Name: 周守琛

Student ID: 2021533042

School: SIST

Year of Entrance: 2021

ShanghaiTech University Final Exam Examination Cover Sheet

Academic Year : 2024 Term: Spring

Course-offering School: SIST

Instructor: Rui Fan

Course Name: Algorithm Design and Analysis / 算法设计与分析

Course Number: CS 240

Exam Instructions for Students:

1. All examination rules must be strictly observed throughout the entire test, and any form of cheating is prohibited.

2. Other than allowable materials, students taking closed-book tests must place their books, notes, tablets and any other electronic devices in places designated by the examiners.

3. Students taking open-book tests may use allowable materials authorized by the examiners. They must complete the exam independently without discussion with each other or exchange of materials.

For Marker's Use:

Section	1	2	3	4	5	6					Total
Marks											
Recheck											

Marker's Signature:

Date:

Rechecker's Signature:

Date:

Instructions for Examiners:

1. The format of the exam papers and answer sheets shall be determined by the school and examiners according to actual needs. All pages should be marked by the page numbers in order (except the cover page). All text should be legible, visually comfortable and easy to bind on the left side. A4 double-sided printing is recommended for the convenience of archiving (There are all-in-one printers in the university).

2.The examiners should make sure that exam questions are correct and appropriate, If errors are found in exam questions during the exam, the examiners should be responsible to respond on site, which will be taking into account in the teaching evaluation.

Instructions for Students

*In all problems in which you are asked to design algorithms, you should **clearly describe** how your algorithm works, provide code or pseudocode when asked to, and argue why your algorithm is correct.*

***Do not** write your answers on the exam paper. Instead, write them on* separate pieces of paper. ***Write** your name and student ID at the top of **each piece** of paper.*

*All answers must be written neatly and legibly in English. If there are **brief parts** of your answer which you cannot express clearly in English, you may write them in Chinese.*

Problem 1

Answer true or false to the following questions, and briefly explain your answers.

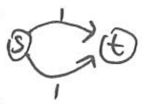
P=NP linear

(a) If 3SAT has a linear time algorithm, then so does every problem in NP.

(b) There is an exponential time algorithm for 3SAT.

(c) When compressing a string using Huffman encoding, the most frequently occurring letter has depth smaller or equal to that of any other letter.

(d) Suppose the maximum (s, t)-flow in a graph has value f. Now we increase the capacity of every edge by 1. Then the maximum (s, t)-flow in the new graph has value at most $f + 1$.



(e) Let (S, T) be a minimum (s, t)-cut in a graph G. If we increase the capacity of every edge by a factor of 2, then the value of the maximum flow in G is increased by 2 times the number of edges from S to T in G.

X2

(5 parts, 5 points / part, 25 points total)

Problem 2

Consider the following type of data compression algorithm. The input is a string y of length n, and a list of k strings $x_1, \dots x_k$ of lengths $m_1, \dots, m_k$ respectively. The problem is to determine the smallest number of copies of strings from $x_1, \dots x_k$ which can be concatenated together to form y. For example, if $x_1 = a, x_2 = ba, x_3 = abab$ and $x_4 = b$, and $y = bababbaababa$, then we can write $y = x_4 x_3 x_2 x_3 x_1$, so we can write y using 5 copies of the x_i's, and this is optimal. Give an efficient algorithm for this problem and analyze its time complexity.

$O(kmn)$

(15 points)

Problem 3

Given two arrays $A = [a_1, \dots, a_n]$ and $B = [b_1, \dots, b_n]$, you want to output an array $[a_1, b_1, a_2, b_2, \dots, a_n, b_n]$. For example, if $A = [1,2,3,4]$ and $B = [5,6,7,8]$, then you need to output $[1,5,2,6,3,7,4,8]$. However, your algorithm is restricted in the amount of space it can use. In particular, it needs to operate *in-place*, and can only use $O(\log n)$ amount of extra space. That is, the algorithm is allowed to use the $2n$ amount of space originally used to store the inputs A, B, and it can also use $O(\log n)$ amount of extra space, but no additional space beyond these can be used. For example, the algorithm cannot simply allocate a new array of size $2n$ for the output. Give an efficient algorithm to solve this problem and analyze its time and space complexity.

$n = 2k$

Hint: Use divide and conquer.

(15 points)

EXAM CONTINUES ON NEXT PAGE

Problem 4

Given an array A of numbers sorted in increasing order, the *diameter* of A is the difference between the largest (i.e. last) and smallest (i.e. first) elements. Furthermore, if we partition A into $k > 1$ subarrays, then the diameter of the partitioning is the largest diameter of any subarray. Given an array A and a value k, your goal is to find a partitioning of A into k subarrays which has the minimum diameter. For example, if $A = [1,5,6,8,13,15]$ and $k = 3$, then you can partition A as $[1], [5,6,8], [13,15]$, which has diameter 3, and this is optimal. Give an efficient algorithm to solve this problem and analyze its time complexity.

dp[i][j][k] 考虑到第 i和第i+1个数的空隙时，放了k个隔断

(15 points)

$i = 1, \dots, n-1$ $\quad dp[i][j] = \min$

integer !

Problem 5 $\quad n = |A| \quad j = 0$

Suppose you have a fair coin, which lands with probability 1/2 on heads or tails when flipped. You are allowed to flip the coin as many times as you wish. Give algorithms to solve the following problems using the results of flipping the coin.

(a) Output a number between 1 to 4 (inclusive), where each value has equal probability (i.e. 1/4) of occurring. How many coin flips does your algorithm use?

— — $\quad$ 2

(5 points)

$\frac{4}{3} \times 2 = \frac{8}{3}$ $\quad FS(\frac{3}{4})$

(b) Output a number between 1 to 3 (inclusive), where each value has equal probability (i.e. 1/3) of occurring. How many coin flips does your algorithm use in expectation?

$P(X = 1,2,3) = \frac{1}{4}$ $\quad$ | | X $\quad$ A: not 4 time

$P(X=4) = \frac{1}{4}$ $\quad P(A=k) = (1-p)^k p$

(10 points)

$$P(X=1/2/3 \mid X \neq 4) = \frac{P(X_i, \neg X_4)}{P(\neg X_4)} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$$

$= (\frac{1}{2})^{k+1}$, $k = 0, \dots$

$E(A) =$

Problem 6

Given an array of n unique numbers, an *approximate median* is a number from the array whose value is between the $n/4$'th to $3n/4$'th largest. For example, given the array [4, 2, 9, 10, 5, 7, 1, 8], values 4, 5, 7, and 8 are all approximate medians. Consider the following fast randomized algorithm to find an approximate median:

1, 2, 4, 5, 7, 8, 9, 10

1. Randomly choose $12 \log n$ different numbers from the array.
2. Sort the numbers you chose.
3. Return the median of the sorted list, i.e. the $6 \log n$'th value.

Show that this algorithm computes an approximate median in $O(\log n \log \log n)$ time with probability $\geq 1 - 2/n$.

$12 \log n (\log(12 \log n))$ **(15 points)**

END OF EXAM

周宇琛 2021533042

Problem 1

(a) F

Since 3SAT is a NP-complete problem, so for any NP problem B, there exist a mapping ~~f: B⇒A~~ $B \xrightarrow{f} A$, i.e. $\forall$ instance X of B, there exist an instance Y of 3SAT. If Y is an yes instance of 3SAT, then X is a yes instance of B. Similarly, if X is yes instance of B, then Y is a yes instance of 3SAT.

And since it has given that 3SAT has a linear time algorithm, i.e. it is polynomial time, So 3SAT is a P problem

$\Rightarrow P = NP$

But any NP problem requires a polynomial time to reduce to 3SAT, the polynomial time may not be a constant.

So for any NP problem, it has a polynomial time algorithm, but may not be linear time.

(b) T

Since 3SAT is a NP-complete problem, and we have learned that $NPC \subseteq EXP$, so there exist an exponential time algorithm for 3SAT.

(c) T

As we have learned, when constructing the Huffman tree, we merge the nodes which has the less and second less frequency.

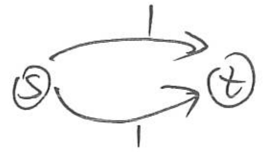
So the nodes with less frequency would be merged first, the most frequence letter would be the last merged node representing the letter,

So it would be the node with depth smaller or equal to any other letter.

周宇琛 2021533042

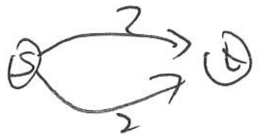
(d) F

We can construct a counter example: suppose the graph is

G: 

, the number on the edge is its capacity,

and its max (s,t)-flow is $f=2$, and add 1 to every edge

G': 

, and its max (s,t)-flow is $f'=4$

which is $f' = f+2 > f+1$ So the statement is wrong.

(e). T

For each flow $f_1, \cdots, f_k$, the total max flow is $f = f_1 + \cdots + f_k$.
after increase by factor 2, the relative bigger or smaller of each edge remains, so the flows are $2f_1, \cdots, 2f_k$, and the max-flow is
$f' = 2f_1 + \cdots + 2f_k = 2(f_1 + \cdots + f_k) = 2f$, so its true.

周宇琛　2021533042

Problem 2:

Let $dp[i]$ represent that the less copies we used to generate $y[1:i]$, where $y[i:j]$ represent the substring of y from i to j (all include)

initial state: all values in dp are $+\infty$, $dp[0]=0$

transition:

for $i=1$ to n:
　for $j=1$ to k: if $(i-m_j \geq 0)$:
　　if (x_j matches $y[i-m_j+1:i]$)
　　　$dp[i] = \min(dp[i], dp[i-m_j]+1)$

for the second if above, "match" represent to compare whether $y[i-m_j+1:i]$ is exactly the same with x_j, if so, return True, otherwise return false.

Solution: the value of $dp[n]$, where $n=|y|$ is the length of y
if $dp[n]$ is $+\infty$, then there is no solution.

time complexity: the two loops take $O(n)$ and $O(k)$ repective.
and the "match" requires to compare, so it takes $O(m_j)$ time for each j.
Let $m = \max_{j=1,\cdots,k} \{m_k\}$

So the time complexity is $O(kmn)$

Problem 3: ~~With WLOG~~

W.L.O.G, we can assume that $n = 2^k$, $k \in \mathbb{Z}_+$

if not, we can add "0"s to the end of each array to make it as $n' = 2^k$, $k \in \mathbb{Z}$

which still takes $O(n)$ space complexity.

And after this, we apply divide-and-conquer.

as a ~~psed~~ pseudo-code:

```
def work(l, r):          <- where l, r are the left and right index
                            we need to rearrange.
    mid <- ⌊(l+r)/2⌋
    if l == r:
        return
    for i = mid+1 to r:
        swap(A[i], B[i-mid])
    work(l, mid)
    work(mid+1, r)
```

for example.

```
A: 1 2 | 3 4         A  1 5 | 3 7         A  1 5  2 6
B: 5 6 | 7 8   =>    B  2 6 | 4 8   =>    B  3 7  4 8
```

then we just need to run work(1, n)

Then we ~~fit~~ return A, B, if we concat [A, B], it would be the output.

Time complexity: Suppose ~~array~~ array with length n takes $T(n)$ time

and we need to swap the elements, so $T(n) = 2T(\frac{n}{2}) + O(n)$

from master theorem, we can get that $T(n) = O(n\log n)$. So time complexity is $O(n\log n)$

And space complexity: $O(n)$ for storing A, B, and extra $O(\log n)$

space and used for the ~~recu~~ recursive stack, no other additional

space were used. So total extra space is $O(\log n)$

周宇璟 2021533042

Problem 4

As we ~~ne~~ can define $n = |A|$ as the length of subarray

And define $dp[i][j][l]$ to represent that the minimum diameter of subarray $[A[i], A[i+1], \ldots, A[j]]$. ~~that~~ with putting ~~k~~ l partitions.

initial: for $i=1$ to n:
  for $j=i$ to n
    $dp[i][j][0] = A[j] - A[i]$

transition: for i ~~in range~~ $(=1$ to n:
  for $j = i$ to ~~range~~ n:
    for $l =$ ~~0~~ to ~~n~~ k:

$$dp[i][j][l] = \min_{m=i,\ldots,j-1} \{ dp[i][m][l-1] + dp[m+1][j][l-1] \}$$

~~Which repres~~

Which representing that putting a partiation ~~at~~ ,

between ~~number~~ ~~n~~ the m-th number and the $m+1$-th number

output. The solution is $dp[1][n][k]$

Time complexity: $O(n^3 l)$

周宇琛 2021533042

Problem 5

(a) we flip the coins two times, and get the result as X_1, X_2 then we concat X_1, X_2 to get a binary number, and translate it in to the $X_{(10)}$ number, then we ~~pt~~ let $X += 1$

Since $X_1, X_2 \overset{i.i.d}{\sim} Bern(\frac{1}{2})$, so $P(X=1)=P(X=2)=P(X=3)=P(X=4)=\frac{1}{4}$

The coin flips is 2.

(b) Similarly with (a), but when we generate $X=4$, we regenerate the number, until $X \neq 4$.

$$P(X=1 | X\neq 4) = \frac{P(X=1, X\neq 4)}{P(X\neq 4)} = \frac{\frac{1}{4}}{1-\frac{1}{4}} = \frac{1}{3}$$

similarly $P(X=2|X\neq 4) = P(X=3|X\neq 4) = \frac{1}{3}$, so we generate uniformly of 1, 2, 3.

Expected time:

Let A be the number of times ~~when need~~ we totally used to generate $X \neq 4$.

So $P(A=k) = (1-\frac{3}{4})^{k-1} \times \frac{3}{4} = (\frac{1}{4})^{k-1}\frac{3}{4}$, $k=1,2,\dots$

So $$E(A) = \sum_{k=1}^{+\infty} k\cdot P(A=k) = \frac{3}{4}\cdot\sum_{k=1}^{+\infty} k(\frac{1}{4})^{k-1} \quad ①$$

$$\frac{1}{4}E(A) = \frac{3}{4}\sum_{k=1}^{+\infty} k(\frac{1}{4})^{k} = \frac{3}{4}\sum_{k=2}^{+\infty}(k-1)(\frac{1}{4})^{k-1} \quad ②$$

$$①-②: \frac{3}{4}E(A) = \frac{3}{4}\cdot\left[1\cdot(\frac{1}{4})^{1-1} + \sum_{k=2}^{+\infty}(\frac{1}{4})^{k-1}\right]$$

$$E(A) = 1 + \frac{(\frac{1}{4})^{2-1}}{1-\frac{1}{4}} = \frac{4}{3}$$

So we are expected to generate $\frac{4}{3}$ times for $X\neq 4$,

i.e. $\frac{4}{3}\times 2 = \frac{8}{3}$ coin flips.

If we requires it to be ~~posit~~ integer, the expected flips is 9.

周宇琛 2021533042

Problen 6

As we need to sort the $12\log n$ ~~of~~ numbers, and there has no other ~~calculati~~ operations, so the total time complexity is

$$O((12\log n)\log(12\log n)) = \cancel{O(\log} \; O(\log n \log\log n)$$

Define $X_i = 1$: the i-th number's chosen index is in range $[\frac{n}{4}, \frac{3n}{4}]$

and $X = \sum_{i=1}^{12\log n} X_i$

Since we are randomly chosen, so $P(X_i=1) = \dfrac{\frac{3}{4}n - \frac{1}{4}n}{n} = \frac{1}{2}$.

i.e. $E(X_i) = \frac{1}{2}$

from linearity of expectation:

$$E(X) = \frac{1}{2} \cdot 12\log n = 6\log n.$$

Since we are chosen the medium value, so if the medium value ranges in $[\frac{n}{4}, \frac{3}{4}n]$

So if we has more than $9(-]n$, the medium value would be in $[\frac{n}{4}, \frac{3}{4}n]$

Consider the error rate:

According to Chernoff bounds, we have, for $0 \le \delta \le 1$

$$P(X \ge (1+\delta)\mu) \le e^{-\frac{1}{3}\mu\delta^2} = e^{-\frac{1}{3}\times\frac{1}{4}\times 12\log n} = \frac{1}{n}$$

and since $\mu = 6\log n$, so $\delta = \frac{1}{2}$

So $P(\text{approxmate medians}) \ge 1 - P(X \ge (1+\delta)\mu)$

$$\ge 1 - \frac{2}{n}$$