Note on Dynamic Programming (II)

Day 1 PM to Day 2 NIGHT—Classical Problem

Hatsune Miku

目录

1 线性模型

1.1 最长上升子序列

二分优化掉内层循环。注意到 LIS 的一个性质，如果当前的数比之前的某个数大，那么它的 LIS 长度比之前的那个数大。注意到之前 $O(n^2)$ 的算法的内层循环的问题。

明显地我们可以使用二分技术优化内层循环，维护一个数组 g[]，使得它单调，然后二分查找它的 LIS 的长度。

1.2 滑雪

dp[i][j] 表示到 (i,j) 这个格子的最长滑雪路线长度。

用推的方法：

$$dp[x][y] \to dp[i][j] |x - x'| + |y - y'| = 1, a[x'][y'] < a[x][y]$$

按照所有点的高度进行排序，然后按照高度由高到低枚举即可。

```
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

#include <algorithm>
#include <vector>
#include <utility>

using namespace std;

const int dx[] = {-1, 1, 0, 0};
const int dy[] = {0, 0, -1, 1};

void update(int &a, int b) {
    if (a < b) {
        a = b;
```

```
    }
}

int a[100007];
int dp[100007];
int n, m, ans;

vector<pair<int, int> > b;

bool cmp(const pair<int, int> &x, const pair<int, int> &y) {
    return a[x.first * m + x.second] > a[y.first * m + y.second];
}

int main(void) {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            scanf("%d", a + (i*m+j));
            b.push_back(make_pair(i, j));
        }
    }

    sort(b.begin(), b.end(), cmp);
    for (int i = 0; i < b.size(); ++i) {
        int x = b[i].first, y = b[i].second;
        update(ans, dp[x * m + y]);
        for (int k = 0; k < 4; ++k) {
            int x1 = x + dx[k], y1 = y + dy[k];
            if (x1 >= 0 && x1 < n && y1 >= 0 && y1 < m) {
                if (a[x1 * m + y1] < a[x * m + y]) {
                    update(dp[x1 * m + y1], dp[x * m + y] + 1);
                }
            }
```

```
        }
    }

    return 0;
}
```

1.3 最长不互斥子序列

给定一个序列，找出最长不互斥子序列，即 $b[i]\,\text{and}\,b[i-1] \neq 0$.

类似 LIS 的做法，维护一个 g[] 数组，使它表示满足不互斥性质的子序列的长度。

```
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

int n, a[100007], dp[100007], g[32];
int ans;

void update(int &a, int b) {
    if (a < b) a = b;
}

int main(void) {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", a+i);
    }
    for (int i = 0; i < n; ++i) {
        int temp = 0;
        for (int k = 0; k < 31; ++k) {
            if (a[i] & (1 << k)) {
                update(temp, g[k]);
            }
```

```
            }
            dp[i] = temp + 1;
            for (int k = 0; k < 31; ++k) {
                if (a[i] & (1 << k)) {
                    update(g[k], dp[i]);
                }
            }

            update(ans, dp[i]);
        }
        printf("%d\n", ans);

        return 0;
}
```

1.4 回文串划分

给一个字符串，划分成最少个回文子串。长度不超过 1000.
令 $dp[i]$ 表示已经将字符串的第 1 到 i 位处理完毕的最少划分次数。

$$dp[i] = \min\{dp[j] + 1\}, j < i, s[j, i]\text{是回文串}.$$

使用字符串 Hash 算法，判断是否是回文串（只需要对一个字符串正过来做一半的 Hash，倒着再做一遍 Hash，判断 Hash 是否相等。）

or:

- 若回文串长度为奇数，可以预处理一个数组，表示以 i 点为中心的最长回文串长度 => $O(n^2)$.
- 若回文串长度为偶数：
  - 额外处理一个数组，表示以 i 和 i+1 为中心的最长回文串长度（使得 a[i]=a[i+1],a[i-1]=a[i-2]）=>$O(n^2)$.
  - 在两个字符之间都插入一个特殊字符，然后这个串的长度就变成了奇数，再按照长度为奇数的方法做.

```cpp
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cmath>

char s[100007];
int a[100007], b[100007];
int n;
int dp_[100007], *dp = dp_ + 1;

void update(int &a, int b) {
    if (a > b) {
        a = b;
    }
}

void init(void) {
    for (int i = 0; i < n; ++i) {
        int x = i - 1, y = i + 1;
        while (x >= 0 && y < n && s[x] == s[y]) {
            --x; ++y;
        }
        a[i] = (i - x);
    }
    for (int i = 0; i < n; ++i) {
        int x = i, y = i + 1;
        while (x >= 0 && y < n && s[x] == s[y]) {
            --x; ++y;
        }
        b[i] = (i - x);
    }
}
```

```
bool is_para(int x, int y) {
    int t = x + y;
    if (t % 2) {
        int ext = b[(t / 2)];
        return ext >= (y - x + 1) / 2;
    } else {
        int ext = a[(t / 2)];
        return ext >= (y - x) / 2 + 1;
    }
}

void work(void) {
    for (int i = 0; i < n; ++i) {
        dp[i] = n + 1;
    }
    dp[-1] = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = -1; j < i; ++j) {
            if (is_para(j+1, i)) {
                update(dp[i], dp[j] + 1);
            }
        }
    }
}

int main(void) {
    scanf("%s", s);
    n = strlen(s);
    init();
    work();
    printf("%d\n", dp[n-1]);
    return 0;
}
```

1.5 传球问题

有 N 个人排成一个环，每个人选择向左或向右传球，最后一个拿到球的人输。问游戏进行 M 轮，第 i 个人输的方案数是多少。N 不超过 30，M 不超过 30。

dp[i,x] 表示经过了 i 轮，第 x 个人拿到球的方案数。

$$\begin{aligned} dp[i,x] &= \min\{dp[i-1,p] + C(p,r)\} \\ dp[i][x] &= \sum_{(p,r)} dp[i-1,p] \end{aligned}$$

1.6 阶梯序列

B 序列是梯子序列，当且仅当：存在 x 使得

$$B(1) \le B(2) \le \cdots \le B(x) \ge B(x+1) \ge \cdots \ge B(N).$$

给定一个序列 A，有 Q 次询问 $A(L \ldots R)$ 是不是梯子序列.

$N, Q \le 10^5$.

做一遍最长不降子序列+最长上升子序列。

预处理 up[i] 表示以 i 为起点向左最大有多少个单调上升的数，同样预处理 dowm[i] 数组表示向右有多少个单调上升的数，然后判断一下区间 [L,R] 内的 up 与 down 的长度。

1.7 区间染色

给定一个长度为 N 的序列，每一位有一个目标颜色，每次可以选择一个区间，将区间内的所有元素改为其目标颜色。设区间内不同颜色的数量为 X，则操作的代价为 X^2。求最小代价。$N \le 5 \times 10^4$. 注意该点需要什么颜色就必须染成什么颜色，不能染成别的颜色。

很容易写出状态转移方程：

$$f[i] = \min\{f[j] + cost(j,i)\}, j < i;$$

1D1D 动态规划标准 DP 模型。

注意到直接输出 n 可以暴力骗分。

可以优化这个状态转移方程为 $f[i] = \min(f[g[i][x]] + x^2)$，其中 $g[i][x]$ 是记录前 i 个方格中有 x 种颜色。

g 数组的求法：对于数组元素 $g[i][x]$，我们有以下状态转移方程：

$$\begin{aligned} g[i+1][x] &= g[i][x] && \text{,第}i+1\text{个方格的颜色是前面所包含的;} \\ g[i][x] &= g[i+1][x] && \text{,第}i+1\text{个方格的颜色不是前面所包含的.} \end{aligned}$$

1.8 最长公共子序列

令 dp[i][j] 表示以第一个串的第 i 位与第二个串的第 j 位结尾的最长公共子序列长度。

状态的转移有三种：

$$dp[i][j] = \max \begin{cases} dp[i-1,j] \\ dp[i,j-1] \\ dp[i-1,j-1]+1, a[i]=b[j]. \end{cases}$$

1.9 最长公共上升子序列

1.9.1 三维 DP

令 dp[i,j,k] 表示以第一个串的第 i 位与第二个串的第 j 位结尾，最后一位是 k 的最长公共上升子序列的长度，则有状态转移方程：

$$dp[i,j,k] = \max \begin{cases} dp[i-1,j,k] \\ dp[i,j-1,k] \\ dp[i-1,j-1,k'], k < k'. \end{cases}$$

1.9.2 降维优化

dp[i,j] 表示以 a 串的第 i 位结尾，b 串的第 i-j 位结尾的 LCS，且最后元素为 b[j]，则有：

$$dp[i,j] = \max \begin{cases} dp[i-1,j] \\ dp[i-1,j'], & j' < j, \\ & b[j'] < b[j], \\ & a[i] = b[j]. \end{cases}$$

2 区间模型

2.1 矩阵乘法

矩阵：一个 $N \times M$ 的矩阵被定义为一个 N 行 M 列的数组，数组中的每个元素都是一个实数。一个向量可以表示为一个 $M \times 1$ 的矩阵。

应用：

矩阵乘向量：

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

有一个任意的向量 $x(x_0, y_0)$ 旋转$\theta°$，则有：

$$x_1 = x_0 \cdot \cos\theta - y_0 \cdot \sin\theta$$
$$y_1 = x_0 \cdot \sin\theta + y_0 \cdot \cos\theta$$

矩阵乘法：定义矩阵乘法运算规则如下：

$$\begin{aligned} C[i,j] &= \sum_{k=1}^{m} A_{ik} B_{kj} \\ &= A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{im}B_{mj} \end{aligned}$$

时间复杂度为 $O(n \times m \times k)$.

矩阵乘法与 DP 的关系：矩阵乘法优化递推转移。

例题：斐波那契数列升级版

题解：矩阵快速幂优化递推转移。

（抱歉，Atom 崩了，推导过程没写，具体过程可参照题解）

2.2 矩阵乘法

明显地，给矩阵乘法加括号的方案数等于卡特兰数。

另外，我们有以下性质：

1. 每一次做矩阵乘法的时候，都是将一个序列内连续区间的矩阵相乘，然后再乘上另外一个区间中的矩阵的乘积，最后再将两个矩阵乘起来。

2. 第一个区间中的矩阵乘积的行数等于第二个区间中矩阵乘积的列数。

定义 $dp[i,j]$ 表示把 $[i,j]$ 这个区间中的矩阵全部乘起来所需要花费的最小代价，则最终答案为 $dp[1][n]$，在转移时，枚举一个 k，表示最后一个乘上去的矩阵是 k，则我们有以下状态转移方程：

$$dp[i,j] = \min\{dp[i,k] + dp[k+1,j] + r[i] \times c[j] \times c[k].\}$$

注意转移时不能按照正常的顺序转移，枚举的应该是区间长度，然后再转移。或者可以使用 DFS 的方式进行转移。

可以使用记忆化搜索的方式进行优化。可以参照第一节记忆化搜索的写法。

可以很容易的发现，在递归的顺序或枚举的拓扑序不是十分明显时，使用 DFS 加上记忆化搜索可以使代码清晰明了。

时间复杂度 $O(n^3)$.

2.3 最优三角剖分（UVa 1626）

把一个 n 个顶点的凸多边形剖分成三角形。每一个三角形有唯一的权值函数 $w(i,j,k)$。求最优剖分方法，最大化权值和。n 不超过 100，假设函数 w 定义如下：

$$w(i,j,k) = e\cos C + i\sin S.$$

设有两个向量：

$$\alpha = (x_2 - x_1, y_2 - y_1)$$
$$\beta = (x_3 - x_1, y_3 - y_1)$$
$$\frac{1}{2}(\alpha \times \beta) = \frac{1}{2}|(\alpha_1\beta_1 - \alpha_2\beta_1)|$$

……（其余内容被留作课后作业）

dp[i,j] 表示以 i,j 两个顶点构成的三角形的最大权值，则有：

$$dp[i,j] = \begin{cases} 0 & ,i = j, \\ \min\limits_{i \le k \le j}\{t[i,k] + t[k+1,j] + w(v_{i-1}v_kv_j)\} & ,i < j. \end{cases}$$

2.4 括号序列

令 $dp[i,j]$ 表示区间 $[i,j)$ 范围内最少需要补多少个括号。

$$dp[i,j]=\begin{cases}0 & ,i=j,\\ 1 & ,i+1=j,\\ dp[i+1,j-1] & ,of\,(s)\,or\,[s],\\ dp[i,k]+dp[k,j] & ,i\le k<j.\end{cases}$$

2.5 区间染色 II

与括号序列有些类似，不妨参照括号序列的状态转移。令 dp[i,j] 把区间 $[i,j)$ 全部染色所需要的最小代价，则有：

$$dp[i,j]=\begin{cases}0 & ,i=j,\\ 1 & ,i+1=j,\\ dp[i,k]+dp[k,j] & ,\textbf{...TODO...}\end{cases}$$

2.6 传球问题 II

假设两个人来回传球，类似等比数列求和公式的处理。

$$\text{等比数列求和公式：}1+p+p^2+\cdots+p^n=\frac{p^n-1}{p-1},p\neq 1.$$

可以把相互传球的人看成一个整体，因为这些人无论相互传球传多少次，球最终总会被传出这些人的手里。考虑矩阵乘法这道题目，合并 $n\times m$ 的代价等于合并 $n\times k$ 的代价加上 $k\times m$ 的代价。

用类似的方法只考虑两个人，假设右边的人拿到了球向左传球的概率为 p_i，定义另外一个 $q_i=1-p_i$ 表示向右传球的概率。则球从右边传出去的概率 $P=q_j+p_jq_ip_j+p_jq_ip_jq_ip_j+\cdots$，容易看出这是一个等比数列，则从左边传出的概率与从右边传出的概率是一个定值，所以这两个人与一个人是没有区别的。

对于两个点而言，我们已经知道球从左边出去和从右边出去的概率是多少，我们可以合并过来第三个人表示在这两个人右边的人所构成的整体，同样的我们也可以合并过来这两个人左边的人，问题就转换成四个人的问题。上述合并过程可以使用递归来实现。

现在我们需要设法求解一下四个人的问题：

（弃疗……） 区间染色代码实现：

```
# include <cstdlib>
# include <cstdio>
# include <cmath>
# include <cstring>

int n=0;

struct person_t
{
        float p_;
        person_t(float pp=0):p_(pp)
        {

        }
        float p (void)
        {
                return p_;
        };
        float q(void)
        {
                return 1-p_;
        };
};


person_t merge_r(person_t x,person_t y)
{
        return person_t(1-y.q()/(1-x.q()*y.p()));
};

person_t merge_l(person_t x,person_t y)
```

```
{
        return person_t(x.p()/(1-x.q()*y.p()));
};

person_t a[100005],b[100005] ,c[100005];

int main(void)
{
        scanf("%d",&n);
        for(int i=1;i<=n;++i) scanf("%lf",&a[i].p_);
        if(n==3)
        {
            //something 只看第一个人向左边传还是右边传
        }
        b[2]=a[2];
        for(int i=3;i<=n;++i) b[i]=merge_l(b[i-1],a[i]);
        c[n]=a[n];
        for(int i=n-1;i>=1;--i) c[i]=merge_r(a[i],c[i+1]);
        //ANSWER i=1;
        //ANSWER i=2;
        for(int i=3;i<=n-1;++i)
        {
                if(i+2<=n) p =merge_r(c[i+2],p);
                if(i-2>=2) p =merge_l(p,b[i-2]);
                float ans=0;
                ans+=p.p()*a[i+1].q(),p.q()/(1-p.q()*a[i+1].q());
                ans+=p.q()*a[i-1].p(),p.q()/(1-p.q()*a[i-1].p());
        }
        return 0;
}
```