A. 全排列

```
signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> a(n);
    iota(all(a), 1);
    do
    {
        for(int i : a) cout << i << ' ';
        cout << '\n';
    }while(next_permutation(all(a)));
    return 0;
}
```

B. N皇后

```
int g[10];
int n, ans;

int col[11], b[21], db[21];

void dfs(int u)
{
        if(u > n)
        {
                ans++;
                return;
        }
        for(int i = 1; i <= n; i ++) // (u, i) -> (x, y)
        {
                if(col[i] == 1 || b[u + i] == 1 || db[i - u + n] == 1) continue;
                col[i] = b[u + i] = db[i - u + n] = 1;
                dfs(u + 1);
                col[i] = b[u + i] = db[i - u + n] = 0;
        }
}

signed main()
{
        cin >> n;
        dfs(1);
        cout << ans << '\n';
        return 0;
}
```

C. 合成质数

在dfs选择数的时候，由于我们是按照从小到大选的，即选择 $a_1a_2a_3$ 和选 $a_3a_2a_1$ 是完全等价的，因此我们从前往后选的时候，假如当前选择了 a_p ，那么就不用再去选 a_p 前面的数了，因为这样会导致重复，因此只要从 a_p 开始往后选即可，这是一个非常重要的 剪枝 。

```
int n, k;
vector<int> a(21),vis(21);
int ans;

bool isPrime(int u)
{
    if(u<2) return false;
    for(int i=2;i<=u/i;i++)
    {
        if(u%i==0) return false;
    }
    return true;
}

// 选了u个数，和为sum，从第p个数开始选
void dfs(int u, int sum, int p)
{
    if(u == k)
    {
        ans += isPrime(sum);
        return;
    }
    for(int i = p; i <= n; i ++)
    {
        if(!vis[i])
        {
            vis[i] = 1;
            dfs(u + 1, sum + a[i], i + 1);
            vis[i] = 0;
        }
    }
}

signed main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i ++) cin >> a[i];
    dfs(0, 0, 1);
    cout << ans;
    return 0;
}
```

D. Wordle

暴力枚举所有三个大写字母构成的字符串，依次去检验他是否能够匹配上之前的约束条件。

```
signed main()
{
    IOS;
    vector<string> word(5);
    vector<int> a(5), b(5);
    for(int i = 0; i < 5; i ++)
        cin >> word[i] >> a[i] >> b[i];
    auto check = [&](string s)
    {
        for(int i = 0; i < 5; i ++)
        {
            int aa = 0, bb = 0;
            for(int j = 0; j < 3; j ++)
            {
                if(s.find(word[i][j]) != string::npos) aa++; // 包含相同字母
                if(s[j] == word[i][j]) bb++; // 且位置相同
            }
            if(aa != a[i] or bb != b[i]) return 0;
        }
        return 1;
    };
    vector<string> ans;
    for (char i = 'A'; i <= 'Z'; i++) {
        for (char j = 'A'; j <= 'Z'; j++) {
            for (char k = 'A'; k <= 'Z'; k++) {
                string s = {i, j, k};
                if (check(s)) {
                    ans.push_back(s);
                }
            }
        }
    }
    cout << ans.size() << '\n';
    for (auto &x : ans) {
        cout << x << '\n';
    }
    return 0;
}
```

E. Orange走迷宫

```
using pii = pair<int,int>;
int g[10];
int n, m;
int graph[1010][1010], vis[1010][1010], dist[1010][1010];
int dx[] = {0,1,0,-1};
int dy[] = {1,0,-1,0};
void bfs()
{
        queue<pii> q;
        q.push({1,1});
        vis[1][1] = 1;
        dist[1][1] = 0;
        while(q.size() > 0)
        {
                auto [x, y] = q.front();
                q.pop();
                for(int i = 0; i < 4; i ++)
                {
                        int tx = x + dx[i];
                        int ty = y + dy[i];
                        if(tx < 1 || tx > n || ty < 1 || ty > m || vis[tx][ty] || graph[tx][ty]) continue;
                        dist[tx][ty] = dist[x][y] + 1;
                        vis[tx][ty] = 1;
                        q.push({tx, ty});
                }
        }
}

signed main()
{
        cin >> n >> m;
        for(int i = 1; i <= n; i ++)
                for(int j = 1; j <= m; j ++)
                        cin >> graph[i][j];
        bfs();
        cout << dist[n][m] << '\n';
        return 0;
}
```

F. 八数码

```
int dx[] = {0, 1, 0, -1};
int dy[] = {1, 0, -1, 0};
map<string, int> vis, dist;
string ts = "12345678x";
int get(int x,int y)
{
        return x * 3 + y;
}
int bfs(string s)
{
        queue<string> q;
        q.push(s);
        vis[s] = 1;
        dist[s] = 0;
        while(q.size() > 0)
        {
                auto u = q.front();
                q.pop();
                int p = u.find("x");
                int x = p / 3;
                int y = p % 3;
                for(int i = 0; i < 4; i ++)
                {
                        int tx = x + dx[i];
                        int ty = y + dy[i];
                        if(tx < 0 || tx >= 3 || ty < 0 || ty >= 3) continue;
                        string origin = u;
                        swap(u[get(x, y)], u[get(tx, ty)]);
                        if(u == ts)
                        {
                                return dist[origin] + 1;
                        }
                        if(vis[u])
                        {
                                u = origin;
                                continue;
                        }
                        dist[u] = dist[origin] + 1;
                        vis[u] = 1;
                        q.push(u);
                        u = origin;
                }
        }
        return -1;
}

signed main()
{
        string s;
        for(int i = 0; i < 9; i ++)
        {
                char c;
                cin >> c;
                s += c;
        }
        cout << bfs(s) << '\n';
        return 0;
}
```

G. 马走日

同样使用bfs即可，跟E本质相同，不过是把位移向量换成了马的走法。

注意一下输出格式即可。

```cpp
int graph[401][401], dist[401][401];
// 马的所有走法
int dx[8] = {-2, -2, 2, 2, 1, -1, 1, -1};
int dy[8] = {-1, 1, -1, 1, 2, -2, -2, 2};
int main()
{
    memset(dist, -1, sizeof(dist)); // 把dist数组初始化称-1
    int n, m, x, y;
    cin >> n >> m >> x >> y;
    queue<array<int, 2>> q;
    q.push({x, y});
    dist[x][y] = 0;
    graph[x][y] = 1;
    while (!q.empty())
    {
        auto [nx, ny] = q.front();
        q.pop();
        for (int i = 0; i < 8; i++)
        {
            int tx = nx + dx[i];
            int ty = ny + dy[i];
            if (tx > 0 && tx <= n && ty > 0 && ty <= m && graph[tx][ty] == 0)
            {
                graph[tx][ty] = 1;
                dist[tx][ty] = dist[nx][ny] + 1;
                q.push({tx, ty});
            }
        }
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            printf("%-5d", dist[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

H. 流星雨

同样是简单的bfs,这里把走迷宫的墙换成了随时间落下的流星雨，也就是说流星雨落下之前是可以走的，当流星雨落下之后就不能走了。我们可以用一个二维数组存储每个格子**最早有流行落下的时刻**，然后每次bfs的时候，如果当前格子流星雨已经落下，那么就不能走，否则可以走。实际上只需要在bfs的过程中加入一个时间维度即可，即(x,y,t)表示走到(x,y)这个格子的时间是t。最后，第一个找到的没有流星雨的格子就是答案。

```cpp
constexpr int N = 300 + 10;
using piii = tuple<int, int, int>;
int mat[N][N], vis[N][N];

signed main()
{
    memset(mat, -1, sizeof mat);
    int n;
    cin >> n;
    int dx[] = {0, 0, -1, 1};
    int dy[] = {-1, 1, 0, 0};
    while (n--)
    {
        int x, y, z;
        cin >> x >> y >> z;
        if (mat[x][y] != -1)
            mat[x][y] = min(mat[x][y], z);
        else
            mat[x][y] = z;
        // 流星雨是一个十字型，因此要标记当前格子的上下左右四个格子
        for (int i = 0; i < 4; i++)
        {
            int px = x + dx[i];
            int py = y + dy[i];
            if (px < 0 || py < 0)
                continue;
            else
            {
                // 特别注意一下，每个格子要记录最早落下的流星雨时间
                if (mat[px][py] != -1)
                    mat[px][py] = min(mat[px][py], z);
                else
                    mat[px][py] = z;
            }
        }
    }
    queue<piii> q;
    q.push({0, 0, 0});
    vis[0][0] = 1;
    while (q.size())
    {
        auto [x, y, t] = q.front();
        q.pop();
        if (mat[x][y] == -1) // 如果这个格子没有流星雨，那么就输出答案
        {
            cout << t;
            return 0;
        }
        for (int i = 0; i < 4; i++)
        {
            int px = x + dx[i];
            int py = y + dy[i];
            // 注意一下，我们走到(px,py)时，时间应该是 t + 1，因为流星雨是同时落下的，因此要判断t + 1是否大于等于流星雨最早落下的时间，否则不能走。
            if (px < 0 || py < 0 || vis[px][py] || (mat[px][py] != -1 && t + 1 >= mat[px][py]))
                continue;
            vis[px][py] = 1;
            q.push({px, py, t + 1});
        }
    }
    cout << -1;
    return 0;
}
```

I. Orange走迷宫II

> DFS: 如果找任意可行解或者全局所有解
> BFS: 如果找全局最优解

因此本题要找所有的解，需要用 DFS ，每次遇到一个分岔路口，就把每条路都看成一个子节点 DFS 一次。每次走到终点都算一种不同的方案，计数即可。

注意一下对于每次 DFS 我们都要标记走过的路，防止重复走。然后每次回溯的时候都记得要取消标记。

```
int graph[6][6], vis[6][6], ans = 0;
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};
int n, m, q, st_x, st_y, ed_x, ed_y;
void dfs(int x, int y)
{
    if (x == ed_x && y == ed_y)
    {
        ans++;
        return;
    }
    for (int i = 0; i < 4; i++)
    {
        int px = x + dx[i], py = y + dy[i];
        if (px < 1 || px > n || py < 1 || py > m)
            continue;
        if (graph[px][py] == 0 && !vis[px][py])
        {
            vis[px][py]++;
            dfs(px, py);
            vis[px][py]--;
        }
    }
}
signed main()
{
    IOS;
    cin >> n >> m >> q >> st_x >> st_y >> ed_x >> ed_y;
    while (q--)
    {
        int u, v;
        cin >> u >> v;
        graph[u][v] = 1;
    }
    vis[st_x][st_y] = 1;
    dfs(st_x, st_y);
    cout << ans;
    return 0;
}
```

J. 全球变暖

本题考察的是连通块搜索问题，我们依次去枚举地图上的每一个点，如果这个点冰川，我们就把它作为起点，去搜索这个冰川的连通块，即与他相连的冰川，然后判断一下，他所有的冰川中，是否存在一个以上四周都是冰川的(即不与海洋直接接触)冰川，如果存在，这说明当前连通块不会沉没，让dfs返回 `1` 即可，否则返回 `0` 。

本题也可以使用bfs完成，可以自行尝试。

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

int vis[1010][1010];
string graph[1010];

int n;

int dfs(int x,int y)
{
    int flag = 1;
    // 检查当前冰川的四周是否都是冰川
    for(int i = 0; i < 4; i ++)
    {
        int px = x + dx[i], py = y + dy[i];
        // 越界表示不满足
        if(px < 0 or px >= n or py < 0 or py >= n)
        {
            flag = 0;
            break;
        }
        flag &= (graph[px][py] == '#');
    }
    // 开始向四周dfs拓展，寻找连通块
    for(int i = 0; i < 4; i ++)
    {
        int px = x + dx[i], py = y + dy[i];
        if(px < 0 or px >= n or py < 0 or py >= n or graph[px][py] == '.' || vis[px][py])
            continue;
        vis[px][py] = 1;
        // 答案每次都用位或计算，因为只要存在一个连通块满足条件，答案就是1
        flag |= dfs(px, py);
    }
    // 返回当前搜索结果，即是否存在一个以上四周都是冰川的冰川
    return flag;
}

signed main()
{
    IOS;
    cin >> n;
    for(int i = 0; i < n; i ++)
        cin >> graph[i];
    int ans = 0;
    // 枚举每一个点
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n; j ++)
        //如果这个点是一个之前没有搜过的冰川，这查找这个冰川的连通块
            if(graph[i][j] == '#' and !vis[i][j])
            {
                vis[i][j] = 1;
                ans += dfs(i, j);
            }
    cout << ans << '\n';
    return 0;
}
```

K. 飞机降落

枚举所有飞机降落的顺序即可，我们可以记录一个当前的时间点 T，对于一架需要降落的飞机，如果他到达时间晚于 T，那么就等到他到达再降落，降落完成时间为 $t_i + l_i$，如果他早于当前时间到达，且盘旋时间 $T - t_i \leq d_i$，说明还在飞机的盘旋时间范围内，让他开始降落即可，降落完成时间为 $T + l_i$。否则，这说明当前这架飞机不能降落，因此当前顺序不合法，直接剪枝即可。

```
#define rep(_x, _y, _z) for (int _x = _y; _x < _z; _x++)
typedef tuple<int,int,int> piii; // 到达时间，盘旋时间，降落需要时间
void solve()
{
    int n;
    cin >> n;
    vector<piii> a(n + 1);
    rep(i,1,n + 1)
    {
        auto &[x,y,z] = a[i];
        cin >> x >> y >> z;
        y += x; // y表示最晚能降落的时间，也就是t + d
    }
    vector<int> vis(n + 1);
    int t = 0;
    bool ok = 0;
    auto dfs = [&](auto dfs, int u)
    {
        if(u == n || ok) // 只要找到一种合法方案，就全部返回。
        {
            ok = 1;
            return;
        }
        for (int i = 1; i <= n; i++)
        {
            auto [x,y,z] = a[i];
            if (vis[i]) // 如果这架飞机已经降落过，就跳过
                continue;
            if(t > y) return; // 如果当前时间已经超过飞机i最晚降落时间，说明飞机i无法降落，剪枝
            int p = t; // 记录一下当前时间用于后面回溯
            t = max(x + z, t + z); // 更新当前时间
            vis[i] = 1; // 标记飞机i降落过
            dfs(dfs,u + 1); // 继续搜索
            // 回溯
            vis[i] = 0;
            t = p;
        }
    };
    dfs(dfs,0);
    cout << (ok ? "YES" : "NO") << '\n';
}

signed main()
{
    int _ = 1;
    cin >> _;
    while (_--)
        solve();
    return 0;
}
```

L. 单词接龙

可以先预处理出所有字符之间得最短后缀与前缀重合的转移数组 g_{ij}，表示第 i 个单词可以接到第 j 个单词后面。

注意一下，我们不知道搜索结束的条件，因此每次搜索都需要更新一次答案。

同时每个单词最多使用2次，因此我们可以把标记数组改成计数数组，即vis_i表示第 i 个单词已经使用了多少次。

```cpp
const int N = 21;

int n;
string word[N];
int g[N][N];
int vis[N];
int ans;

void dfs(string dragon, int last)
{
    ans = max((int)dragon.size(), ans);

    vis[last] ++ ;

    for (int i = 0; i < n; i ++ )
        if (g[last][i] && vis[i] < 2)
            dfs(dragon + word[i].substr(g[last][i]), i);

    vis[last] -- ;
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i ++ ) cin >> word[i];
    char start;
    cin >> start;

    // 预处理可以拼接的单词
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n; j ++ )
        {
            string a = word[i], b = word[j];
            for (int k = 1; k < min(a.size(), b.size()); k ++ )
                if (a.substr(a.size() - k, k) == b.substr(0, k))
                {
                    g[i][j] = k;
                    break;
                }
        }

    for (int i = 0; i < n; i ++ )
        if (word[i][0] == start)
            dfs(word[i], i);

    cout << ans << '\n';

    return 0;
}
```

M. kotori和素因子

```
#include<stdio.h>
#define N 1010
int prime[168] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,16
bool st[168];         //第几个素数被用过了
int f[10][100];         //第i个数的质因子有第几个素数
int n,cnt[10];         //第i个数有几个质因子

int ans = 0x3f3f3f3f;
void dfs(int num, int sum)
{
    if(num == n){
        if(sum < ans)  ans = sum;
        return;
    }else{
        for(int i = 0; i < cnt[num]; i ++)
        {
            if(!st[f[num][i]]){
                st[f[num][i]] = true;
                dfs(num + 1, sum + prime[f[num][i]]);
                st[f[num][i]] = false;
            }
        }
    }
}
int main()
{
    scanf("%d",&n);
    for(int i = 0; i < n; i ++)
    {
        int a;
        scanf("%d",&a);
        for(int j = 0; a != 1; j ++)
        {
            if(a % prime[j] == 0)
            {
                f[i][cnt[i]++] = j;         //质因子有第几个素数
                while(a % prime[j] == 0)
                    a /= prime[j];
            }
        }
    }
    dfs(0,0);
    if(ans == 0x3f3f3f3f)  ans = -1;
    printf("%d",ans);
    return 0;
}
```