Introduction to Algorithm Notes
Created by He Entong

1 Time Consumption of Bucket Sort

Since the time consumption of applying INSERTION SORT to every bucket is $O(n_i^2)$, and going through every bucket have time consumption of $\Theta(n)$, we obtain

$$\begin{aligned} T(n) &= \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \\ E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \end{aligned} \tag{1}$$

$$\begin{aligned} E[n_i^2] &= \sum_{x=0}^{n} x^2 P(n_i = x | \sum_{j=0}^{n-1} n_j = n) \\ &= \sum_{x=0}^{n} x^2 (\frac{1}{n})^x (\frac{n-1}{n})^{n-x} \\ &= \frac{d^2 t}{dt^2} M_{n_i}(t)\Big|_{t=0} \\ M_{n_i}(t) &= \sum_{x=0}^{n} (\frac{1}{n} e^t)^x (\frac{n-1}{n})^{n-x} \\ &= (\frac{1}{n} e^t + \frac{n-1}{n})^n \\ E[n_i^2] &= \frac{d^2 t}{dt^2} M_{n_i}(t)\Big|_{t=0} \\ &= e^t (\frac{1}{n} e^t + \frac{n-1}{n})^{n-1} + \frac{n-1}{n} e^{2t} (\frac{1}{n} e^t + \frac{n-1}{n})^{n-2}\Big|_{t=0} \\ &= 2 - \frac{1}{n} \end{aligned} \tag{2}$$

Substitute (2) into (1) we obtain

$$\begin{aligned} E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} O(2 - \frac{1}{n}) \\ &= \Theta(n) + O(2n-1) \\ &= \Theta(n) \end{aligned} \tag{3}$$

2 Structure of Recursion

2.1 Structure

A recursion should be consist of two parts: the basement return and sub-recursion return. That is

$$\text{Recursion} = \begin{cases} \textbf{Basement Return,} & \text{if there is no sub-problem for this recursion layer.} \\ \textbf{Recursion for Sub-problem,} & \text{if the problem is still dividable.} \end{cases}$$

2.2 Alternative Iteration

$$\text{Recursive Iteration} = \begin{cases} \textbf{Return after Loop Ends}, \text{ if the base return condition is met.} \\ \\ \textbf{Update the Analyzed Object with Sub-problem}, \\ \text{if the problem is still dividable} \end{cases}$$

3 Time Complexity of RANDOMIZED-SELECT Algorithm

The logic of the time consumption of the RANDOMIZED-SELECT algorithm can be written as

$$\text{PARTITION(array)} \begin{cases} \textbf{RANDOMIZED-SELECT(left array)}, \text{ if } i < k. \\ \textbf{return } k \textbf{ (Basement Return)}, \text{ if i == k} \\ \textbf{RANDOMIZED-SELECT(right array)}, \text{ if } i > k \end{cases} \tag{4}$$

Define the indicator random variable as

$$X_k = I\Big\{\text{The } kth \text{ smallest element is selected as the pivot}\Big\} \tag{5}$$

The process of recursion instantly come to an end when the targeted element is selected as the pivot. That is, $k == i$.

The time consumption of PARTITION algorithm is $O(n)$. To get the time ceiling we assume that the targeted element constantly falls into the right sub-array. In every randomized selection for pivot each element is equally to be selected. To get the time ceiling we assume that the targeted element constantly falls into the right sub-array. That gives us the following equation that

$$T(n) = \sum_{k=1}^{n} X_k \left(T\Big(\max(k-1, n-k)\Big) + O(n) \right) \tag{6}$$

Also

$$\max(k-1, n-k) = \begin{cases} k-1, & k > \lceil \frac{n}{2} \rceil \\ n-k, & k \leq \lceil \frac{n}{2} \rceil \end{cases} \tag{7}$$

Hence the original equation is

$$\begin{aligned} T(n) &= \sum_{k=1}^{n} X_k \left(T\Big(\max(k-1, n-k)\Big) + O(n) \right) \\ &= \sum_{k=1}^{n} X_k \; T\Big(\max(k-1, n-k)\Big) + \sum_{k=1}^{n} X_k \; O(n) \\ E\Big[T(n)\Big] &= E\Big[\sum_{k=1}^{n} X_k \; T\Big(\max(k-1, n-k)\Big)\Big] + E\Big[\sum_{k=1}^{n} X_k \; O(n)\Big] \\ &= \sum_{k=1}^{n} E\Big[X_k\Big] E\Big[T\Big(\max(k-1, n-k)\Big)\Big] + n \cdot \frac{1}{n} O(n) \\ &= \frac{1}{n}\Big\{ \sum_{k=1}^{\lceil \frac{n}{2} \rceil} E\Big[T(n-k)\Big] + \sum_{k=\lceil \frac{n}{2} \rceil + 1}^{n} E\Big[T(k-1)\Big] \Big\} + O(n) \\ &= \frac{1}{n}\Big\{ \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E\Big[T(k)\Big] + \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} E\Big[T(k)\Big] \Big\} + O(n) \\ &\leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E\Big[T(k)\Big] + O(n) \end{aligned} \tag{8}$$

This gives us a recurrence of a characteristic sequence that

$$\begin{aligned} X_n &= \frac{2}{n}\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} X_k + O(n) \\ &\leq \frac{2}{n}\left(n-1-\lfloor\frac{n}{2}\rfloor\right)\ \max(X_{\lfloor \frac{n}{2} \rfloor}, X_{\lfloor \frac{n}{2} \rfloor+1}, \ldots, X_n) + O(n) \\ &\leq \frac{2}{n}\left(n-\lceil\frac{n}{2}\rceil\right)\max\ (X_{\lfloor \frac{n}{2} \rfloor}, X_{\lfloor \frac{n}{2} \rfloor+1}, \ldots, X_n) + O(n) \\ &\leq \frac{2}{n}\left(n-\frac{n}{2}\right)\ \max\ (X_{\lfloor \frac{n}{2} \rfloor}, X_{\lfloor \frac{n}{2} \rfloor+1}, \ldots, X_n) + O(n) \\ &= \max\ (X_{\lfloor \frac{n}{2} \rfloor}, X_{\lfloor \frac{n}{2} \rfloor+1}, \ldots, X_n) + O(n) \end{aligned} \tag{9}$$

We give $X_n = O(n)$ as a tentative upper bound. Then

$$\begin{aligned} &\forall\ n \in N^+, \exists\ C_{max}, C' \\ &\max\ (X_{\lfloor \frac{n}{2} \rfloor}, X_{\lfloor \frac{n}{2} \rfloor+1}, \ldots, X_n) + O(n) \leq C_{max}n + C'n \\ &\therefore\ \ \exists\ C_n \leq C_{max} + C' \end{aligned} \tag{10}$$

Hence, $X_n = O(n)$ is a tenable attempt. We successfully conclude that the upper bound of the time complexity of RANDOMIZED-SELECT Algorithm is **linear**.

4 Binary Calculation Method

4.1 Transform Between Decimal System and Binary System

4.1.1 Decimal to Binary

We calculate the transformation recursively.

$$\text{DecToBin(int)} = \begin{cases} \text{ConnectRecords(Records)}, & \text{if int} == 0 \\ \text{DecToBin(int} - 2^{n-1}\text{)}, & \text{if } 2^{n-1} \leq \text{int} < 2^n\text{. Record } n \end{cases} \tag{11}$$

$$\text{That gives int} = \sum_{i \text{ in recorded n}} 2^i$$

ConnectRecords(Records) $= 0 *$ (max n recorded) with the n^{th} digit recorded overwritten as 1

4.1.2 Binary to Decimal

$$\text{BinToDec(bin)} = \sum_{n=0}^{\text{highest digit}} 2^n \delta_{1,n^{th}\ \text{digit}} \tag{12}$$

4.2 Multiplication

The multiplication in binary system obeys the following rule

$$\begin{aligned} &0 \times 0 \to 0 \quad 1 \times 0 \to 0 \\ &0 \times 1 \to 0 \quad 1 \times 1 \to 1 \end{aligned} \tag{13}$$

And the calculation is exactly the same as the one in decimal system.

5 Multiplication Hashing

We tend to make the output of every **hashing function** unique, that is, to minimize the quantity of collision. We use the hashing function

$$\begin{aligned} h(k) &= (\theta k \bmod 2^w)\text{rsh}(w-r) \\ \theta &\equiv 1(\text{mod}2), \quad 2^{w-1} < \theta < 2^w \end{aligned} \tag{14}$$

That means we use only r digits as the valid hash value.

6 Universal Hashing

6.1 Concept

We want to find a set of function when we tend to map the universe U to a hash table of m slots. The set, namely the **universal hashing**, has the following features.

$$\begin{aligned} \mathscr{H} &= \{h_1, h_2, h_3, \ \dots \ , h_m\} \\ \forall h_i \in \mathscr{H}, &\quad h_i : U \to \{0, 1, 2, \dots, m-1\} \\ \forall\ k, l \in U, &\quad \Pr\left\{h(k) = h(l), h \in U\right\} \leq \frac{1}{m} \end{aligned}$$

6.2 Implementation

We select an arbitrary prime m. In base m we decompose key k into $r+1$ digits.

$$\begin{aligned} k &\ \to \ < k_0, k_1, k_2, \dots, k_r >, \quad k_i \leq m-1 \\ a &\ \to \ < a_0, a_1, a_2, \dots, a_r >, \quad a_i \text{ randomly picked from } \{0, 1, 2, \dots, m-1\} \end{aligned} \tag{15}$$

The hash function is defined as

$$h_a(k) = (a \cdot k) \bmod m \tag{16}$$

Now we will give the proof on the rationality of the hash function above.

Assume that $x, y \ \in U$. No collision occurs requires that they have difference in at least one digit in base m. Since all the digits are symmetric in our assumption, we assume that the only difference occurs in digit 0, which gives us a upper bound of the probability of collision. Collision case gives us that

$$\begin{aligned} h_a(x) &= h_a(y) \\ (a \cdot k) \bmod m &= (b \cdot k) \bmod m \\ \sum_{i=0}^{r} a_i x_i &\equiv \sum_{i=0}^{r} a_i y_i \pmod m \\ a_0(x_0 - y_0) &\equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \pmod m \end{aligned} \tag{17}$$

We have the lemma that

$$\begin{aligned} &\forall \text{ prime } m, \quad \mathbb{Z}_m = \{1, 2, 3, \dots, m-1\} \\ &\forall\ z \in \ \mathbb{Z}_m, \quad \exists\ z^{-1} \in \ \mathbb{Z}_m, \quad z z^{-1} \equiv 1(\text{mod } m) \end{aligned}$$

Hence

$$\begin{aligned} &\exists\ (x_0 - y_0)^{-1} \in 0, 1, 2, \dots, m-1 \\ &a_0 \equiv -\sum_{i=1}^{r} a_i (x_i - y_i)(x_0 - y_0)^{-1} \pmod m \end{aligned} \tag{18}$$

It gives us that once other digits are fixed, there is only one choice for the different digit. We obtain that the probability for this case is that

$$\Pr\{\text{Collision Occurs with One-digit Difference}\} = \frac{m^r}{m^{r+1}} = \frac{1}{m} \tag{19}$$

Hence the hash equation is well-defined to be a **universal hash function**.

7 The Expectation of the Height to a Randomized BST

7.1 Introduction

When we wish to build a **Binary Search Tree** using a random number generator, some conclusion can be drawn that the BST we build will have a good performance in its average height. We will prove this point in the following part.

7.2 Jensen Inequality

If $F(x)$ is a convex function, and X is a random variable, then

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n} F(x_i) &\geq F(\frac{1}{n}\sum_{i=1}^{n} x_i) \\ F(E[X]) &\geq E[F(X)] \end{aligned} \tag{20}$$

7.3 Derivation

The height of a RANDOMIZED BST with n nodes is denoted as X_n. We use an auxiliary variable $Y_n = 2^{X_n}$ to make the derivation clear. In construction of a sub-tree, we use an indicator random variable

$$R_k = I\{\text{The } k^{th} \text{ element in the array be the parent}\} \tag{21}$$

Then

$$\begin{aligned} X_n &= 1 + \max(X_{k-1}, X_{n-k}) \\ Y_n &= 2 \times \max(Y_{k-1}, Y_{n-k}) \end{aligned} \tag{22}$$

Given that the k^{th} element is selected as the parent

So transit the form using the indicator random variable

$$\begin{aligned} E[Y_n] &= E[2\sum_{i=1}^{n} R_i \ \max(Y_{i-1}, Y_{n-i})] \\ &= 2\sum_{i=1}^{n} E[R_i \ \max(Y_{i-1}, Y_{n-i})] \\ &= 2\sum_{i=1}^{n} E[R_i] \ E[\max(Y_{i-1}, Y_{n-i})] \\ &\leq \frac{2}{n}\sum_{i=1}^{n} E[Y_{i-1}] + E[Y_{n-i}] \\ &= \frac{4}{n}\sum_{i=0}^{n-1} E[Y_i] \end{aligned} \tag{23}$$

So using the characteristic sequence again

$$\begin{aligned} a_n &= \frac{4}{n}\sum_{i=0}^{n-1} a_i \\ a_0 &= a_1 = O(1) \end{aligned} \tag{24}$$

We can see that the solution to a_n has the form of a polynomial. Assume that a_n is a combination C^3_{n+3}. Substitute in we obtain

$$\begin{aligned}\frac{4}{n}\sum_{i=0}^{n-1} C^3_{i+3} &= \frac{4}{n}C^4_{n+3} \\ &= \frac{4}{n}\frac{(n+3)(n+2)(n+1)n}{4!} \\ &= \frac{(n+3)(n+2)(n+1)}{3!} \\ &= C^3_{n+3}\end{aligned} \tag{25}$$

The assumption indeed make sense. Applying **Jensen Inequality** we conclude

$$\begin{aligned}2^{E[X_n]} \leq E[2^{X_n}] = E[Y_n] \leq C^3_{n+3} = O(n^3) \\ E[X_n] \leq \lg\Big(O(n^3)\Big) = O(\lg n)\end{aligned} \tag{26}$$

Hence the RANDOMIZED BST has a low-order expectation in height, which gives a good performance in all the manipulations applied on it.

8 Implementation of Matrix Multiplication in Python

Assume that two matrix, denoted as M_A, M_B, where A.col == B.row

$$\begin{aligned}M_A &= [\ [A_{ij} \text{ for i in range(A.row)}] \text{ for j in range(A.col) }] \\ M_B &= [\ [B_{ij} \text{ for i in range(B.row)}] \text{ for j in range(B.col) }]\end{aligned} \tag{27}$$

Then the multiplication function $Mul(M_A, M_B)$ is defined as

$Mul(M_A, M_B) =$
[[$Multiplication$($M_A[i], [M_B[k][j]$ for k in range(B.row)]) for j in range(B.col)] for i in range(A.row)]

Where $Multiplication$ function returns the inner product of two vectors.

9 BOTTOM-UP RECURSION: Optimizing Time Complexity with Extra Storage

When creating a recursion, if the recursion to the relevant sub-problem is called every time a new superior sub-problem is called, we call the recursion **sub-problem overlapped**. This largely affect the total complexity of the program. To optimize, we can use an extra array to store the sub-problems that has been called and calculated. This is the **Bottom-up Recursion**

9.1 Implementation

- Decide the **sub-problem domain**, and initialize the array to get ready for storage
- - **Find the state shifting equation**
  - **Iteration** Calculate the sub-problems and uplift the pointers, the superior problem is the combination of sub-problems

Using the **Bottom-up Rod Cutting** program to measure the maximum profit. Dictionary is used to store the calculated sub-problems.

```
def LCS(str1, str2):
    for i in range(-1, len(str1)):
        for j in range(-1, len(str2)):
            Dict[(i, j)] = 0
    for i in range(len(str1)):
        for j in range(len(str2)):
            if str1[i] == str2[j]:
                Dict[(i, j)] = Dict[(i-1, j-1)] + 1
            elif Dict[(i-1, j)] >= Dict[(i, j-1)]:
                Dict[(i, j)] = Dict[(i-1, j)]
            else:
                Dict[(i, j)] = Dict[(i, j-1)]
    D = {v:k for (k, v) in Dict.items()}
    return max(D)
```

Caution: In the process of separating the problems, if we can guarantee that in the two sub-problems generated, one is definitely an empty problem to reach the optimal solution, we can directly turn to **Greedy Algorithm**.

10 Catalan Number

We are interested in the number of different binary trees with n nodes, which following will deduce. The binary tree mentioned above can be expressed as the combination of a root node and the summation of two sub-trees.

$$b_n = \sum_{i=0}^{n-1} b_i b_{n-i-1} \tag{28}$$

$$\text{Base Case: } \ b_0 = 1$$

The generating function of the sequence will give

$$G(b,x) = \sum_{n=0}^{\infty} b_n x^n \tag{29}$$

An implicit feature of $G(b,x)$ is

$$\begin{aligned} G(b,x) &= \sum_{n=1}^{\infty}\left(\sum_{i=0}^{n-1} b_i b_{n-i-1}\right)x^n + b_0 \\ &= \sum_{n=0}^{\infty}\left(\sum_{i=0}^{n} b_i b_{n-i}\right)x^{n+1} + 1 \\ &= x\cdot\sum_{n=0}^{\infty}\left(\sum_{i=0}^{n} b_i b_{n-i}\right)x^{n} + 1 \\ &= xG^2(b,x) + 1 \\ \text{Or, } \ G(b,x) &= \frac{1-\sqrt{1-4x}}{2x} \\ &= \frac{1}{2x}\left(1-\left(1-2x-\sum_{n=2}^{\infty}\frac{(2n-3)!}{2^{2n-2}(n-2)!}\frac{1}{n!}(4x)^n\right)\right) \\ &= \sum_{k=0}^{\infty}\frac{(2n)!}{n!n!}\frac{1}{n+1}x^n \end{aligned} \tag{30}$$

Compare (29) and (30) we obtain

$$\begin{aligned} b_n = \frac{(2n)!}{n!n!}\frac{1}{n+1} &= \frac{1}{n+1}C_{2n}^n \\ &= \frac{1}{n+1}\frac{\sqrt{4\pi n}(\frac{2n}{e})^{2n}(1+\Theta(\frac{1}{n})}{(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))^2} \\ &= \frac{4^n}{\sqrt{\pi}n^{\frac{3}{2}}}\left(1+O(\frac{1}{n})\right) \end{aligned} \tag{31}$$

Hence we obtain the exact growing order of a binary search tree of n nodes.

11 Longest Common Sub-string with k Mismatches

11.1 A Degraded Model

If there is no 'k mismatches' condition, the state shifting equation will be

$$\text{LCS}[i,j] = \begin{cases} \text{LCS}[i-1,j-1], & \text{if str1[i]} \neq \text{str2[j]} \\ \max\{\text{LCS}[i-1,j-1]+1,\ \text{LCS}[i-1,j],\ \text{LCS}[i,j-1]\}, & \text{if str1[i]} == \text{str2} \end{cases} \tag{32}$$

With the base case

$$\text{LCS}[0,j] == \text{LCS}[i,0] == 0$$

11.2 k Mismatches

Compared to the degraded model, some limitations are added such that

$$\left|\left|\left\{0 \leq h \leq l-1 \middle| str1[i-h] \neq str2[j-h]\right\}\right|\right| \leq k, \quad 0 \leq i < n, \quad 0 \leq j < m \tag{33}$$

$$\text{LCS}[i,j] = \max\{l \mid l \leq \min(i,j)+1\} \tag{34}$$