算法设计第四次作业

第一题 交换硬币游戏

【解题思路】

- 首先考虑贪心地每次取两端上较大面值的硬币。但是可以举出反例序列$\{8,15,3,7\}$.如果采用贪心策略，先手可以取到$8+7=15$,后手可以取到$15+3=18>15$;然而先手完全可以取到$7+15=22$,这样只留给后手$8+3=11$.由此证明了贪心策略不是理性的选取方式，不能利用好先手的优势。
- 设计$dp[i][j]$数组，用来表示**对于第i个到第j个硬币，先手可以获得的最大领先金额**。由$dp[i][j]$的定义，可以很快发现$dp[1][n]$中数值的正负表明答案结果。$dp[1][n]>0$表示对于整个问题而言先手采取明智的策略可以最终取得优势，应该选择先手；否则表示先手在双方都绝对理性的情况下没法最终获得优势，因此选后手。
- 对于$dp[i][j]$的计算，可以由$dp[i+1][j]$和$dp[i][j-1]$转移得到。以$dp[i+1][j]$为例，表示第$i+1$个到第j个硬币，先手可以获得的最大领先金额，对于$dp[i][j]$而言，$dp[i+1][j]$的先手相当于后手，所以在转移方程中需要取相反数再加上新取到的$coin[i]$。$dp[i][j-1]$同理，最后两者取较大值。
- 整理得状态转移方程

$$dp[i][j]=max(coin[i]-dp[i+1][j],coin[j]-dp[i][j-1])$$

【代码】

```cpp
#include<bits/stdc++.h>

using namespace std;

const int N = 1e3;
int coin[N];
int dp[N][N];
int main () {
    int n;cin>>n;
    for (int i=1;i<=n;i++) {
        cin>>coin[i];
    }
    for (int i=1;i<=n;i++) dp[i][i] = coin[i];

    for (int i=n-1;i>=1;i--) {
        for (int j=i+1;j<=n;j++) {
            dp[i][j] = max(coin[i]-dp[i+1][j],coin[j]-dp[i][j-1]);
        }
    }
    if (dp[1][n]>0) cout<<"first"<<endl;
    else cout<<"second"<<endl;

    // for (int i=1;i<=n;i++) {
    //     for (int j=1;j<=n;j++) {
    //         cout<<dp[i][j]<<" ";
    //     }
    //     cout<<endl;
    // }
```

```
    return 0;
}

```

【输入输出】

测试数据1:

```
9
3 6 8 1 3 8 9 10 2
```

```
second
```

测试数据2:

```
4
8 15 3 7
```

```
first
```

【复杂度分析】

从代码的两层for循环易得算法时间复杂度$O(n^2)$,

用到二维dp数组，因此空间复杂度$O(n^2)$.

第二题 编辑距离

【解题思路】

- 设计$dp[i][j]$数组表示$A[1;i]$和$B[1;j]$两个字串之间的最小编辑次数。
- 易得如果$A[i-1] == B[j-1]$, 则$dp[i][j] = dp[i-1][j-1]$.
- 由于可以对A做3种操作，所以在计算$dp[i][j]$时考虑从3个方向转移的情况，分别是$dp[i-1][j]$, $dp[i][j-1]$和$dp[i-1][j-1]$。
  - $dp[i-1][j]$表示对A删除字符$A[i]$的结果
  - $dp[i][j-1]$表示对A插入字符$A[i]$的结果
  - $dp[i-1][j-1]$表示对A修改字符$A[i]$的结果
- 达到上述三种状态都需要一步操作，对应3种不同操作，整理得到状态转移方程

$$dp[i][j] = min\{dp[i-1][j], dp[i][j-1], dp[i-1][j-1])\} + 1$$

- 在求$dp[i][j]$数组之前需要对第0行和第0列初始化，$dp[i][0] = i$,$dp[0][i] = i$.

【代码】

```cpp
#include<bits/stdc++.h>


using namespace std;

const int N = 1e3;
int dp[N][N];
int main () {
    string A, B;
    cin>>A>>B;
    int len_A = A.length();
    int len_B = B.length();
    for (int i=0;i<=len_A;i++) dp[i][0] = i;
    for (int i=0;i<=len_B;i++) dp[0][i] = i;
    for (int i=1;i<=len_A;i++) {
        for (int j=1;j<=len_B;j++) {
            if (A[i-1] == B[j-1]) dp[i][j] =  dp[i-1][j-1];
            else dp[i][j] = min(dp[i-1][j], min(dp[i][j-1], dp[i-1][j-1])) + 1;
        }
    }

    cout<<dp[len_A][len_B]<<endl;

    // for (int i=0;i<=len_A;i++) {
    //     for (int j=0;j<=len_B;j++) {
    //         cout<<dp[i][j]<<" ";
    //     }
    //     cout<<endl;
    // }

    return 0;
}
```

【输入输出】

测试数据1:

```
string
bhrinten
```

```
5
```

测试数据2:

```
string
strring
```

```
1
```

【复杂度分析】

从代码的两层for循环易得算法时间复杂度$O(nm)$,

用到二维dp数组，因此空间复杂度$O(nm)$.

其中n,m分别表示字符串A和字符串B的长度。

第三题 哈夫曼编码

要证对于任意长度n的的序列，哈夫曼编码的贪心构造方法得到的平均码长L最小,其中码长L的计算公式：

$$L=\sum_{i=1}^{n}w_il_i$$

基本情况

当$n=2$时， 直接把两个字符合并，得到哈夫曼编码分别是0和1.这两个编码长度都是1，显然是最优的。

归纳假设

假设对某个k， 我们有一组字符$\{a_1,a_2,\ldots,a_k\}$和它们的频率$\{w_1,w_2,\ldots,w_k\}$。我们通过哈夫曼编码的构造规则得到的编码$\{c_1,c_2,\ldots,c_k\}$是最优的，即满足

$$L=\sum_{i=1}^{n}w_il_i$$

是最小的。

归纳证明

考虑$n=k+1$的情况，有$\{a_1,a_2,\ldots,a_k，a_{k+1}\}$和$\{w_1,w_2,\ldots,w_k,w_{k+1}\}$.不妨设$w_j$和$w_k$是最小的两个频率。

将w_j和w_k合并成一个新的节点z，其频率$w_z=w_j+w_k$.

得到k个节点$\{a_1,a_2,\ldots,a_{j-1},a_z,a_{j+1},\ldots,a_{k-1}，a_{k+1}\}$和$\{w_1,w_2,\ldots,w_{j-1},w_z,w_{j+1},\ldots,w_{k-1},w_{k+1}\}$,由假设我们可以从$k$个节点得到最优的编码结果，对于$a_j$和$a_k$，只需要在$z$的编码之后分别加上$0$和$1$即可，因此$a_j$和$a_k$的码长都是$l_z+1$, 得到

$$
\begin{aligned}
L_{k+1} &= \sum_{i=1,i\neq j,k}^{k+1} w_i l_i + w_j(l_z+1) + w_k(l_z+1) \\
&= \sum_{i=1,i\neq j,k}^{k+1} w_i l_i + (w_j+w_k)(l_z+1) \\
&= \sum_{i=1,i\neq j,k}^{k+1} w_i l_i + w_z l_z + (w_j+w_k)
\end{aligned}
$$

整理得到，

$$
L_{k+1} = L_k + (w_j + w_k)
$$

可以得到L_{k+1}都可以通过L_k加上序列中最小的两项w_j和w_k得到，因此一定是最小的，由此归纳总结得哈夫曼编码的贪心选择策略所得的平均码长是最优的。