Some Exercises on the "Three Basic Techniques"

By Yufei Tao's Teaching Team

Department of Computer Science and Engineering
Chinese University of Hong Kong

You have learned three basic techniques in algorithm design:

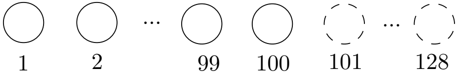
- Recursion
- Repeating (till success)
- Geometric Series.

In this tutorial, we will discuss some exercises that can be solved using these techniques.

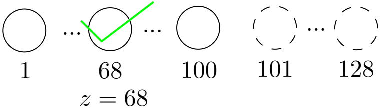
Exercise 1

Recall that our RAM model has an atomic operation RANDOM(x, y) which, given integers x, y, returns an integer chosen uniformly at random from $[x, y]$.

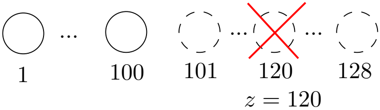
Suppose that you are allowed to call the operation only with $x = 1$ and $y = 128$. Describe an algorithm to obtain a uniformly random number between 1 and 100. Your algorithm must finish in $O(1)$ expected time.



Call RANDOM(1,128) and let z be its return value. Output z if it is in $[1, 100]$.



Otherwise, repeat from the beginning.



We need to call the operator at most twice in expectation because each time z has probability $100/128$ to fall in the range we want. Therefore, our algorithm finishes in $O(1)$ expected time.

Exercise 2

Suppose that we enforce a harder constraint that you are allowed to call RANDOM(x, y) only with $x = 0$ and $y = 1$. Describe an algorithm to generate a uniformly random number in $[1, n]$ for an arbitrary integer n. Your algorithm must finish in $O(\log n)$ expected time.

Suppose n is a power of 2; then how can we use recursion to solve this problem?

1. Set $z = $RANDOM$(x, y)$.
2. If $z = 0$, we have a subproblem: generate a uniformly random number in the first half of the range;
   If $z = 1$, we have a subproblem: generate a uniformly random number in the second half of the range.

Considering the subproblem solved, we finish the algorithm.

Analysis of the Algorithm

$$
\begin{aligned}
f(1) &= O(1) \\
f(n) &\leq f(n/2) + O(1) \quad \text{, for } n > 1
\end{aligned}
$$

Thus, we have

$$f(n) = O(\log n)$$

Think: Why does the algorithm require n to be a power of 2?

Next, we will extend our algorithm to support values of n that are not powers of 2.

First, obtain the smallest power of 2 that is at least n.

- Try $1, 2, 4, ...,$ until reaching m such that $n \leq m < 2n$. This takes $O(\log n)$ time.

We have known how to generate a uniformly random number y in $[1, m]$ in $O(\log n)$ time.

If $y \leq n$, return y; otherwise, repeat the algorithm. At most 2 repeats are needed in expectation. The overall time is there $O(\log n)$ in expectation.

Exercise 3

Recall the k-selection problem:

> You are given a set S of n integers in an array and an integer $k \in [1, n]$. Find the k-th smallest integer of S.

Suppose there is a deterministic algorithm $\mathcal{A}_1$ which returns the median of n integers in $O(n)$ time. Can you use $\mathcal{A}_1$ as a blackbox to solve k-selection in $O(n)$ time?

Consider the following algorithm.

1. Get the median v of S from $\mathcal{A}_1(S)$.
2. Divide S into S_1 and S_2 where
   - $S_1 =$ the set of elements in S less than or equal to v;
   - $S_2 =$ the set of elements in S greater than v.
3. If $|S_1| \geq k$, then return $S' = S_1$ and $k' = k$; else return $S' = S_2$ and $k' = k - |S_1|$

Since $\mathcal{A}_1$ is deterministic, we always succeed in obtaining a subproblem with size no larger than $\lceil \frac{|S|}{2} \rceil$.

Analysis of the Algorithm

$$f(1) = O(1)$$
$$f(n) \le f(n/2) + O(n)$$

Thus, $f(n) = O(n)$.

What if $\mathcal{A}_1$ returns the $\lceil \frac{4}{5} n \rceil$-th smallest integer of n integers in $O(n)$ time. Can you still use $\mathcal{A}_1$ as a blackbox to solve k-selection in $O(n)$ time?

Instead of shrinking the size of subproblem by half, we shrink it by $\frac{4}{5}$.

We can still use $\mathcal{A}_1$ to shrink the problem size by a constant factor. From the geometric series we know that the total cost will be $O(n)$.

Think: If $\mathcal{A}_1$ returns the $\lceil \frac{99}{100} n \rceil$-th smallest integer of n integers in $O(n)$ time, can you still use $\mathcal{A}_1$ as a blackbox to solve k-selection in $O(n)$ time?

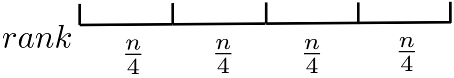
Exercise 4

Let's still focus on the k-selection problem. In the lecture, we shrink the input size of the subproblem into at most $\frac{2}{3}n$. Now, we want to shrink the input size into at most $\frac{n}{2}$. Give an algorithm to achieve the purpose in $O(n)$ expected time.

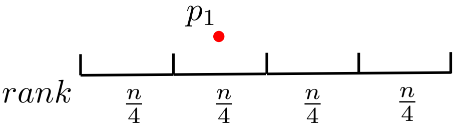
A simple solution: run our "$\frac{2n}{3}$-algorithm" twice. The number of remaining elements becomes at most $\frac{4n}{9}$.

Next, let us look at another way to achieve the purpose, assuming for simplicity that n is a multiple of 4.

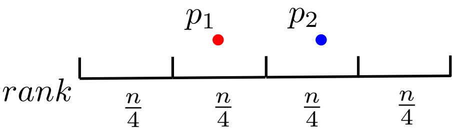
First, divide the rank space into 4 equal partitions.



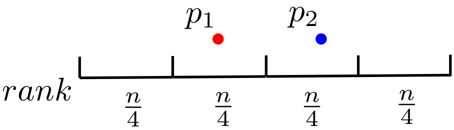
Second, take an element p_1 from S uniformly at random. Repeat until $rank(p_1)$ is in range $[\frac{n}{4}, \frac{n}{2}]$.



Third, take an element p_2 from S uniformly at random. Repeat until $rank(p_2)$ is in range $[\frac{1}{2}n, \frac{3}{4}n]$.



- If $k \leq rank(p_1)$, set $S' =$ the set of elements in S less than or equal to p_1, $k' = k$.
- If $rank(p_1) < k < rank(p_2)$, set $S' =$ the set of elements in S larger than p_1 and smaller than p_2, $k' = k - rank(p_1)$.
- If $k \geq rank(p_2)$, set $S' =$ the set of elements in S larger than or equal to p_2, $k' = k - rank(p_2)$.



In any case, we have $|S'| \leq \frac{n}{4} + \frac{n}{4} = \frac{n}{2}$.

In expectation, 4 repeats are needed for p_1, and 4 repeats for p_2 (think: why?).