# 1 Formula

Eular Formular: |V| - |E| + |F| = 2 | |=| | | | | | | | | | | | | | |=Ω | | | | | | | | | | | | | | | Catalan Number: $C_n = (4n-2)/(n+1)C_{n-1}$ Burnside 引理可以通过 dfs 序列化 Floyd 算法别忘了设置 dist[i][i] = 0 想着比赛的时候可以打表初始化一定不要忘记提交时记得把所有的调试信息都删掉想着可以用二分的方法，把问题转化为判定问题。对于几何问题，没想法就先动手画画图，别上来就解析法。数组一定要开的足够打大，能用 LL 就别用 int 数位 dp 一定要写暴力 check

# 2 Edit Esp

```
int main() {
    static const int stksz = 10000000;
    static int stk[stksz], espbak;
    __asm__ __volatile__ ( "movl␣%%esp,␣%0\n\tmovl␣%1,␣%%esp\n\t" : "=g"(
        espbak) : "g"(stk+stksz-1)  );
    solve();
    exit(0);
}
```

# 3 Java Header

```
import java.io.*;
import java.util.*;
import java.math.*;

class Task {
    void solve( int ri, InputReader in, PrintWriter out ) {
        BigDecimal a = new BigDecimal("23213432.2142143");
        a = a.round( new MathContext(10, RoundingMode.HALF_UP) );
        out.println( a.toPlainString() );
    }
}

public class Main {
    public static void main(String []args) {
        InputStream insm = System.in;
        OutputStream outsm = System.out;
        InputReader in = new InputReader( insm );
        PrintWriter out = new PrintWriter( outsm );
        Task task = new Task();
        task.solve(1, in, out);
        out.close();
    }
}

class InputReader {
    private BufferedReader reader;
    private StringTokenizer tokenizer;

    public InputReader( InputStream sm ) {
        reader = new BufferedReader( new InputStreamReader(sm) );
        tokenizer = null;
    }

    public String next() {
        while ( tokenizer == null || !tokenizer.hasMoreTokens() ) {
            try {
                tokenizer = new StringTokenizer( reader.readLine() );
            } catch( IOException e ) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }
}
```

# 4 Dancing Links 精确覆盖 (矩阵处理)

```
const int maxN = 60 * 20, maxM = 60 * 10;
const int max_size = maxN * maxM;
const int inf = 0x3f3f3f3f;
int L[max_size], R[max_size], U[max_size], D[max_size], CH[max_size], RH[
    max_size];
int S[maxM], O[maxM];
int head, size;
int node(int up, int down, int left, int right) {
    U[size] = up, D[size] = down;
    L[size] = left, R[size] = right;
    D[up] = U[down] = R[left] = L[right] = size;
    return size++;
}
bool mat[maxN][maxM];
void init(int N, int M) {
    size = 0;
    head = node(0, 0, 0, 0);
    for (int j = 1; j <= M; ++j) {
        CH[j] = node(size, size, L[head], head), S[j] = 0;
    }
    for (int i = 1; i <= N; ++i) {
        int row = -1, k;
        for (int j = 1; j <= M; ++j) {
            if (!mat[i][j]) continue;
            if (row == -1) {
                row = node(U[CH[j]], CH[j], size, size);
                RH[row] = i, CH[row] = CH[j], ++S[j];
            } else {
                k = node(U[CH[j]], CH[j], L[row], row);
                RH[k] = i, CH[k] = CH[j], ++S[j];
            }
        }
    }
}
void remove(const int &c) {
    L[R[c]] = L[c], R[L[c]] = R[c];
    for (int i = D[c]; i != c; i = D[i]) {
        for (int j = R[i]; j != i; j = R[j]) {
            U[D[j]] = U[j], D[U[j]] = D[j];
            --S[CH[j]];
        }
    }
}
void resume(const int &c) {
    for (int i = U[c]; i != c; i = U[i]) {
        for (int j = L[i]; j != i; j = L[j]) {
            ++S[CH[j]];
            U[D[j]] = D[U[j]] = j;
        }
    }
    L[R[c]] = R[L[c]] = c;
}
int len;
bool DLX(const int &k) {
    if (R[head] == head) {
        len = k - 1;
        return true;
    }
    int s = inf, c;
    for (int t = R[head]; t != head; t = R[t]) {
        if (S[t] < s) s = S[t], c = t;
    }
    remove(c);
    for (int i = D[c]; i != c; i = D[i]) {
        O[k] = RH[i];
        for (int j = R[i]; j != i; j = R[j]) {
            remove(CH[j]);
        }
        if (DLX(k + 1)) {
            return true;
        }
        for (int j = L[i]; j != i; j = L[j]) {
```

```
            resume(CH[j]);
        }
    }
    resume(c);
    return false;
}
```

# 5 Dancing Links 重复覆盖 (矩阵处理)

```
const int head = 0;
const int INF=10000000;
const int maxn = 1700;
const int maxd = 1000000;
int N, M, K, n, m, cnt, res;
int mat[maxn][maxn], s[maxn], l[maxd], r[maxd], u[maxd], d[maxd], c[maxd],
    o[maxn], row[maxd];
bool use[maxn];
void makegragh(int &n, int &m) {
    memset(mat, 0, sizeof(mat));
    //init
}
void initial(int n, int m) {
    memset(use, false, sizeof(use));
    res = n + 1;
    int i, j, rowh;
    memset(s, 0, sizeof(s));
    for(i=head; i<=m; i++) {
        r[i]=(i+1)%(m+1);
        l[i]=(i-1+m+1)%(m+1);
        u[i]=d[i]=i;
    }
    cnt=m+1;
    for(i=1; i<=n; i++) {
        rowh=-1;
        for(j=1; j<=m; j++) {
            if(mat[i][j])
            {
                s[j]++; u[cnt]=u[j]; d[u[j]]=cnt;
                u[j]=cnt; d[cnt]=j; row[cnt]=i; c[cnt]=j;
                if(rowh==-1) {
                    l[cnt]=r[cnt]=cnt; rowh=cnt;
                }
                else {
                    l[cnt] = l[rowh]; r[l[rowh]] = cnt;
                    r[cnt] = rowh; l[rowh] = cnt;
                }
                cnt++;
            }
        }
    }
}
void remove(int c) {
    for(int i=d[c]; i!=c; i=d[i]) {
        r[l[i]]=r[i]; l[r[i]]=l[i];
    }
}
void resume(int c) {
    for(int i=d[c]; i!=c; i=d[i])
        r[l[i]]=l[r[i]]=i;
}
int h() {
    bool has[maxn];
    memset(has, false, sizeof(has));
    int ans=0;
    for(int i=r[head]; i!=head; i=r[i])
        if(!has[i]) {
            ans++;
            for(int j=d[i]; j!=i; j=d[j])
                for(int k=r[j]; k!=j; k=r[k])
                    has[c[k]]=true;
        }
    return ans;
```

```
}
bool dfs(int k) {
    if(k+h()>=res)return false;//A* cut
    if(r[head]==head) {
        if(k<res) res=k;
        return true;
    }
    int ms=INF, cur=0;
    for(int t=r[head]; t!=head; t=r[t])
        if(s[t]<ms) {
            ms=s[t]; cur=t;
        }
    for(int i=d[cur]; i!=cur; i=d[i]) {
        remove(i);
        for(int j=r[i]; j!=i; j=r[j]) {
            remove(j); s[c[j]]--;
        }
        dfs(k+1);
        for(int j=l[i]; j!=i; j=l[j]) {
            resume(j); s[c[j]]++;
        }
        resume(i);
    }
    return false;
}
```

leftist

```
//TODO
```

treap by HL

kmp

exkmp

迪卡尔树

SA

AC 自动机

Manacher