int:-2147483648 ~ 2147483647(-2^31~2^31-1)

long long:-9223372036854775808~9223372036854775807(-2^63~2^63-1)

unsigned long long:0~18446744073709551615 //20位

快速幂与质因数分解

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
int q;
ll n, k, cnt, ans;
ll fpow(ll a, ll b){//快速幂精简版 a^11 = a^8 * a^2 * a^1
    ll ans = 1;
    while(b){
        if(b&1)ans *= a;
        a *= a;
        b >>= 1;
    }
    return ans;
}
int main() {
    cin >> q;
    while(q--) {
        vector<pair<ll, ll> > vt;
        cin >> n >> k;
        //进行质因数分解
        for(int i = 2; i * i <= n; i++) {
            cnt = 0;
            if(n % i == 0) {
                while(n % i == 0) {
                    n /= i;
                    cnt++;
                }
                vt.push_back(make_pair(i, cnt)); //i是质因数，cnt是幂次
            }
        }
        if(n > 1) vt.push_back(make_pair(n, 1));

        ans = 1;
        for(int i = 0; i < vt.size(); i++) {
            if(vt[i].second >= k) {
                ans *= fpow(vt[i].first, vt[i].second);
            }
        }

        cout << ans << endl;
    }
    return 0;
}
```

素数筛

```
bool visit[N];                                                  //visit[i] = true
表示i被筛掉，不是素数
int prime[N];
int optimal_E_sieve(int n){
    for(int i = 2; i <= sqrt(n); i++)                           //筛选掉非素数
        if(!visit[i])
            for(int j = i * i; j <= n; j += i)visit[j] = true;  //标记非素数
    //下面来记录素数
    int k = 0;
    for(int i = 2; i <= n; i++)
        if(!visit[i])prime[++k] = i;                            //记录素数
prime[1]=2, prime[2]=3, ......
    return k;
}
```

gcd和lcm

```
#include<iostream>
#include <algorithm>
using namespace std;
typedef long long ll;
ll gcd(ll a, ll b){return b ? gcd(b, a % b) : a;}//最小公倍数
ll lcm(ll a, ll b){return a / gcd(a, b) * b;}//最大公约数
int main(){
    ll a, b;
    cin >> a >> b;
    cout << __gcd(a,b) << ' ' << gcd(a, b) << ' ' << lcm(a, b);
    return 0;
}
```

dfs

```
ans;                                    //答案，常常用全局变量表示
void dfs(层数,其他参数)
{
    if (到达目的地,或者出局)             //到达最底层，或者满足条件退出
    {
        更新答案ans;                    //答案一般用全局变量表示，ans是最优解
        return;                         //递归返回，即返回到上一层
    }
    (剪枝)                              //在进一步DFS之前剪枝
    for (用i遍历下一层所有可能的情况)    //对每一个情况继续DFS
        if (used[i] == 0)               //如果状态i没有处理过，就可以进入下一层dfs
        {
            used[i] = 1;                //标记状态i为已经使用，在后续dfs时不能再使用
            dfs(层数+1,其他参数);       //下一层，即搜小规模后继续dfs
            used[i] = 0;                //恢复状态i，回溯时，不影响上一层对这个状态的使
用
        }
    return;                             //返回到上一层
```

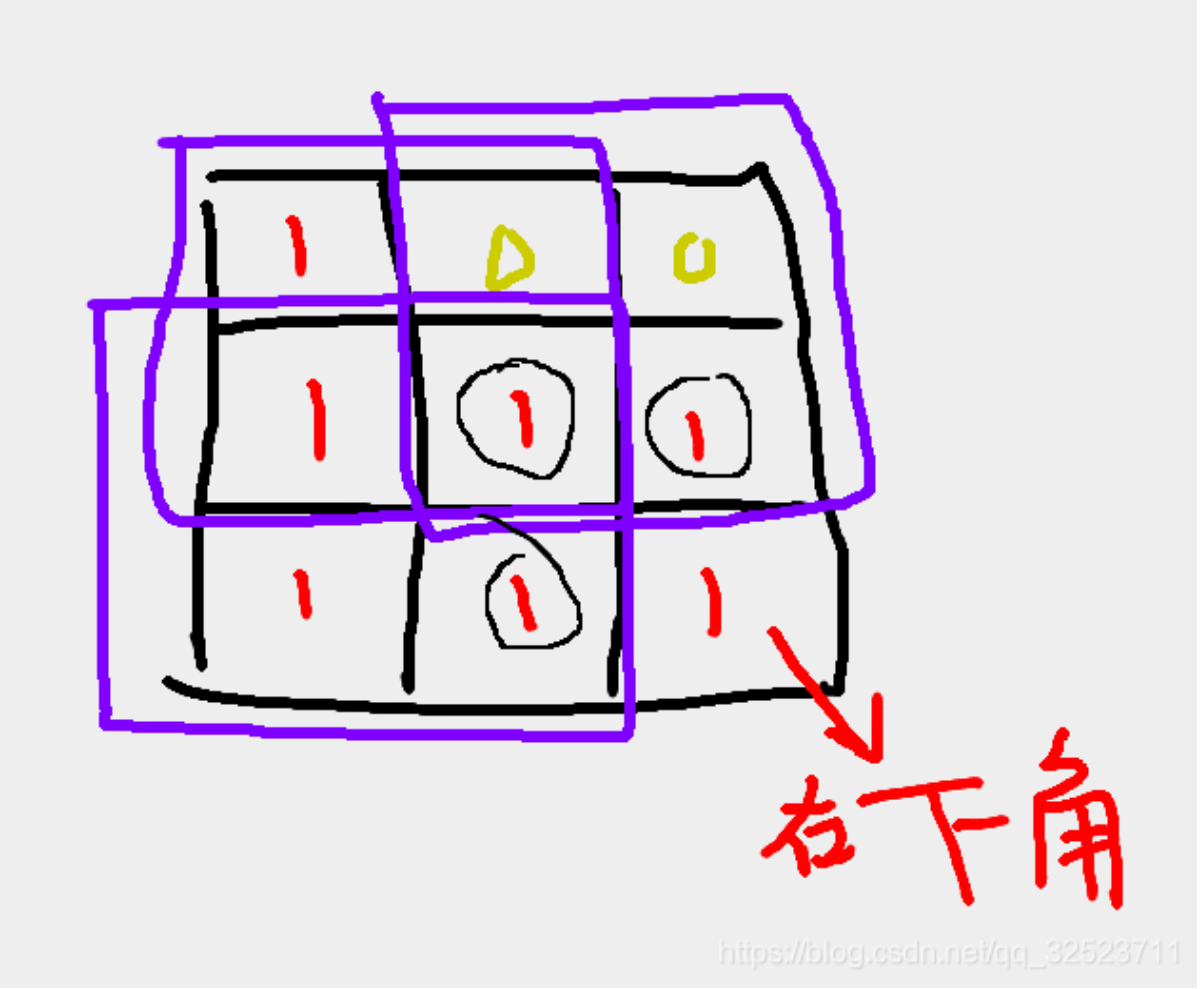
```
}
```

dfs典例：选数

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;
typedef long long ll;
int n, k;
ll ans = 0;
ll x[25];
bool isprime(ll y){
    if(y < 2)return false;
    for(ll i = 2; i * i <= y; i++)if(y % i == 0)return false;
    return true;
}
void dfs(int depth, int sum, ll res){
    if(sum == k){
        cout << res << endl;
        if(isprime(res)) ans++;
        return;
    }
    //或者：
    //if(depth > n)return;
    // dfs(depth + 1, sum + 1, res + x[depth]);
    // dfs(depth + 1, sum, res);
    for(int i = depth; i <= n; i++)
       dfs(i + 1, sum + 1, res + x[i]);
    return;
}
int main(){
    cin >> n >> k;
    for(int i = 1; i <= n; i++)cin >> x[i];
    dfs(1, 0, 0);
    cout << ans;
}
```

最大正方形

matrix[i][j] 为 '0' 或 '1'，找到最大的正方形，并返回其面积




```
//dp[i][j]表示以第i行第j列为右下角所能构成的最大正方形边长，则递推式为:
//dp[i][j] = 1 + min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]);
class Solution {
public:
    int maximalSquare(vector<vector<char>>& matrix) {
        if(matrix.size() == 0 || matrix[0].size() == 0) return 0;
        int row = matrix.size(), col = matrix[0].size();
        vector<vector<int>> dp(row+1, vector<int>(col+1, 0));//dp[i][j]表示以i，j为
右下角的正方形的宽
        int res = 0;//最大的宽
        for(int i = 1; i <= row; ++i){
            for(int j = 1; j <= col; ++j){
                if(matrix[i-1][j-1] == '1'){
                    dp[i][j] = 1 + min(dp[i-1][j-1], min(dp[i-1][j], dp[i][j-
1]));
                    res = max(res, dp[i][j]);
                }
            }
        }
        return res*res;
    }
};
```

string

```
string s;       // 初始化
s.size();       // 返回s的长度
s += x;         // 把x加到s的末尾
s.find(x);      // 查找x在s中第一次出现的位置，没有找到则返回string::npos
s.substr(pos, len);     // 从位置pos开始截取长度为len的子串
s.insert(pos, t);       // 在位置pos处插入字符串t
s.erase(pos, len);      // 从位置pos开始删除长度为len的子串
getline(cin, s); //读入带空格的字符串，如果之前有换行符，要先getchar();
```

vector

```
vector<int> v;      // 初始化
vector<int> v(n);   // 初始化v的长度为n
v.push_back(x);     // 向v的尾部插入x
v.pop_back();       // 弹出v的最后一个元素
v.insert(v.begin() + i, x);     // 在第i个位置插入x
v.erase(v.begin() + i);         // 删除第i个位置的元素
v.erase(v.begin() + i, v.begin() + j);  // 删除区间[i:j)里的元素
v.size();           // 返回v中元素个数
v.clear();          // 清空
v.empty();          // 判空
v.back();           // 返回最后一个元素
v.front();          // 返回第一个元素
v.resize(n);        // 调整v的长度为n，多删少补
v.resize(n, 0);     // 调整v的长度为n，多删少用0补
reverse(v.begin(), v.end());    // 翻转
sort(v.begin(), v.end());       // 排序
find(v.begin(), v.end(), target); //查找target，并返回其位置
lower_bound(v.begin(), v.end(), x) - v.begin(); // 二分出第一个大于等于x元素的位置
upper_bound(v.begin(), v.end(), x) - v.begin(); // 二分出第一个大于x元素的位置
for (auto x : v) cout << x << ' ';      // 遍历
for (int i = 0; i < v.size(); i++) {    // 遍历
    cout << v[i] << ' ';
}
```

stack

```
stack<int> q;   //以int型为例
int x;
q.push(x);      //将x压入栈顶
q.top();        //返回栈顶的元素
q.pop();        //删除栈顶的元素
q.size();       //返回栈中元素的个数
q.empty();      //检查栈是否为空,若为空返回true,否则返回false
```

set

```
set<int> s;      // 初始化
s.insert(x);     // 插入元素
s.count(x);      // 判断容器中是否存在x
s.size();        // 返回容器长度
s.erase(x);      // 删除集合中的元素x
s.clear();       // 清空集合中元素
s.empty();       // 判断是否为空
s.begin();       // 返回第一个元素的迭代器
s.end();         // 返回最后一个元素加1的迭代器
s.rbegin();      // 反向迭代器
s.find(x);       // 返回元素x的迭代器，没找到返回s.end()
s.lower_bound(x);   // 返回第一个大于等于x的迭代器，没找到返回s.end()
s.upper_bound(x);   // 返回第一个大于x的迭代器，没找到返回s.end()
for (auto x : s) {  // 正向遍历
    cout << x << ' ';
}
for (auto it = --s.end(); it != --s.begin(); it--) {    // 反向遍历
    cout << *it << ' ';
}
// 自定义结构体，按关键字y从小到大排序
struct node {
    int x, y;
    bool operator < (const node& a) const {
        return y < a.y;
    }
};
set<node> s;
s.insert({1, 2});
s.insert({2, 1});
s.insert({3, 3});
for (auto it : s) {
    cout << it.x << ' ' << it.y << endl;
}
```