原文地址:http://blog.csdn.net/y990041769/article/details/24194605

区间 dp 顾名思义就是在一个区间上进行的一系列动态规划。对一些经典的区间 dp 总结在这里。

1）石子归并问题

题目链接：http://acm.nyist.edu.cn/JudgeOnline/problem.php?pid=737

描述： 有 N 堆石子排成一排，每堆石子有一定的数量。现要将 N 堆石子并成为一堆。合并的过程只能每次将相邻的两堆石子堆成一堆，每次合并花费的代价为这两堆石子的和，经过 N-1 次合并后成为一堆。求出总的代价最小值。

分析：要求 n 个石子归并，我们根据 dp 的思想划分成子问题，先求出每两个合并的最小代价，然后每三个的最小代价，依次知道 n 个。

定义状态 dp [i] [j]为从第 i 个石子到第 j 个石子的合并最小代价。

那么 **dp [i] [j] = min(dp [i] [k] + dp [k+1] [j])**

那么我们就可以从小到大依次枚举让石子合并，直到所有的石子都合并。

这个问题可以用到平行四边形优化，用一个 s【i】【j】=k 表示区间 i---j 从 k 点分开才是最优的，这样的话我们就可以优化掉一层复杂度，变为 O（n^2）.

代码：

[cpp] view plain copy print?

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
#define N 210
int dp[N][N],sum[N];
int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        int a[N];sum[0]=0;
        for(int i=1;i<=n;i++){
            scanf("%d",&a[i]);
            sum[i]=sum[i-1]+a[i];
        }
        memset(dp,0,sizeof(dp));
        int i,j,l,k;
        for(l = 2; l <= n; ++l)
        {
            for(i = 1; i <= n - l + 1; ++i)
            {
```

```
                j = i + l - 1;
                dp[i][j] = 2100000000;
                for(k = i; k < j; ++k)
                {
                    dp[i][j] = std::min(dp[i][j],dp[i][k] + dp[k + 1][j] + sum[j] - sum[i-1]);
                }
            }
        }
        printf("%d\n", dp[1][n]);
    }
    return 0;
}
```

平行四边形优化代码：

[cpp] view plain copy print?

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define N 210
int dp[N][N],sum[N],s[N][N];
int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        int a[N];sum[0]=0;
        memset(s,0,sizeof(s));
        for(int i=1;i<=n;i++){
            scanf("%d",&a[i]);
            s[i][i]=i;
            sum[i]=sum[i-1]+a[i];
        }
        memset(dp,0,sizeof(dp));
        int i,j,l,k;
        for(l = 2; l <= n; ++l)
        {
            for(i = 1; i <= n - l + 1; ++i)
            {
                j = i + l - 1;
                dp[i][j] = 2100000000;
                for(k = s[i][j-1]; k <= s[i+1][j]; ++k)
                {
```

```
                    if(dp[i][j]>dp[i][k] + dp[k + 1][j] + sum[j] - sum[i-1])
                    {
                        dp[i][j]=dp[i][k] + dp[k + 1][j] + sum[j] - sum[i-1];
                        s[i][j]=k;
                    }
                }
            }
        }
        printf("%d\n", dp[1][n]);
    }
    return 0;
}
```

2)括号匹配

题目链接：

poj2955,http://poj.org/problem?id=2955

nyoj 15 http://acm.nyist.edu.cn/JudgeOnline/problem.php?pid=15

描述：给出一串的只有‘(’ ‘)’ '[’ ']'四种括号组成的串，让你求解需要最少添加括号数让串中的所有括号完全匹配。

分析：我们求出这个串的最大匹配，然后串的总长度-最大匹配就是答案。

方法 1:首先能想到的是转化成 LCS（最长公共子序列），枚举中间点，求所有的 LCS 中的最大值 * 2 就是最大匹配。但是复杂度较高，光 LCS 一次就 O(n^2)的复杂度。

方法 2：

首先考虑怎么样定义 dp 让它满足具有通过子结构来求解、

定义 dp [i] [j] 为串中第 i 个到第 j 个括号的最大匹配数目

那么我们假如知道了 i 到 j 区间的最大匹配，那么 i+1 到 j+1 区间的是不是就可以很简单的得到。

那么 假如第 i 个和第 j 个是一对匹配的括号那么 **dp [i] [j] = dp [i+1] [j-1] + 2 ;**

那么我们只需要从小到大枚举所有 i 和 j 中间的括号数目，然后满足匹配就用上面式子 dp，然后每次更新 dp [i] [j]为最大值即可。

更新最大值的方法是枚举 i 和 j 的中间值，然后让 **dp[i] [j] = max (dp [i] [j] , dp [i] [f] + dp [f+1] [j]) ;**

如果要求打印路径，即输出匹配后的括号。见

http://blog.csdn.net/y990041769/article/details/24238547 **详细讲解**

代码：

[cpp] view plain copy print?

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
using namespace std;
const int  N = 120;
int dp[N][N];
int main()
{
    string s;
    while(cin>>s)
    {
        if(s=="end")
            break;
        memset(dp,0,sizeof(dp));
        for(int i=1;i<s.size();i++)
        {
            for(int j=0,k=i;k<s.size();j++,k++)
            {
                if(s[j]=='('&&s[k]==')' || s[j]=='['&&s[k]==']')

                    dp[j][k]=dp[j+1][k-1]+2;
                for(int f=j;f<k;f++)
                    dp[j][k]=max(dp[j][k],dp[j][f]+dp[f+1][k]);
            }
        }
        cout<<dp[0][s.size()-1]<<endl;
    }
    return 0;
}
```

3）整数划分问题

题目链接：nyoj746 http://acm.nyist.net/JudgeOnline/problem.php?pid=746

题目描述：给出两个整数 n , m ,要求在 n 中加入 m - 1 个乘号，将 n 分成 m 段，求出这 m 段的最大乘积

分析：根据区间 dp 的思想，我们定义 dp [i] [j]为从开始到 i 中加入 j 个乘号得到的最大值。

那么我们可以依次计算加入 1----m-1 个乘号的结果

而每次放入 x 个乘号的最大值只需枚举第 x 个乘号的放的位置即可

dp [i] [j] = MAX (dp [i] [j] , dp [k] [j-1] * a [k+1] [i]) ;

代码：

[cpp] view plain copy print?

```cpp
#include <cstdio>
#include <cstring>
#define MAX(a,b) a>b?a:b
long long a[20][20];
long long dp[25][25];
int main()
{
    int T,m;
    scanf("%d",&T);
    getchar();
    while(T--)
    {
        char s[22];
        scanf("%s",s+1);
        scanf("%d",&m);
        int l=strlen(s),ok=1;
        memset(a,0,sizeof(a));
        for(int i=1;i<l;i++)
        {
            if(s[i]=='0')
                ok=0;
            for(int j=i;j<l;j++)
            {
                a[i][j]=a[i][j-1]*10+(s[j]-'0');
            }
        }
```

```
        if(ok==0&&l-1==m||l-1<m)
        {
            printf("0\n");continue;
        }
        long long x,ans;
        memset(dp,0,sizeof(dp));
        for(int i=0;i<l;i++)
            dp[i][1]=a[1][i];
        ans=0;
        if(m==1)
            ans=dp[l-1][1];
        for(int j=2;j<=m;j++)
        {
            for(int i=j;i<l;i++)
            {
                ans=a[i][i];
                for(int k=1;k<i;k++)
                {
                    dp[i][j]=MAX(dp[i][j],dp[k][j-1]*a[k+1][i]);

                }
            }
        }
        printf("%lld\n",dp[l-1][m]);
    }
    return 0;
}
```