算法作业-Week5 （动态规划）

1. 数值和最小的路径

求解点数值矩阵最小路径。随机产生一个 n 行 m 列的整数矩阵，在整数矩阵中寻找从左上角到右下角、每步可以向下(D)或向右(R)或斜向右下(O)的一条**数值和最小的路径**。

分析：

到达一个位置的最小路径 = 到达**左上、上侧、左侧位置**之中最小的路径+当前位置数值
设 a[i][j]是从(1,1)到(i,j)的最小路径数值和，则有：

$$a[i][j] = \min\left(a[i-1][j], a[i][j-1], a[i-1][j-1]\right)$$

按照此式逐行递推，同时记录路径，最终逆序输出。
为了避免数组越界，我先计算了左边界和右边界。

代码：

```
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
#define random(x) rand() % x

using namespace std;

const int MAXN = 100;
int n, m;                  // n 行 m 列
int a[MAXN][MAXN];         // 整数矩阵
char road[MAXN][MAXN];     // 记录路径

void generate() {  // 生成随机矩阵
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) a[i][j] = random(100);
}

int solve() {  // 逐行递推
    // 计算左边界
    for (int i = 2; i <= n; i++) {
        a[i][1] = a[i][1] + a[i - 1][1];
        road[i][1] = 'D';
    }
    // 计算上边界
```

```
    for (int i = 2; i <= m; i++) {
        a[1][i] = a[1][i] + a[1][i - 1];
        road[1][i] = 'R';
    }
    // 逐行递推 a[i][j] = a[i][j] + min(min(a[i-1][j],a[i][j-1]),a[i-
1][j-1]);
    for (int i = 2; i <= n; i++) {
        for (int j = 2; j <= m; j++) {
            // 递推求最小值
            int min_x = min(min(a[i - 1][j], a[i][j - 1]), a[i - 1][j -
 1]);
            a[i][j] = a[i][j] + min_x;
            // 记录路径
            if (a[i - 1][j] == min_x)
                road[i][j] = 'D';
            else if (a[i][j - 1] == min_x)
                road[i][j] = 'R';
            else
                road[i][j] = 'O';
        }
    }
    return a[n][m];
}
void print_road() {  // 打印路径
    char ans[100];
    int len = 0, r = n, c = m;

    while(!((r==1&&c==1))){
        ans[len++] = road[r][c];
        if(road[r][c]=='D') r--;
        else if(road[r][c]=='R') c--;
        else{
            r--;
            c--;
        }
    }
    for(int i=len-1;i>=0;i--){
        cout<<ans[i];
    }
    cout<<endl;
}
int main() {
    srand(int(time(0)));  // 重置随机数种子
    cin >> n >> m;        // 读入行、列数
```

```
    generate();           // 生成随机矩阵

    // 打印原始随机矩阵
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }

    // 打印最小路径和
    cout << solve()<<endl;
    print_road();
    return 0;
}
```

运行结果：

```
4 6
63 9 41 47 23 4
13 90 28 19 94 32
47 30 4 48 41 49
69 30 31 28 16 90
238
RODORR
```

说明：63+9+28+4+28+16+90=238

2. 最佳加法表达式

有一个由数字 1,2,3…9 组成的数字串（长度不超过 500），问如何将 M（1<=M<=20）个加号插入这个数字串中，使得形成的基本算数表达式的值最小。

注：

（1）加号不能加在数字串的最前面或者最末尾，也不应有两个或以上的加号相邻；

（2）M 的值一定小于数字串长度

例：数字串 79846，若需要加入两个加号，则最佳方案为 79+8+46=133

分析：

设：$V[i][j]$ = 在前 j 个数中插入 i 个加号的最小和

若 i=0，则：

$$V[i][j] = \text{前 j 个数字构成的整数}$$

若 j<i+1 ：

$$V[i][j] = INF\text{（无穷大）}$$

否则：

$$V[i][j] = min\{V[i-1][k] + num[k+1][j]\}, \quad k = i, \ldots j-1$$

再加上大整数加法，即可处理掉数据过大的问题。

在枚举加号数量、数字数量和 k 时，使用了三重循环，循环次数分别为 m, n, n，故总的时间复杂度为$O(mn^2)$

```
#include <algorithm>
#include <cstdio>
#include <iostream>
#include <string>
using namespace std;
const int MAXN = 500;
const int MAXM = 20;
string INF;
string num[MAXN][MAXN];  // num[i][j] = 从第 i 个数到第 j 个数构成的数值
string V[MAXM][MAXN];  // V[i][j] = 在前 j 个数中插入 i 个加号的最小和

// 大整数加法,num3 = num1+num2
void add(string& num1, string& num2, string& num3) {
    reverse(num1.begin(), num1.end());
    reverse(num2.begin(), num2.end());
    int len1 = num1.length();
    int len2 = num2.length();
    int maxl = MAXN, c = 0;  // c 是进位标志
    for (int i = 0; i < maxl; i++) {
        int t;
        if (i < len1 && i < len2)
            t = num1[i] + num2[i] - 2 * '0' + c;
        else if (i < len1 && i >= len2)
            t = num1[i] - '0' + c;
        else if (i >= len1 && i < len2)
            t = num2[i] - '0' + c;
        else
            break;
        num3.append(1, t % 10 + '0');
        c = t / 10;
    }

    while (c) {
        num3.append(1, c % 10 + '0');
        c /= 10;
    }
```

```cpp
    reverse(num1.begin(), num1.end());
    reverse(num2.begin(), num2.end());
    reverse(num3.begin(), num3.end());
}

// 大整数比较，相当于小于号
bool lt(string& num1, string& num2) {
    int len1 = num1.length(), len2 = num2.length();
    if (len1 > len2)
        return false;
    else if (len1 < len2)
        return true;
    for (int i = 0; i <= len1; i++) {
        if (num1[i] > num2[i])
            return false;
        else if (num1[i] < num2[i])
            return true;
    }
    return false;
}

int main() {
    for (int i = 0; i < MAXN; i++) INF.push_back('9');
    string S;
    int n, m;
    cin >> m >> S;
    n = S.length();
    // 提前计算 num[i][j]
    for (int i = 0; i < n; i++) {
        num[i + 1][i + 1] = S.substr(i, 1);
    }
    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            num[i][j] = S.substr(i - 1, j - i + 1);
        }
    }

    for (int i = 1; i <= n; i++) {  // 0 个加号的情况
        V[0][i] = num[1][i];
    }

    // V[i][j] = 在 j 个数里面插入 i 个加号时的最小和
    for (int i = 1; i <= m; i++) {      // 枚举加号数量
        for (int j = 1; j <= n; j++) {  // 枚举数字个数
```

```
            if (j < i + 1) {            // 加号数量过多
                V[i][j] = INF;
            } else {  // 正常情况下，从 i 到 j-1 枚举 k，求
                string min_x = INF;
                string temp;
                for (int k = i; k <= j - 1; k++) {
                    temp.clear();
                    add(V[i - 1][k], num[k + 1][j], temp);
                    if (lt(temp,min_x)) min_x = temp;
                }
                V[i][j] = min_x;
            }
        }
    }
    cout << V[m][n] << endl;
}
```

运行结果 1：

```
2 123456
102
```

说明：12+34+56=102

运行结果 2：

```
2 79846
133
```

说明：79+8+46=133

3.找钱

设有 n 种不同面值的货币，存于数组 S[1:n]中。现用这些货币来找钱，各种货币使用的个数不限。

（1）当只用面值为 S[1],S[2}…S[i]来找钱时，所用的货币的最小个数记为 C(i, j)，写出 C(i, j)的递推式。

递推式如下：

$$C(i,j) = \min\{C(i-1, j-k*S[i]) + k\} \quad k = 0,1,2 \ldots j/S[i]$$

（2）设计一个动态规划算法计算 C(n, j)，1<=j<=L，只使用一个规模为 L 的数组，并分析算法的时间复杂度。

分析：

可以只使用规模为 L 的一维数组完成 DP，状态转移方程如下：

$$C(j) = \min\{C(j), C(j - S[i]) + 1\} \quad j = S[i], S[i] + 1 \ldots L$$

最坏情况下，算法需要对 n 个硬币各执行 L 次循环，因此时间复杂度为$O(nL)$。

代码：

```
#include <algorithm>
#include <iostream>
using namespace std;
const int MAXL = 1000;
const int INF = 1e9;
int S[1000 + 10], n, L, C[MAXL + 10];
int main() {
    for (int i = 0; i < MAXL; i++) C[i] = INF;

    cin >> n >> L;
    for (int i = 1; i <= n; i++) cin >> S[i];

    for (int i = 1; i <= n; i++) {
        for (int j = S[i]; j <= L; j++) {
            C[j] = min(C[j], C[j - S[i]] + 1);
        }
    }
    cout<<C[L]<<endl;
    return 0;
}
```

运行结果

```
7 145
1 2 5 10 20 50 100
4
```

说明：145 = 100+20+20+5