HW1 - Theoretical part

Xinyuan Cao
Uni: xc2461

September 25, 2018

1. (a) Use induction to prove that Horner's rule can solve this problem.
   i. **base** $n = 0$.
   From the definition of polynomial, we have $p(x) = a_0$. From the routine, we have $z = a_0$. So $p(x) = z$, and we have Horner's rule holds for $n = 0$.
   ii. **Inductive Hypothesis** Assume that Horner's rule is true for $n \geq 1$, that is, $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = z$ for n.
   iii. **Inductive Step** For $n + 1$, we consider the first loop.
   Initially, $z = a_{n+1}$. When $i = n$, $z = zx + a_n = a_{n+1}x + a_n$.
   Denote $z' = a_{n+1}x + a_n$.
   The following loops are the same as the loops in the situation for n, with the initialization $z = a_{n+1}x + a_n$.
   Denote $p(x) = a'_0 + a'_1 x + \cdots + a'_n x^n$, where $a'_i = a_i$ for each $1 \leq i \leq n - 1$, and $a'_n = a_{n+1}x + a_n$.
   Using the hypothesis, we finally get $z = p'(x)$.
   So we have $z = p'(x) = a_0 + a_1 x + \cdots + a'_n x^n = a_0 + a_1 x + \cdots + (a_{n+1}x + a_n)x^n = a_0 + a_1 x + \cdots + a_n x^n + a_{n+1}x^{n+1}$.
   iv. **Conclusion** It follows that Horner's rule solves the problem and stores the solution in z for all n. #

   (b) There are n loops in the routine. Inside each loop, $z = zx + a_i$ contains one addition and one multiplication. So there are totally n additions and n multiplications in this routine. So the running time of this routine is $O(n)$.
   So in all, the routine uses n additions and n multiplications as a function of n.
   Let $n = 2^k$, $p(x) = x^n$. We define an algorithm as below:

   Calculate Polynomial(x^{2^k})
       if $k == 0$ **then**
           return x
       else return [**Calculate Polynomial**($x^{2^{k-1}}$)]^2
       end if

   Firstly, I will use induction prove its correctness.

i. **Base** $k = 0$. Then return x. $p(x) = x$.It's correct.

ii. **Inductive Hypothesis** Assume that the algorithm is correct for $k \geq 0$. So we have the result of the algorithm is equal to $p(x) = x^{2^k}$

iii. **Inductive Step** For $k + 1$, the result of the algorithm is the square of the result for k, that is the square of x^{2^k}. So the result for $k+1 = x^{2^{k+1}} = p(x)$. So it holds for $k + 1$.

iv. **Conclusion** The algorithm is correct for all k.

Next, I'll prove this method is better. Assume $T(2^k)$ is the running time of this method. So we get,

$$T(2^k) = T(2^{k-1}) + \Theta(1)$$
$$T(n) = T(n/2) + \Theta(1)$$

By Master Theorem, we have $T(n) = \Theta(\log n)$.
For the method in (a), the running time is $O(n)$. So my method is better. #

2. Let $v = (v_1, v_2, \ldots, v_n)^T$, and denote $v_{p:g} = (v_p, c_{p+1}, \ldots, c_q)$. The algorithm is shown as below.

For H_k and v, the inputs are k and v.

Compute matrix-vector(k, v)
  if $k == 0$ **then**
    return v
  else
    vector1 = **Compute matrix-vector(**$k - 1$**,**$v_{1:2^{k-1}}$**)**
    vector2 = **Compute matrix-vector(**$k - 1$**,**$v_{2^{k-1}+1:2^k}$**)**
    return $\binom{vector1+vector2}{vector1-vector2}$
  end if

Then I'll prove its correctness.

(a) **Base** $k = 0$. Because H_0 is 1*1 matrix (1), v is a real number. So $H_0 v = v$. So the algorithm is correct for $k = 0$.

(b) **Inductive Hypothesis** Assume the algorithm is correct for $k \geq 0$. So for any column vector v with length of $n = 2^k$, the output of **Compute matrix-vector)**(k, v) **is equal to** $H_k v$.

(c) **Inductive Step** For $k + 1$, we have

$$H_{k+1} = \begin{pmatrix} H_k & H_k \\ H_k & -H_k \end{pmatrix}$$

For the vector v, we have $v = \begin{pmatrix} v_{1:2^k} \\ v_{2^k+1:x^{k+1}} \end{pmatrix}$, where $v_{1:2^k}$ and $v_{2^k+1:1}$ are two column vectors of length 2^k. From the inductive step, we have

$$vector1 = \textbf{Compute matrix-vector}(k, v_{1:2^k}) = H_k v_{1:2^k}$$

$$vector2 = \textbf{Compute matrix-vector}(k, v_{2^k+1:2^{k+1}}) = H_k v_{2^k+1:2^{k+1}}$$

$$H_{k+1}v = \begin{pmatrix} H_k v_{1:2^k} + H_k v_{2^k+1:2^{k+1}} \\ H_k v_{1:2^k} - H_k v_{2^k+1:2^{k+1}} \end{pmatrix} = \begin{pmatrix} vector1 + vector2 \\ vector1 - vector2 \end{pmatrix}$$

So $H_{k+1}v$ is equal to the result of the algorithm. It's correct for $k+1$.

(d) **Conclusion** The algorithm is correct for all k.

Finally we will analyze the running time, denote $T(n) = T(2^k)$.

From the algorithm we have,

$$T(n) = 2T(n/2) + \Theta(n)$$

By Master Theorem, we get $T(n) = O(n \log n)$. #

3. (a) Let the observations are $y_1, y_2, \ldots y_n, \ldots$, the samples we maintain are $X_1, X_2, \ldots, X_n, \ldots$ y_i denotes the i_{th} observation, and X_i is the sample we store right after the i_{th} item appears.

   Based on the algorithm, after the k-th item appears,

$$P(X_k = y_k) = 1/k$$

$$P(X_k = X_{k-1}) = 1 - 1/k$$

   Then I'll prove the correctness with induction.

   i. **Base** k=1. On one hand, there is only one observation y_1, so we should store $X_1 = y_1$. On the other hand, by the algorithm, $P(X_1 = y_1) = 1$, so $X_1 = y_1$. The algorithm is correct.

   ii. **Inductive Hypothesis** Assume the algorithm is correct for $k \geq 1$. That is X_k is uniformly distributed over $y_1, \ldots, y_k$. That is, for $1 \leq i \leq k$, we have,

$$P(X_k = y_i) = 1/k$$

   .

   iii. **Inductive Step** For $k+1$, by algorithm we have,

$$P(X_{k+1} = y_k) = 1/(k+1)$$

$$P(X_{k+1} = X_k) = 1 - 1/(k+1) = k/(k+1)$$

   For $1 \leq i \leq k$, using the hypothesis we get,

$$\begin{aligned} P(X_{k+1} = y_i) &= P(X_{k+1} = X_k, X_k = y_i) = P(X_{k+1} = X_k)P(X_k = y_i) \\ &= \frac{k}{k+1}\frac{1}{k} = \frac{1}{k+1} \end{aligned}$$

   So we have, for $1 \leq i \leq k+1$, $P(X_{k+1} = y_i) = 1/(k+1)$. That shows the algorithm is right for $k+1$.

iv. **Conclusion** This algorithm solves the problem. #

(b) By new algorithm we have,

$$P(X_k = y_k) = 1/2$$

$$P(X_k = X_{k-1}) = 1/2$$

Then I will use induction to prove the following distribution:

$$P(X_k = y_i) = \begin{cases} (\frac{1}{2})^{k-i+1} & 2 \le i \le k \\ (\frac{1}{2})^{k-1} & i = 1 \end{cases}$$

i. **Base** $k = 1$, $P(X_1 = y_1) = 1 = (1/2)^0$. It satisfies the distribution.

ii. **Inductive Hypothesis** Assume the above distribution is correct for $k \ge 1$. That is,

$$P(X_k = y_i) = \begin{cases} (\frac{1}{2})^{k-i+1} & 2 \le i \le k \\ (\frac{1}{2})^{k-1} & i = 1 \end{cases}$$

iii. **Inductive Step**
For $k + 1$, we have

$$P(X_{k+1} = y_{k+1}) = 1/2 = (1/2)^{(k+1)-(k+1)+1}$$

For $2 \le i \le k$,

$$\begin{aligned} P(X_{k+1} = y_i) &= P(X_{k+1} = X_k, X_k = y_i) = P(X_{k+1} = X_k)P(X_k = y_i) \\ &= \frac{1}{2}(\frac{1}{2})^{k-i+1} = (\frac{1}{2})^{(k+1)-i+1} \end{aligned}$$

For $i = 1$,

$$\begin{aligned} P(X_{k+1} = y_1) &= P(X_{k+1} = X_k, X_k = y_1) = P(X_{k+1} = X_k)P(X_k = y_1) \\ &= \frac{1}{2}(\frac{1}{2})^{k-1} = (\frac{1}{2})^{(k+1)-1} \end{aligned}$$

So it's correct for $k + 1$.

iv. **Conclusion** So the distribution of the stored item is:

$$P(X_k = y_i) = \begin{cases} (\frac{1}{2})^{k-i+1} & 2 \le i \le k \\ (\frac{1}{2})^{k-1} & i = 1 \end{cases}$$

4. (a) Denote the entries of A, B, C as a_{ij}, b_{ij}, c_{ij}, $1 \le i, j \le n$.

Input is theree matrices A, B, C.If $AB = C$, the output is True, else False.

Check Matrices(A, B, C)
 //Check each element of AB and C
 for $i = 1$ to n **do**
  for $j = 1$ to n **do**
   //compute each element of AB
   $x_{ij} = 0$
   for $k = 1$ to n **do**
    $x_{ij} = x_{ij} + a_{ik}b_{kj}$
   end for
   if $z_{ij} \neq c_{ij}$
    return False
   end if
  end for
 end for
return True

This algorithm is naturally true because I calculate each element of AB by matrix multiplication, and compare it with the corresponding element of C.

There are three loops (n times for each) in this algorithm. In the most inside loop, the running time is $O(1)$. So the running time of this algorithm is $O(n^3)$.

(b) i. Since M is a non-zero matrix, there is definitely a non-zero element. Let's assume $m_{pq} \neq 0$. Denote $m_p = (m_{p1}, m_{p2}, \ldots, m_{pn})$. Because $Mx = 0$ ensures each of the element of Mx is equal to 0. So $Pr[Mx = 0] \le Pr[m_p x = 0]$.

Because $x = 0$ fits all $m_p x = 0$, there exists a x that satisfies $m_p x = 0$. Assume that $x = (x_1, x_2, \ldots x_n)^T$ satisfies $m_p x = 0$.

Define a map $\phi : R^n \to R^n$, s.t. $\phi(x) = x^*$, where $x^* = (x_1, x_2, \ldots, (1 - x_q), \ldots, x_n)^T$. ϕ is injective. Then

$$m_p x^* = m_p x + (1 - 2x_q)m_{pq} \neq m_p x = 0$$

The inequality holds because $x_q \in \{0, 1\}$.

Each element of x is equally likely to be 1 or 0. For each x satisfies, we have $\phi(x)$ does not satisfy, where ϕ is an injective function. So $Pr[m_p x = 0] \le 1/2$

Therefore $Pr[Mx = 0] \le Pr[m_p x = 0] \le 1/2$. #

ii. Let $M = AB - C$. Because $AB \neq C$, we have M is a non-zero matrix. Using the result of 4(b)i, we have $Pr[Mx = 0] \le 1/2$. So we show $Pr[ABx = Cx] = Pr[(AB - C)x = 0] = Pr[Mx = 0] \le 1/2$. #

The new algorithm is shown as below.

Rand Check Matrices(A, B, C)

```
for i = 1 to n do
    x_i = Bernoulli(1/2)
end for
//Compute ABx = y
for j = 1 to n do
    ω_j = 0
    for k = 1 to n do
        ω_j = ω_j + b_jk x_k
    end for
end for
for i = 1 to n do
    y_i = 0
    for j = 1 to n do
        y_i = y_i + a_ij ω_j
    end for
end for
//Compute Cx = z
for i = 1 to n do
    z_i = 0
    for j = 1 to n do
        z_i = z_i + c_ij x_k
    end for
end for
// Compare y and z
for i = 1 to n do
    if y_i ≠ z_i
        return False
    end if
end for
return True
```

The running time of the algorithm is

$$T(n) = \Theta(n) + O(n^2) + O(n^2) + O(n^2) + O(n) = O(n^2)$$

Then I'll compute its success probability.

$$Pr[ABx \neq Cx|AB \neq C] = 1 - Pr[ABx = Cx|Ab \neq C] \geq 1/2$$

$$Pr[ABx = Cx|AB = C] = 1$$

So the probability that it returns the correct answer is no less than 1/2.

To improve the success probability, we could implement the above algorithm for many times. If each time returns True, then we think $AB = C$, else $AB \neq C$.
The detailed algorithm is shown is below. The input is matrices A, B, C and the times that **Rand Check Matrices** will be implemented.

Improved Rand Check Matrices(A, B, C, n)
    for $i = 1$ to n **do**
        if Rand Check Matrices(A, B, C) $==$ False
            return False
        end if
    end for
    return True

Finally I'll prove it's a better algorithm with respect to success rate.

$$\begin{aligned} Pr[\text{it's incorrect when } AB \neq C] &= Pr[ABx_1 = Cx_1, ABx_2 = Cx_2, \ldots, ABx_n = Cx_n] \\ &= Pr[ABx_1 = Cx_1]Pr[ABx_2 = Cx_2]\ldots Pr[ABx_n = Cx_n] \\ &\leq (1/2)^n \end{aligned}$$

So we have the success probability is no less than $1 - (1/2)^n$. #