CSE 321_ Homework 04

Aikeboer Aizezi
131044026

1. Horner's Algorithm: (for reference)

Input: $f, g \in Z[x], i, l$.

Output: $F(g) = a_{i+l-1} g^{l-1} + \dots + a_{i+1} g + a_i$.

Res = $a_{i+l-1}$

For $j = l-2$ down to $j=0$ do:

Res = Res $\cdot g$

Res = Res + $a_{i+j}$

Return Res.

Devide and Conquer Algorithm:

Input: $f, g \in Z[x]$

Output: $h = f(g)$

let $l = 4$, $i = 1$, $k_i = \lceil \frac{n-1}{l} \rceil$

for $j = 0$ to $k_i - 1$

$h_{i,j}$ = HornersAlgorithm$(f, g, jl, l)$

$G = g^l$

while ($k_i > 1$) do:

$k_{i+1} = \lceil k_i / 2 \rceil$

for $j=0$ to $k_{i+1}$ do:

$h_{i+1,j} = h_{i,2j} + h_{i,2j+1} \cdot G$

clear $h_{i,2j}$ and $h_{i,2j+1}$

If $k+1 > 1$ then $G = G^2$

$i = i+1$

return $h = h_{i,0}$.

No. of additions: $\sum_{l-2}^{0} 1 + \sum_{k_i}^{l} \sum_{j=0}^{k_{i+1}} 1 = (n \cdot m \log(n) \cdot \log(m \cdot n))$

multiplications: $(n \cdot m \cdot \log(n) \cdot \log(nm))$

2.
```
def indexFind (A, i, j, key) {
    int mid = (i+j)/2 ;
    if (A[mid] == key && mid == key);
        return mid;
    else if (A[mid] < key)
        return indexFind (A, i, mid-1, key);
    else
        return -1;
}
```

3. let $n$ be the number of disks.

let $D$ be a direction, $\bar{D}$ be the opposite direction.

if n is odd ;

move smallest disk in direction $D$ (from peg 1 to 2, or from peg 2 to 3)

if $n$ is even :

move smallest dis in direction $\bar{D}$ (from peg 3 to 2, or from peg 2 to 1).

make the only other legal move.

4. Input ; $A[0 \dots n-1]$, $(i \le j)$

Output : smallest (min), largest (max) elements in the array

```
def findMinMax(A[i...j], min, max){
    if (i == j){
        min = A[i];
        max = A[i];
    }
    elseif ( (i-j) == 1){
        if (A[i] == A[j]){
            min = A[i] ;
            max = A[j];
        } else {
            min = A[j];
            max = A[i];
        }
    else {
        m = (i+j)/2 ;
        findMinMax (A[i...m], min, max);
        findMinMax (A[m...j], Min, Max);
        if(min < Min)
            min = Min;
        if(max < Max)
            max = Max;
    }
}
```

⑤ We can represent this problem using a binary tree, where the parential nodes represents breakable pieces and leaves represent 1-by-1 pieces of the original bar. Since only one bar can be broken at a time, any break increases the number of the pieces by 1. Hence to get $n \cdot m$ 1-by-1 pieces from n-by-n piece $nm-1$ breaks are needed.

proof by induction:

(1) when there is only one square we need no breaks.

(2) Assume that for numbers $1 \le m < N$ we have already shown that it takes exactly $m-1$ breaks to split a bar consisting of $m$ squares. If we take a bar with $N > 1$ squares and then splitting that bar into two with $m_1$ and $m_2$ squares. By the induction it will take $m_1 - 1$ breaks to split the first bar and $m_2 - 1$ to split the second one. the total will be $1 + (m_1 - 1) + (m_2 - 1) = N - 1$ since $m_1 + m_2 = N$.

⑥ (a) In both techniques we devide a problem into smaller instances of the same problem.

(b) Devide and conquer algorithms don not store solutions to smaller instances while dynamic programming algorithms store solutions to smaller instances.

⑦ World series odds:

(a) If team A wins a game whose probablity is $p$, A will need $i-1$ more wins to win the series while B will need $j$ wins. and if A looses a game whose probability is $q = 1 - p$, A will need $i$ wins while B will need $j-1$ more wins to win the series.

So $P(i,j) = pP(i-1,j) + qP(i,j-1)$ for $i$ and $j$ bigger than 0.

the initial conditions are: $P(0,j)=1$ for $j>0$ and $P(i,0)=0$ for $i>0$.

(b) Dynamic programming table with values rounded-off to two digits after the dot.

| i \ j | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | - | 1 | 1 | 1 | 1 |
| 1 | 0 | 0.4 | 0.64 | 0.78 | 0.87 |
| 2 | 0 | 0.16 | 0.35 | 0.52 | 0.66 |
| 3 | 0 | 0.06 | 0.18 | 0.32 | 0.46 |
| 4 | 0 | 0.03 | 0.09 | 0.18 | 0.29 |

$P(0,j)=1$ for $j=1$ to $j=4$.

$P(i,0)=0$ for $i=1$ to $i=4$.

Since the probability of team A winning a game is 0.4 then $P(1,1)=0.4$.

$P(1,2)=1-q^2=1-(0.6)^2=0.64$

$P(1,3)=1-q^3=0.78$

$P(1,4)=1-q^4=0.87$

$q=1-p$ probability of team A to lose.

$P(2,1)=P(1,1)\cdot 0.4=0.16$

$P(3,1)=P(2,1)\cdot 0.4=0.064$

$P(4,1)=P(3,1)\cdot 0.4=0.27$.

$\vdots \qquad \vdots \qquad \vdots$

$P(4,4) \simeq 0.29$

(c) Input: The number of wins N needed to win the series and the probability P of one particular team to win a game.

```
def worldSeries (m, p){
    double P[n+1][n+1];
    int q = 1-p;
    for (int j=1 ,i<=n, ++j)
        P[0][j] = 1.0;
    for (int i=1 , i<=n, ++i){
        P[i][0] = 0;
        for(int j=1 , j<=n, ++j)
            P[i][j] = p* P[i-1][j] + q*P[i][j];
    }
    return P[n][n];
}
```

Both the time and the space efficiency are in $\Theta(n^2)$. because each entry of the n+1 by n+1 matrix is computed in $\Theta(1)$ time.

8. let the binary matix be M[R][C], auxiliary matrix S.

(1) Construct a sum matrix S[R][C] for the given M[R][C].

a) Cope first row and fist column as it is from M[][] to S[][]

b) for other enties, use following expressions to construct S[][]

if M[i][j] is 1:

S[i][j] = min (S[i][j-1], S[i-1][j], S[i-1][j-1]) +1.

else

S[i][j]=0.

2) find the maximum entry in S[R][C].

3) Using the value and coordinates of maximum entry in S[i], print submatrix of M[][].