A&DS Worksheet 2 solutions

Dimitri Tabatadze

October 3, 2023

1. By induction on n, it can be proven that

$$\left\lceil \frac{n}{2} \right\rceil = \left\lfloor \frac{n+1}{2} \right\rfloor.$$

Base case. $n = 1, n = 2$

$$\left\lceil \frac{1}{2} \right\rceil = \left\lfloor \frac{1+1}{2} \right\rfloor \implies 1 = 1$$
$$\left\lceil \frac{2}{2} \right\rceil = \left\lfloor \frac{2+1}{2} \right\rfloor \implies 1 = 1$$

Induction step. $n \to n+2$

$$\left\lceil \frac{n+2}{2} \right\rceil = \left\lceil \frac{n}{2} + 1 \right\rceil = \left\lceil \frac{n}{2} \right\rceil + 1 = \left\lfloor \frac{n+1}{2} \right\rfloor + 1 = \left\lfloor \frac{n+1}{2} + 1 \right\rfloor = \left\lfloor \frac{n+2+1}{2} \right\rfloor.$$

Same goes for

$$\frac{n}{2} - \frac{1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2} \iff$$
$$\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2}$$

Base case. $n = 1, n = 2$

$$\frac{1-1}{2} \leq \left\lfloor \frac{1}{2} \right\rfloor \leq \frac{1}{2} \implies 0 \leq 0 \leq \frac{1}{2}$$
$$\frac{2-1}{2} \leq \left\lfloor \frac{2}{2} \right\rfloor \leq \frac{2}{2} \implies \frac{1}{2} \leq 1 \leq 1$$

Induction step. $n \to n+2$

$$\begin{aligned}
\frac{(n+2)-1}{2} \leq \left\lfloor \frac{(n+2)}{2} \right\rfloor \leq \frac{(n+2)}{2} \quad &\iff \\
\frac{n-1}{2} + 1 \leq \left\lfloor \frac{n}{2} + 1 \right\rfloor \leq \frac{n}{2} + 1 \quad &\iff \\
\frac{n-1}{2} + 1 \leq \left\lfloor \frac{n}{2} \right\rfloor + 1 \leq \frac{n}{2} + 1 \quad &\iff \\
\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2}
\end{aligned}$$

We used this to prove lemma 1 on slide 9. Lemma 1 says that every step of

2. It can be proven that a binary tree with depth d has at most 2^d leaves by induction on d.

   Let $n = 2^d$ be the induction hypothesis.

   Base case. $d = 0$ A binary tree with depth $d = 0$ can only consist of a single node, which is both the root and the only leaf of the tree. Hence, this tree has one leaf $n = 2^0$.

   Induction step. If we had a tree with d depth and $n = 2^d$ leaves, we can give every leaf two (the maximum possible amount) children. This will increase the depth to $d + 1$ and double the leaf count giving us $n = 2^{d+1}$.

3. For the given data, we can say that

$$\begin{aligned}
W &= (S, p) \\
S &= \{(\mathrm{M}, \mathrm{S}), (\mathrm{M}, \mathrm{DNS}), (\mathrm{F}, \mathrm{S}), (\mathrm{F}, \mathrm{DNS})\} \\
p(x) &= \begin{cases} 0.18 & \text{if } x = (\mathrm{M}, \mathrm{S}) \\ 0.31 & \text{if } x = (\mathrm{M}, \mathrm{DNS}) \\ 0.14 & \text{if } x = (\mathrm{F}, \mathrm{S}) \\ 0.37 & \text{if } x = (\mathrm{F}, \mathrm{DNS}) \end{cases}
\end{aligned}$$

   (1) $\frac{0.18}{0.32} = \frac{9}{16}$
   (2) $\frac{0.18}{0.49} = \frac{18}{49}$
   (3) $\frac{0.37}{0.68} = \frac{37}{68}$

4. (1) Let $W_1 = (S_1, p_1)$ be the probability space of a 6 sided die, then

$$p_1(i) = \frac{1}{6},\ i \in S_1$$

where $S_1 = \{1, 2, 3, 4, 5, 6\}$. The probability space of two independent six sided dice would then be $W = W_1 \times W_1 = (S, p)$. Now, with $A_i = \{(a, b)|a + b = i\}$ the probability $p(A_i)$ can easily be defined as

$$p(A_i) = |A_i| \cdot \frac{1}{36}$$

where $|A_i|$ is the size of the set A_i, which can also be written as

$$|A_i| = 6 - |i - 7|$$

so we get

$$p(A_i) = \frac{6 - |i - 7|}{36}$$

(2) $E(a + b) = E(a) + E(b)$

5. Custom linked list class:

```
package Mergesort;

public class LinkedList {
    int value = 0;
    LinkedList next = null;

    public LinkedList(int value) {
        this.value = value;
    }

    public static LinkedList fromList(int[] arr, int n) {
        LinkedList res = new LinkedList(arr[0]);
        LinkedList res_head = res;
        for (int i = 1; i < n; i++) {
            res.next = new LinkedList(arr[i]);
            res = res.next;
        }
        return res_head;
    }

    public LinkedList nth(int n) {
        LinkedList res = this;
        while (res != null && n-- > 0) {
            res = res.next;
        }
        return res;
    }

    public int length() {
        int l = 1;
        LinkedList s = this.next;
        while (s != null) {
            l++;
            s = s.next;
        }
        return l;
    }

    public void print() {
        print(length());
    }

    public void print(int d) {
        LinkedList s = this;
        while (s != null && d-- >= 0) {
            System.out.print("" + s.value);
            if (s.next != null) {
                System.out.print(" ");
            }
            s = s.next;
        }
        System.out.println("");
    }
}
```

(1)

```
package Mergesort;

public class One {
    /// sort the first n elements
    static LinkedList mergeSort(LinkedList arr, int n) {
        if (n == 1) {
            return arr;
        }

        LinkedList tail = arr.nth(n);
        LinkedList a = mergeSort(arr, n/2);
        LinkedList b = mergeSort(arr.nth(n / 2), n - n/2);

        int al = n / 2;
        int bl = n - n / 2;

        LinkedList r = new LinkedList(69420);
        LinkedList res = r;
        for (int i = 0; i < n; i++) {
            if (bl == 0 || (al > 0 && a.value <= b.value)) {
                r.next = a;
                a = a.next;
                al--;
            } else {
                r.next = b;
                b = b.next;
                bl--;
            }
            r = r.next;
        }
        r.next = tail;
        return res.next;
    }
}
```

(2)

```java
package Mergesort;

public class Two {
    static LinkedList mergeSort(LinkedList arr, int n) {
        LinkedList head = new LinkedList(666);
        head.next = arr;

        LinkedList res = new LinkedList(69420);
        for (int w = 2; w / 2 < n; w *= 2) {
            for (int h = 0; h + w / 2 < n; h += w) {
                LinkedList tail = head.nth(h + w + 1);
                LinkedList a = head.nth(h + 1);
                LinkedList b = head.nth(h + w / 2 + 1);

                int al = w / 2;
                int bl = Math.min(w / 2, n - h - w/2);

                LinkedList r = res;
                for (int i = 0; i < w; i++) {
                    boolean pick_a = bl == 0;
                    if (!pick_a) {
                        pick_a = al > 0 && a.value <= b.value;
                    }

                    if (pick_a) {
                        r.next = a;
                        a = a.next;
                        al--;
                    } else {
                        r.next = b;
                        b = b.next;
                        bl--;
                    }
                    r = r.next;
                }
                r.next = tail;
                head.nth(h).next = res.next;
            }
        }
        return head.next;
    }
}
```

(3)

```java
package Mergesort;

public class Three {
    static int min(int a, int b) {
        return a < b ? a : b;
    }

    static void mergeSort(int[] arr) {
        int[] brr;

        for (int p = 2; p / 2 <= arr.length; p *= 2) {
            brr = arr.clone();
            for (int l = 0; l < arr.length; l += p) {
                if (arr.length - l < p / 2) break;

                int r = min(l + p - 1, arr.length - 1);
                int m = l + p / 2 - 1;

                int L = l, R = m + 1;
                for (int i = l; i <= r; i++) {
                    if (L > m || R > r) {
                        if (L > m) {
                            arr[i] = brr[R];
                            R++;
                        } else {
                            arr[i] = brr[L];
                            L++;
                        }
                    } else {
                        if (brr[L] > brr[R]) {
                            arr[i] = brr[R];
                            R++;
                        } else {
                            arr[i] = brr[L];
                            L++;
                        }
                    }
                }
            }
        }
    }
}
```