CS31005 Algorithms – II

Class Test 3

Date: 18-Nov-2020 Maximum marks: 50

Instructions

- **Answer all of Q1–Q4, and only one of Q5 and Q6.** Be brief and precise.
- If you use any algorithm/result/formula covered in the lectures/tutorials, just mention it, do not elaborate.

---

1. Consider the following instance of the set-cover problem, where X is the universal set, and S_i are the subsets of X.

$$X = \{1,2,3,4,5,6,7,8,9,10\},$$

$$\begin{aligned} S_1 &= \{3,8,9\}, \\ S_2 &= \{1,4,8,10\}, \\ S_3 &= \{2,8,9,10\}, \\ S_4 &= \{5,6,10\}, \\ S_5 &= \{3,7,9\}, \\ S_6 &= \{1,3,5,7,9\}, \\ S_7 &= \{2,4,5\}, \\ S_8 &= \{1,2,4,6\}. \end{aligned}$$

You run the greedy set-cover algorithm (as taught in the class) on this instance. Prove/Disprove: The algorithm produces an optimal solution for this instance. **(8)**

Solution *True.* In the first iteration, the greedy algorithm chooses the set S_6, and updates the remaining sets as

$$X = \{2,4,6,8,10\},$$

$$\begin{aligned} S_1 &= \{8\}, \\ S_2 &= \{4,8,10\}, \\ S_3 &= \{2,8,10\}, \\ S_4 &= \{6,10\}, \\ S_7 &= \{2,4\}, \\ S_8 &= \{2,4,6\}. \end{aligned}$$

Now, there are three contenders S_2, S_3, S_8. Let us see what happens if we include each of these in the cover. In each case, only the non-empty sets that remain are shown.

S_2		S_3		S_8	
$X = \{2,6\}$,	$S_3 = \{2\}$,	$X = \{4,6\}$,	$S_2 = \{4\}$,	$X = \{8,10\}$,	$S_1 = \{8\}$,
	$S_4 = \{6\}$,		$S_4 = \{6\}$,		$S_2 = \{8,10\}$,
	$S_7 = \{2\}$,		$S_7 = \{4\}$,		$S_3 = \{8,10\}$,
	$S_8 = \{2,6\}$.		$S_8 = \{4,6\}$.		$S_4 = \{10\}$.

In each case, we need to select only one more set. Thus the greedy algorithm always produces a set cover with three subsets for this instance. On the other hand, simple size arguments reveal that X cannot be covered by only two of the given subsets.

2. Consider a Bloom Filter with 15 bits for storing positive integers, with all bits set to 0 initially. The filter uses the following three hash functions to hash an integer key x: $h_1(x) = x \bmod m$, $h_2(x) = (4x+3) \bmod m$, and $h_3(x) = (5x+4) \bmod m$, where m is the size of the filter.

(a) Choose four integers as follows: one integer randomly from 10 to 25, one integer randomly from 26 to 40, one integer randomly from 41 to 55, and one integer randomly from 56 to 75. Make sure that the four integers you choose are distinct modulo 15. Insert the integers in the bloom filter. Your answer should first show the set of integers chosen in ascending order. Then, for each insertion, show the values of the hash functions and the bloom filter after the insertion. Do not give any explanation or write anything else. **(6)**

(b) Now, find one integer x such that $h_1(x), h_2(x), h_3(x)$ are not all the same as for any of the four integers inserted (that is, for each of these four integers y, at most two of the following equalities may hold: $h_1(x) = h_1(y)$, $h_2(x) = h_2(y)$, and $h_3(x) = h_3(y)$) such that searching for x will cause a false positive. Show the integer x chosen by you as a false positive and the hash function values for it. Do not give any explanation or write anything else. **(4)**

3. You are given a stream of songs lasting for $t_1, t_2, t_3, \ldots, t_n$ seconds. You have two write-once devices D_1 and D_2, each capable of storing songs of total duration T seconds. When the i-th song starts, you have three options: (i) copy the song to D_1, (ii) copy the song to D_2, and (iii) discard the song. The copy of a song to a device is allowed only when there is enough memory left in that device. Assume that the individual song durations t_i are known to you beforehand, and these durations satisfy $t_i \leqslant T$ for all i, and $\sum_{i=1}^{n} t_i \leqslant 2T$. Your goal is to maximize the total copy time in the two output devices together. To that effect, you run the following greedy algorithm. Derive a tight approximation ratio of the algorithm. Prove that the approximation ratio is tight. **(6 + 4)**

```
for $i = 1, 2, 3, \ldots, n$ (in that order) {
    If $D_1$ can accommodate the $i$-th song, copy the $i$-th song to $D_1$,
    else if $D_2$ can accommodate the $i$-th song, copy the $i$-th song to $D_2$,
    else discard the $i$-th song, and continue.
}
```

Solution If all the songs can be copied to D_1 and D_2, we have obtained an optimal solution. So assume that some song(s) can be copied to neither D_1 nor D_2. Let the k-th song be the first such song. Under the assumption $t_k \leqslant T$, this implies that both D_1 and D_2 store some songs at the time when the k-th song comes. This implies that the first $k-1$ songs could not be copied to D_1 alone because of its capacity constraint, that is, the total length of the songs copied so far is already $> T$. Since the two devices can store a maximum of $2T$ seconds of songs, the approximation ratio is $> \frac{1}{2}$.

In order to see that this approximation ratio is tight, start with any $\varepsilon > 0$. Take $T \geqslant \frac{1}{\varepsilon}$. Consider the four songs of durations $1, \frac{T}{2}, \frac{T}{2}, T-1$. The greedy algorithm copies the first two songs to D_1 and the third song to D_2, and discards the last song. All the songs can be copied to the devices (we have $1 + (T-1) = \frac{T}{2} + \frac{T}{2} = T$), so the approximation ratio is $\frac{T+1}{2T} = \frac{1}{2} + \frac{1}{T} \leqslant \frac{1}{2} + \varepsilon$.

4. You have only two weeks left to solve the pending assignments of Algorithms–II. Let there be n such assignments with the i-th assignment demanding time t_i. Assume that you can solve all the assignments in one week, but since you have other courses too, you would like to distribute the Algorithms–II assignments in the available two weeks as evenly as possible. That is, you want to partition the assignments into two subsets A, B such that $\left| \sum_{a \in A} t_a - \sum_{b \in B} t_b \right|$ is minimized. Prove that this problem cannot have any polynomial-time approximation algorithm (unless $\mathrm{P} = \mathrm{NP}$). **(7)**

Solution Assume that there exists a polynomial-time ρ-approximation algorithm $\mathscr{A}$ for the given problem. Using $\mathscr{A}$, we can solve the partition problem in polynomial time as follows. Let $S = \{a_1, a_2, \ldots, a_n\}$ be an instance of the partition problem with $\sum_{i=1}^{n} a_i = 2T$. We pass S to $\mathscr{A}$. If the partition problem on the given instance has a solution, we have $\mathrm{OPT} = 0$ on the converted instance. Since $\mathscr{A}$ is a ρ-approximation algorithm, it outputs a partition A, B of S with load imbalance $\leqslant \rho \times \mathrm{OPT} = 0$. Since the load imbalance cannot be negative, it must be equal to 0. On the other hand, if the partition problem has no solution for the given instance, $\mathrm{OPT} > 0$, so $\mathscr{A}$ returns a suboptimal solution with load imbalance > 0 as well.

Answer either Q5 or Q6. Do <u>not</u> answer *both* Q5 and Q6. In case you do, we will grade *only* the first one we see, and ignore the other.

5. Consider an experiment in which two people are each asked to choose n numbers one by one independently and randomly from a very large range of positive integers. The numbers chosen need not be distinct, so a person can choose the same number more than once also. At the end, we want to find out if they have chosen the same collection of numbers (the <u>same set of distinct numbers</u>, each repeated the <u>same number of times</u>) or not. You are asked to design a Monte Carlo algorithm that outputs YES if they have chosen the same set of numbers, NO otherwise. The NO answer must always be correct, but the YES answer may be wrong with a probability of at most 1/2. Your algorithm should run in $O(n)$ time and $O(n)$ space.

Your solution must use hashing in some form. You do not need to specify any hash function explicitly, just explain how your algorithm will work using hashing.

Your answer should have:

(a) The pseudocode for the algorithm. **(8)**

Solution Let A and B be two hash tables of size $2n$ each, initialized to 0.
Let h be a random hash function.
For each number x of person 1, set $A[h(x)] = A[h(x)] + 1$.
For each number y of person 2, set $B[h(y)] = B[h(y)] + 1$.

```
For $i = 1$ to $2n$
    If $A[i] \neq B[i]$, return NO.
Return YES.
```

[You can also do with one hash table with increment for Person 1 and decrement for Person 2. Check for all entries = 0 at the end]

(b) A proof that the error probability is at most 1/2. (State any assumptions you make, if any.) **(7)**

Solution If the two sets of numbers are different, there exists at least one number p chosen by Person 1 such that p is chosen a different number of times by Person 2 (including 0 times when Person 2 has not chosen p). Suppose $h(p) = q$. Since the answer is YES, $A[q] = B[q]$. Since p is chosen a different number of times by Person 2, there must exist at least one number $r \neq p$ that hashes to the same index q to make $A[q] = B[q]$. Now, the expected number of numbers hashing to any one entry of the table is $n/(2n) = 1/2$. Thus, the probability that a number different than p hashes to the same index as p is less than $1/2$. Hence, with probability at least $1/2$, no other number hashes to the same index as p. In that case, there is no chance of $A[q] = B[q]$, as the difference in the number of occurrences of p in the two sets cannot be canceled out by any other number. Hence, the difference will be detected with probability at least $1/2$. Hence the probability of a wrong YES answer (the difference is not detected) is at most $1/2$.

6. Consider the following variation of the traveling salesperson problem on a complete undirected weighted graph of n nodes with positive edge weights: The salesperson starts from a particular city x, and travels to a particular city y going through every other city exactly once. There is no requirement of coming back to city x. The goal is to find a path such that the maximum edge weight on the path is the minimum among all such paths. You are asked to design a branch-and-bound algorithm for this problem with DFS order of generating the state-space tree.

(a) Show the structure of the solution (one sentence only). **(2)**

Solution $\langle x = x_1, x_2, x_3, \ldots, x_{n-1}, x_n = y\rangle$, where each x_i is a city in the path.

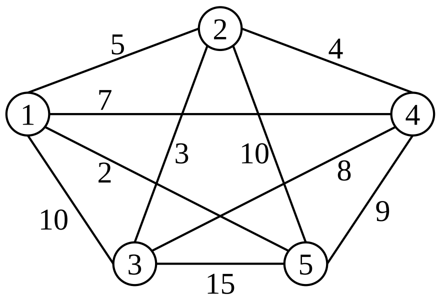
(b) Show the bounding function you will use to prune the state-space tree. The bounding function will be used as follows at a node u: Before generating any child v of u, u will apply the bounding function on the state represented by v; v will be generated if and only if the bounding function returns true, otherwise v will not be generated. Your bounding function must prune nodes leading to both infeasible solutions, and feasible solutions that are not optimal. Write the bounding function clearly describing any notations/terms that you use. Do not explain the working of the bounding function. **(7)**

Solution Suppose you are applying the bounding function to a node corresponding to the partial solution $\langle x = x_1, x_2, x_3, \ldots, x_i\rangle$.

Pruning infeasible solutions: Prune if x_i is a city already in the path, or if $x_i = y$ but all other cities are not covered.
    If $(x_i = x_j)$ for some $1 \leqslant j \leqslant i-1$ OR if $(x_i = y$ and $i \neq n)$, then return false.

Pruning feasible but non-optimal solutions: If $F(s_i) + v_{lower}(i+1) > v_{upper}$, then return false, where $F(s_i) = \max\{w(x_j, x_{j+1}) \mid 1 \leqslant j \leqslant i-1\}$ is the maximum weighted edge in the partial path seen so far. Let $W = \min\{w(x_i, x_j) \mid x_j \notin \{x_1, x_2, \ldots, x_i\}\}$ be the minimum-weight edge from x_i to nodes not in the partial path. We take $v_{lower}(i+1) = W - F(s_i)$ if $W > F(s_i)$ (this will be minimum increase in max weight edge), 0 otherwise (the current max-weight edge stays as max-weight). Finally, v_{upper} is the max weight of any edge in the graph initially. As each full solution S is found, update v_{upper} as the minimum of current value of v_{upper} and the weight of the maximum-weight edge in S.

(c) Consider the following graph, with the numbers inside the nodes indicating the IDs of the nodes, and the numbers on the edges showing the weights of the edges. Show the state-space tree generated by applying the branch-and-bound algorithm to this graph, with node 2 as the starting city, node 5 as the final destination city, and neighboring nodes explored in increasing order of their IDs wherever required (strictly follow this order). You should only show the nodes generated (strictly as per the method of generating nodes mentioned above). Do not show any node not generated. **(6)**


2
1
4
3
5
5
4
7
3
10
2
8
10
9
15