1 Classical Problems

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 $b[i]$ 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);
    }
}
```

```c
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

$$B(1) \leq B(2) \leq \cdots \leq B(x) \geq B(x+1) \geq \cdots \geq B(N).$$

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

$N, Q \leq 10^5$.

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。$N \leq 5 \times 10^4$. 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：

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