cd_{i-1} for some c. Thus

$$\left(\sum_{j=1}^{n} x_j^* d_j\right) - \left(\sum_{j=1}^{n} x_j d_j\right) = cd_{i-1} + \delta,$$

this value is *not* a multiple of d_{i-1}, and in particular, is not 0. Thus i does not exist and $X = X^*$. This contradicts that X and X^* are both solutions to the coin changing problem.

4. [*12 marks*] Suppose we are given n intervals, say $I_j = [s_j, f_j)$ for $1 \leq j \leq n$, and for each I_j we have a profit $p_j > 0$. A *feasible solution* is a subset of *pairwise disjoint* intervals. The *optimal solution* is a feasible solution such that the *sum of the profits* of the selected intervals is maximized.

Suppose that the intervals are pre-sorted by finishing time, i.e., $f_1 \leq \cdots \leq f_n$. For each j, $1 \leq j \leq n$, let $\mathsf{last}(j) = \max\{i : f_i \leq s_j\}$. If $f_i > s_j$ for all i, define $\mathsf{last}(j) = 0$.

Also, let $P(j)$ denote the maximum achievable profit for the subproblem consisting of the first j intervals. Thus, $P(n)$ is the solution for the given problem instance. You may find it useful to define $P(0) = 0$.

(a) [*2 marks*] It is easy to compute all n values $\mathsf{last}(1), \ldots, \mathsf{last}(n)$ in time $O(n^2)$. Give a brief, high-level description of a method to compute these n values in time $O(n \log n)$.

(b) [*4 marks*] Give a recurrence relation that can be used to compute the values $P(j)$, $j = 1, \ldots, n$. This part of the algorithm should take time $\Theta(n)$. In general, $P(j)$ will depend on two previous P-values. The $\mathsf{last}(j)$ values will be used in the recurrence relation, and you need to handle the situation where $\mathsf{last}(j) = 0$ correctly. You should give a brief explanation justifying the correctness of your recurrence relation.

(c) [*6 marks*] Solve the following problem instance by dynamic programming, using the recurrence relation that you presented in part (b). Give the values $\mathsf{last}(1), \ldots, \mathsf{last}(n)$ as well as $P(1), \ldots, P(n)$ and show all the steps carried out to perform these computations. Finally, determine the set of intervals that yields the maximum profit $P(n)$.

interval	$[s_j, f_j)$	p_j
I_1	$[2, 3)$	2
I_2	$[2, 4)$	3
I_3	$[1, 6)$	5
I_4	$[3, 8)$	4
I_5	$[4, 9)$	4
I_6	$[6, 11)$	1
I_7	$[5, 12)$	3
I_8	$[12, 13)$	1
I_9	$[9, 14)$	3
I_{10}	$[11, 15)$	2
I_{11}	$[14, 16)$	2
I_{12}	$[13, 18)$	4

Answer:

(a) Sort the s_j all in order of increasing time in $\mathcal{O}(n \log n)$ operations, keeping track of the indices j. We can then simultaneously scan through the f_i and s_j, keeping track of the index of the right-most f_i left of each s_j to give us $\mathsf{last}(j)$.

(b) We claim that

$$P(j) = \max(P(j-1), p_j + P(\mathsf{last}(j))),$$

where $P(0) = 0$. To see this, note that an optimal solution for $\{I_1, \ldots, I_j\}$ can either contain or not contain I_j. If the optimal solution does not contain I_j, then it is an optimal solution for $\{I_1, \ldots, I_{j-1}\}$, i.e., it gives profit $P(j-1)$. If it does contain I_j, then the remaining intervals that comprise the optimal solution $P(j)$ give an optimal solution for the subset of intervals in $\{I_1, \ldots, I_{j-1}\}$ not intersecting I_j. These are exactly $\{I_1, \ldots, I_{\mathsf{last}(j)}\}$, which have optimal solution $P(\mathsf{last}(j))$.

(c) For this example $\mathsf{last}(j)$ may be computed by inspection. We complete the table, with the optimal solution in **bolded red**:

interval	$[s_j, f_j)$	p_j	$\mathsf{last}(j)$	$p_j + P(\mathsf{last}(j))$	P_{j-1}
I_1	$[2, 3)$	2	0	**2**	0
I_2	$[2, 4)$	3	0	**3**	2
I_3	$[1, 6)$	5	0	**5**	3
I_4	$[3, 8)$	4	1	**6**	5
I_5	$[4, 9)$	4	2	**7**	6
I_6	$[6, 11)$	1	3	6	**7**
I_7	$[5, 12)$	3	2	6	**7**
I_8	$[12, 13)$	1	7	**8**	7
I_9	$[9, 14)$	3	5	**10**	8
I_{10}	$[11, 15)$	2	6	9	**10**
I_{11}	$[14, 16)$	2	9	**12**	10
I_{12}	$[13, 18)$	4	8	**12**	**12**

The optimal value is 12. There are two optimal solutions with intervals indexed by $\{2, 5, 8, 12\}$ or $\{2, 5, 9, 11\}$, which can be obtained by backtracking through the table above.

Namely, if the optimal value in the j-th row is in the last column, then the optimal solution giving $P(j)$ comprises I_j and the optimal solution giving $P(\mathsf{last}(j))$, otherwise, it is the optimal solution giving $P(j-1)$.