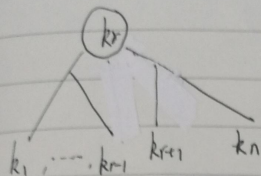
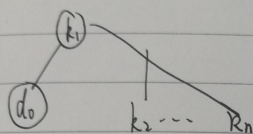


Exercise 1.

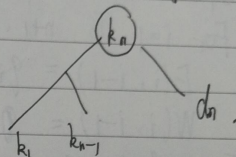
(a) $K = \{k_1, k_2, \dots, k_n\}$ 的优化解的根必为 K 中某个 k_r



① 若 $r=1$, 左子树仅包含 d_0



② 若 $r=n$, 右子树仅包含 d_n



如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, \dots, k_j\}$ 的子树 T' , 则 T' 是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 的子问题的优化解.

故对其子问题:

$K_{ij} = \{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的根必为 K_{ij} 中某个 k_r , 结构同上.

只要对于每个 $k_r \in K$, 确定 $\{k_i, \dots, k_{r-1}\}$ 和 $\{k_{r+1}, \dots, k_j\}$ 的优化解, 我们就可以求出 K 的优化解.

(b) 由上述结构分析可知

$$E(i, j) = p_r + E(i, r-1) + W(i, r-1) + E(r+1, j) + W(r+1, j)$$

(其中 W 为合并子树时深度加 1 所造成的代价)

$$W(i, j) = p_r + W(i, r-1) + W(r+1, j)$$

故有递归方程

$$W(i, i-1) = q_{i-1}$$

$$W(i, j) = W(i, r-1) + p_r + W(r+1, j) = W(i, j-1) + p_j + q_j$$

$$E(i, i-1) = q_{i-1}$$

$$E(i, j) = E(i, r-1) + E(r+1, j) + W(i, j)$$

(c1) 数据结构:

数组 E: 存储优化解的搜索代价

数组 W: 存储代价增量

数组 Root: 记录于问题优化解的根

Optimal-BST (p, q, n)

For $i=1$ To $n+1$ Do

$$E(i, i-1) = q_{i-1}$$

$$W(i, i-1) = 0$$

For $l=1$ To n Do

For $i=1$ To $n-l+1$ Do

$$j = i+l-1$$

$$E(i, j) = \infty$$

$$W(i, j) = W(i, j-1) + q_i + p_j$$

For $r=i$ To j Do

$$t = E(i, r-1) + E(r+1, j) + W(i, j)$$

$$\text{If } t < E(i, j)$$

$$\text{Then } E(i, j) = t$$

$$\text{Root}(i, j) = r$$

Return E and Root

(c2) 时间复杂度: 三层循环, 时间复杂度为 $O(n^3)$

空间复杂度: 二个 $(n+1) \times (n+1)$ 数组, 一个 $n \times n$ 数组, 空间复杂度为 $O(n^2)$

Exercise 2

1) 设字符串 $X = \langle x_0, \dots, x_n \rangle$, 则令 X_i 表示 $\langle x_0, \dots, x_i \rangle$.
 设有两字符串 A, B , 其长度分别为 m, n . 其最短编辑距离为 D_{AB}
 则其优化子结构为

- ① 如果 $x_m = y_n$, 则 $D_{AB} = D_{A_{m-1}B_{n-1}}$
- ② 如果 $x_m \neq y_n$, 则选取以下三种方案中 D 最小的为 D_{AB}
 则 1) 修改: 将 a_i 改为 b_i , 则 $D_{AB} = D_{A_{m-1}B_{n-1}} + 1$.
 2) 删除: 删除 a_i , 则 $D_{AB} = D_{A_{m-1}B_n} + 1$.
 3) 插入: 将 b 添加至 a 前, 则 $D_{AB} = D_{A_mB_{n-1}} + 1$.
 以此类推, 直至 A, B 完全一样时即可求出 D_{AB} .

1) 建立二维数组 $edit[i][j]$, 其中 i 表示 A 字符串从第 0 个字符到第 i 个字符, j 表示 B 字符串从第 0 个字符到第 j 个字符. $edit[i][j]$ 表示二者间的最短编辑距离.

由 1) 分析建立递归方程:

$$\begin{array}{l|l}
 edit[i][j] & \begin{array}{l} 0 \quad i=0, j=0 \\ j \quad i=0, j>0 \\ i \quad i>0, j=0 \\ edit[i-1][j-1] \quad i>0, j>0 \text{ 且 } A[i] = B[j] \\ \min \{ edit[i-1][j-1] + 1, edit[i-1][j] + 1, edit[i][j-1] + 1 \} \end{array}
 \end{array}$$

(c) 代码实现:

```

import java.util.Scanner;

public class edit {

    public static void main(String[] args) {
        int i, j;
        Scanner sc = new Scanner(System.in);
        System.out.println("Input:");
        String A = sc.next();
        String B = sc.next();
        int[][] edit = new int[A.length()+1][B.length()+1];
        for(j = 0; j <= B.length(); j++)
        {
            edit[0][j] = j;
        }
        for(i = 1; i <= A.length(); i++)
        {
            edit[i][0] = i;
        }
        for(i = 0; i < A.length(); i++)
        {

```

```

    int m = i + 1;
    for( j = 0; j < B.length(); j++)
    {

        int n = j + 1;
        if(A.charAt(i) == B.charAt(j))
        {
            edit[m][n] = edit[m-1][n-1];
        }
        else
        {
            edit[m][n] = edit[m-1][n-1] + 1;
            if((edit[m][n-1]+1) < edit[m][n])
            {
                edit[m][n] = edit[m][n-1] + 1;
            }
            if((edit[m-1][n]+1) < edit[m][n])
            {
                edit[m][n] = edit[m-1][n] + 1;
            }
        }
    }
}
System.out.println("Output:");
System.out.println(edit[A.length()][B.length()]);
}
}

```

(d) 空间复杂度: 数组大小 $(m+1) \times (n+1)$. 故空间复杂度为 $O(m \cdot n)$
 时间复杂度: $T_1(n) = O(m)$ $T_2(n) = O(n)$ $T_3(n) = O(m \cdot n)$
 故时间复杂度为 $O(m \cdot n)$.

结果:

```

Input:
horse
ros
Output:
3

```


Exercise 3

矩阵元素 $V(i, j)$ —— 对于前 i 个物品而言, 使背包装上价值为 j 的物品, 所需的最小重量, 其中 $0 \leq i \leq n$, $0 \leq j \leq \sum V_i$.

递归方程

$$V(n, j) = \begin{cases} W_n & j = V_n \\ +\infty & j \neq V_n \end{cases}$$

$$V(i, j) = \min \{ V(i+1, j), V(i+1, j - V_i) + W_i \} \quad (i \leq n)$$

则在值为 C 的 $V(i, j)$, j 最大的便是最优解

Exercise 4

思路:

| | | |
|---|-----|-------|
| 0 | 0 0 | 0 0 0 |
| 1 | 0 1 | 0 0 1 |
| | 1 1 | 0 1 1 |
| | 1 0 | 0 1 0 |
| | | 1 1 0 |
| | | 1 1 1 |
| | | 1 0 1 |
| | | 1 0 0 |

如图, 由格雷码的性质, 格雷码中间切割后, 取出第一位数字, 前半部分与后半部分对称, 而前半部分第一位数字为 0, 后半部分第一位数字为 1。由此联想

到利用分治算法（不是代价类函数不好用动态规划算法？）表示格雷码。

代码实现：

```
import java.lang.Math;
import java.util.Scanner;

public class GrayCode {
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int i;
        System.out.println("Input:");
        int n = sc.nextInt();
        int[] a = divide(n);
        System.out.println("Output:");
        System.out.print("[ ");
        for(i = 0; i < a.length ; i++)
        {
            System.out.print(" " + a[i] + " ");
        }
        System.out.print(" ]");
    }

    public static int[] divide(int n)
    {
        int i , j;
        if(n == 1)
        {
            return new int[] {0,1};
        }
        else
        {
            int[] temp = divide(n-1);
            int[] ret = new int[temp.length * 2];
            for(i = 0 ; i < temp.length ; i++)
            {
                ret[i] = temp[i];
            }
            for(i = temp.length , j = temp.length - 1 ; i < ret.length && j >=
0 ; i++ , j--)
            {
                ret[i] = temp[j] + (int)Math.pow(2, n-1);
            }
        }
    }
}
```

```
        return ret;
    }
}
```

结果:

Input:

2

Output:

[0 1 3 2]

Input:

3

Output:

[0 1 3 2 6 7 5 4]