template

1、ac自动机

```
#include <queue>
#include <cstdlib>
#include <cmath>
#include <cstdio>
#include <string>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;
const int maxn = 2 * 1e6 + 9;

int trie[maxn][26]; //字典树
int cntword[maxn];  //记录该单词出现次数
int fail[maxn];     //失败时的回溯指针
int cnt = 0;

void insertWords(string s)
{
    int root = 0;
    for (int i = 0; i < s.size(); i++)
    {
        int next = s[i] - 'a';
        if (!trie[root][next])
            trie[root][next] = ++cnt;
        root = trie[root][next];
    }
    cntword[root]++; //当前节点单词数+1
}
void getFail()
{
    queue<int> q;
    for (int i = 0; i < 26; i++)
    { //将第二层所有出现了的字母扔进队列
        if (trie[0][i])
        {
            fail[trie[0][i]] = 0;
            q.push(trie[0][i]);
        }
    }

    //fail[now]    ->当前节点now的失败指针指向的地方
    //tire[now][i] -> 下一个字母为i+'a'的节点的下标为tire[now][i]
    while (!q.empty())
    {
        int now = q.front();
        q.pop();

        for (int i = 0; i < 26; i++)
        { //查询26个字母
```

```
            if (trie[now][i])
            {
                //如果有这个子节点为字母i+'a',则
                //让这个节点的失败指针指向(((他父亲节点)的失败指针所指向的那个节点)的下一个节点)
                //有点绕,为了方便理解特意加了括号

                fail[trie[now][i]] = trie[fail[now]][i];
                q.push(trie[now][i]);
            }
            else //否则就让当前节点的这个子节点
                //指向当前节点fail指针的这个子节点
                trie[now][i] = trie[fail[now]][i];
        }
    }
}

int query(string s)
{
    int now = 0, ans = 0;
    for (int i = 0; i < s.size(); i++)
    {                                   //遍历文本串
        now = trie[now][s[i] - 'a']; //从s[i]点开始寻找
        for (int j = now; j && cntword[j] != -1; j = fail[j])
        {
            //一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).
            ans += cntword[j];
            cntword[j] = -1; //将遍历国后的节点标记,防止重复计算
        }
    }
    return ans;
}

int main()
{
    int n;
    string s;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        insertWords(s);
    }
    fail[0] = 0;
    getFail();
    cin >> s;
    cout << query(s) << endl;
    return 0;
}
```

2、n皇后问题

```
#include <cstdio>
#include <iostream>
using namespace std;
int ans[14], check[3][28] = {0}, sum = 0, n;
```

```
void eq(int cont)
{
    if (cont > n)
    {
        sum++;
        if (sum > 3)
            return;
        else
        {
            for (int i = 1; i <= n; i++)
                printf("%d ", ans[i]);
            printf("\n");
            return;
        }
    }
    for (int i = 1; i <= n; i++)
    {
        if ((!check[0][i]) && (!check[1][cont + i]) && (!check[2][cont - i +
n]))
        {
            ans[cont] = i;
            check[0][i] = 1;
            check[1][cont + i] = 1;
            check[2][cont - i + n] = 1;
            eq(cont + 1);
            check[0][i] = 0;
            check[1][cont + i] = 0;
            check[2][cont - i + n] = 0;
        }
    }
}
int main()
{
    scanf("%d", &n);
    eq(1);
    printf("%d", sum);
    return 0;
}

#include <iostream>
#include <cstdio>
int x[15], n;
int sols = 0;
using namespace std;
int place(int row)
{
    for (int j = 0; j < row; j++)
    {
        if (row - x[row] == j - x[j] || row + x[row] == j + x[j] || x[j] ==
x[row])
            return 0;
    }
    return 1;
}
void f(int row)
{
    if (n == row)
    {
```

```
        sols++;
        if (sols > 3)
            return;
        for (int k = 0; k < n; k++)
            printf("%d ", x[k] + 1);
        printf("\n");
    }
    else
    {
        for (int i = 0; i < n; i++)
        {
            x[row] = i;
            if (place(row))
                f(row + 1);
        }
    }
}
int main()
{
    scanf("%d", &n);
    f(0);
    printf("%d\n", sols);
    return 0;
}
```

3、O(nlogn)最长不下降子序列

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main()
{
    int n, x;
    vector<int> v, vec;
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        cin >> x;
        v.push_back(x);
    }
    //reverse(v.begin(),v.end()); 可以得到最长不上升或者下降子序列
    for (int i = 0; i < (int)v.size(); ++i)
    {
        if (i == 0)
            vec.push_back(v[0]);
        else
        {
            if (v[i] >= /*>  可以得到最长上升子序列*/ vec[vec.size() - 1])
                vec.push_back(v[i]);
            else
                *upper_bound(vec.begin(), vec.end(), v[i]) = v[i];
        }
    }
    for (int i = 0; i < (int)vec.size(); ++i)
    {
```

```
        cout << vec[i] << ' ';
    }
    cout << endl;
    cout << "len == " << vec.size() << endl;
}
```

4、queue

1. 定义：先进先出

```
queue<typename> name;
```

2. queue容器内元素的访问：

   由于队列(queue)本身就是一种先进先出的限制性数据结构，因此在STL中只能通过front()来访问队首元素，或是通过 back()来访问队尾元素。

```
#include<cstdio>
#include<queue>
using namespace std;
int main(){
    queue<int> q;
    for(int i=1;i<=5;i++) {
        q.push(i);      //push(i)用以将i压入队列，因此一次入队 1 2 3 4 5
    }
    printf("%d %d\n",q.front(),q.back()) ;  //输出对头元素 1  和  队尾元素 5
    return 0;
}
```

   运行结果：

```
1 5
```

3. queue常用函数实例解析：
   - push()：push(x) 可将 x 压入队列，时间复杂度为 O(1)。实例见“queue容器内元素的访问”。
   - front()、back()：front()和back()可分别获得队首元素和对微元素，时间复杂度为O(1)。实例见“queue容器内元素的访问”。
   - pop()：令队首元素出队，时间复杂度为O(1)。

```
#include<cstdio>
#include<queue>
using namespace std;
int main(){
    queue<int> q;
    for(int i=1;i<=5;i++) {
        q.push(i);      //push(i)用以将i压入队列，因此一次入队 1 2 3 4 5
    }
    for(int i=1;i<=3;i++) {
        q.pop();    //队首元素出队列（1 2 3）出队列
    }
    printf("%d\n",q.front()) ;  //输出对头元素 4
    return 0;
}
```

运行结果：

```
4
```

- empty()：检测是否为空，返回true则为空，返回false则非空，时间复杂度O(1)。

```
#include<cstdio>
#include<queue>
using namespace std;
int main(){
    queue<int> q;
    printf("%d\n",q.empty());//先开始没有元素，为空(true)；输出 1
    for(int i=1;i<=5;i++) {
        q.push(i);      //push(i)用以将i压入队列，因此一次入队 1 2 3 4 5
    }
    printf("%d\n",q.empty());//压入元素，为非看(false)；输出 0
    return 0;
}
```

运行结果：

```
1
0
```

- size()：返回queue内元素的个数，时间复杂度为O(1)。

```
#include<cstdio>
#include<queue>
using namespace std;
int main(){
    queue<int> q;
    for(int i=1;i<=5;i++) {
        q.push(i);      //push(i)用以将i压入队列，因此一次入队 1 2 3 4 5
    }
    printf("%d\n",q.size());// 输出队列元素个数
    return 0;
}
```

运行结果：

```
5
```

5、sscanf与sprintf

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    int n;
    scanf("%d", &n);
    getchar();
    char flag;
```

```
    while (n--)
    {
        char line[200];
        gets(line);
        int a, b; // operands
        if (line[0] >= 'a' && line[0] <= 'c')
        {
            sscanf(line, "%c %d %d", &flag, &a, &b);
        }
        else
        {
            sscanf(line, "%d %d", &a, &b);
        }

        char ans[200];
        switch (flag)
        {
        case 'a':
            sprintf(ans, "%d+%d=%d", a, b, a + b);
            break;
        case 'b':
            sprintf(ans, "%d-%d=%d", a, b, a - b);
            break;
        case 'c':
            sprintf(ans, "%d*%d=%d", a, b, a * b);
            break;
        }
        puts(ans);
        printf("%zd\n", strlen(ans));
    }
    return 0;
}
```

6、string

```
void *memccpy (void *dest, const void *src, int c, size_t n);
```
从src所指向的对象复制n个字符到dest所指向的对象中。如果复制过程中遇到了字符c则停止复制，返回指针指向dest中字符c的下一个位置；否则返回NULL。

```
void *memcpy (void *dest, const void *src, size_t n);
```
从src所指向的对象复制n个字符到dest所指向的对象中。返回指针为dest的值。

```
void *memchr (const void *s, int c, size_t n);
```
在s所指向的对象的前n个字符中搜索字符c。如果搜索到，返回指针指向字符c第一次出现的位置；否则返回NULL。

```
int memcmp (const void *s1, const void *s2, size_t n);
```
比较s1所指向的对象和s2所指向的对象的前n个字符。返回值是s1与s2第一个不同的字符差值。

```
int memicmp (const void *s1, const void *s2, size_t n);
```
比较s1所指向的对象和s2所指向的对象的前n个字符，忽略大小写。返回值是s1与s2第一个不同的字符差值。

```
void *memmove (void *dest, const void *src, size_t n);
```
从src所指向的对象复制n个字符到dest所指向的对象中。返回指针为dest的值。不会发生内存重叠。

```
void *memset (void *s, int c, size_t n);
```
设置s所指向的对象的前n个字符为字符c。返回指针为s的值。

```
char *stpcpy (char *dest, const char *src);
```
复制字符串src到dest中。返回指针为dest + len(src)的值。

```
char *strcpy (char *dest, const char *src);
```
复制字符串src到dest中。返回指针为dest的值。

```
char *strcat (char *dest, const char *src);
```
将字符串src添加到dest尾部。返回指针为dest的值。

```
char *strchr (const char *s, int c);
```
在字符串s中搜索字符c。如果搜索到，返回指针指向字符c第一次出现的位置；否则返回NULL。

```
int strcmp (const char *s1, const char *s2);
```
比较字符串s1和字符串s2。返回值是s1与s2第一个不同的字符差值。

```
int stricmp (const char *s1, const char *s2);
```
比较字符串s1和字符串s2，忽略大小写。返回值是s1与s2第一个不同的字符差值。

```
size_t strcspn (const char *s1, const char *s2);
```
返回值是字符串s1的完全由不包含在字符串s2中的字符组成的初始串长度。

```
size_t strspn (const char *s1, const char *s2);
```
返回值是字符串s1的完全由包含在字符串s2中的字符组成的初始串长度。

```
char *strdup (const char *s);
```
得到一个字符串s的复制。返回指针指向复制后的字符串的首地址。

```
char *strerror(int errnum);
```
返回指针指向由errnum所关联的出错消息字符串的首地址。errnum的宏定义见errno.h。

```
size_t strlen (const char *s);
```
返回值是字符串s的长度。不包括结束符'/0'。

```
char *strlwr (char *s);
```
将字符串s全部转换成小写。返回指针为s的值。

```
char *strupr (char *s);
```
将字符串s全部转换成大写。返回指针为s的值。

```
char *strncat (char *dest, const char *src, size_t maxlen);
```
将字符串src添加到dest尾部，最多添加maxlen个字符。返回指针为dest的值。

```
int strncmp (const char *s1, const char *s2, size_t maxlen);
```
比较字符串s1和字符串s2，最多比较maxlen个字符。返回值是s1与s2第一个不同的字符差值。

```
char *strncpy (char *dest, const char *src, size_t maxlen);
```
复制字符串src到dest中，最多复制maxlen个字符。返回指针为dest的值。

```
int strnicmp(const char *s1, const char *s2, size_t maxlen);
```
比较字符串s1和字符串s2，忽略大小写，最多比较maxlen个字符。返回值是s1与s2第一个不同的字符差值。

```
char *strnset (char *s, int ch, size_t n);
```
设置字符串s中的前n个字符全为字符c。返回指针为s的值。

```
char *strset (char *s, int ch);
```
设置字符串s中的字符全为字符c。返回指针为s的值。

```
char *strpbrk (const char *s1, const char *s2);
返回指针指向字符串s1中字符串s2的任意字符第一次出现的位置；如果未出现返回NULL。

char *strrchr (const char *s, int c);
在字符串s中搜索字符c。如果搜索到，返回指针指向字符c最后一次出现的位置；否则返回NULL。

char *strrev (char *s);
将字符串全部翻转，返回指针指向翻转后的字符串。

char *strstr (const char *s1, const char *s2);
在字符串s1中搜索字符串s2。如果搜索到，返回指针指向字符串s2第一次出现的位置；否则返回NULL。

char *strtok (char *s1, const char *s2);
用字符串s2中的字符做分隔符将字符串s1分割。返回指针指向分割后的字符串。第一次调用后需用NULLL替代s1作为第一个参数。
```

7、unique 去重

```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
int arr[10005];
int main()
{
    int n, k;
    cin >> n >> k;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    sort(arr, arr + n);
    int len = unique(arr, arr + n) - arr;
    if (len <= n)
        cout << arr[k - 1];
    else
        cout << "NO RESULT";
    return 0;
}
```

8、vector

```
随机访问频繁
1.1 vector 说明
vector是向量类型，可以容纳许多类型的数据，因此也被称为容器
(可以理解为动态数组，是封装好了的类）
进行vector操作前应添加头文件#include <vector>
1.2 vector初始化：
方式1.

//定义具有10个整型元素的向量（尖括号为元素类型名，它可以是任何合法的数据类型），不具有初值，其值不确定
vector<int>a(10);
1
2
方式2.
```

```
//定义具有10个整型元素的向量，且给出的每个元素初值为1
vector<int>a(10,1);
1
2
方式3.

//用向量b给向量a赋值，a的值完全等价于b的值
vector<int>a(b);
1
2
方式4.

//将向量b中从0-2（共三个）的元素赋值给a，a的类型为int型
vector<int>a(b.begin(),b.begin+3);
1
2
方式5.

//从数组中获得初值
int b[7]={1,2,3,4,5,6,7};
vector<int> a(b,b+7）;
1
2
3
1.3 vector对象的常用内置函数使用（举例说明）
#include <vector>
vector<int> a,b;
//b为向量，将b的0-2个元素赋值给向量a
a.assign(b.begin(),b.begin()+3);
//a含有4个值为2的元素
a.assign(4,2);
//返回a的最后一个元素
a.back();
//返回a的第一个元素
a.front();
//返回a的第i元素,当且仅当a存在
a[i];
//清空a中的元素
a.clear();
//判断a是否为空，空则返回true，非空则返回false
a.empty();
//删除a向量的最后一个元素
a.pop_back();
//删除a中第一个（从第0个算起）到第二个元素，也就是说删除的元素从a.begin()+1算起（包括它）一直到a.begin()+3（不包括它）结束
a.erase(a.begin()+1,a.begin()+3);
//在a的最后一个向量后插入一个元素，其值为5
a.push_back(5);
//在a的第一个元素（从第0个算起）位置插入数值5,
a.insert(a.begin()+1,5);
//在a的第一个元素（从第0个算起）位置插入3个数，其值都为5
a.insert(a.begin()+1,3,5);
//b为数组，在a的第一个元素（从第0个元素算起）的位置插入b的第三个元素到第5个元素（不包括b+6）
a.insert(a.begin()+1,b+3,b+6);
//返回a中元素的个数
a.size();
//返回a在内存中总共可以容纳的元素个数
```

```
a.capacity();
//将a的现有元素个数调整至10个，多则删，少则补，其值随机
a.resize(10);
//将a的现有元素个数调整至10个，多则删，少则补，其值为2
a.resize(10,2);
//将a的容量扩充至100，
a.reserve(100);
//b为向量，将a中的元素和b中的元素整体交换
a.swap(b);
//b为向量，向量的比较操作还有 != >= > <= <
a==b;

*下标只能用来获取已经存在的元素，不能进行赋值

#include <algorithm>
//对a中的从a.begin()（包括它）到a.end()（不包括它）的元素进行从小到大排列
sort(a.begin(),a.end());
//对a中的从a.begin()（包括它）到a.end()（不包括它）的元素倒置，但不排列，如a中元素为
1,3,2,4,倒置后为4,2,3,1
reverse(a.begin(),a.end());
//把a中的从a.begin()（包括它）到a.end()（不包括它）的元素复制到b中，从b.begin()+1的位置（包
括它）开始复制，覆盖掉原有元素
copy(a.begin(),a.end(),b.begin()+1);
//在a中的从a.begin()（包括它）到a.end()（不包括它）的元素中查找10，若存在返回其在向量中的位置
find(a.begin(),a.end(),10);
```

9、并查集（出了错就找wsh版）

```
#include <iostream>
#include <cstdio>
using namespace std;
int set[100]; //记录父节点
int rank[100];
int per[100]; //记录深度
void init()   //初始化节点
{
    for (int i = 0; i <= 100; ++i)
    {
        set[i] = i;
        rank[i] = 0;
    }
}
int find(int x)
{
    while (set[x] != x)
        x = set[x];
    return x;
}
void merge(int a, int b)
{
    int fa = find(a);
    int fb = find(b);
    if (fb != fa)
    {
        if (rank[fa] < rank[fb])
        {
            set[fa] = fb;
```

```
        }
        else
        {
            set[fb] = fa;
            if (rank[fa] == rank[fb])
                rank[fa]++;
        }
    }
}
int main()
{
    init();
    int m, n;
    cin >> m >> n;
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        merge(a, b);
    }
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        if (find(a) == find(b))
        {
            cout << "yes" << endl;
        }
        else
        {
            cout << "no" << endl;
        }
    }
    return 0;
}
```

10、单hash

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;
typedef unsigned long long ull;
const int maxn = 10000 + 5, maxm = 1e5 + 5;
int n, m, ans = 1;
int base = 131, a[maxn], mol = (1 << 16) - 1;
char s[maxn];
inline int read()
{
    int s = 0, w = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            w = -1;
```

```
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
        s = s * 10 + ch - '0', ch = getchar();
    return s * w;
}
int hashh(char s[])
{
    int anss = 0, len = strlen(s);
    for (int i = 0; i < len; i++)
    {
        anss = (anss * base + s[i]) % mol;
    }
    return anss;
}
int main()
{
    n = read();
    for (int i = 1; i <= n; i++)
    {
        cin >> s;
        a[i] = hashh(s);
    }
    sort(a + 1, a + n + 1);
    for (int i = 2; i <= n; i++)
    {
        if (a[i - 1] != a[i])
            ans++;
    }
    cout << ans;
}
```

11、双hash

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;
typedef unsigned long long ull;
const int maxn = 10000 + 5, maxm = 1e5 + 5;
int n, m, ans = 1;
ull base1 = 131, base2 = 789, mol1 = 19260817, mol2 = 19660813;
char s[maxn];
struct Node
{
    ull x, y;
} a[maxn];
inline int read()
{
    int s = 0, w = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            w = -1;
```

```
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
        s = s * 10 + ch - '0', ch = getchar();
    return s * w;
}
ull hash1(char s[])
{
    ull anss = 0;
    int len = strlen(s);
    for (int i = 0; i < len; i++)
    {
        anss = (anss * base1 + (ull)s[i]) % mol1;
    }
    return anss;
}
ull hash2(char s[])
{
    ull anss = 0;
    int len = strlen(s);
    for (int i = 0; i < len; i++)
    {
        anss = (anss * base1 + (ull)s[i]) % mol2;
    }
    return anss;
}
bool cmp(Node A, Node B) { return A.x < B.x; }
int main()
{
    n = read();
    for (int i = 1; i <= n; i++)
    {
        cin >> s;
        a[i].x = hash1(s);
        a[i].y = hash2(s);
    }
    sort(a + 1, a + n + 1, cmp);
    for (int i = 2; i <= n; i++)
    {
        if (a[i - 1].x != a[i].x || a[i - 1].y != a[i].y)
            ans++;
    }
    cout << ans;
}
```

12、递归

```
#include <iostream>
typedef long long ll;
using namespace std;

int sum(int n)
{
    // 1 + 2 + ... + n
    int ans = 0;
    for (int k = 1; k <= n; ++k)
```

```
    {
        ans += k;
    }
    return ans;
}

// n! = 1·2·3·....·n

// recursive  递归
// n! = (n - 1)! · n, 0! = 1, n 是非负整数
// 1. 函数自己调用自己的过程就叫递归
// 3! = 2! * 3 = 1! * 2 * 3 = 0! * 1 * 2 * 3 = 1 * 1 * 2 * 3 = 6
// 2. 在定义域内函数的调用必需要能终止

// sigma(n) = sigma(n-1) + n
// sigma(1) = 1

int sigma(int n)
{
    if (n == 1)
        return 1;
    return (sigma(n - 1)) + n; // 左递归展开
}

// n = 3
// sigma(3)
// (sigma(2)) + 3
// ((sigma(1)) + 2) + 3
// ((1) + 2) + 3

int sigma_tail(int n)
{
    if (n == 1)
        return 1;
    return n + (sigma_tail(n - 1)); // 右递归展开 / 尾递归 -> 尾递归优化 -> 循环
}

// n = 3
// sigma(3)
// 3 + (sigma(2))
// 3 + (2 + (sigma(1)))
// 3 + (2 + (1))

void print_front_rec(int n)
{
    if (n == 0)
        return;
    cout << n << endl;
    print_front_rec(n - 1); // 右展开
}

void print_after_rec(int n)
{
    if (n == 0)
        return;
    print_after_rec(n - 1); // 左展开
    cout << n << endl;
}
```

```
// 0 1 1 2 3 5 8 13 21 ..

// fib(n) = fib(n - 1) + fib(n - 2)
// specified: fib(0) = 0, fib(1) = 1

// 时间复杂度 O(2^N) 指数爆炸
// 刚才这里说错了，空间复杂就是递归树深度O(N)
ll fib(ll n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fib(n - 1) + fib(n - 2); // complexity
}

// 死递归炸栈
// segmentation fault
void foo()
{
    foo();
}

int main()
{
    // DP hd 2602 01背包

    // cout << fib(50) << endl;
    // print_front_rec(4);
    cout << endl
         << endl
         << endl;
    // print_after_rec(4);
    // foo(); // runtime error 运行时错误  RE  runtime exception 运行时异常
    return 0;
}
```

13、覆盖接海报

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <queue>
#include <cstring>
using namespace std;
const int maxn = 1e3 + 10;
vector<int> v[maxn];
typedef struct node
{
    int x1, y1, x2, y2;
};
node x[maxn];
int vis[maxn], jiao[4];
int judge(node a, node b, int i) //判断两者是否覆盖，覆盖掉了哪个角
{
    bool flag = 1;
```

```
    if (a.x2 > b.x1 && a.x2 <= b.x2 && a.y2 > b.y1 && a.y2 <= b.y2)
        jiao[0] = i;
    if (a.x1 >= b.x1 && a.x1 < b.x2 && a.y2 > b.y1 && a.y2 <= b.y2)
        jiao[1] = i;
    if (a.x1 >= b.x1 && a.x1 < b.x2 && a.y1 >= b.y1 && a.y1 < b.y2)
        jiao[2] = i;
    if (a.x2 > b.x1 && a.x2 <= b.x2 && a.y1 >= b.y1 && a.y1 < b.y2)
        jiao[3] = i;
    if (a.x2 <= b.x1 || a.y1 >= b.y2 || a.x1 >= b.x2 || a.y2 <= b.y1)
        flag = 0;
    return flag;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, w, h;
    while (cin >> w >> h >> n)
    {
        memset(jiao, 0, sizeof(jiao));
        int ans = 0, index = 0;
        for (int i = 1; i <= n; i++)
            cin >> x[i].x1 >> x[i].y1 >> x[i].x2 >> x[i].y2;
        for (int i = n; i >= 1; i--)
        {
            memset(vis, 0, sizeof(vis));
            for (int j = n; j > i; j--)
            {
                if (judge(x[i], x[j], i))
                    v[i].push_back(j); /*如果x[j]覆盖了x[i]，把j放进i*/
            }
            int f = 0;
            for (int j = 0; j < 4; j++)
            { /*计数有几个角被覆盖*/
                if (jiao[j] == i)
                    f++;
            }
            if (f == 4) /*四个角全部被覆盖，没办法揭掉*/
                continue;
            queue<int> q;
            q.push(i);
            vis[i] = 1;
            int cnt = 1;        /*当前海报算一张，初始值为1*/
            while (!q.empty()) /*计算有几张海报盖着当前的海报*/
            {
                int t = q.front();
                q.pop();
                for (int j = 0; j < v[t].size(); j++)
                {
                    if (!vis[v[t][j]])
                    {
                        q.push(v[t][j]);
                        vis[v[t][j]] = 1;
                        cnt++;
                    }
                }
            }
```

```
            if (cnt >= ans)
            {
                ans = cnt;
                index = i;
            }
        }
        for (int i = 1; i <= n; i++)
            v[i].clear();
        cout << ans << " " << index << endl;
    }
    return 0;
}
```

14、集合子集

```
#include <iostream>
const int Maxn = 32;
bool flag[Maxn];
using namespace std;
void PowerSet(int i, int n)
{
    if (i > n)
    { //输出幂集的一个元素：
        for (int j = 1; j <= n; j++)
            if (flag[j] == true)
                cout << j << ' ';
        cout << endl;
        return;
    }
    else
    {
        flag[i] = true; //取第i个元素；
        PowerSet(i + 1, n);
        flag[i] = false; //不取第i个元素；
        PowerSet(i + 1, n);
    }
}
int main()
{
    int i;
    for (i = 0; i <= Maxn; i++)
        flag[i] = false;
    PowerSet(1, 5);
}
```

15、结构体排序

```
#include<bits/stdc++.h>
using namespace std;

// 排序要求：按分数从高到低输出上线考生的考号与分数，其间用1空格分隔。若有多名考生分数相同，则按他们考号的升序输出。
struct Student {
    int  id;   //考生准考证号
    int  num;  //存题号
    int  sum;  // 考生的最后得分
```

```
};

int cmp(Student a, Student b) {
    if(a.sum == b.sum) {
        return a.id < b.id;
    }
    return a.sum >b.sum;
}

int main() {
    struct Student student[10];
    for(int i = 0; i < 10; i++) {
        student[i].id = i;
        student[i].num = i;
        student[i].sum = i;
    }
    sort(student, student + 10, cmp);
    return 0;
}
```

16、快速求质数

```
#include <iostream>
#include <cstdio>
#include <cmath>
using namespace std;

int main()
{
    // 埃氏筛

    // 0 表示质数，1 非质数（1 非质非合）
    int arr[101] = {0}; // 0 去掉，1~100
    arr[1] = 1;          // 1 既不是质数也不是合数

    for (int i = 2; i <= 100; ++i)
    {
        if (arr[i] == 1)
            continue;
        for (int i_times = i * 2; i_times <= 100; i_times += i)
        {
            arr[i_times] = 1; // i 的 n 倍必然是合数
        }
    }

    int cnt = 0;
    for (int i = 1; i <= 100; ++i)
    {
        cnt += (arr[i] + 1) % 2;
    }
    cout << cnt << endl;
    return 0;
}
```

17、全排列问题

```
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
int n;
char a[15];
char re[15];
int vis[15];

void dfs(int step)
{
    int i;
    if (step == n + 1)
    {

        for (i = 1; i <= n; i++)
            printf("    %c", re[i]);
        printf("\n");
        return;
    }
    for (i = 1; i <= n; i++)
    {
        if (vis[i] == 0)
        {
            re[step] = a[i];
            vis[i] = 1;
            dfs(step + 1);
            vis[i] = 0;
        }
    }
    return;
}

int main()
{
    int T;
    scanf("%d", &T);
    for (int i = 1; i <= T; i++)
    {
        a[i] = i + '0';
    }
    n = strlen(a + 1);
    dfs(1);
    return 0;
}
```

18、随机取数和C语言与文件

```
#include <stdio.h>
// 文件指针
// stdin - standard in 标准输入流
// stdout - standard out 标准输出流
// stderr - standard error 标准错误流
```

```c
int main(void)
{
    // 文件指针
    FILE *fp;                   // file pointer
    fp = fopen("qw.txt", "r"); // file open
    char str[100];
    // 从文件流 fp 读取 99 个字符到 str，并且在 str[99]='\0'
    // fgets 的中止条件
    // 1. 超过了 count - 1 个字符
    // 2. 不够了
    // 3. 换行符
    fgets(str, 100, fp); // file get string
    fclose(fp);

    fp = fopen("qw.txt", "a");
    // 输入重定向
    fprintf(fp, "%s", str);
    fclose(fp);

    return 0;
}

#include <stdio.h>
#include <stdlib.h> // srand, rand
#include <time.h>   // time

int main(void)
{
    // psdueo-random number  伪随机数
    int arr[10];
    for (int i = 0; i < 10; i++)
        arr[i] = i;

    srand(time(NULL)); // seed
    srand(rand());

    int idx = rand() % 10; // index 目录 索引
    printf("%d\n", arr[idx]);
}
```

19、随机取数排序数组

```c
#include <iostream>  // I/O stream I/O 流 Input/Output
#include <stdio.h>   // C 语言输入输出
#include <stdlib.h>  // srand
#include <time.h>    // time
#include <algorithm> // 提供 std::random_shuffle, std::sort, std::reverse
using namespace std; // 为 C++ 标准库函数省去 std:: 修饰

int rng(int n)
{
    srand(rand());
    return rand() % n;
}

int main()
```

```
{
    srand(time(NULL)); // 为 random_shuffle 内部使用的 rand 函数提供伪随机数种子

    int arr[] = {1, 2, 3, 4, 5};

    // random 随机 shuffle 洗牌 ，范围 前闭后开区间 [arr, arr+5)
    random_shuffle(arr, arr + 5, rng);

    // 遍历输出
    for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    putchar('\n');

    // 从小到大排序
    sort(arr, arr + 5);
    // 逆序 （从大到小）
    reverse(arr, arr + 5);

    // 遍历输出
    for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    putchar('\n');

    return 0;
}
```

20、拓扑排序-最大食物链计数

```
#include <bits/stdc++.h>
using namespace std;
#define mod 80112002
最大食物链计数（拓扑排序）：
int s[5010][5010]; // 邻接矩阵存储关系
int x[5010];       // 出度
int y[5010];       // 入度
int r[5010];       // 存储路径数
int main()
{
    int n, m;
    cin >> n >> m;
    int a, b;
    for (int i = 0; i < m; i++)
    {
        cin >> a >> b;
        s[a][b] = 1; // a->b
        x[a]++;      // 出度+1
        y[b]++;      // 入度+1
    }
    queue<int> q;
    for (int i = 1; i <= n; i++)
    {
        if (y[i] == 0)
        {              // 入度为0的进入队列
            r[i] = 1; // 到达该点路径数为1
            q.push(i);
        }
    }
```

```
    int len = 0; // 路径总数
    while (!q.empty())
    {
        int p = q.front(); // 取队首元素
        q.pop();
        for (int i = 1; i <= n; i++)
        { // 搜索与该点相关联的点
            if (s[p][i] == 0)
                continue;
            r[i] = (r[i] + r[p]) % mod; // 路径数更新
            y[i]--;                    // 入度--
            if (y[i] == 0)
            { // 入度为0
                if (x[i] == 0)
                {                              // 出度为0
                    len = (len + r[i]) % mod; // 更新路径总数
                    continue;
                }
                q.push(i); // 入度为0出度不为0进入队列
            }
        }
    }
    cout << len << endl;
    return 0;
}
```

21、质数环

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
int a[20], vis[20], n;
int prime[40] = {0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 0, 0, 0,
                 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0};
void dfs(int step)
{
    int i;
    if (step == n + 1 && prime[a[n] + a[1]])
    {
        for (i = 1; i < n; i++)
            printf("%d ", a[i]);
        cout<<a[n]<<endl;
        return;
    }
    for (i = 2; i <= n; i++)
    {
        if (!vis[i] && prime[i + a[step - 1]])
        //如果这个数没用过，并且这个数和上一个放到环里的数之和是素数
        {
            a[step] = i;
            vis[i] = 1;
            dfs(step + 1);
            vis[i] = 0; //回溯
        }
    }
```

```
}

int main()
{
    int k = 1;
    a[1] = 1;

    while (cin>>n)
    {
        if(k>1){
            cout<<endl;
        }
        cout<<"Case "<<k++<<":"<<endl;
        memset(vis, 0, sizeof(vis));
        dfs(2);
    }
    return 0;
}
```

22、自然溢出hash

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;
typedef unsigned long long ull;
const int maxn = 10000 + 5, maxm = 1e9 + 5;
int n, m, ans = 1;
ull base = 131, a[maxn];
char s[maxn];
inline int read()
{
    int s = 0, w = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            w = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
        s = s * 10 + ch - '0', ch = getchar();
    return s * w;
}
ull hashh(char s[])
{
    int len = strlen(s);
    ull anss = 0;
    for (int i = 0; i < len; i++)
    {
        anss = anss * base + (ull)s[i];
    }
    return anss;
}
int main()
```

```
{
    n = read();
    for (int i = 1; i <= n; i++)
    {
        cin >> s;
        a[i] = hashh(s);
    }
    sort(a + 1, a + n + 1);
    for (int i = 2; i <= n; i++)
    {
        if (a[i] != a[i - 1])
            ans++;
    }
    cout << ans;
}
```

23、最大公因数

```
int gcd(ll a, ll b){
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

24、最小生成树

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
int n, m;
struct node
{
    int s, e, w;
} edge[200000];
int f[200000];
int cmp(node a, node b)
{
    return a.w < b.w;
}
int find(int x)
{
    if (f[x] == -1)
        return x;
    return f[x] = find(f[x]);
}
int main()
{
    int n, m;
    int s, e, w;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++)
    {
        scanf("%d%d%d", &s, &e, &w);
        edge[i].s = s;
        edge[i].e = e;
```

```
        edge[i].w = w;
    }
    memset(f, -1, sizeof(f));
    sort(edge + 1, edge + m + 1, cmp);
    int sum = 0;
    int t1, t2;
    for (int i = 1; i <= m; i++)
    {
        t1 = find(edge[i].s);
        t2 = find(edge[i].e);
        if (t1 != t2)
        {
            f[t1] = t2;
            sum += edge[i].w;
        }
    }
    printf("%d", sum);
    return 0;
}
```

25、岛屿问题

1. 四个方向

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 1005

int n;
char mapp[MAXN][MAXN];
int vis[MAXN][MAXN];
int wsad[4][2] = {0, 1, 0, -1, 1, 0, -1, 0};

struct node {
    int x, y;
};

void bfs(node x) {
    queue<node> q;
    q.push(x);
    node t, nx;
    vis[x.x][x.y] = 1;
    mapp[x.x][x.y] = 'o';
    while (!q.empty()) {
        t = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            nx.x = t.x + wsad[i][0];
            nx.y = t.y + wsad[i][1];

            if ((nx.x >= 0 && nx.x < n) && (nx.y >= 0 && nx.y < n) && \
                vis[nx.x][nx.y] == 0 && mapp[nx.x][nx.y] != 'o') {
                vis[nx.x][nx.y] = 1;
                mapp[nx.x][nx.y] = 'o';
                q.push(nx);
            }
        }
```

```
    }
}

void Print() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << mapp[i][j] << ' ';
        }
        cout << endl;
    }
    cout << endl;
}

int main() {
    while(cin >> n) {
        int ans = 0;
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cin >> mapp[i][j];
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (mapp[i][j] == 'x') {
                    node x;
                    x.x = i; x.y = j;
                    bfs(x);
                    // Print();
                    ans++;
                }
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

2. 八个方向

```
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
char daoyu[505][505];
int vis[505][505];
int Max; //记录面积
void f(int x, int y)
{
    if (daoyu[x][y] == '0' || vis[x][y] == 1) //寻找陆地
        return;
    Max++;
    vis[x][y] = 1; //找过的变为陆地
    /*八个方向寻找*/
    f(x + 1, y);
    f(x + 1, y + 1);
    f(x, y + 1);
```

```
    f(x - 1, y + 1);
    f(x - 1, y);
    f(x - 1, y - 1);
    f(x, y - 1);
    f(x + 1, y - 1);
}
int main()
{
    int m, n, k;
    while (scanf("%d %d %d", &m, &n, &k) != EOF)
    {
        int ans = 0, max_ = 0, i, j;
        memset(daoyu, '0', sizeof(daoyu));
        memset(vis, 0, sizeof(vis));
        while (k--)
        {
            int a, b;
            cin >> a >> b;
            daoyu[a][b] = '1';
        }
        //      for(int i=1;i<=m;i++){
        //          for(int j=1;j<=n;j++){
        //              cout<<daoyu[i][j];
        //          }
        //          cout<<endl;
        //      }
        for (i = 0; i < m; i++)
        {
            for (j = 0; j < n; j++)
            {
                if (daoyu[i][j] == '1' && !vis[i][j]) //遇到未曾访问过的小岛开始
查找
                {
                    Max = 0;
                    ans++;
                    f(i, j);
                    if (max_ < Max)
                        max_ = Max;
                }
            }
        }
        //        printf("%d %d\n",ans,max_);
        cout << max_ << endl;
    }
    return 0;
}
```