Search

// BFS (double-ended)

int de\_bfs()

{

    if (st==te) return 0;

    unordered\_map<string, int> vis;

    queue<string> q;

    q.push(st); q.push(te);

    d[st]=d[te]=0;

    vis[st]=1; vis[te]=2;

    while (q.size())

    {

        string u=q.front(); q.pop();

        string ne=u;

        int x, y, cnt=0;

        while (cnt<9)

        {

            s[cnt/3][cnt%3]=ne[cnt]-'0';

            if (ne[cnt]=='0') { x=cnt/3; y=cnt%3; }

            cnt++;

        }

        for (int i=0; i<4; i++)

        {

            int tx=x+dx[i], ty=y+dy[i];

            if (0>tx || tx>=3 || 0>ty || ty>=3) continue;

            swap(s[x][y], s[tx][ty]);

            ne="";

            for (int j=0; j<3; j++) for (int k=0; k<3; k++) ne+=(char)('0'+s[j][k]);

            if (vis[ne]==vis[u])

            {

                swap(s[x][y], s[tx][ty]);

                continue;

            }

            if ((vis[ne]==1 && vis[u]==2) || (vis[ne]==2 && vis[u]==1)) return d[ne]+d[u]+1;

            q.push(ne);

            d[ne]=d[u]+1;

            vis[ne]=vis[u];

            swap(s[x][y], s[tx][ty]);

        }

    }

    return -1;

}

// Rabin-Karp Algorithm

struct node

{

    const int mod[2]={1000000007, 1000000009};

    int pwr[2][N];

    unordered\_map<string, vector<vector<int>>> h;

    node()

    {

        pwr[0][0]=pwr[1][0]=1;

        for (int i=1; i<N; i++)

        {

            pwr[0][i]=(pwr[0][i-1]\*131)%mod[0];

            pwr[1][i]=(pwr[1][i-1]\*131)%mod[1];

        }

    }

    void insert(string x)

    {

        h[x].resize(2, vector<int>(lx+1));

        h[x][0][0]=h[x][1][0]=0;

        for (int i=0; i<lx; i++)

        {

            h[x][0][i+1]=(h[x][0][i]+x[i]\*pwr[0][i])%mod[0];

            h[x][1][i+1]=(h[x][1][i]+x[i]\*pwr[1][i])%mod[1];

        }

    }

    vector<int> query(string x, string y)

    {

        vector<int> ans;

        for (int i=0; i+ly-1<lx; i++) if (check(i, x, y)) ans.push\_back(i);

        return ans;

    }

    bool check(int pos, string x, string y)

    {

        bool t1=calc(0, h[x][0][pos+ly]-h[x][0][pos]) == calc(0, h[y][0][ly]\*pwr[0][pos]);

        bool t2=calc(1, h[x][1][pos+ly]-h[x][1][pos]) == calc(1, h[y][1][ly]\*pwr[1][pos]);

        return t1 && t2;

    }

    int calc(int i, int x) { return (x%mod[i]+mod[i])%mod[i]; }

} \*rk;