22年算法设计课程机试回忆版

一共四个题，一百分

22年（21级）

注意：回忆版正确与否不做评价，支持开源禁止倒卖。

建议：复习ICT实验作业题（八皇后、最短下降路径、矩阵链相乘、全排列）

1、高精度乘法

给定两个非负整数（不含前导 00） A 和 B，请你计算 A×B 的值。

输入格式

共两行，第一行包含整数 A，第二行包含整数 B。

输出格式

共一行，包含 A×B的值。

数据范围

1≤A的长度≤100000

0≤B≤10000

题解1 高精度x低精度

```cpp
#include <iostream>
#include <vector>

using namespace std;

vector <int> mul(vector <int> & A, int b) {
    vector <int> C;

    int t = 0;
    for (int i = 0; i < A.size(); i ++) {
        t += A[i] * b;       // t + A[i] * b = 7218
        C.push_back(t % 10); // 只取个位 8
        t /= 10;             // 721 看作 进位
    }

    while (t) {            // 处理最后剩余的 t
```

```
        C.push_back(t % 10);
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}

int main() {
    string a;
    int b;
    cin >> a >> b;

    vector <int> A;
    for (int i = a.size() - 1; i >= 0; i --) A.push_back(a[i] - '0');

    auto C = mul(A, b);

    for (int i = C.size() - 1; i >= 0; i --) {
        cout << C[i];
    }

    return 0;
}
```

2.找零钱（贪心）

问题描述：

设有6 种不同面值的硬币，各硬币的面值分别为5 分，1 角，2 角，5 角，1 元，2 元。现要用这些面值的硬币来购物和找钱。购物时可以使用的各种面值的硬币个数存于数组Coins［1:6］中，商店里各面值的硬币有足够多。在1 次购物中希望使用最少硬币个数。

例如，1 次购物需要付款0.55 元，没有5 角的硬币，只好用2*20+10+5 共4 枚硬币来付款。如果付出1 元，找回4 角5 分，同样需要4 枚硬币。但是如果付出1.05 元（1 枚1 元和1 枚5 分），找回5 角，只需要3 枚硬币。这个方案用的硬币个数最少。

编程任务：

对于给定的各种面值的硬币个数和付款金额，编程计算使用硬币个数最少的交易方案。

数据输入：

每一行有6 个整数和1 个有2 位小数的实数。分别表示

可以使用的各种面值的硬币个数和付款金额。

结果输出:

结果应分行输出，每行一个数据。如果不可能完成交易，则输出”impossible”。

输入

2 4 2 2 1 0 0.95

2 4 2 0 1 0 0.55

输出文件示例

2

3

```
1
```

3.最长公共子序列

【问题描述】给定两个字符串text1和text2，返回这两个字符串的最长公共子序列的长度。如果不存在公共子序列，则返回0。一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。

【输入形式】输入的第1行中有一个字符串，表示text1；输入的第2行中有一个字符串，表示text2。

【输出形式】输出1行一个整数，表示text1和text2的最长公共子序列的长度。

【样例输入】

abcde

ace

【样例输出】

3

【样例说明】

最长公共子序列是ace，长度为3

【说明】

1<=text1.length, text2.length<=1000

text1和text2仅有小写英文字符组成。

```
1 /*
2     线性DP
```

```
    1.状态表示
        1.1集合: f[i][j] 表示 a[1~i] 与 b[1~j]的公共子序列的集合
        1.2属性:Max
    2.状态计算: f[i][j] 可以分为四种情况转移而来
        1) 00 a[i] b[j] 都不包含  f[i-1][j-1]
        2) 10 包含a[i] 不包含 b[j]  f[i][j-1]
        3) 01 包含b[j] 不包含a[i]   f[i-1][j]
        4) 11 if a[i] == b[j]  包含 f[i-1][j-1] + 1;
   */
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5+10;
int main()
 {
    char a[N],b[N];
    int f[110][110];
    scanf("%s",a + 1);
    scanf("%s",b + 1);
    int n = strlen(a + 1),m = strlen(b + 1);
    for(int i = 1 ; i <= n ; i ++)
    {
        for(int j = 1 ; j <= m; j ++)
        {
            f[i][j] = max(f[i-1][j] , f[i][j-1]);
            if(a[i] == b[j])
            {
                f[i][j] = max(f[i][j],f[i-1][j-1] + 1);
            }
        }
    }
    cout<<f[n][m]<<endl;
 }
```

4.单源最短路径

【问题描述】

给定带权有向图G =(V,E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源（默认源点为顶点1）。现在要计算从源到所有其他各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

【输入形式】

第一行是顶点数n和边数m

随后n行是顶点编号

随后m行是所有边的起点 终点和边长

【输出形式】

输出n-1行，分别为源点到其他各点的最短路径

【样例输入】

5 6

1

2

3

4

5

1 2 5

2 5 1

1 3 2

3 4 3

4 5 2

1 4 1

【样例输出】

5

2

1

3

```
#include<iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 510, M = 100010;

int h[N], e[M], ne[M], w[M], idx;//邻接表存储图
int state[N];//state 记录是否找到了源点到该节点的最短距离
int dist[N];//dist 数组保存源点到其余各个节点的距离
int n, m;//图的节点个数和边数

void add(int a, int b, int c)//插入边
```

```
14 {
15     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
16 }
17
18 void Dijkstra()
19 {
20     memset(dist, 0x3f, sizeof(dist));//dist 数组的各个元素为无穷大
21     dist[1] = 0;//源点到源点的距离为置为 0
22     for (int i = 0; i < n; i++)
23     {
24         int t = -1;
25         for (int j = 1; j <= n; j++)//遍历 dist 数组，找到没有确定最短路径的节点中距
   离源点最近的点t
26         {
27             if (!state[j] && (t == -1 || dist[j] < dist[t]))
28                 t = j;
29         }
30
31         state[t] = 1;//state[i] 置为 1。
32
33         for (int j = h[t]; j != -1; j = ne[j])//遍历 t 所有可以到达的节点 i
34         {
35             int i = e[j];
36             dist[i] = min(dist[i], dist[t] + w[j]);//更新 dist[j]
37         }
38
39
40     }
41 }
42
43 int main()
44 {
45     memset(h, -1, sizeof(h));//邻接表初始化
46
47     cin >> n >> m;
48     while (m--)//读入 m 条边
49     {
50         int a, b, w;
51         cin >> a >> b >> w;
52         add(a, b, w);
53     }
54
55     Dijkstra();
56     if (dist[n] != 0x3f3f3f3f)//如果dist[n]被更新了，则存在路径
57         cout << dist[n];
58     else
59         cout << "-1";
```

```
60 }
```

5.矩阵链相乘（不完全回忆，好像有）

【问题描述】给定n个矩阵M1,M2...Mn，他们的维数分别是r1*c1,r2*c2...rn*cn，要求使用【动态规划】的策略求解矩阵连乘的最优计算代价(总乘法次数最少)。题目保证矩阵相乘一定是有效的。

例如有三个矩阵M1，M2，M3，他们的维度分别是2*10,10*2，2*10。按照矩阵乘法的结合律，可以先把M1和M2相乘，然后把结果和M3相乘，总的乘法次数为2*10*2+2*2*10=80次；也可以先把M2和M3相乘，再用M1去相乘，这种方式下总的乘法次数为10*2*10+2*10*10=400次。因此最优计算代价为80。

【输入形式】输入的第1行中有1个数字n，表示矩阵的个数；接下来n行，每行2个整数ri和ci，分别表示矩阵Mi的行数和列数。

【输出形式】输出1行中有一个数字，表示n个矩阵相乘的最优计算代价。

【样例输入】

3

2 10

10 2

2 10

【样例输出】

80

【说明】

n>=2

1<=ri,ci<=20

```cpp
/*
   矩阵链，求最小乘法次数
   DP:
   1.状态表示:
    1.1 集合: f[i][j] : 表示矩阵i 到 j 相乘次数
    1.2 属性: Min
   2 状态计算:Ai…j = Ai…k * Ak+1 …j
   f[i][j] = min(f[i][k] + f[k+1][j],f[i][j]) + mat[i].r*mat[k].c*mat[j].c
  */
#include<bits/stdc++.h>
using namespace std;
const int N = 110;
struct Mat{
```

```
    int r,c;
 }mat[N];
int main()
 {
    int n, f[N][N];
    int p[N];
    cin>>n;
    for(int i = 1 ; i <= n ; i ++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        p[i-1] = a ; p[i] = b;
    }
    for(int len = 2 ; len <= n; len ++)
    {
        for(int i = 1; i <= n - len + 1 ; i ++)
        {//j - i + 1 = len
            int j = i + len -1;
            f[i][j] = 1e9;
            for(int k = i; k <= j - 1; k ++)
            {
                f[i][j] = min(f[i][j],f[i][k]+f[k+1][j]+p[i-1]*p[k]*p[j]);
            }
        }
    }
    cout<<f[1][n];
    return 0;
 }
```

6.全排列（不完全回忆）

【问题描述】给定一个含有n个数字的数组(数组中元素各不相同)，设计一个回溯算法生成其所有的排列。要求以字典序升序的方式进行输出。

【输入形式】输入的第1行包含1个整数n。接下来1行包含n个整数，表示n个数字。

【输出形式】输出包含若干行，每行含有n个数字，表示其一种排列方式。

【样例输入】

3

1 2 3

【样例输出】

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5+10;
int n,path[N];
bool st[N];
int a[N];
void dfs(int t )
 {
    //递归出口
    if(t == n)
    {
        for(int i = 0 ; i < n ; i ++)
            cout<<path[i]<< " ";
        puts("");
        return ;
    }
    for(int i = 0 ; i < n ; i ++)
    {
        if(!st[i])
        {
            st[i] = true;
            path[t] = a[i];
             //回溯
            dfs(t + 1);
            st[i] = false;
         }
    }
 }
int main()
 {
    cin>>n;
    for(int i = 0 ; i < n ;i ++)scanf("%d",&a[i]);
    sort(a,a+n);
    dfs(0);
    return 0;
 }
```