ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2022

Problem Set 4

Dai ZiJia 2022233158

Due: 23:59, Dec. 18, 2022

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

Suppose that there are n items need to be placed in some bins. The capacity of each bin is 1, and the volume of items are not same and all volumes are smaller than 1. We want to use the fewest number of bins to place all items. Please design a 2-approximation algorithm to solve this problem.

Solution:

2-approximation algorithm: We can start to open a random bin and place items one by one in it. If the current item doesn't have an opened bin to fit in, we will open a new bin to put it in.

Proof: Suppose that the optimal solution need s' bins, the algorithm needs s bins. If items in two bins are less than half capacity of two bins', then the algorithm will make the items in the latter bin will be put into the first bin, so there will be at least $s-1$ bins that are half full:

$$\sum_{i=1}^{n} w_i > \frac{1}{2}(s-1) \Rightarrow s-1 < 2\sum_{i=1}^{n} w_i \Rightarrow s \leq 2\sum_{i=1}^{n} w_i,$$

for $\sum_{i=1}^{n} w_i < s'$, it can be infer that $s < 2s'$. So the algorithm is 2-approximation.

Problem 2:

In HW3, you've considered the problem that optimally assign m items to two players. Now suppose there are n players and each of them has a unique valuation and an upper bound. Provide a 2-approximation algorithm to solve the problem and prove it.

Solution:

Problem 3:

Suppose to perform a sequence of n operations on a data structure. The i-th operation costs i if i is an exact power of 2, otherwise i-th operation costs 1.
a) Using accounting method to determine the amortized cost per operation.
b) Using potential method to determine the amortized cost per operation.

Solution:

Accounting method: We let c_i' be the charge for the i-th operation, where c_i is the true cost and b_i is the balance after the i-th operation. We can set $c_i' = 3$ for each operation cost. In 10 times iterations they appears to be:

i	1	2	3	4	5	6	7	8	9	10
c_i	1	2	1	4	1	1	1	8	1	1
c_i'	3	3	3	3	3	3	3	3	3	3
b_i	2	3	5	4	6	8	10	5	7	9

Suppose m refer to the m-th operation, if m is not an exact power of 2, there will add $3 - 1 = 2$ to balance. If m is an exact power of 2. The b_m will be:

$$b_m = b_{\frac{m}{2}} + \sum_{i=\frac{m}{2}}^{m} c_i - \sum_{i=\frac{m}{2}}^{m} c_i'$$

$$= b_{\frac{m}{2}} + 2 \cdot (m - \frac{m}{2} - 1) + 3 - m = b_{\frac{m}{2}} + 1$$

Since $b_1 = 2, b_2 = 3$, the balance will always be larger than zero, which means:

$$T(n) = \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} c_i' = 3n$$

In conclusion, it cost $O(3n) = O(n)$ in n operations, each operation cost $O(1)$.

Potential method: We can use the potential function $\Phi(h) = 2x - y$, where x is the current number of elements, y is the next an exact power of 2, for example if $x = 6$, then $y = 8$. It can be inferred that $\Phi(h_1) = 0, \Phi(h_i) \geq 0$ for all i. We define the amortized time of an operation is $c + \Phi(h') - \Phi(h)$. There will be two cases:

- If $x < y$, then true cost $c_i = 1$, x add 1, and y does not change. Then the potential is $\Phi(h') - \Phi(h) = 2(x + 1) - 2x = 2$, so the amortized time is 3.
- If $x = y$, then the next y will get doubled, so the true cost $c_i = x$. But the potential is $\Phi(h') - \Phi(h) = 2(x + 1) - 2x - (x - 1) = 3 - x$, so amortized time is $x + (3 - x) = 3$.

In above cases, the amortized time is O(1).

Problem 4:

Given a function rand2() that returns 0 or 1 with equal probability, implement rand3() using rand2() that returns 0, 1 or 2 with equal probability. Minimize the number of calls to rand2() method. Prove the correctness.

Solution:

Algorithm 1 Rand3()

Input: rand2()
Output: rand3()

```
1: int x,y;
2: while x==1 and y==0 do
3:     x = rand2()
4:     y = rand2()
5: return x+y
```

Proof: To prove that our algorithm returns 0,1,2 with equal probability, we start to discuss about the case $p(RAND3() == 1)$, because there is no case that $x == 1$ and $y == 0$ at the same time. It can be inferred that:

$$p(RAND3() = 1) = p(x+y = 1) = \frac{p(x = 0, y = 1)}{p(x = 0, y = 0) + p(x = 0, y = 1) + p(x = 1, y = 1)}$$

For $p(x = 0, y = 0) = p(x = 0, y = 1) = p(x = 1, y = 1) = \frac{1}{4}$, $p(RAND3() = 1) = \frac{1}{3}$.

Evidenced by the same token, $p(RAND3() = 0) = p(RAND3() = 2) = \frac{1}{3}$.

Problem 5:

Suppose that for some decision problem, we have an algorithm which on any instance computes the correct answer with probability at least 2/3. We wish to reduce the probability of error by running the algorithm n times on the same input using independent randomness between trials and taking the most common result. Using Chernoff bounds, give an upper bound on the probability that this new algorithm produces an incorrect result.

Solution:

Suppose $EX = \mu$ is the expectation to get incorrect result in n times. $\mu = (1-p)n = \frac{n}{3}$. If we want algorithm produces an incorrect result, the error outputs must happen more than $\frac{n}{2}$ times:

$$P(X \geq (1+\delta)\mu) = P(X \geq (1+\frac{1}{2})\frac{n}{3}) \leq e^{-\frac{\mu\delta^2}{3}} = e^{-\frac{\frac{n}{3}\cdot\frac{1}{4}}{3}} = e^{-\frac{n}{36}}$$