

# 算法设计实验报告



实验名称：动态规划综合实验

姓名：蒋雪枫

班级：2017211314

学号：2017213508

专业：网络工程

指导教师：叶文老师

2019 年 11 月 13 日

## 一、综述

动态规划算法和分治法的思想有相似之处，它们都意图通过减小问题的规模来提高我们解决问题的效率。毫无疑问，动态规划是非常重要的算法。回顾大学生活，学生接触过它好几次，每次遇到的时候都需要再学一下，翻翻书，看看网上的一些资料，觉得“噢，我懂了”。但真正让自己面对题目独自编程的时候，老是写不太对，从而产生一种莫名的恐惧感。

本次实验，学生会结合叶文老师布置的任务和自己对于动态规划算法的再一次理解展开说明，并且这一次试着去接触自己不太擅长的 C/C++ 语言来完成本次实验，觉得学计算机的怎么也得不求精通 C++ 也得看得懂也会用 C++，于是这次抛弃了自己较为擅长的语言，希望这次实验进行得顺利：)

在实验之后，自己也会多多练习与总结这种题目，来培养自己面对此类问题的“内功”。实验过程中遇到了一些小障碍，一些总结会在后面一一道来。

## 二、简单谈谈动态规划

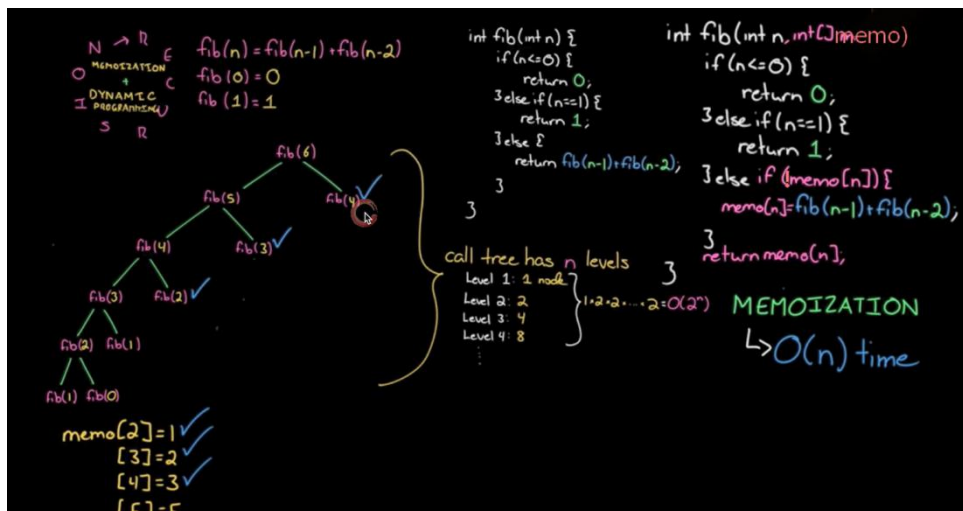
正如前面所说，动态规划是一种非常重要的算法，经常在面试题中出现，缺乏相应练习的同学（比如我）就很容易做不出来从而凉凉。但其实，动态规划并不是一个很玄学的算法，也不是一个所谓的“高瞻远瞩地规划的算法”。Dynamic Programming，这里的 programming 其实是递推的意思，所以就是动态递推。

学生认为，动态规划有四个要点：

- 1) 递推+记忆化==>动态递推
- 2) 状态的定义很重要，一维(DP[])还是二维(DP[][]))才能够解决问题？
- 3) 状态转移方程
- 4) 最优子结构性质

以下图为例子，还是我们经常谈及的斐波那契数列问题。如果我们以递归来分析该问题，则我们可以直接这么处理“return fib(n-1)+fib(n-1) if n>=1 else n”，这样会导致我们在推进问题的时候有了很多不必要的计算，从递归树也很容易看出它的复杂度是  $O(2^n)$ 。而如果我们加上记忆化，保留一些子问题的答案，那么我们则可以避免重复计算，在  $O(n)$  复杂度以内解决问题。这里，我们就实现了通过记忆化来减少问题规模的目标。而我们现在从底向上分析问题，先计算 fib(0) 和 fib(1)，我们来动态递推问题的答案，则我们就有了“动态规划”的想法了。(这里的话，如果用动态规划，四行之内就可以算出斐波那契数列的项了)

动态规划也不是万能的，有时也会遇到负载较大，代价较大的情况，这时候要么我们就考虑优化算法，要么就考虑放弃最优解选择启发式。比如斐波那契数列就有  $O(1)$  的算法。



### 三、最长公共子序列问题

首先，我们先要知道什么是公共子序列。比如  $X = \text{【A,B,C,B,D,A,B】}$  ,  $Y = \text{【B,D,C,A,B,A】}$  , 则它们最长的公共子序列就是  $\text{【B, C, B, A】}$  。

在本次实验中，我们通过有意初始化 ABCD 字符串，并试图从中提取它们之间的最长子序列，来分析公共子序列问题。

$$c[i][j] = \begin{cases} 0 & i = 0, \text{或} j = 0 \\ c[i-1][j-1] + 1 & i, j > 0; x_i = y_j \\ \max\{c[i][j-1], c[i-1][j]\} & i, j > 0; x_i \neq y_j \end{cases}$$

这是状态转移方程，其中我们使用  $c[i][j]$  来表示  $x_i$  和  $y_j$  两个序列的最长子序列长度。这里  $i$  和  $j$  表示下标+1 的自身局部序列。核心代码如下：

```
void LCSLength(char *x, char *y, int m, int n, int c[][MAXLEN], int b[][MAXLEN])
{
    int i, j;
    //这里，如果 X 序列和 Y 序列有一个为空串，它们最大子序列长度就是 0
    for(i=0; i<=m; i++)
        c[i][0]=0;
    for(j=1; j<=n; j++)
        c[0][j]=0;
    for(i=1; i<=m; i++)
    {
        for(j=1; j<=n; j++)
        {
            //然后，我们开始“动态递推”，开始填写一个最长子序列表 Table[m][n]=c[m][n]
            //如果 X[i-1]==Y[j-1]，则找到了一个构成子序列的字符
            //比如，第一个 X="ILOVEYOU"，第二个是"IHATEYOU"，则我们填的第一个项目就是 c[1][1]=1
            //这意味着“X 第一个字符和 Y 的第一个字符构成的仅有一个字符的小串儿”就是其 LCS
            if(x[i-1]==y[j-1])
            {
                c[i][j]=c[i-1][j-1]+1;
                b[i][j]=0;
            }
            else if(c[i-1][j]>=c[i][j-1])
            {
                c[i][j]=c[i-1][j];
                b[i][j]=1;
            }
            else
            {
                c[i][j]=c[i][j-1];
                b[i][j]=-1;
            }
        }
    }
}
```

```

    }
}
}
//这里，我们根据 b 矩阵留下来的信息进行判断
//如果 b[i][j]=0 的话，这说明 X[1:i]和 Y[1:j]的末位字符相同，加入 LCS 当中
//如果 b[i][j]=1 的话，这说明去 Yj 串不动，Xi 串往前挪动一个指针，可以往前展望下一个 LCS 字符
//为-1 同理
//由于函数是递归调用的，所以先打印 LCS 前面的字符，再打印 LCS 后面的字符
void PrintSeq(int b[][MAXLEN],char *x,int i,int j)
{
    if(i==0||j==0)
        return;
    if(b[i][j]==0)
    {
        PrintSeq(b,x,i-1,j-1);
        printf("%c",x[i-1]);
    }
    else if(b[i][j]==1)
        PrintSeq(b,x,i-1,j);
    else
        PrintSeq(b,x,i,j-1);
}

```

我们分别计算 AB，CD，AD 和 CB 的 LCS，运行结果如下：

```

C:\Users\Administrator\Desktop\Algorithm\Algorithm Practica\Homework03\Homework03.exe
analgorithm+is+any+welldefined+computational+procedure+that+takes+some+values+as+input+and+produces+some+values+as+output
=====
analgorithm+is+any+welldefined+computational+procedure+that+takes+some+values+as+input+and+produces+some+values+as+output
=====
analgorithm+is+any+welldefined+computational+procedure+that+takes+some+values+as+input+and+produces+some+values+as+output
=====
analgorithm+is+any+welldefined+computational+procedure+that+takes+some+values+as+input+and+produces+some+values+as+output
=====
Process exited after 0.1201 seconds with return value 0
请按任意键继续. . .

```

当我们完成第一个实验的时候，我们就很清楚了。其实动态规划，就是动态递推。

## 四、最大字段和

最大子段和有多种解法，比较常见的是  $O(N^2)$ 和  $O(N^3)$  的算法。在《算法设计与分析》第三章的内容中，我们还学习了分治法解决这种题的办法，即算算左边，算算右边，再从中间开始，算算左右和。这种想法是源于这样的子段要么全在左边，要么全在右边，要么左右均有。而本次实验我们使用动态规划的方法，在线处理( $O(N)$ 复杂度)这个问题。

状态转移方程：

$$b[j] = \max \{b[j-1] + a[j], a[j]\}, \quad 1 \leq j \leq n$$

核心代码如下：

```
int MaxSum(int n,int *a,int &l,int &r)
{
    int sum=0,b=0;
    int pos=1;
    for(int i=0;i<=n;i++)
    {
        if(b>0)
        {
            b+=a[i];
        }
        else
        {
            b=a[i];
            pos=i;
        }
        if(b>sum)
        {
            sum=b;
            l=pos;
            r=i;
        }
    }
    return sum;
}
```

分别将两个序列读入 stdin 中，得到如下的结果：

序列一：

最大字段和为:2818

最大字段位于第42个元素到第358个元素

序列二：

最大字段和为:377

最大字段位于第71个元素到第141个元素

## 五、凸多边形最优三角剖分问题

老师在课堂上已经给我们介绍了凸多边形的最优三角剖分问题。这种问题在几何上，我们可以把它解体为一个二叉树结构，而恰好我们的矩阵最优连乘方法也可以展开成一个二叉树。所以这两种问题具有同构性，它们都对应于同一个模型，解法也应该相似。

```

.cpp 附件1.最长公共子序列输入文件-4-26.txt 附件2.最大子段和输入数据-序列2.txt 附件2.最大子段和输入数据-序列1.txt MaxSubPartSum.cpp TriangleCut1.cpp
int main()
{
    ifstream in("附件3-2.21个基站凸多边形数据.txt");
    int COUNT, ID, i, j;
    double Long, Lati, ans, sum=0.0;
    while(!in.eof())
    {
        in>>ID>>Long>>Lati>>i;
        node[i]=Node(ID, Long, Lati, i);
        // memset(dp[i], 0, sizeof(dp[i]));
        printf("%d %f %f %d\n", ID, Long, Lati, i);
    }
    COUNT=i;
    ans=solve(COUNT);
    out<<"最优三角形的剖分结果为"<<endl;
    // printf("calledB");
    Recall(1, COUNT);

    printf("%.6f千米", ans);
    out<<"最小边长弦长总和为"<<ans<<"千米"<<endl;
    out.close();
    in.close();
    return 0;
}
567580 102.675000 24.995970 15
566022 102.676000 24.992480 16
566038 102.686000 24.980320 17
567508 102.704000 24.971840 18
565676 102.740000 24.957210 19
565466 102.746000 24.955500 20
0 19 20
0 18 19
0 12 18
0 6 12
0 1 6
1 5 6
1 3 5
1 2 3
3 4 5
6 8 12
6 7 8
8 9 12
9 11 12
9 10 11
12 15 18
12 13 15
13 14 15
15 17 18
15 16 17
295.846995千米
Process exited after 0.2636 seconds with return value 0
请按任意键继续. . .

```

```

main.cpp 附件1.最长公共子序列输入文件-4-26.txt 附件2.最大子段和输入数据-序列2.txt 附件2.最大子段和输入数据-序列1.txt MaxSubPartSum.cpp TriangleCut1.cpp
98
99 int main()
100 {
101     ifstream in("附件3-2.29个基站凸多边形数据.txt", ios::in);
102     int COUNT, ID, i, j;
103     double Long, Lati, ans, sum=0.0;
104     while(!in.eof())
105     {
106         in>>ID>>Long>>Lati>>i;
107         node[i]=Node(ID, Long, Lati, i);
108         // memset(dp[i], 0, sizeof(dp[i]));
109         printf("%d %f %f %d\n", ID, Long, Lati, i);
110     }
111     COUNT=i;
112     ans=solve(COUNT);
113     out<<"最优三角形的剖分结果为"<<endl;
114     // printf("calledB");
115     Recall(1, COUNT);

116
117     printf("%.6f千米", ans);
118     out<<"最小边长弦长总和为"<<ans<<"千米"<<endl;
119     out.close();
120     in.close();
121     return 0;
122 }
123
565668 102.732000 24.982210 28
565668 102.732000 24.982210 28
0 1 28
1 2 28
2 24 28
2 5 24
2 3 5
3 4 5
5 16 24
5 10 16
5 8 10
5 7 8
5 6 7
8 9 10
10 13 16
10 12 13
10 11 12
13 15 16
13 14 15
16 21 24
16 19 21
16 17 19
17 18 19
19 20 21
21 23 24
21 22 23
24 27 28
24 26 27
24 25 26
194.329148千米

```

而对于启发式算法，其设计思想是：

- E.g. 三角剖分

$$t[i][j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{t[i][k] + t[k+1][j] + w(v_{i-1}v_kv_j)\} & i < j \end{cases}$$

穷搜

- 不需要搜索、比较  $P\{v_{i-1}, v_i, \dots, v_j\}$  中的每个  $v_k$ ，而是按照一定启发式信息，优先选取某个（或最多常数  $K=5$  个）剖分点  $v_k$ ，e.g.  $v_k = v_3$ ，避免最内层循环，将算法复杂性由  $O(n^3)$  降为  $O(n^2)$

即将 DP 算法最后的 BruteForce 限制次数，变成  $O(N^2)$ 。介于算法平时分已经到手，这里就不多计算了，就对上面的 Solve 函数的最后一次循环限定次数， $O(N^3)$  变  $O(N^2)$  而已。

源代码如下：

```

#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdio>

```

```

#include<cstring>
#include<algorithm>
using namespace std;
#define M_PI 3.14159265358979323846
const double Earth_R=6378.137;
ofstream out("最优三角剖分结果.txt");

struct Node
{
    int ENodeID,seq;
    double LongiTude,LatiTude;
    double radLat,radLon;
    Node(){
    }
    Node(int ID,double Long,double Lati,int ord):ENodeID(ID),LongiTude(Long),LatiTude(Lati),seq(ord)
    {
        radLat=LatiTude*M_PI/180.0;
        radLon=LongiTude*M_PI/180.0;
    }
}node[50];
double dist(int a,int b)
{
    struct Node p1=node[a],p2=node[b];
    return Earth_R*acos(cos(p1.radLat)*cos(p2.radLat)*cos(p1.radLon-p2.radLon)+sin(p1.radLat)*sin(p2.radLat));
}
double dp[50][50],s[50][50],cunt;
double Weight(int a,int b,int c)
{
    return dist(a,b)+dist(a,c)+dist(b,c);
}
double solve(int n)
{
    for(int i=1;i<=n;i++) dp[i][i]=0;
    for(int r=2;r<=n;r++)
    {
        for(int i=1;i<=n-r+1;i++)
        {
            int j=i+r-1;
            dp[i][j]=dp[i+1][j]+Weight(i-1,i,j);
            s[i][j]=i;
            for(int k=i+1;k<=i+r-1;k++)
            {

```

```

        double u=dp[i][k]+dp[k+1][j]+Weight(i-1,k,j);
        if(u<dp[i][j])
        {
            dp[i][j]=u;
            s[i][j]=k;
        }
    }
}

return dp[1][n];
}

void Recall(int h,int n)
{
    if(n==h) return;
    cout<<h-1<<" "<<s[h][n]<<" "<<n<<endl;
    out<<"点"<<h-1<<"", "<<s[h][n]<<"", "<<n<<endl;
    Recall(h,s[h][n]);
    Recall(s[h][n]+1,n);
}

int main()
{
    ifstream in("附件 3-2.29 个基站凸多边形数据.txt",ios::in);
    int COUNT,ID,i,j;
    double Long,Lati,ans,sum=0.0;
    while(!in.eof())
    {
        in>>ID>>Long>>Lati>>i;
        node[i]=Node(ID,Long,Lati,i);
        // memset(dp[i],0,sizeof(dp[i]));
        printf("%d %f %f %d\n",ID,Long,Lati,i);
    }
    COUNT=i;
    ans=solve(COUNT);
    out<<"最优三角形的剖分结果为"<<endl;
    // printf("calledB");
    Recall(1,COUNT);

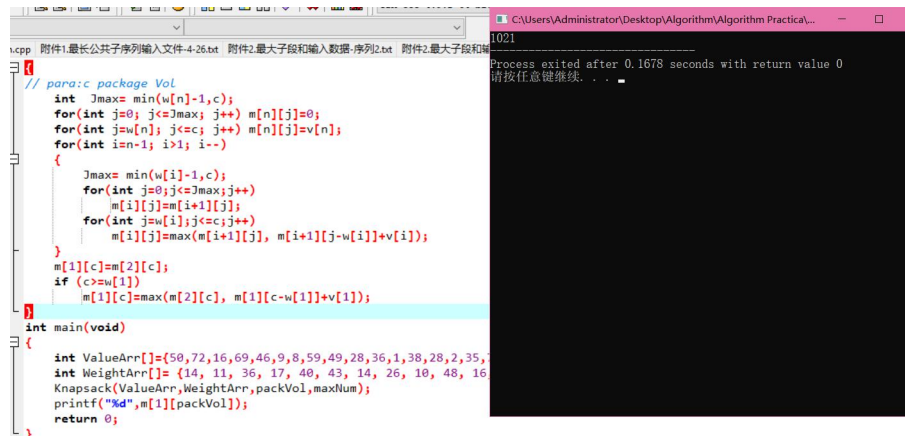
    printf("%.6f 千米",ans);
    out<<"最小边长弦长总和为"<<ans<<"千米"<<endl;
    out.close();
    in.close();
    return 0;
}

```



## 六、0-1 背包

背包问题是常见的动态规划问题,在此不多赘述.



```
// para:c package Vol
int Jmax= min(w[n]-1,c);
for(int j=0; j<=Jmax; j++) m[n][j]=0;
for(int j=w[n]; j<=c; j++) m[n][j]=v[n];
for(int i=n-1; i>1; i--)
{
    Jmax= min(w[i]-1,c);
    for(int j=0; j<=Jmax; j++)
        m[i][j]=m[i+1][j];
    for(int j=w[i]; j<=c; j++)
        m[i][j]=max(m[i+1][j], m[i+1][j-w[i]]+v[i]);
}
m[1][c]=m[2][c];
if (c>=w[1])
    m[1][c]=max(m[2][c], m[1][c-w[1]]+v[1]);

int main(void)
{
    int ValueArr[]={50,72,16,69,46,9,8,59,49,28,36,1,38,28,2,35,
    int WeightArr[]={14, 11, 36, 17, 40, 43, 14, 26, 10, 48, 16
    Knapsack(ValueArr,WeightArr,packVol,maxNum);
    printf("%d",m[1][packVol]);
    return 0;
}
```

1021  
Process exited after 0.1678 seconds with return value 0  
请按任意键继续. . .

## 七、小结

关于动态规划问题,想说的还有很多.本次实验让我亲自实现了一些实际背景下的算法,但其实学生自己还没有把一些基本模型(板子)吃透.最近比较忙,所以本次实验完成得比较仓促,一些实验要求没按标准完成,不过等这两周闲下来了肯定会重新再把这些东西拿出来重新理解与记忆,哎,这两周真的太忙了!