National University of Singapore

CS3230: Design and Analysis of Algorithms

(Semester 2: AY2020/21)

Time Allowed: 2 hours

Instructions

- This paper consists of FOUR questions and comprises of TEN (10) printed pages, including this page.
- Answer ALL the questions.
- Write ALL your answers in this assessment book.
- This is an OPEN BOOK assessment.
- Please write your Student Number (that starts with "A") below and on the top of every page. Do not write your name.

Student Number:

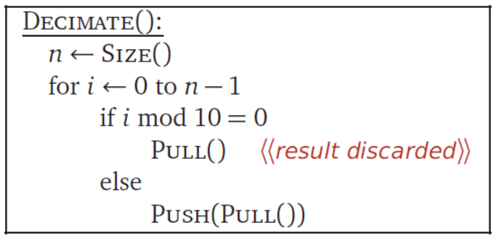
Question	Maximum	Score
Q1	15	
Q2	25	
Q3	15	
Q4	25	
Total	**80**	

Question 1 [15 marks]: Consider a standard (FIFO) queue that supports the following operations:

- PUSH(x): Add item x at the end of the queue.
- PULL(): Remove and return the first item present in the queue.
- SIZE(): Return the number of elements present in the queue.

We can easily implement such a queue using a doubly-linked list so that each of the above three operations takes $O(1)$ worst-case time. (No need to show this. You can assume such an implementation.) Now, suppose we are asked to consider the following new operation:

- DECIMATE(): Remove every tenth element from the queue starting from the beginning.



```
DECIMATE():
  $n \leftarrow$ SIZE()
  for $i \leftarrow 0$ to $n-1$
      if $i \bmod 10 = 0$
          PULL()    ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL())
```

Figure 1: Pseudocode of the DECIMATE operation (assuming starting index to be 0)

The above DECIMATE operation takes $O(n)$ time in the worst-case (where n is the number of items present in the queue). Show that in any intermixed sequence of PUSH, PULL and DECIMATE operations, the amortized cost of each of them is $O(1)$. (Use any one of the three methods we have learned in the module, i.e., aggregate method, accounting method, or potential method.)

Solution: Suppose at any point of time the number of elements present in the queue is ℓ. Then the worst-case (actual) running time of PUSH(x), PULL(), SIZE() and DECIMATE() are c_1, c_2, c_3 and $c_4\ell$ respectively, for some constant $c_1, c_2, c_3, c_4 > 0$. Now, we use the potential method to establish the desired bound on the amortized costs of the above four operations.

Let us consider the following potential function: $\phi = 10c_4\times$ the number of elements present in the queue. Clearly, in the beginning $\phi = 0$, and at any point of time $\phi \geq 0$. So ϕ is a valid potential function. Recall, in the potential method, the amortized cost of an operation is equal to the sum of its actual cost and the potential difference after the application of that particular operation.

Next, consider the amortized cost of each of the four operations:

- PUSH(x): After this operation, the potential difference is $10c_4$, and thus the amortized cost is $c_1 + 10c_4$.
- PULL(): After this operation, the potential difference is $-10c_4$, and thus the amortized cost is $c_2 - 10c_4$.
- SIZE(): After this operation, the potential difference is 0, and thus the amortized cost is c_3.
- DECIMATE(): Let the number of elements present in the queue is ℓ. After this operation, the potential difference is $-10c_4\lfloor\frac{\ell}{10}\rfloor$, and thus the amortized cost is $c_4\ell - 10c_4\lfloor\frac{\ell}{10}\rfloor \leq 10c_4$.

So, we conclude that the amortized cost of each of the four operations is $O(1)$.

Question 2 [25 marks]: For any set S of integers and two non-negative integers r, ℓ, we call a subset $S' \subseteq S$ an (r, ℓ)-*spaced subset* if the sum of all the integers in S' is r and for every two distinct integers $a, b \in S'$, $|a - b| \geq \ell$. Describe an algorithm that given two non-negative integers r and ℓ, and a set S of n distinct non-negative integers, counts the number of possible (r, ℓ)-spaced subsets of S in $O(rn \log n)$ time. (Assume, a single machine word is large enough to hold any integer computed during your algorithm.)

[Significant partial credits will be awarded if your algorithm runs in time $O(rn^2)$.]

Solution: First, sort the integers in S into an array $A[0, \cdots, n-1]$ (of size n). Then for $i \in \{0, 1, \cdots, n\}$ and $j \in \{0, 1, \cdots, r\}$, define $x(i, j)$ as the number of (j, ℓ)-separated subsets of $A[i], A[i+1], \cdots, A[n-1]$. Note, the number of possible (r, ℓ)-spaced subsets of S is $x(0, r)$.

As base case, $x(n, 0) = 1$ (corresponds to the empty subset S'), and $x(n, j) = 0$ for all $j > 0$. To get the optimal substructure property, sum the number of (j, ℓ)-separated subsets using $A[i]$ with the ones that do not use $A[i]$. So we do the following:

If $A[i] \leq j$ is used, then

- No integer in $A[i], A[i+1], \cdots, A[n-1]$ smaller than $A[i] + \ell$ can be used.
- Let $f(i)$ be the smallest index greater than i such that $A[f(i)] - A[i] \geq \ell$.
- Use the value $x(f(i), j - A[i])$.

Else, $A[i]$ is not used and we can use the value $x(i+1, j)$.

The recursive formula for the $x(i, j)$ is the following:

$$x(i,j) = \sum \Big(\begin{cases} x(f(i), j - A[i]) & \text{if } A[i] \leq j \\ x(i+1, j) & \text{always} \end{cases} \Big).$$

The number of subproblems is $(n+1) \times (r+1) = O(rn)$. Time taken per subproblem is $O(\log n)$ (since we can find $f(i)$ by performing a binary search over the sorted array A). Sorting of integers in S (to form A) can also be done in $O(n \log n)$ time (by using merge sort). So the total time taken is $O(rn \log n)$.

Question 3 [15 marks]: A set H is said to be a *hitting set* for a family of sets $\{S_1, S_2, \cdots, S_n\}$ if and only if for all $1 \leq i \leq n$, $H \cap S_i \neq \phi$ (i.e., H has a non-empty intersection with all the sets S_i).

Consider the following hitting set problem: Given a family of finite integer sets $\{S_1, S_2, \cdots, S_n\}$ and a positive integer K, decide whether there exists a hitting set of size at most K. Show that the hitting set problem is NP-complete.

(You may show a reduction from any of the NP-complete problems introduced in the lectures/ tutorials/ assignments/ practice set, including Circuit Satisfiability, CNF-SAT, 3-SAT, Vertex Cover, Independent Set, Max-Clique, Hamiltonian Cycle, Traveling Sales Person Problem.)

Solution: The hitting set problem is clearly in NP because a hitting set H of size at most K can act as a valid certificate. Verifying the certificate also can easily be done in polynomial time by checking whether it is of size at most K and also has a non-empty intersection with all the input sets.

Next, we can show a polynomial-time reduction from the vertex cover problem. Take an instance (G, k) of the vertex cover problem. Then for each edge $e = (u, v)$ of the graph G, create a set $S_e = \{u, v\}$. (Consider an arbitrary integer labeling of the vertices.) Then consider the family of sets $\mathcal{F} = \{S_e | e \text{ is an edge in } G\}$ and set $K = k$. This constitute an instance of the hitting set problem. It is easy to see that (G, k) is a yes-instance of the vertex cover if and only if $(\mathcal{F}, k)$ is a yes-instance of the hitting set. Hence, we deduce that the hitting set problem is NP-complete.

Question 4 [25 marks]: Consider the following *shortest superstring* problem: Given a set of n strings $S = \{x_1, x_2, \cdots, x_n\}$ (all the strings are over some arbitrary finite alphabet Σ), find a shortest string s over the alphabet Σ that contains each x_i as a substring. Assume, no string x_i is a substring of another string x_j, for $j \neq i$. (E.g., for the set $S = \{abab, abc, cab\}$, a shortest superstring is *cababc*.)

For any string z, define $set(z) := \{x_i \in S \mid x_i \text{ is a substring of } z\}$. For any two distinct strings $x_i, x_j \in S$ and an integer $k > 0$, if the last k symbols of x_i are the same as the first k symbols of x_j, define $y_{i,j,k}$ to be the string obtained by overlapping these k positions of x_i and x_j (i.e., remove the last k symbols from x_i and then concatenate x_j after this leftover string). E.g., let $x_i = ababcda$ and $x_j = bcdab$. Then $y_{i,j,4} = ababcdab$. However, $y_{i,j,3}$ is not defined since the last three symbols of x_i are not the same as the first three symbols of x_j.

Next, consider the set cover problem we have seen in lecture 12. Given S, create a set cover instance $\mathcal{S}$ as follows: For all the valid choices of i, j, k, create $y_{i,j,k}$, and let M be the set of all these strings. The universe for the set cover instance is S, and the input sets are $set(z)$, for all $z \in S \cup M$. Let the *cost* of $set(z)$ be equal to the length of the string z.

Now, use the following algorithm for the shortest superstring problem:

1. Use any approximation algorithm $\mathcal{A}$ for the set cover problem (e.g., that covered in the lecture) to find a cover for the instance $\mathcal{S}$ (described above). Let $set(z_1), \cdots, set(z_r)$ be the sets output by that algorithm.

2. Concatenate the strings $z_1, \cdots, z_r$ in any arbitrary order.

3. Output the resulting string.

Let the approximation ratio of the set cover algorithm $\mathcal{A}$ be $\alpha(n)$. Then show that the approximation ratio of the above algorithm for the shortest superstring problem is $2\alpha(n)$.

[**Hint:** Take a shortest superstring s of the input S, and then consider the starting positions of x_i's in it. Next, observe how some of the strings $y_{i,j,k}$ appear (with overlap) in s. Then consider the corresponding sets $set(y_{i,j,k})$ to form a valid set cover of the instance $\mathcal{S}$. Using this, establish a relation between the optimum cost of the set cover instance $\mathcal{S}$ and the length of s.]

Solution: Let s be an arbitrary shortest superstring of the strings in S. Let $\texttt{OPT}$ denote the length of s and $\texttt{OPT}_{\mathcal{S}}$ denote the cost of an optimum solution of the set cover instance $\mathcal{S}$. Also, let x be the string output by the algorithm described in the question.

First, we will argue that x is a valid superstring for the strings in S, i.e., each $x_i \in S$ is a substring of x. Since $set(z_1), \cdots, set(z_r)$ is a valid set cover of the universe S, each of x_i must belong to some $set(z_j)$, for $1 \leq j \leq r$. Then, x_i must be a substring of some z_j, for $1 \leq j \leq r$, and thus also a substring of x. Further note, the cost of the solution $set(z_1), \cdots, set(z_r)$ is the length of x. Since the approximation ratio of $\mathcal{A}$ is $\alpha(n)$, the length of x must be $\leq \alpha\texttt{OPT}_{\mathcal{S}}$.

Now, to show the claim stated in the question, it suffices to prove that $\texttt{OPT}_{\mathcal{S}} \leq 2\texttt{OPT}$. Let s be an arbitrary shortest superstring of the strings in S. Consider the leftmost occurrence of the strings $x_1, x_2, \cdots, x_n$ in s. Since no string is S is a substring of another string in S, the starting indices of the leftmost occurrence are distinct. Similarly, their end indices are also distinct. From now on, we will consider the strings in S ordered according to their start indices (equivalently, end indices) in s. Without loss of generality, assume that $x_1, x_2, \cdots, x_n$ are numbered according to the above order. Take x_1 and consider the largest index i such that x_i has a non-empty overlap with x_1. If the overlap between x_1 and x_i is k, take the string $\pi_1 = y_{1,i,k}$. Next consider the string x_{i+1} and in a similar fashion take the largest index j such that x_j has a non-empty overlap with x_{i+1}. If the overlap between x_{i+1} and x_j is k', take the string $\pi_2 = y_{i+1,j,k'}$. We continue until no string in S is left. Collect all the strings $\pi_1, \pi_2, \cdots, \pi_\ell$ we get. Clearly, the set $\{set(\pi_1), set(\pi_2), \cdots, set(\pi_\ell)\}$ form a valid set cover for the instance $\mathcal{S}$, with cost equal to the sum of the lengths of π_i's. Observe, π_i has no overlap with π_{i+2} for each $1 \leq i \leq \ell - 2$. Thus, the sum of the lengths of π_i's is at most twice the length of s, i.e., $\leq 2\texttt{OPT}$. Hence, $\texttt{OPT}_{\mathcal{S}} \leq 2\texttt{OPT}$.

—End of Paper—