天津大学

程序设计综合实践课程报告

动态规划实验

学生姓名 陈秋澄

学院名称 智能与计算学部

专　　业 大类

学　　号 3022244290

1. 冬冬爬楼梯

1.1 题目分析

因为仅使用递归会超限，故结合大整数高精度加法求解。

1.2 题目代码

```
#include<iostream>
using namespace std;
long long n,m,a[3004][1000];
void f(int x){              //按位储存在数组中
    for(int i=1;i<=m;i++)   //计算每一位
        a[x][i]=a[x-1][i]+a[x-2][i]+a[x-3][i];
    for(int i=1;i<=m;i++){  //进位
        if(a[x][i]>=10){
            a[x][i+1]+=a[x][i]/10;
            a[x][i]%=10;
        }
        if(a[x][m+1]!=0)    //操作后进位 位数加 1
            m++;
    }
}
int main(){
    while(cin>>n){
        m=1;               //位数初始为个位
        a[0][1]=1;
        a[1][1]=1;
        a[2][1]=2;
        for(int x=3;x<=n;x++)
            f(x);
        for(int i=m;i>0;i--)
            cout<<a[n][i];
        cout<<endl;
        for(long long i=1;i<=n;i++){
            for(long long k=0;k<=m;k++)
                a[i][k]=0;
        }
    }
    return 0;
}
```

2. 最大子段和

2.1 题目分析

假设我们已经知道了以 **j** 结尾的最大子段和为 **k**，并且由前 **j** 项所得的最大和为 **sum**。

那么以下有这么几种情况：

假设 **k+nums[j+1]<0**,那么我们可以得知以 **j+1** 结尾的最大子段和是个负数（这里要特别说明一下，当数据特殊的情况下，**k** 可能是不存在的，我们这种时候需要把 **k** 当作 **0** 处理，可以理解为从 **j+1** 重新开始计数）。我们下一步去看看这个 **k+nums[j+1]**是不是**>sum**,如果是，那么 **sum=k+nums[j+1]**，记录下新的最大值。因为以 **j+1** 结尾的最大和是个负数，它会妨碍下一个数的最大和，所以必须从下一个重新记数，**k=0**。

假设 **k+num[j+1]>=0**,同 **1** 的情况，我们先看看和 **sum** 的大小，观察是否是我们要找的那个 **sum**。但是因为以 **j+1** 结尾的子段和是非负数，我们把它保留到 **j+2** 中

如果用 **temp** 来存贮 **k** 的值的话，**temp** 初始值**=0**，**sum=num[0]**，不然的话 **sum** 计算可能出问题

要判断要在更改 **temp** 之前，不然置零的 **temp** 会对 **sum**（尤其是负数多的用例）产生干扰

2.2 题目代码

```
#include <iostream>
#include <stdio.h>
using namespace std;
int a[100], n,m,sum,temp;
int main() {
    cin>>m;                          //样例个数
    while(m>0){
        cin >> n;
        int* a = NULL;
        a = new int[n + 1];
```

```
        for (int i = 1; i <= n; i++)
            cin >> a[i];
        temp=0;
        sum=0;
        for (int i = 1; i <= n; i++) {//求最大子段和算法
            if (temp > 0)
                temp += a[i];
            else
                temp = a[i];
            if (temp > sum)
                sum = temp;
        }
        cout << sum << endl;
        m--;
    }
    system("pause");
    return 0;
}
```

3. 最大子阵和

3.1 题目分析

基本思路【PA】:（可参考最大字段和）

利用前缀和的思路，先把它上下加起来，再把得到的新矩阵的左右加起来。

然后运用暴力求解，其中递推式为 dp[i][j] = max(dp[i][j], sum[i][j] - sum[k - 1][j] - sum[i][l - 1] + sum[k - 1][l - 1])。

3.2 题目代码

```
#include<iostream>
#include<cstring>
using namespace std;
int n;
int m[147][147];
int sum[147][147];
int dp[147];
int main() {
    while(cin >> n) {
        int maxn = -1e+7;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cin >> m[i][j];
                sum[i][j] = sum[i - 1][j] + m[i][j];
            }                          // sum 数组就是这个新矩阵
        }
        memset(dp, 0, sizeof dp);
        for (int i = 1; i <= n; i++) {//只把原矩阵的上下加起来得到矩阵，然
//后再求最大子段和即可
            for (int j = i; j <= n; j++) {
                for (int k = 1; k <= n; k++) {
                    int temp = sum[j][k] - sum[i - 1][k];
                    dp[k] = max(temp, dp[k - 1] + temp);
                    maxn = max(maxn, dp[k]);
                }
                memset(dp, 0, sizeof dp);
            }
        }
```

```
        cout << maxn << endl;
    }
    return 0;
}
```

4. 最长上升子序列

4.1 题目分析

这个题就是经典的动态规划思路：用一个数组表示最长子序列长度。

可以使用 lower_bound()函数返回一个迭代器，指向 map 中键值>=key 的第一个元素。

lower_bound：这个函数从已经排好序的的序列 a 中利用二分搜索找出指向 ai>=k 的 ai 的最小指针。

4.2 题目代码

```
#include <iostream>
#define inf 0x7fffffff
using namespace std;

int main()
{
    int n;
    while(cin>>n)
    {
        int a[1100];
        int dp[1100];
        int i;
        fill(dp,dp+n,inf);//fill()可以返回当前填充字符，或者设置当前填充字
//符为 ch 。填充字符被定义为用来填充字符，当一个数字比较指定宽度 T 小时。默认的
//填充字符是空格。
        for(i=0; i<n; i++)
            cin>>a[i];
        for(i=0;i<n;i++)
        {
            *lower_bound(dp,dp+n,a[i])=a[i];
        }lower_bound() //函数返回一个迭代器，指向 map 中键值>=key 的第一个元
素。
//lower_bound：这个函数从已经排好序的的序列 a 中利用二分搜索找出指向 ai>=k
//的 ai 的最小指针。
        cout<<lower_bound(dp,dp+n,inf)-dp<<endl;
    }
    return 0;
```

```
}
```

5. 最小乘车费用

5.1 题目分析

首先，可以任意换车表明这是一道完全背包的题目，如果把 1-10 公里当成十个物品的话，对应的车票费用为价值，那么本质上变为 10 个物品都有无限个的背包。

其次，设置状态。w[i]为十个公里数的重量，对应的也就是 1-10，v[i]表示每种公里数的费用，dp[i][j]表示前 i 种公里费用走 j 公里的最小花费。

再次，状态转移。当 j<w[i]时，此时的花费只能继承前 i-1 种的费用，否则的话，枚举可以行驶的公里数个数（0-j/w[i]）然后依次选择最小值即可。

然后，初始状态是当行驶的公里数为 0 的时候，价值都为 0，因为不行使，即 dp[i][0]=0，其他都设为最大值 maxn，因为要求最小值。

最后，可用滚动数组进行空间优化，求目标值 dp[10][n]。

5.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
int w[15],v[15],dp[110];
const int maxn=pow(10,9);
int main()
{
    for(int i=1;i<=10;i++)
    {
        w[i]=i;
        cin>>v[i];
    }
    int n;
    cin>>n;

    for(int i=1;i<=n;i++)
    dp[i]=maxn;//初始化状态，除了第 0 项全部赋 maxn

    for(int i=1;i<=10;i++)//进行 10 次滚动数组
    {
```

```
        for(int j=w[i];j<=n;j++)//从第 w[i]项开始枚举
        {
            dp[j]=min(dp[j],dp[j-w[i]]+v[i]);//状态转移方程
        }
    }

    cout<<dp[n];//输出目标值
    return 0;
}
```

6. 方格取数

6.1 题目分析

运用动态规划，在 **n*n** 的方格阵中，从左上角出发，每次只能往正下方或右边走，找出一种路线方案，使得所经历方格中数字和最大，输出这个值。

6.2 题目代码

```
#include<iostream>
using namespace std;
int dp[1010][1010],N;
int fun (int **a,int N){       //找出对应矩阵
    int sum1=0,sum2=0;
    for (int i=0;i<N;i++){
        sum1+=a[0][i];
        dp[0][i]=sum1;
        sum2+=a[i][0];
        dp[i][0]=sum2;
    }
    for(int i=1;i<N;i++){
        for(int j=1;j<N;j++){
            dp[i][j]=((dp[i-1][j]>dp[i][j-1])?dp[i-1][j]:dp[i][j-
1])+a[i][j];                    //简化代码
        }
    }
    return dp[N-1][N-1];
}

int main(){
    cin>>N;
    int **a=new int *[N];
    for(int i=0;i<N;i++)
        a[i]=new int [N];
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++)
            cin>>a[i][j];
    }
    cout << fun(a,N)<<endl;
}
```

7. 01 背包

7.1 题目分析

首先，d[j]=max(d[j],d[j-w[i]]+c[i])；

其中 d 数组表示当前容量可以装的最大价值，w[i] 是重量，c[i] 是价值。

在公式中，我们在装和不装中选一种：

1、不装：就是当前的最大重量 d[j]

2、装：先在当前容量 j 中给 当前重量 w[i] 预留一个位置 （d[j-w[i]]），然后在加上当前价值 c[i]

最后，用 max 函数在它们当中选大的那个就可以了。

公式中有 i 有 j（这是一个双重循环）。

7.2 题目代码

```
#include<iostream>
using namespace std;
int d[2000],w[50],c[50];//d 数组的下标表示容量
int v,n;
int main()
{
    cin>>v>>n;//v 表示容量，n 表示数量
    for(int i=1;i<=n;i++)
        cin>>w[i]>>c[i];
    for(int i=1;i<=n;i++)
    {
        for(int j=v;j>=w[i];j--)//01 背包中，第二重循环要倒序，从 v 到 w[i]
        {
            d[j]=max(d[j],d[j-w[i]]+c[i]);
        }
    }
    cout<<d[v];
    return 0;
}
```

8. 完全背包

8.1 题目分析

本题是动态规划问题，判断能否恰好装满背包：若背包恰好装满了，输出结果；背包不能恰好装满，则输出“NO”。

8.2 题目代码

```
#include <iostream>
#include<algorithm>
using namespace std;
long long V,N,T,v[50000],w[50000],f[100010];
int main(){
    cin>>T;
    while (T--){
        cin >> N >> V;//输入物体数和背包容量
        for (long i = 0; i < 100010; i++)//初始化所有的无效状态
            f[i] = -999;
        f[0] = 0;//f[0]是有效状态
        for (int i = 1; i <= N; i++)
            cin >> v[i] >> w[i];//输入每个物体的体积和价值
        for (int i = 1; i <= N; i++){//动态规划过程
            for (long long j = v[i]; j <=V; j++)
                f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
        if (f[V] > 0)//判断能否恰好装满背包
            cout << f[V] << endl;//背包恰好装满了，输出结果
        else
            cout << "NO" << endl;//背包不能恰好装满
    }
    system("pause");
    return 0;
}
```