Introduction to Algorithms **Solution Set 5**
CS 4820, Spring 2014

(1) *(10 points)*

Consider the following generalization of the median-finding problem: given a sequence $A = (a_1, \ldots, a_n)$ and integers $k_1 < k_2 < \cdots < k_t$ between 1 and n, output a sequence $B = (b_1, \ldots, b_t)$ such that b_i is the $(k_i)^{\text{th}}$ largest element among the elements in A for all i from 1 to t. You may assume the elements of A are all distinct.

Show that there exists a randomized algorithm for this problem with expected running time $O(n \log t)$.

Since a sorting-based algorithm can solve this problem in time $O(n \log n)$, the interesting case for the algorithm occurs when $\log t$ is significantly smaller than $\log n$. Accordingly, algorithms that solve the problem by sorting the entire sequence $(a_1, \ldots, a_n)$ will not be considered acceptable solutions to this problem.

Solution 1

Algorithm. We will describe a recursive algorithm $\textsc{Select}(a_1, \ldots, a_n; k_1, \ldots, k_t)$. For $t = 1$, our algorithm finds the $(k_1)^{\text{th}}$ largest element of A in time $O(n)$ using the randomized median-finding algorithm presented in class. (Recall that the median-finding algorithm can be used to find the k^{th} largest element for any value of k, not just for $k = n/2$.)

For $t > 1$, let $s = \lceil t/2 \rceil$. Our algorithm first finds the $(k_s)^{\text{th}}$ largest element of A in time $O(n)$, again using the randomized median-finding algorithm presented in class. Denote this element by x. We divide $a_1, \ldots, a_n$ into two sequences $a'_1, \ldots, a'_p$ and $a''_1, \ldots, a''_q$ consisting of the elements greater than x and less than x, respectively. (Note that $p = k_s - 1$ and $q = n - k_s$.) For $i = 1, \ldots, t - s$, let $\ell_i = k_{s+i} - k_s$. Recursively let

$$(b'_1, \ldots, b'_{s-1}) = \textsc{Select}(a'_1, \ldots, a'_p; k_1, \ldots, k_{s-1})$$
$$(b''_1, \ldots, b''_{t-s}) = \textsc{Select}(a''_1, \ldots, a''_q; \ell_1, \ldots, \ell_{t-s}).$$

Return the sequence

$$(b_1, \ldots, b_t) = (b'_1, \ldots, b'_{s-1}, x, b''_1, \ldots, b''_{t-s}).$$

Proof of correctness. The proof of correctness is by strong induction on t. In the base case $t = 1$ we are running the algorithm presented in class and in Section 13.5 of the book. The proof of correctness is contained in both the lecture and the book. For $t > 1$, we know from the $t = 1$ case that the first step of our algorithm correctly sets x equal to the $(k_s)^{\text{th}}$ largest element of A. Thus the sequence $A' = (a'_1, \ldots, a'_p)$ consists of the $k_s - 1$ largest elements of A. By the induction hypothesis, $\textsc{Select}(a'_1, \ldots, a'_p; k_1, \ldots, k_{s-1})$ will return a sequence $(b'_1, \ldots, b'_{s-1})$ such that b'_i is the $(k_i)^{\text{th}}$ largest element of A', hence also the $(k_i)^{\text{th}}$ largest element of A. Again using the induction hypothesis, $\textsc{Select}(a''_1, \ldots, a''_q; \ell_1, \ldots, \ell_{t-s})$ returns a sequence $(b''_1, \ldots, b''_{t-s})$ such that b''_i is the $(\ell_i)^{\text{th}}$ largest element of $A'' = (a''_1, \ldots, a''_q)$. Since each element of A'' is also less than x and less than all $k_s - 1$ of the elements in A', it means that the $(\ell_i)^{\text{th}}$ largest element of A'' is the $(\ell_i + k_s)^{\text{th}}$ largest element of A. This completes the proof of correctness since, by

construction, $\ell_i + k_s = k_{s+i}$, and our algorithm places b''_i in the $(s+i)^{\text{th}}$ position of its output sequence.

Running time. Let $T(n,t)$ denote the running time of our algorithm. Let C denote an absolute constant large enough that the median-finding algorithm presented in class has expected running time bounded above by Cn. We will prove that $T(n,t) \le (C+2)n(1+\log t)$ by induction over t. The base case $t = 1$ follows from our definition of C. The case $t > 1$ satisfies the recurrence

$$T(n,t) \le Cn + n + T(p, s-1) + T(q, t-s) + n, \tag{1}$$

where p, q, s are as defined above. (Note that $s = \lceil t/2 \rceil$, whereas the values of p and q are dependent on the input data.) The first term on the right side accounts for finding the $(k_s)^{\text{th}}$ largest element of A, the second term accounts for splitting the sequence A into A' and A'', the next two terms account for solving the two subproblems recursively, and the final term accounts for forming the combined output sequence $(b_1, \ldots, b_t)$.

Both $s-1$ and $t-s$ are bounded above by $t/2$. Therefore, by the induction hypothesis,

$$T(p, s-1) \le (C+2)p(1+\log(t/2)) = (C+2)p\log(t)$$
$$T(q, t-s) \le (C+2)q(1+\log(t/2)) = (C+2)q\log(t).$$

Substituting these upper bounds into (1) we obtain

$$T(n,t) \le (C+2)n + (C+2)p\log(t) + (C+2)q\log(t) \le (C+2)n(1+\log t),$$

where we have used the fact that $p + q < n$. This completes the induction step and concludes the runtime analysis.

Solution 2

Algorithm. A recursive algorithm RandSplit$(a_1, \ldots, a_n; k_1, \ldots, k_t)$ is described by the following pseudocode.

repeat
  Sample x uniformly at random from sequence A.
  Let $A' = (a'_1, \ldots, a'_p)$ consist of all elements of A greater than x.
  Let $A'' = (a''_1, \ldots, a''_q)$ consist of all elements of A less than or equal to x.
until $\lfloor n/4 \rfloor \le p \le \lceil 3n/4 \rceil$
if $t = 1$ and $k_1 = p + 1$ **then**
  return the one-element sequence (x).
else
  Let $k_0 = 0$. Let $s \in \{0, 1, \ldots, t\}$ be the largest index such that $k_s \le p$.
  if $s > 0$ **then**
    Let $(b'_1, \ldots, b'_s) =$ RandSplit$(a'_1, \ldots, a'_p; k_1, \ldots, k_s)$.
  end if
  if $s < t$ **then**
    Let $(b''_1, \ldots, b''_{t-s}) =$ RandSplit$(a''_1, \ldots, a''_q; k_{s+1} - p, \ldots, k_t - p)$.
  end if
  return $(b'_1, \ldots, b'_s, b''_1, \ldots, b''_{t-s})$.
end if

The algorithm chooses a random splitter x and splits the sequence A into the subsequences consisting of elements greater than x and those less than or equal to x. It repeats this step (sampling a new random splitter independently on each repetition) until both subsequences have $\lceil 3n/4 \rceil$ or fewer elements in them. Then it recursively selects the required elements from both subsequences. Note that if one of the two subsequences does not contain any of the elements that need to be selected (the cases $s = 0$ and $s = t$ in the pseudocode) then we skip the unnecessary recursive call. This fact will be important in the running time analysis.

Proof of correctness. The proof of correctness is by strong induction on t. It is almost identical to the proof of correctness for Solution 1, so we omit the proof of correctness from this solution.

Running time. The repeat block performs $O(n)$ work per iteration. Each iteration has probability at least $1/2$ of succeeding (i.e., producing a split such that the index p belongs to the required range), so the expected number of repeat-block iterations is at most $2 = 1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots$. Finding the largest index s such that $k_s \leq p$ takes time $O(\log t)$ using binary search. Forming the output sequence takes time $O(t)$. Thus, other than work performed in recursive calls, the expected number of operations performed by the algorithm is $O(n + t + \log t) = O(n)$. Fix a constant c such that the expected number of operations performed, other than work performed in recursive calls, is less than cn. Then the running time satisfies the recurrence

$$T(n, t) \leq cn + T(p, s) + T(q, t - s), \tag{2}$$

where p, q, s are as defined above.

Now, for a constant $C > 0$ to be specified later, let us adopt as our inductive hypothesis that $T(n, t) \leq Cn \log(2t)$. Choose C large enough that this hypothesis is satisfied whenever $1 \leq t \leq n \leq 4$, so that we may assume henceforth that $n \geq 5$. In particular, this means $\lceil 3n/4 \rceil \leq 4n/5$.

In the base case $t = 1$, at least one of $s, t - s$ is equal to zero, and p, q are both bounded above by $4n/5$, so (2) reduces in the $t = 1$ case to

$$T(n, 1) \leq cn + T(4n/5, 1), \tag{3}$$

whose solution is $T(n, 1) = O(n)$. Assume henceforth that C is chosen to be large enough that $T(n, 1) < Cn$ for all n, so the base case of our induction is satisfied.

For $t > 1$ the inductive hypothesis combined with Equation (2) implies that

$$T(n, t) \leq cn + C[p \log(2s) + q \log(2(t - s))]. \tag{4}$$

If $s = 0$, then (4) reduces to $T(n, t) \leq cn + Cq \log(2t)$. Using the fact that $q \leq 4n/5$, we see that the right side is less than or equal to $Cn \log(2t)$, as desired, provided that $C \geq 5c$. The case $t - s = 0$ is dealt with similarly. For the case that both s and $t - s$ are strictly positive, we use the following lemma.

Lemma 1. *For any numbers $x, y, z, w > 0$ we have*

$$x \log(z) + y \log(w) \leq (x + y) \left[\log(z + w) - \frac{x}{x + y} \log\left(\frac{x + y}{x}\right) - \frac{y}{x + y} \log\left(\frac{x + y}{y}\right) \right]. \tag{5}$$

Proof. Let $\alpha = \frac{x}{x+y}$. Dividing both sides of (5) by $x + y$ we see that it is equivalent to

$$\alpha \log(z) + (1 - \alpha) \log(w) \leq \log(z + w) - \alpha \log(1/\alpha) - (1 - \alpha) \log(1/(1 - \alpha)). \tag{6}$$

Rearranging terms in (6) we see that it is equivalent to

$$\alpha \log(z/\alpha) + (1-\alpha)\log(w/(1-\alpha)) \leq \log(z+w), \tag{7}$$

which holds because the logarithm is a concave function. (Here we are using the fact that $z+w$ is the weighted average of z/α and $w/(1-\alpha)$ with weights α and $1-\alpha$, respectively. The value of a concave function at the weighted average of two numbers is greater than or equal to the weighted average of the function values at those two numbers.) □

If $s > 0$ and $t - s > 0$ then we may apply Lemma 1 and use the fact that $p + q = n$ to obtain obtain

$$cn + C[p\log(2s) + q\log(2(t-s))] \leq cn + Cn\left[\log(2t) - (p/n)\log(n/p) - (q/n)\log(n/q)\right]. \tag{8}$$

The function $H(y) = y\log(1/y) + (1-y)\log(1/(1-y))$, defined on the interval $0 < y < 1$, is concave and satisfies $H(y) = H(1-y)$, so its minimum value on the interval $\frac{1}{5} \leq y \leq \frac{4}{5}$ is equal to $H(\frac{1}{5})$. By our choice of splitter, we know that $n/5 \leq p \leq 4n/5$, so $y = p/n$ belongs to the interval $\frac{1}{5} \leq y \leq \frac{4}{5}$, implying that $(p/n)\log(n/p) + (q/n)\log(n/q) \geq H(\frac{1}{5})$. Hence

$$cn + Cn\left[\log(2t) - (p/n)\log(n/p) - (q/n)\log(n/q)\right] \leq cn + Cn\log(2t) - CH(\tfrac{1}{5})n. \tag{9}$$

If we choose C large enough that $CH(\frac{1}{5}) \geq c$, then we can combine (4), (8), (9) to obtain $T(n,t) \leq Cn\log(2t)$, which confirms the induction step.

(2) *(15 points)* Let G be a flow network with vertex set V, edge set $E = \{e_1, e_2, \ldots, e_m\}$, source $s \in V$ and sink $t \in V$. If x is a vector in the vector space $\mathbb{R}^m$, we will say that:

- x is a *flow vector* if there exists a flow f such that $x_i = f(e_i)$ for all i.
- x is a *path vector* if there exists a directed path from s to t such that $x_i = 1$ if e_i belongs to the path and $x_i = 0$ otherwise.
- x is a *cycle vector* if there exists a directed cycle such that $x_i = 1$ if e_i belongs to the cycle and $x_i = 0$ otherwise.

(2.a) Prove that every flow vector can be represented as a non-negative weighted sum of path vectors and cycle vectors. To prove this, you should design a polynomial-time algorithm to compute such a representation. As usual, you should analyze your algorithm's running time and prove its correctness.

(2.b) Give an example of a flow network having a flow vector that cannot be represented as a non-negative weighted sum of path vectors. Your example should specify both the flow network G and the flow vector x.

(2.c) Prove the following statement, or give a counterexample: *if a flow f is computed using the Ford-Fulkerson algorithm, then the corresponding flow vector x can always be represented as a non-negative weighted sum of path vectors.* (If you are giving a counterexample, your counterexample should specify not only the flow network G and the flow f, but also the sequence of augmenting paths used in the Ford-Fulkerson execution that computes f.)

Solution

(2.a) Algorithm. We will use the following algorithm, which repeatedly finds a directed s-t path or cycle made up of edges with a positive amount of flow. It inserts the corresponding path vector or cycle vector (weighted by the minimum flow value of any edge in the path or cycle) into its list of summands, and it updates the flow by subtracting the path or cycle vector with the aforementioned weight.

1: Initialize an empty list of summands.
2: **while** $\{e \mid f(e) > 0\}$ contains a directed s-t path or cycle **do**
3: Let S denote the edge set of this path or cycle.
4: Let $b = \min\{f(e) \mid e \in S\}$.
5: Let x be the path vector or cycle vector corresponding to S. Add $b \cdot x$ to the list of summands.
6: **for all** edges $e \in S$ **do**
7: $f(e) = f(e) - b$.
8: **end for**
9: **end while**

The second line of the algorithm requires us to find either a directed s-t path or a directed cycle, in the subgraph of G consisting of edges e such that $f(e) > 0$. For each of these search problems (finding an s-t path, finding a cycle) we use either DFS or BFS. This completes the specification of the algorithm.

Proof of correctness. We begin the proof of correctness by observing the following properties of the algorithm.

1. At the start and end of each while-loop iteration, f is a flow.
2. For every edge e, the value $f(e)$ never increases during the algorithm's execution.
3. For every while-loop iteration, there is at least one edge e such that $f(e) > 0$ at the start of the iteration, and $f(e) = 0$ at the end of the iteration.

The first two properties are consequences of the way we update f: we modify $f(e)$ only in loop iterations where $e \in S$; in those iterations, we modify $f(e)$ by subtracting b, where $b \leq f(e)$ by construction. Thus, $f(e)$ never increases and we never assign a negative value to it, thus ensuring that $0 \leq f(e) \leq c(e)$ as required in the definition of flow. Flow conservation is also ensured, because for every vertex $v \neq s, t$, the edge set S contains exactly the same number of edges entering v as leaving v (this number is either 0 or 1), which means that $f^{\text{out}}(v)$ and $f^{\text{in}}(v)$ change by the same amount when we modify f, thus preserving the property of flow conservation. By our choice of b, in every loop iteration there is at least one edge e such that $f(e) = b$, and this results in setting $f(e) = 0$ before the end of the iteration.

The proof of correctness now proceeds by induction on the number of nonzero components in the flow vector. In the base case, the flow vector is equal to the zero vector and our algorithm outputs an empty list, corresponding to the trivial representation of the zero vector as an empty sum. For the induction step, if x is a nonzero flow vector corresponding to the flow f, we first prove that the conditional in the while-loop will pass, i.e. that there is either a directed s-t path or a directed cycle made up of edges such that $f(e) > 0$. There are two cases.

Case 1: value$(f) > 0$**.** In this case, let A be the set of vertices u such that G contains a directed path P from s to u with $f(e) > 0$ for every edge e in P. Let B denote the set $V \setminus A$. We have

$$\sum_{v \in A} f^{\text{out}}(v) - f^{\text{in}(v)} = \begin{cases} 0 & \text{if } t \in A \\ \text{value}(f) & \text{if } t \notin A \end{cases} \tag{10}$$

because $s \in A$ and the flow conservation equation ensures that the only other potential nonzero summand on the left side occurs when $v = t$. For any vertex sets X, Y, let $E(X, Y)$ denote the set of edges whose tail belongs to X and head belongs to Y. We have

$$\sum_{v \in A} f^{\text{out}}(v) - f^{\text{in}(v)} = \sum_{e \in E(A,A)} [f(e) - f(e)] + \sum_{e \in E(A,B)} f(e) - \sum_{e \in E(B,A)} f(e) = - \sum_{e \in E(B,A)} f(e), \tag{11}$$

where we have used the fact that $f(e) = 0$ for all $e \in E(A, B)$ by the definition of the sets A and B. The right side of Equation (11) is clearly not a positive number, so the left side is also non-positive, and combining this fact with Equation (10) we find that $t \in A$. Thus, there is a directed s-t path made up of edges e such that $f(e) > 0$.

Case 2: value$(f) = 0$**.** Since we are not in the base case of the induction, the edge set $E' = \{e \mid f(e) > 0\}$ is nonempty. We need to prove that this set contains a directed cycle. The alternative is that (V, E') is a directed acyclic graph with a nonempty edge set. Every (finite) such graph contains a sink vertex, meaning a vertex v that has at least one incoming edge but no outgoing edges. For such a vertex v we have $f^{\text{in}}(v) > 0$ but $f^{\text{out}}(v) = 0$. When $v \neq s, t$ this violates flow conservation, whereas when $v \in \{s, t\}$ it violates the hypothesis that value$(f) = 0$. In either case, we reach a contradiction, so E' contains a directed cycle as claimed.

Given that the conditional in the while-loop passes, the algorithm inserts another summand (a path vector or cycle vector, with weight b) into its list and updates the flow vector by modifying $f(e)$ to $f(e) - b$ for every $e \in S$. By our choice of b, this sets $f(e)$ to zero for at least one edge $e \in S$, which strictly decreases the number of nonzero components in the flow vector. By our induction hypothesis, the remaining while-loop iterations compute a valid decomposition of the remaining flow as a weighted sum of path vectors and cycle vectors, which completes the proof of correctness.

Running time. In each while-loop iteration, the time required to find the set S using DFS or BFS is $O(m + n)$. The remaining steps in the loop require $O(1)$ operations for each element of S. Since $|S| < n$, this amounts to $O(n)$ operations altogether, so in total a single loop iteration runs in time $O(m + n)$. Above, we have proven that the number of edges e with $f(e) > 0$ strictly decreases in each iteration of the while-loop, which means that the combined number of iterations is bounded by m, and our algorithm's running time is bounded by $O(m^2 + mn)$.

(2.b) See solution to part 2.c.

(2.c) Consider a graph with vertex set $\{s, a, b, c, d, e, t\}$ and with the following edges.

- Edges of capacity 3: $\{(s, a), (a, t), (s, e), (e, t)\}$.
- Edges of capacity 1 from a to each of b, c, d.
- Edges of capacity 1 from each of b, c, d to e.
- Edges of capacity 1 forming a 3-cycle: $\{(b, c), (c, d), (d, b)\}$.

Consider an execution of the Ford-Fulkerson algorithm that uses the following sequence of augmenting paths.

1. $P_0 = (s, a, b, c, e, t)$.
2. $P_1 = (s, a, c, d, e, t)$.
3. $P_2 = (s, a, d, b, e, t)$.
4. $P_3 = (s, e, b, a, t)$.
5. $P_4 = (s, e, c, a, t)$,
6. $P_5 = (s, e, d, a, t)$.

The algorithm will send one unit of flow on each of the augmenting paths listed above. The first three paths saturate edges (s, a) and (e, t) while placing a unit of flow on the 3-cycle $\{(b, c), (c, d), (d, b)\}$ and on the edges linking these three vertices to $\{a, e\}$. The next three paths saturate edges (s, e) and (a, t) while removing the flow from the edges links $\{a, e\}$ to $\{b, c, d\}$. However, the flow on the 3-cycle remains, so the final flow is represented by the values.

- $f(s, a) = f(s, e) = f(a, t) = f(e, t) = 3$.
- $f(b, c) = f(c, d) = f(d, b) = 1$.
- All other flow values are zero.

The edges $\{(b, c), (c, d), (d, b)\}$ do not belong to any directed s-t path made up of edges with positive flow values. Therefore, in any decomposition of the flow vector as a weighted sum of path vectors and cycle vectors, the edges $\{(b, c), (c, d), (d, b)\}$ cannot belong to any of the path vectors and must instead belong to a cycle vector.