CS601 Algorithms and Complexity **2021 Jul-Nov**

Endsem Solution

Total Marks: 60

Part 1 Que 1. We have n TAs and k courses. For each TA, we are given a list of suitable courses. A TA is either a PhD or an Mtech student. Each course has a specific requirement, which can be in one of the following forms:

- at least d PhD TAs and at least b Mtech TAs.
- at least c TAs in total, out of which at least b should be from PhD
- at least b TAs.

We want to check whether it is possible to fulfill requirements of every course and if it is possible then find a TA allocation. Show that there is a polynomial time algorithm for this problem. One way to do that is to convert any instance of this problem into an instance of bipartite matching and use the bipartite matching algorithm (no need to describe the bipartite matching algorithm).

Answer. First, for any course requiring b TAs in total, we will create b TA positions required to be fulfilled. The idea is to create a bipartite graph where there will be a vertex for each TA and also a vertex for each TA position. If a TA X is eligible for a TA position Y, then we add an edge between X and Y. Let's describe the edges of the graph in more detail. We will consider the three given forms in which a course can give its requirement.

- Suppose a course requires at least d PhD TAs and at least b Mtech TAs. We will create $d+b$ vertices corresponding to these TA positions. Naturally, each of the d PhD TA positions will be connected to all PhD TAs who are eligible for this course. Similarly, each of the b Mtech TA positions will be connected to all Mtech TAs who are eligible for this course.
- Suppose a course requires at least c TAs in total, out of which at least b should be from PhD. Create b vertices for PhD TA positions and connect each of them to all the PhD TAs who are eligible for this course. Then create $c-b$ vertices for general TA positions and connect each of them to both PhD and Mtech TAs who are eligible for this course.
- Suppose a course requires at least b TAs. Then simply create b vertices for general TA positions and connect each of them to both PhD and Mtech TAs who are eligible for this course.

Clearly this graph would be bipartite. Any matching in this graph will give a valid TA allocation because a TA will get assigned to at most one TA position and a TA position will get at most one TA. Similarly, any valid TA allocation will correspond to a matching in the above graph. Thus, to see whether it is possible to fulfill all the TA requirements, we simply need to compute a maximum matching in the above graph and check whether all vertices for TA positions have been matched. Since maximum matching can be found in polynomial time, a desired TA allocation can be found in polynomial time.

Part 1 Que 2. This problem has multiple parts. Even if you are not able to solve initial parts, you can assume them and solve the later parts.

Consider the problem of finding the kth smallest number from a given set of distinct numbers. One can do this easily in $O(n \log n)$ time via sorting. We want to find an $O(n)$ time algorithm. Let us propose the following randomized algorithm, called quickselect. It takes as input an array of integers A , and a number k.

Quickselect(A, k):
Select a pivot element uniformly randomly from A, say it is p.
Compare p with every number in A and generate two arrays:
L : numbers in A which are less than p.
Let $|L| = \ell$.
G : numbers in A which are greater than p.
If $\ell = k - 1$ then return p.
If $\ell < k - 1$ then
return Quickselect$(G, k - \ell - 1)$.
If $\ell > k - 1$ then
return Quickselect(L, k).

We want to show that the expected number of comparisons in the Quickselect algorithm will be $O(n)$. For any two indices $i < j$, consider the ith smallest element a_i and the jth smallest element a_j in the array. We want compute the probability that a_i and a_j will ever be compared in the algorithm. Note that this comparison can happen only when one of a_i and a_j will be selected as pivot. Consider three cases:

1. $i < j < k$: Prove that the probability that a_i and a_j will ever be compared in the algorithm is $2/(k - i + 1)$. You can prove this inductively. Note that the probability is independent of n [3 marks].

2. $i \leq k \leq j$: Prove that the probability that a_i and a_j will ever be compared in the algorithm is $2/(j - i + 1)$. You can prove this inductively. Note that the probability is independent of n [3 marks].

3. $k < i < j$: This is similar to the first case. The probability in this case will be $2/(j - k + 1)$ by similar arguments. No need to prove anything.

Using the probabilities in above three cases show that the expected number of total comparisons will be $O(n)$ [4 marks]. You may need to use linearity of expectation.

Answer. As mentioned in the question we will prove the probabilities using induction based on the size of the array. Note that a_i and a_j can possibly be compared only when one of them is chosen as the pivot element. Moreover, when we pick something as a pivot, it will be thrown out and will not be compared with anything in the later rounds. Hence, a pair of elements can only be compared once in the algorithm.

No marks will be deducted if the base case of the induction is not explicitly discussed. The main argument is the induction step.

Case 1 ($i < j < k$): Recall that each of the n elements of the array are equally probable to be the pivot, i.e., they have probability $1/n$ of selected as pivot p. Let's consider various scenarios for the choice of p.

If $p = a_i$ or $p = a_j$, then clearly a_i and a_j will be compared with each other as the pivot is compared with every other element. Clearly, this will happen with probability $2/n$.

If $a_i < p < a_j < a_k$, then in the next recursive call only elements larger than p will be considered, i.e., a_i will be thrown out. Hence a_i and a_j can never be compared with each other. Same conclusion can be drawn when $a_i < a_j < p < a_k$. The total probability for these events will be $(k - i - 2)/n$.

If $a_k < p$, then we will go into the recursive call Quickselect(L, k). The probability for this kind of pivot is $(n - k)/n$. L will contain both a_i and a_j and hence, there is a possibility that they can be compared later on. By induction hypothesis, the probability that a_i and a_j will be compared in the recursive call

Quickselect(L, k) is $2/(k-i+1)$. Note that a_i still has rank i in L and a_k still has rank k in L, hence, the same probability expression $2/(k-i+1)$ is valid.

If $p < a_i$, then we will go into the recursive call Quickselect$(G, k-\ell-1\ )$. The probability for this kind of pivot is $(i-1)/n$. G will contain both a_i and a_j and hence, again there is a possibility that they can be compared later on. By induction hypothesis, the probability that a_i and a_j will be compared in the recursive call Quickselect$(G, k-\ell-1)$ is $2/(k-i+1)$. Note that the new rank of a_i in G will be $i-\ell-1$ and the new rank of a_k in G will be $k-\ell-1$. We can write $(k-\ell-1)-(i-\ell-1)+1 = k-i+1$, hence, the same probability expression $2/(k-i+1)$ is valid.

Putting all these cases together, we get the probability that a_i and a_j will be compared

$$\begin{aligned}
&= \frac{2}{n} + \frac{n-k}{n} \times \frac{2}{k-i+1} + \frac{i-1}{n} \times \frac{2}{k-i+1} \\
&= \frac{2}{n} + \frac{n-k+i-1}{n} \times \frac{2}{k-i+1} \\
&= \frac{2}{n}\left(1 + \frac{n-k+i-1}{k-i+1}\right) \\
&= \frac{2}{n}\left(\frac{n}{k-i+1}\right) \\
&= \frac{2}{k-i+1}
\end{aligned}$$

For the base case of induction, we need to consider the call after which no further recursive call with a_i and a_j present is possible. This will be when $n = k$ and $i = 1$. In this call, a_i and a_j will be compared precisely when the current pivot is a_i or a_j. Hence, the probability is $2/n = 2/k = 2/(k-i+1)$.

Case 2 ($i \le k \le j$): Let's again consider various scenarios for the choice of p.

If $p = a_i$ or $p = a_j$, then clearly a_i and a_j will be compared with each other. Clearly, this will happen with probability $2/n$.

If $a_i < p < a_j$, then in the next recursive call, definitely either a_i or a_j will be thrown out. Hence a_i and a_j will never be compared with each other. The probability for this kind pivot will be $(j-i-1)/n$.

If $a_j < p$, then we will go into the recursive call Quickselect$(L, k\ )$. The probability for this kind of pivot is $(n-j)/n$. L will contain both a_i and a_j and hence, there is a possibility that they can be compared later on. By induction hypothesis, the probability that a_i and a_j will be compared in the recursive call Quickselect(L, k) is $2/(j-i+1)$. Note that a_i and a_j will still have ranks i and j in L, hence, the same probability expression $2/(j-i+1)$ is valid.

If $p < a_i$, then we will go into the recursive call Quickselect$(G, k-\ell-1\ )$. The probability for this kind of pivot is $(i-1)/n$. G will contain both a_i and a_j and hence, again there is a possibility that they can be compared later on. By induction hypothesis, the probability that a_i and a_j will be compared in the recursive call Quickselect$(G, k-\ell-1)$ is $2/(j-i+1)$. Note that the new rank of a_i in G will be $i-\ell-1$ and the new rank of a_j in G will be $j-\ell-1$. We can write $(k-\ell-1)-(i-\ell-1)+1 = k-i+1$, hence, the same probability expression $2/(j-i+1)$ is valid.

Putting all these cases together, we get the probability that a_i and a_j will be compared

$$\begin{aligned}
&= \frac{2}{n} + \frac{n-j}{n} \times \frac{2}{j-i+1} + \frac{i-1}{n} \times \frac{2}{j-i+1} \\
&= \frac{2}{n} + \frac{n-j+i-1}{n} \times \frac{2}{j-i+1} \\
&= \frac{2}{n}\left(1 + \frac{n-j+i-1}{j-i+1}\right) \\
&= \frac{2}{n}\left(\frac{n}{j-i+1}\right) \\
&= \frac{2}{j-i+1}
\end{aligned}$$

For the base case of induction, we need to consider the call after which no further recursive call with a_i

and a_j present is possible. This will be when $n = j$ and $i = 1$. In this call, a_i and a_j will be compared precisely when the current pivot is a_i or a_j. Hence, the probability is $2/n = 2/j = 2/(j - i + 1)$.

Case 3 ($k < i < j$) : This is similar to the first case. The probability in this case will be $2/(j - k + 1)$ by similar arguments.

Expected number of total comparisons. To find the expected number of total comparisons we will use linearity of expectation. Let's define a random variable $X_{i,j}$ which is 1 if a_i and a_j are compared with each other and 0 otherwise. We can write

$$\mathbb{E}[X_{i,j}] = 1 \times \Pr(X_{i,j} = 1) + 0 \times \Pr(X_{i,j} = 0) = \Pr(X_{i,j} = 1) = \Pr(a_i \text{ and } a_j \text{ are compared}).$$

Let X be the total number of comparisons. Clearly,

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}.$$

Using linearity of expectation,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{i,j}] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr(a_i \text{ and } a_j \text{ are compared}).$$

Let's fix k and break this sum into three parts: $i < j < k$, $i \leq k \leq j$, and $k < i < j$. We will use the value of $\Pr(a_i \text{ and } a_j \text{ are compared})$ as derived above for the three cases.

Case 1 ($i < j < k$):

$$\begin{aligned}
\sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \Pr(a_i \text{ and } a_j \text{ are compared}) &= \sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} 2/(k - i + 1) \\
&= \sum_{i=1}^{k-2} (k - i - 1) \times 2/(k - i + 1) \\
&\leq \sum_{i=1}^{k-2} 2 \\
&= 2(k - 2).
\end{aligned}$$

Case 2 ($i \leq k \leq j$):

$$\sum_{i=1}^{k} \sum_{j=k}^{n} \Pr(a_i \text{ and } a_j \text{ are compared}) = \sum_{i=1}^{k} \sum_{j=k}^{n} 2/(j - i + 1)$$

There are $n - 1$ possible values for the difference $j - i$. If we fix the difference of j and i to a certain value h, then the maximum possible value of j can be $k + h$, while the minimum possible value of j is k. Hence, the number of such pairs (i, j) is at most $h + 1$. Thus, we can write,

$$\begin{aligned}
\sum_{i=1}^{k} \sum_{j=k}^{n} 2/(j - i + 1) &\leq \sum_{h=1}^{n-1} (h + 1) \times 2/(h + 1) \\
&= 2(n - 1).
\end{aligned}$$

Case 3 ($k < i < j$):

$$\begin{aligned}
\sum_{i=k+1}^{n-1} \sum_{j=i+1}^{n} \Pr(a_i \text{ and } a_j \text{ are compared}) &= \sum_{j=k+2}^{n} \sum_{i=k+1}^{j-1} \Pr(a_i \text{ and } a_j \text{ are compared}) \\
&= \sum_{j=k+2}^{n} \sum_{i=k+1}^{j-1} 2/(j-k+1) \\
&= \sum_{j=k+2}^{n} (j-1-k) \times 2/(j-k+1) \\
&\leq \sum_{j=k+2}^{n} 2 \\
&\leq 2(n-k-1).
\end{aligned}$$

Adding the three cases together we get that the expected number of total comparisons is at most $4n$.

Part 1 Que 3. For two binary strings A and B, another string C is called their superposition if C can be partitioned into two subsequences such that one of the subsequences is A and the other is B. For example, for A = 011010 and B = 10010, the string C=01010101100 is their superposition. You can see that in the string C=01010101100, bold letters give you A and light letters give you B.

There are two sources which are generating some kind of repeated signals. Source 1 sends a signal A of the form x^* for some binary string x, that is, repetition of x multiple times. For example, if x is 110 then A can be for example, 110110110110110110. Source 2 sends a signal B of the form $(y|z)^*$ for some binary strings y and z. That is, B is made by concatenating y and z any number of times, in any order. For example, if $y = 00$ and $z = 101$ then B can be for example, 101000010110110100.

Design an efficient algorithm that takes as input binary strings x, y, and z and another binary string T and outputs whether T is a superposition of two signals A and B as described above. That is, whether one can partition T into two subsequences A and B, where A is of the form x^* and B is of the form $(y|z)^*$.

Answer. Let $x = x_1x_2\cdots x_p$, $y = y_1y_2\cdots y_q$, and $z = z_1z_2\cdots z_r$. Let $T = T_1T_2\cdots T_n$.

Let's try to build a recursive algorithm. Let T have n bits. If T is indeed a superposition of A and B, then the last bit of T will either come from A or come from B. In case 1, we will try to see if $T_1T_2\cdots T_{n-1}$ is a superposition of A' and B where A' is of the form $x^*x_1x_2\cdots x_{p-1}$ and B is of the form $(y|z)^*$. In case 2, we will try to see if $T_1T_2\cdots T_{n-1}$ is a superposition of A and B' where A is of the form x^* and B' is of the form $(y|z)^*y_1y_2\cdots y_{q-1}$ or $(y|z)^*z_1z_2\cdots z_{r-1}$ depending on whether T_n matches with y_q, or z_r, or both.

This motivates us to define the following Boolean variables. For $0 \le i \le n$, $0 \le j \le p$, $0 \le k \le q$, $0 \le \ell \le r$, define

$XY(i,j,k)$: supposed to be true if and only if $T_1T_2\cdots T_i$ is a superposition of A and B such that A is of the form $x^*x_1x_2\cdots x_j$ and B is of the form $(y|z)^*y_1y_2\cdots y_k$.

$XZ(i,j,\ell)$: supposed to be true if and only if $T_1T_2\cdots T_i$ is a superposition of A and B such that A is of the form $x^*x_1x_2\cdots x_j$ and B is of the form $(y|z)^*z_1z_2\cdots z_\ell$.

We will compute these variables recursively.

---

1. Initialize $XY(0,0,0)$ and $XZ(0,0,0)$ as True.
2. Initialize all other entries as false.
3. for $i = 1$ to n:
4. for $j = 1$ to p and for $k = 1$ to q:
5. $XY(i,j,k) \leftarrow \big(XY(i-1,j-1,k)$ and $(T_i == x_j)\big)$ or $\big(XY(i-1,j,k-1)$ and $(T_i == y_k)\big)$;
6. for $j = 1$ to p and for $\ell = 1$ to r:
7. $XZ(i,j,\ell) \leftarrow \big(XZ(i-1,j-1,\ell)$ and $(T_i == x_j)\big)$ or $\big(XZ(i-1,j,\ell-1)$ and $(T_i == z_\ell)\big)$;
8. for $k = 1$ to q:
9. $XY(i,0,k) \leftarrow XY(i,p,k)$;
10. for $\ell = 1$ to r:
11. $XZ(i,0,\ell) \leftarrow XZ(i,p,\ell)$;
12. for $j = 0$ to p:
13. $XY(i,j,0) \leftarrow XY(i,j,q)$ or $XZ(i,j,r)$;
14. $XZ(i,j,0) \leftarrow XY(i,j,0)$;
15. Output $XY(n,0,0)$;

The update rule for $XY(i,j,k)$ and $XZ(i,j,\ell)$ are natural. Let's go over the updates in boundary cases. The update in line 9 is done because if a string A is of the form $x^*x_1x_2\cdots x_p$ then A can also be said to be of the form x^*. Line 11 has a similar logic. The argument behind Line 13 and 14 is that if a string B is of the form $(y|z)^*y_1y_2\cdots y_q$ or of the form $(y|z)^*z_1z_2\cdots z_r$, we can also say that B is of the form $(y|z)^*$.

Part 2 Que 1. Suppose we are storing a set of files on a hard disk which has only a sequential access. That is, to read a certain memory location you have to traverse through all the memory locations before it. Suppose there are$n$files with their lengths being $\ell(1), \ell(2), \ldots, \ell(n)$. If they are stored in the order$1, 2, \ldots, n$then to read the$k$th file, the time it takes will be

$$\sum_{h=1}^{k} \ell(h).$$

We are also given the access frequencies for the files $f(1), f(2), \ldots, f(n)$. We want to arrange the files in an optimal order $i_1, i_2, \ldots, i_n$so that the average access time

$$\sum_{j=1}^{n} f(i_j) \sum_{h=1}^{j} \ell(i_h).$$

is minimized.

Show that the optimal order is simply the increasing order of$\ell(i)/f(i)$. You can show this in small steps. That is, just consider two consecutive files which are not in the correct order, swap them and show that the average access time goes down.

Answer. Let qth and pth file satisfy $\ell(q)/f(q) > \ell(p)/f(p)$. Equivalently, $f(q)\ell(p) - f(p)\ell(q) < 0$. Suppose the files are ordered such that the qth file comes just before the pth file. Let the average access time for this order be T. Now, let us swap the pth and qth file. Observe that except these two, all other files will have the same access time as before. The qth file will get its access time increased by $\ell(p)$, while the pth file will get its access time reduced by $\ell(q)$. Hence, the net change in the average access time is

$$f(q)\ell(p) - f(p)\ell(q) < 0.$$

That is, the average access time has reduced.

Starting from an arbitrary ordering, we repeatedly do this swap on any two consecutive files with $\ell(q)/f(q) > \ell(p)/f(p)$. Finally, we will get the files ordered in increasing order of $\ell(i)/f(i)$. Since the average access time reduces in every swap, we can conclude that increasing order of $\ell(i)/f(i)$ is better than every other order.

Part 2 Que 2. A ministry wants to create a scientific advisory panel which has a representation from every field. It is a given a list of experts with their areas of expertise. One person can be expert in multiple areas. They want to find a subset of experts such that every subject area is covered by at least one expert in the subset.

For example, suppose there are five subject areas A, B, C, D, E. Suppose
P1 is an expert in A, B, C,
P2 is an expert in B, D, E,
P3 is an expert in A, C, D,
P4 is an expert in A, E.

Then P1, P2 is a subset of experts that covers every subject area. But, P1, P3 does not cover every subject area, it leaves out E.

Show that the following problem is NP-hard: given a list of experts with their areas of expertise and a number k , is there a subset of k experts that covers every subject area?

You can assume NP-completeness for a problem, only if it was done in the lectures. There were two such problems, satisfiability and independent set.

Hint: You can start with the assumption that the independent set problem in a graph is NP-complete. One thing you can try is to relate independent set to vertex cover and then relate vertex cover to the above problem. Or you can try to directly relate independent set to the above problem.

Answer. Let's recall the independent set problem. Given a graph G and a number ℓ, is there an independent set in G of size ℓ? We are assuming that the independent set problem is NP-complete. We will reduce the independent set problem to the given problem statement, which we will refer as the *experts* problem. That will prove that the *experts* problem is NP-hard.

Reduction: We start with an instance of the independent set problem, that is a graph G and a number ℓ. Let the vertices of G be $v_1, v_2, \ldots, v_n$. We will generate an instance $E_{G,\ell}$ of the *experts* problem as follows: (i) for every vertex v_i in G, let us create an expert p_i, (ii) for every edge (v_i, v_j) in the graph G, we create a subject $s_{i,j}$. The subject $s_{i,j}$ has exactly two experts p_i and p_j. Finally let us define k as $n - \ell$.

Claim: G has an independent set of size ℓ if and only if there is a set of $k = n - \ell$ experts covering every subject.

Proving the forward direction of the claim: Let I be an independent set of vertices of size ℓ. Consider the set of experts $\{p_i : v_i \notin I\}$, that is, the complement set of I. Clearly, the set has $n - \ell$ experts. Now, we argue that this set of experts covers every subject area. Take any subject $s_{i,j}$. By construction, there is an edge (v_i, v_j) in G. Since I is an independent set, it cannot contain both v_i and v_j. That means, at least one of p_i and p_j will fall into the chosen set of experts (as it is complement of I). Hence, the subject $s_{i,j}$ has at least one expert in the chosen set.

Proving the backward direction of the claim: Let P be a set of $k = n - \ell$ experts that covers every subject area. Consider the set of vertices $I = \{v_i : p_i \notin P\}$, that is, the complement set of P. Clearly, I has ℓ vertices. We argue that I is an independent set. Consider any edge (v_i, v_j) in G. There is a corresponding subject $s_{i,j}$ by construction. Hence, one of p_i and p_j must be present in the set of experts P. That means, at most one of v_i and v_j can be in I (because it is complement of P). We have argued that for any edge, at most one endpoint can be in I. Thus, I is indeed an independent set.

This finishes the reduction.

Part 2 Que 3. Consider a version of the previous problem where we want to find a panel with minimum number of experts which will cover every subject area. Since it is NP-hard, we do not expect an efficient algorithm. Let us consider the following algorithm that can give an approximate solution.

0. Let S be initialized to the set of subjects.
1. Pick an arbitrary subject in S and include all the experts in that subject into your panel.
2. Remove all those subjects from S that are covered by the selected experts.
3. If S is non-empty, go to 1.
4. Output the panel formed.

Let's run this algorithm on the example given in the previous question. Say, in line 1 we pick the subject C. Then we include all the experts in C, which are P1 and P3, into our panel. Now, P1 and P3 together cover A, B, C, D. So, in line 2, we remove these from S and are left with only E. Now, we go back to line 1. We include all the experts in E, which are P2 and P4, into our panel. Now, E is covered and and S is empty. We output our panel P1, P3, P2, P4. Note that the optimal solution has only two experts in this example.

Assume that in the given instance, any subject has at most 3 experts. Assuming this prove that the above algorithm is a 3-approximation algorithm.

Hint: a natural lower bound on the optimal solution can be a set of subjects where no two subjects have any common experts.

Answer. Let $s_1, s_2, \ldots, s_k$ be all the subjects that were picked in line 1 during the algorithm. Observe that when we pick s_1, we remove all those subjects from S that are covered by any expert in s_1. That means, s_2 is a subject that is not covered by any expert in s_1. In other words, s_1 and s_2 don't have any experts in common. By the same logic, any two subjects in $\{s_1, s_2, \ldots, s_k\}$ cannot have any experts in common. From this we can conclude that the optimal covering set of experts must have at least k experts because the subjects $s_1, s_2, \ldots, s_k$ all need different experts.

Now, recall that by assumption any subject has at most 3 experts. Hence for each subject in $\{s_1, s_2, \ldots, s_k\}$, the algorithm includes at most 3 experts into our panel. That means, the algorithm outputs a set with at most $3k$ experts.

To conclude, the optimal solution has at least k experts, while the algorithm outputs at most $3k$ experts. It follows that the number of experts selected by the algorithm is at most 3 times the optimal number of experts. Thus, it's a 3-approximation algorithm.