|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 2022212080 | 算法思路(30%) | 编码实现与算法分析(50%) | 实验报告(20%) | 得分 |
| 姓名 | 刘纪彤 |  |  |  |  |
| 评语 |  | | | | |

### 《算法设计与分析》实验报告

实验3 动态规划算法

**一、实验目的**

1. 加深对动态规划法的理解，了解其基本思想和算法流程。

2. 学习如何在具体问题中应用动态规划法设计算法。

3. 分析动态规划法与分治法、回溯法的区别和联系，以及它们在不同类型问题中的适用性。

4. 提高算法设计能力和编程实践能力。

5. 掌握动态规划法的核心要素：最优子结构、边界条件、状态转移方程。

**二、实验内容(题目)**

给定由n个整数（可能为负整数）组成的序列a1,a2,…,an，求该序列形如：的子段和的最大值。当所有整数均为负数时定义其最大子段和为0。

分别采用穷举法、分治法、动态规划法完成。

**三、算法设计思路**

分治法：将数组分为两部分，分别求左右两部分的最大子段和，再求跨越中间的最大子段和，三者取最大值。之后递归求解。

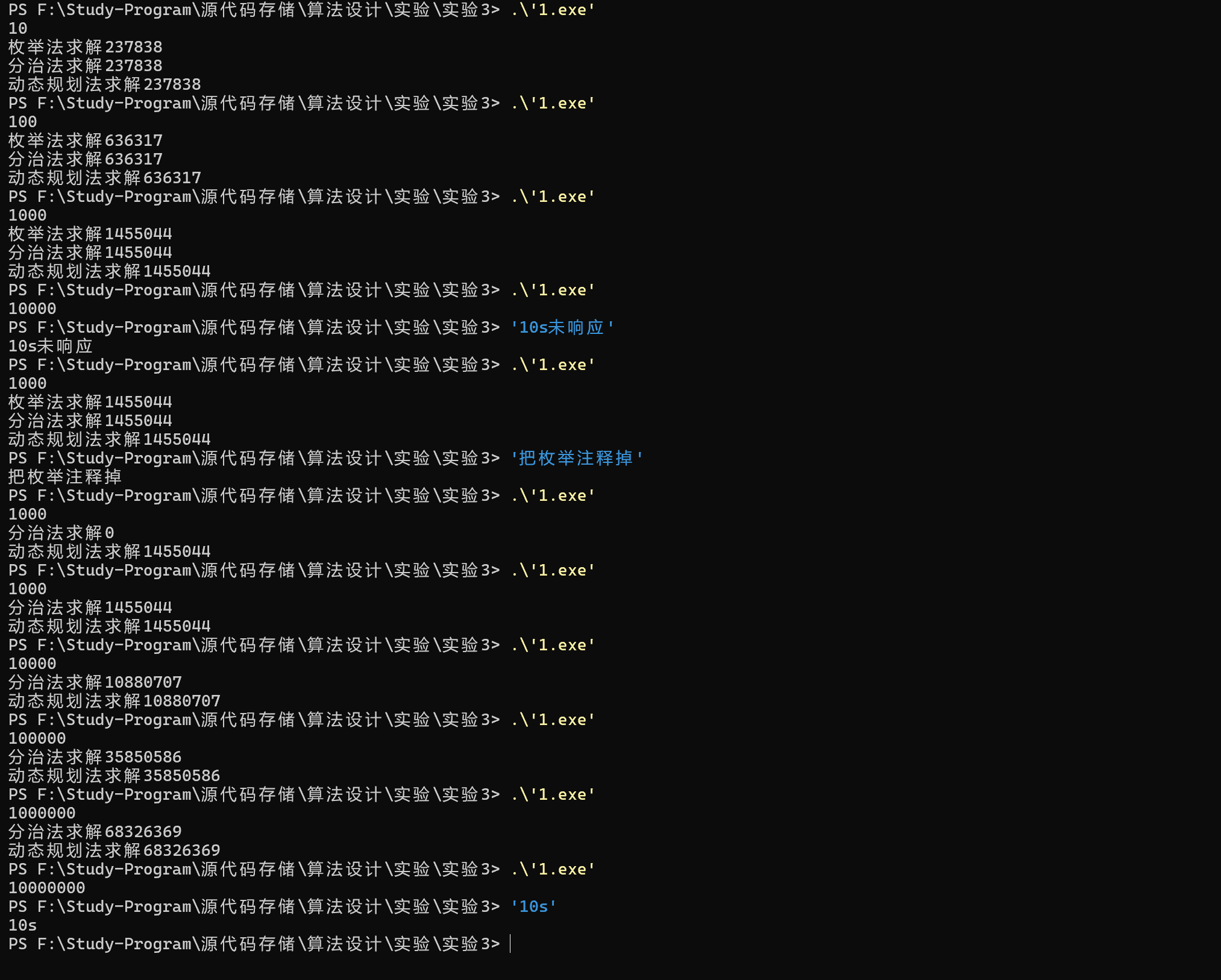
动态规划法：设dp[i]为以a[i]结尾的最大子段和，dp[i]=max(dp[i-1]+a[i],a[i])，最后取dp数组中的最大值即可。

枚举法：枚举所有的子段，求出最大值。

**四、各功能模块设计**

#include<bits/stdc++.h>  
//给定由n个整数（可能为负整数）组成的序列a1,a2,…,an，求该序列形如： 的子段和的最大值。当所有整数均为负数时定义其最大子段和为0。  
using namespace std;  
//枚举法求解  
int max\_subarray\_0(vector<int> a){  
 int n=a.size();  
 int max\_sum=0;  
 for(int i=0;i<n;i++){  
 for(int j=i;j<n;j++){  
 int sum=0;  
 for(int k=i;k<=j;k++){  
 sum+=a[k];  
 }  
 max\_sum=max(max\_sum,sum);  
 }  
 }  
 return max\_sum;  
}  
//分治法求解  
int max\_subarray\_1(vector<int>a)  
{  
 int n=a.size();  
 if(n==1){  
 return max(0,a[0]);  
 }  
 vector<int> a1(a.begin(),a.begin()+n/2);  
 vector<int> a2(a.begin()+n/2,a.end());  
 int sum1=max\_subarray\_1(a1);  
 int sum2=max\_subarray\_1(a2);  
 int sum3=0;  
 int sum4=0;  
 int sum=0;  
 for(int i=n/2-1;i>=0;i--){  
 sum+=a[i];  
 sum3=max(sum3,sum);  
 }  
 sum=0;  
 for(int i=n/2;i<n;i++){  
 sum+=a[i];  
 sum4=max(sum4,sum);  
 }  
 return max(max(sum1,sum2),sum3+sum4);  
}  
//动态规划法  
int max\_subarray\_2(vector<int>a)  
{  
 int pre=0;  
 int ans=a[0];  
 for(int &X:a)  
 {  
 pre=max(pre+X,X);  
 ans=max(pre,ans);  
 }  
 return ans;  
}  
int main(){  
 int n;  
 cin>>n;  
 vector<int> a(n);  
 //调用随机数生成数组(区间在-1e5~1e5之间)  
 for(int i=0;i<n;i++){  
 a[i]=(1.0\*rand()/(RAND\_MAX + 1))\*200001-100000;  
 }  
 // //输出数组  
 // for(int i=0;i<n;i++){  
 // cout<<a[i]<<" ";  
 // }  
 // cout<<endl;  
 int max\_sum=0;  
 max\_sum=max\_subarray\_0(a);  
 cout<<"枚举法求解";  
 cout<<max\_sum<<endl;  
 max\_sum=0;  
 max\_sum=max\_subarray\_1(a);  
 cout<<"分治法求解";  
 cout<<max\_sum<<endl;  
 max\_sum=0;  
 max\_sum=max\_subarray\_2(a);  
 cout<<"动态规划法求解";  
 cout<<max\_sum<<endl;  
 return 0;  
}

**五、运行结果与分析**



3-1

如图所示3-1输出分治、动态规划，并以10s为界限进行测试，从时间上讲动态规划法、分治一定比枚举快，经过理论计算，动态规划和分治算法时间复杂度为O(n)。枚举为

**六、实验总结**

通过本次实验我已经了解并掌握了动态规划法的基本逻辑，结合前述所学的递归和分治知识，能够将动态规划的思想利用已学的递归结合在一起，能够使用分治的思路解决实际应用中的问题，受益匪浅。