# 西安交通大学实验报告

课程名称：算法设计与分析　　实 验 名 称：动态规划

学　　院：电子与信息学部　　实 验 日 期：2023 年 10 月 28 日

姓　　名：林圣翔　　班级：计试 2201　　学号：2223312202

**诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，没有抄袭。**

## 一、实验目的

1.深入了解动态规划算法及相关经典问题

2.掌握用动态规划解题的基本步骤

3.运用区间动态规划的思想解决钢管分割问题

## 二、实验环境

系统类型：64 位操作系统, 基于 x64 的处理器

处理器：12th Gen Intel(R) Core(TM) i7-1260P　　2.10 GHz

软件：Dev-C++ 5.9.2 TDM-GCC 4.8.1 64-bit Debug

## 三、问题描述

设有一个长度为 L 的钢条，在钢条上标有 n 个位置点（p1,p2,…,pn）。现在需要按钢条上标注的位置将钢条切割为 n+1 段，假定每次切割所需要的代价与所切割的钢条长度成正比。请编写一个算法，能够确定一个切割方案，使切割的总代价最小。

## 四、问题分析

由题意可得，将一根钢条切成小段，且需切割位置点和段数已经给出，由 n 个位置点和 n+1 的段数可以得出，其切割结果唯一且可以求出，但由于每次切割所需代价与切割钢条长度成正比，导致不同的切割方案可能产生不同的总代价，故要求我们确定一个切割方案，使切割的总代价最小。因为始末状态已知，我们不妨将题意转变一下，即已知若干个按顺序排列的小段，只能组合相邻小段，每次组合所需要的代价与所组合而成的钢条长度成正比，现确定一个组合方案，使组合的总代价最小。不难看出，这是一个不影响结果的等价转化。通过转化，我们可以发现，这其实就是区间动态规划中一个非常经典的合并石子问题。我们定义 dp[i][j]为合并第 i 根钢条到第 j 根钢条的最小代价，则转移方程为 dp[i][j]=min(dp[i][k]+dp[k+1][j]+cost[i][j]) i≤k＜j,cost[i][j]表示合并第 i 根钢条到第 j 根钢条的花费代价，具体所指的值以及实现方式详见下面的算法设计和算法实现。因为总共要分成 n+1 段，故 dp[1][n+1]就是答案。

## 五、算法设计

首先我们对题目的输入数据进行处理。输入的是 n 个坐标点的乱序坐标，我们需要先讲他们排序，采用 c++自带的 sort()函数，时间复杂度为 O（nlogn），然后我们计算最后每段钢条的长度，len[i]=p[i]-p[i-1] i≤k≤n+1,为了节省空间，算法实现代码中并没有另外开一个 len[]数组，而是利用 c++赋值覆盖的特性，利用倒向遍历，巧妙地将 p[]数组的存储数据由原先的坐标点变成的所需的钢条长度。之后，为了之后求第 i 根钢条到第 j 根钢条的花费代价的简便性，我们在此运用前缀和思想以及 c++赋值覆盖的特性，让

p[]数组中原 p[i]所记录的第 i 根钢条长度变为前 i 个钢条长度之和。至此，我们对输入数据的处理阶段已经完成。

根据题目分析，我们在这里采用区间动态规划的算法。针对这道题目，有如下两种的主要设计方式。

①采用记忆化搜索，计算大区间[i,j]的最优值时，合并其两个子区间[i,k]和[k+1,j]，对所有可能的合并（i≤k＜j，即 k 在 i 和 j 之间移动）进行枚举，从而找到该区域最优合并。最终，一个大区间将会递归成若干个小区间，最小的区间也就是每段小钢管自身，返回 0。这是一个不断递归的过程，为了减少时间复杂度，我们需要做一些优化，即记录下每次递归所得到的结果，这样可以避免大量重复的计算。

②从小区间开始，自底向上递推。先在小区间内进行 DP 得到最优解，然后逐步合并小区间为大区间。我们用 len 来表示当前区间长度，len 的大小是有 2 到 n+1。然后针对每一个 len,枚举其区间起点 i，1≤i≤n+1-len+1，对应的区间终点 j 即为 i+len-1。接下来我们通过变量 k,i≤k＜j 来遍历所有可能情况。故我们这里是通过了三重循环来实现了这个算法。

尽管这两种设计方式在思路实现上有些许不同，但通过计算，其复杂度均为 O(n3)，我们可以采用四边形不等式对其进行优化，将时间复杂度提升至 O(n2)，这题的最优算法是 Garsia-Wachs 算法，复杂度为 O(nlogn)。不过，由于本题的测试数据不大，用 O(n3)的时间复杂度足以应对，而且在思路和代码实现上均较为简单。

## 五、算法实现

这里根据算法设计的两种不同的思路给出两种实现方式。

①记忆化搜索的实现

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e4+10;
int dp[N][N];
int solve(int l,int r,int *p)
{
    if(dp[l][r]!=0) return dp[l][r];
    if(l==r)
    {
        dp[l][r]=0;
        return 0;
    }
    int ans=0x7fffffff;
    for(int i=l;i<=r-1;i++) ans=min(solve(l,i,p)+solve(i+1,r,p)+p[r]-p[l-1],ans);
    dp[l][r]=ans;
    return ans;
}
void MinCost(int L,int n,int *p)
{
    sort(p,p+n+1);
    for(int i=n+1;i>=1;i--) p[i]=p[i]-p[i-1];
    for(int i=1;i<=n+1;i++) p[i]=p[i]+p[i-1];
    int ans=solve(1,n+1,p);
    cout<<ans;
}
int main()
{
    int L,n,*p;
    cin>>L>>n;
    p=new int[n+2];
    p[0]=0;
    p[n+1]=L;
    for(int i=1;i<n+1;i++) cin>>p[i];
    MinCost(L,n,p);
    return 0;
}
```

②自低向上递推思想的实现

```cpp
#include<bits/stdc++.h>
using namespace std;
void MinCost(int L,int n,int *p)
{
    int inf=0x7fffffff,**dp=new int *[n+2];
    for(int i=0;i<=n+1;i++) dp[i]=new int[n+2];
    sort(p,p+n+1);
    for(int i=n+1;i>=1;i--) p[i]=p[i]-p[i-1];
    for(int i=1;i<=n+1;i++) p[i]=p[i]+p[i-1];
    for(int i=1;i<=n+1;i++) dp[i][i]=0;
    for(int len=2;len<=n+1;len++)
    {
        for(int i=1;i<=n+1-len+1;i++)
        {
            int j=i+len-1;
            dp[i][j]=inf;
            for(int k=i;k<=j-1;k++) dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+p[j]-p[i-1]);
        }
    }
    cout<<dp[1][n+1];
}
int main()
{
    int L,n,*p;
    cin>>L>>n;
    p=new int[n+2];
    p[0]=0;
    p[n+1]=L;
    for(int i=1;i<n+1;i++) cin>>p[i];
    MinCost(L,n,p);
    return 0;
}
```

## 七、运行结果

首先我们可以制作一个简单的随机数据生成器，受版面的限制以及记录的方便，这里随机数据生成 n 和 L 较小，但只要稍微修改下面程序的参数，即可生成想要数据规模的数据。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
map<int,int> mp;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n,L,p[N];
    srand(time(0));
    n=rand()%100;
    L=rand()%1000;
    for(int i=1;i<=n;i++)
    {
        p[i]=rand()%(L+1);
        if(mp[p[i]]) i--;
        mp[p[i]]=1;
    }
    cout<<L<<" "<<n<<"\n";
    for(int i=1;i<=n;i++) cout<<p[i]<<" ";
    cout<<"\n";
    return 0;
}
```

第一组以下是部分数据输入与输出

生成数据程序输出：

```
345 31
31 213 339 182 24 323 178 81 122 273 39 8 50 286 220 46 263 167 16 297 312 176 47 63 160 22 20 96 189 9 126
```

①算法程序输入和输出：

```
345 31
31 213 339 182 24 323 178 81 122 273 39 8 50 286 220 46 263 167 16 297 312 176 47 63 160 22 20 96 189 9 126
1612
```

②算法程序输入和输出：

```
345 31
31 213 339 182 24 323 178 81 122 273 39 8 50 286 220 46 263 167 16 297 312 176 47 63 160 22 20 96 189 9 126
1612
```

答案为 1612，程序运行正确

第二组

生成数据程序输出：

```
481 29
153 90 77 182 357 211 294 194 393 458 100 125 114 35 284 455 71 480 224 370 220 107 238 459 380 142 395 65 173
```

①算法程序输入和输出：

```
481 29
153 90 77 182 357 211 294 194 393 458 100 125 114 35 284 455 71 480 224 370 220 107 238 459 380 142 395 65 173
2184
```

②算法程序输入和输出：

```
481 29
153 90 77 182 357 211 294 194 393 458 100 125 114 35 284 455 71 480 224 370 220 107 238 459 380 142 395 65 173
2184
```

答案为 2184，程序运行正确

第三组

生成数据程序输出：

```
997 52
142 870 980 124 917 203 920 747 658 147 79 510 554 256 3 566 474 490 349 432 680 775 242 529 213 643 811 461 520 814 616
 606 861 273 534 430 274 344 514 397 871 84 771 961 549 445 964 664 64 363 728 411
```

①算法程序输入和输出：

```
997 52
142 870 980 124 917 203 920 747 658 147 79 510 554 256 3 566 474 490 349 432 680 775 242 529 213 643 811 461 520 814 616
 606 861 273 534 430 274 344 514 397 871 84 771 961 549 445 964 664 64 363 728 411
5452
```

②算法程序输入和输出：

```
997 52
142 870 980 124 917 203 920 747 658 147 79 510 554 256 3 566 474 490 349 432 680 775 242 529 213 643 811 461 520 814 616
 606 861 273 534 430 274 344 514 397 871 84 771 961 549 445 964 664 64 363 728 411
5452
```

答案为 5452，程序运行正确

……

为了进一步验证程序的正确性，我们可以通过对这两个代码的输出结果进行比较，即一个设计了一个简单的对拍程序，T 表示测试的总数据量输出 Accepted！表示两个程序在 T 组数据下输出结果完全一致，代码如下：

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10;
map<int,int> mp;
int dp[N][N],p[N];
int solve(int l,int r,int *p)
{
    if(dp[l][r]!=0) return dp[l][r];
    if(l==r)
    {
        dp[l][r]=0;
        return 0;
    }
    int ans=0x7fffffff;
    for(int i=l; i<=r-1; i++) ans=min(solve(l,i,p)+solve(i+1,r,p)+p[r]-p[l-1],ans);
    dp[l][r]=ans;
    return ans;
}
int MinCost1(int L,int n,int *p)
{
    memset(dp,0,sizeof(dp));
    int ans=solve(1,n+1,p);
    return ans;
}
int MinCost2(int L,int n,int *p)
{
    int inf=0x7fffffff,**dp=new int *[n+2];
    for(int i=0; i<=n+1; i++) dp[i]=new int[n+2];
    for(int i=1; i<=n+1; i++) dp[i][i]=0;
    for(int len=2; len<=n+1; len++)
```

```
    {
        for(int i=1; i<=n+1-len+1; i++)
        {
            int j=i+len-1;
            dp[i][j]=inf;
            for(int k=i; k<=j-1; k++) dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+p[j]-p[i-1]);
        }
    }
    return dp[1][n+1];
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n,L,cnt=0,T,flag=1;
    cin>>T;
    while(T--)
    {
        srand(time(0));
        n=rand()%100;
        L=rand()%1000;
        mp.clear();
        for(int i=1; i<=n; i++)
        {
            p[i]=rand()%(L+1);
            if(mp[p[i]]||p[i]==0) i--;
            mp[p[i]]=1;
        }
        p[0]=0;
        p[n+1]=L;
        sort(p,p+n+1);
        for(int i=n+1; i>=1; i--) p[i]=p[i]-p[i-1];
        for(int i=1; i<=n+1; i++) p[i]=p[i]+p[i-1];
        int ans2=MinCost2(L,n,p);
        int ans1=MinCost1(L,n,p);
        if(ans1!=ans2)
        {
            flag=0;
            cout<<"Wrong Answer!\n"<<ans1<<"    "<<ans2<<"\n";
            cout<<L<<" "<<n<<"\n";
            for(int i=1; i<=n; i++) cout<<p[i]<<" ";
            break;
        }
    }
    if(flag) cout<<"Accepted!\n";
    return 0;
}
```

输入输出结果：

测试 1000 组随机数据输入输出结果：

```
1000
Accepted!
```

测试 10000 组随机数据输入输出结果：

```
10000
Accepted!
```