TITLEPAGE

Bc., Ing., Ph.D.

October 22, 2018

Die

Die

- Hardcode all six possibilities
- Rotate
- ...
- Profit $ $ $

Horsemeat - alternative solution (matrix multiplication)

Horsemeat - alternative solution (matrix multiplication)

```
    int x;
    int beg=0;
    for(int i=0;i<4;i++) {
        cin>>x;
        beg+=x;
        if(i<3)
            beg<<=8;
    }

    stavy1.insert({beg, 1.0});

    vector<double> prs(2);
    int pl=0; // 0 white, 1 black

    while(true) {
        // move
        for(auto && s: stavy1) {
            // if(s.second==0) continue; // WTF

            int stav=s.first;
            int x,y, x2, y2;

            if(pl==0) {
                x=(stav>>24)&255;
                y=(stav>>16)&255;
                x2=(stav>>8)&255;
                y2=(stav)&255;
            }else{
                x2=(stav>>24)&255;
                y2=(stav>>16)&255;
                x=(stav>>8)&255;
                y=(stav)&255;
            }

            int no_news=0;

            set<pair<int,int>> tahy = { {-2, -1},
                                        {-2, +1},
                                        {+2, -1},
                                        {+2, +1},
                                        {-1, -2},
                                        {-1, +2},
                                        {+1, -2},
                                        {+1, +2} };

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8)
                    no_news++;
            }

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8) {
                    int newState=0;
                    if(pl==0) {
                        newState+=nx<<24;
                        newState+=ny<<16;
                        newState+=x2<<8;
                        newState+=y2;
                    }else{
                        newState+=x2<<24;
                        newState+=y2<<16;
                        newState+=nx<<8;
                        newState+=ny;
                    }

                    //cout<<nx<<" "<<ny<<" "<<newState<<"AAAAA"<<endl;

                    if(nx==x2 && ny==y2) {
                    //cout<<"PLUS"<<s.second/no_news<<" "<<pl<<endl;
                        prs[pl]+=s.second/no_news;
                    }else
                    stavy2[newState]+=s.second/no_news;
                }
            }

        }

        // sun
        double sun=0.0;
        for(auto && s: stavy2) {
            sun+=s.second;
        }

        if(sun<0.000001) break;

        pl=1-pl;


        stavy1=std::move(stavy2);
        stavy2.clear();
    }

    //cout<<prs[0]<<' '<<prs[1];

    if(prs[0]<prs[1]) cout<<"black"<<endl;
    else cout<<"white"<<endl;

    return 0;
}
```

Horsemeat - alternative solution (matrix multiplication)

```
    int x;
    int beg=0;
    for(int i=0;i<4;i++) {
        cin>>x;
        beg+=x;
        if(i<3)
            beg<<=8;
    }

    stavy1.insert({beg, 1.0});

    vector<double> prs(2);
    int pl=0; // 0 white, 1 black

    while(true) {
        // nowe
        for(auto && s: stavy1) {
            // if(s.second==0) continue; // WTF

            int stav=s.first;
            int x,y, x2, y2;

            if(pl==0) {
                x=(stav>>24)&255;
                y=(stav>>16)&255;
                x2=(stav>>8)&255;
                y2=(stav)&255;
            }else{
                x2=(stav>>24)&255;
                y2=(stav>>16)&255;
                x=(stav>>8)&255;
                y=(stav)&255;
            }

            int no_news=0;

            set<pair<int,int>> tahy = { {-2, -1},
                                        {-2, +1},
                                        {+2, -1},
                                        {+2, +1},
                                        {-1, -2},
                                        {-1, +2},
                                        {+1, -2},
                                        {+1, +2} };

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8)
                    no_news++;
            }

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8) {
                    int newState=0;
                    if(pl==0) {
                        newState+=nx<<24;
                        newState+=ny<<16;
                        newState+=x2<<8;
                        newState+=y2;
                    }else{
                        newState+=x2<<24;
                        newState+=y2<<16;
                        newState+=nx<<8;
                        newState+=ny;
                    }

                    //cout<<nx<<" "<<ny<<" "<<newState<<"AAAAA"<<endl;

                    if(nx==x2 && ny==y2) {
                    //cout<<"PLUS"<<s.second/no_news<<" "<<pl<<endl;
                        prs[pl]+=s.second/no_news;
                    }else
                    stavy2[newState]+=s.second/no_news;
                }
            }

        }

        // sun
        double sun=0.0;
        for(auto && s: stavy2) {
            sun+=s.second;
        }

        if(sun<0.000001) break;

        pl=1-pl;


        stavy1=std::move(stavy2);
        stavy2.clear();
    }

    //cout<<prs[0]<<' '<<prs[1];

    if(prs[0]<prs[1]) cout<<"black"<<endl;
    else cout<<"white"<<endl;

    return 0;
}
```

```
#define pb push_back
#define mp make_pair

typedef vector<int> vi;
typedef pair<int,int> pii;
typedef long long ll;
typedef long double ld;

const int maxn = 1000 + 5;

typedef array<array<ld, 8>, 8> stan;

stan pusty_stan() {
    stan s;
    rep (i, 8) rep(j, 8) s[i][j] = 0.0;
    return s;
}

bool in_range(int x) {
    return x >=0 && x < 8;
}

bool valid(int i, int j, int dx, int dy) {
    bool kon = ((abs(dx) == 2 && abs(dy) == 1) || (abs(dx) == 1 && abs(dy) == 2));
    return kon && in_range(dx + i) && in_range(dy + j);
}

stan kolejny(const stan& obecny, const stan& drugi) {
    stan res = pusty_stan();

    rep(i, 8) {
        rep(j, 8) {
            vector<pii> pos;
            for (int k = -2; k <= 2; k++) {
                for (int l = -2; l <= 2; l++) {
                    if (!valid(i, j, k, l)) continue;
                    pos.push_back({i + k, j + l});
                }
            }

            ld ile = (ld)pos.size();
            for (auto p : pos) {
                int x = p.first;
                int y = p.second;
                res[x][y] += obecny[i][j] * ((ld)1.0 - obecny[i][j] * drugi[i][j]) / ile;
            }
        }
    }

    return res;
}

pair<ld, ld> go(int iter, stan& graczA, stan& graczB, ld wynikA, ld wynikB) {
    //printf("a: %Lf, b: %Lf\n", wynikA, wynikB);
    if (iter >= 60) {
        return {wynikA, wynikB};
    }

    auto nowyA = kolejny(graczA, graczB);
    rep(i, 8) {
        rep(j, 8) {
            wynikA += nowyA[i][j] * graczB[i][j];
        }
    }

    auto nowyB = kolejny(graczB, nowyA);
    rep(i, 8) {
        rep(j, 8) {
            wynikB += nowyA[i][j] * nowyB[i][j];
        }
    }

    return go(iter + 1, nowyA, nowyB, wynikA, wynikB);
}

int main(){
    int a, b;
    int p, q;

    scanf("%d %d %d %d", &a, &b, &p, &q);
    a--;
    b--;
    p--;
    q--;

    auto graczA = pusty_stan();
    graczA[a][b] = 1.0;

    auto graczB = pusty_stan();
    graczB[p][q] = 1.0;

    auto wynik = go(0, graczA, graczB, 0.0, 0.0);

    //printf("%Lf, %Lf\n", wynik.first, wynik.second);

    if (abs(wynik.first - wynik.second) < (ld)0.000001) {
        puts("draw");
    } else if (wynik.first > wynik.second) {
        puts("white");
    } else {
        puts("black");
    }
}
```

Horsemeat - alternative solution (matrix multiplication)

```
    int x;
    int beg=0;
    for(int i=0;i<4;i++) {
        cin>>x;
        beg+=x;
        if(i<3)
            beg<<=8;
    }

    stavy1.insert({beg, 1.0});

    vector<double> prs(2);
    int pl=0; // 0 white, 1 black

    while(true) {
        // move
        for(auto && s: stavy1) {
            // if(s.second==0) continue; // WTF

            int stav=s.first;
            int x,y, x2, y2;

            if(pl==0) {
                x=(stav>>24)&255;
                y=(stav>>16)&255;
                x2=(stav>>8)&255;
                y2=(stav)&255;
            }else{
                x2=(stav>>24)&255;
                y2=(stav>>16)&255;
                x=(stav>>8)&255;
                y=(stav)&255;
            }

            int no_news=0;

            set<pair<int,int>> tahy = { {-2, -1},
                                        {-2, +1},
                                        {+2, -1},
                                        {+2, +1},
                                        {-1, -2},
                                        {-1, +2},
                                        {+1, -2},
                                        {+1, +2} };

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8)
                    no_news++;
            }

            for(auto && t: tahy) {
                int nx,ny;
                nx=x+t.first;
                ny=y+t.second;
                if(nx>=1 && nx<=8 && ny>=1 && ny<=8) {
                    int newState=0;
                    if(pl==0) {
                        newState+=nx<<24;
                        newState+=ny<<16;
                        newState+=x2<<8;
                        newState+=y2;
                    }else{
                        newState+=x2<<24;
                        newState+=y2<<16;
                        newState+=nx<<8;
                        newState+=ny;
                    }

                    //cout<<nx<<" "<<ny<<" "<<newState<<"AAAAA"<<endl;

                    if(nx==x2 && ny==y2) {
                    //cout<<"PLUS"<<s.second/no_news<<" "<<pl<<endl;
                        prs[pl]+=s.second/no_news;
                    }else
                    stavy2[newState]+=s.second/no_news;
                }
            }

        }

        // sun
        double sun=0.0;
        for(auto && s: stavy2) {
            sun+=s.second;
        }

        if(sun<0.000001) break;

        pl=1-pl;


        stavy1=std::move(stavy2);
        stavy2.clear();
    }

    //cout<<prs[0]<<' '<<prs[1];

    if(prs[0]<prs[1]) cout<<"black"<<endl;
    else cout<<"white"<<endl;

    return 0;
}
```

```
#define pb push_back
#define mp make_pair

typedef vector<int> vi;
typedef pair<int,int> pii;
typedef long long ll;
typedef long double ld;

const int maxn = 1000 + 5;

typedef array<array<ld, 8>, 8> stan;

stan pusty_stan() {
    stan s;
    rep (i, 8) rep(j, 8) s[i][j] = 0.0;
    return s;
}

bool in_range(int x) {
    return x >=0 && x < 8;
}

bool valid(int i, int j, int dx, int dy) {
    bool kon = ((abs(dx) == 2 && abs(dy) == 1) || (abs(dx) == 1 && abs(dy) == 2));
    return kon && in_range(dx + i) && in_range(dy + j);
}

stan kolejny(const stan& obecny, const stan& drugi) {
    stan res = pusty_stan();

    rep(i, 8) {
        rep(j, 8) {
            vector<pii> pos;
            for (int k = -2; k <= 2; k++) {
                for (int l = -2; l <= 2; l++) {
                    if (!valid(i, j, k, l)) continue;
                    pos.push_back({i + k, j + l});
                }
            }

            ld ile = (ld)pos.size();
            for (auto p : pos) {
                int x = p.first;
                int y = p.second;
                res[x][y] += obecny[i][j] * ((ld)1.0 - obecny[i][j] * drugi[i][j]) / ile;
            }
        }
    }

    return res;
}

pair<ld, ld> go(int iter, stan& graczA, stan& graczB, ld wynikA, ld wynikB) {
    //printf("a: %Lf, b: %Lf\n", wynikA, wynikB);
    if (iter >= 60) {
        return {wynikA, wynikB};
    }

    auto nowyA = kolejny(graczA, graczB);
    rep(i, 8) {
        rep(j, 8) {
            wynikA += nowyA[i][j] * graczB[i][j];
        }
    }

    auto nowyB = kolejny(graczB, nowyA);
    rep(i, 8) {
        rep(j, 8) {
            wynikB += nowyA[i][j] * nowyB[i][j];
        }
    }

    return go(iter + 1, nowyA, nowyB, wynikA, wynikB);
}

int main(){
    int a, b;
    int p, q;

    scanf("%d %d %d %d", &a, &b, &p, &q);
    a--;
    b--;
    p--;
    q--;

    auto graczA = pusty_stan();
    graczA[a][b] = 1.0;

    auto graczB = pusty_stan();
    graczB[p][q] = 1.0;

    auto wynik = go(0, graczA, graczB, 0.0, 0.0);

    //printf("%Lf, %Lf\n", wynik.first, wynik.second);

    if (abs(wynik.first - wynik.second) < (ld)0.000001) {
        puts("draw");
    } else if (wynik.first > wynik.second) {
        puts("white");
    } else {
        puts("black");
    }
}
```

```

#include <bits/stdc++.h>

using namespace std;

int n = 10000;

int isin(int a) {
    return a<8 && a>=0;
}

int posmoves(int x, int y) {
    int p = 0;
    if(isin(x+1) && isin(y+2)) p++;
    if(isin(x+1) && isin(y-2)) p++;
    if(isin(x+2) && isin(y+1)) p++;
    if(isin(x+2) && isin(y-1)) p++;
    if(isin(x-1) && isin(y+2)) p++;
    if(isin(x-1) && isin(y-2)) p++;
    if(isin(x-2) && isin(y+1)) p++;
    if(isin(x-2) && isin(y-1)) p++;
    return p;
}

vector<vector<vector<long double> > > b(8, vector<vector<long double>>( 8, vector<long double> (n,0)));
vector<vector<vector<long double> > > w(8, vector<vector<long double>>( 8, vector<long double> (n,0)));

int main() {
    int bx, by, wx, wy;
    cin >> bx >> by >> wx >> wy;
    bx--; by--; wx--; wy--;
    b[bx][by][0]=1;
    w[wx][wy][0]=1;
    for(int i = 1; i<n; i++) {
        for(int j = 0; j<8; j++) {
            for(int k = 0; k<8; k++) {

                if(isin(j+1) && isin(k+2)) { w[j][k][i] += w[j+1][k+2][i-1]/posmoves(j+1,k+2); }
                if(isin(j+1) && isin(k-2)) { w[j][k][i] += w[j+1][k-2][i-1]/posmoves(j+1,k-2); }
                if(isin(j+2) && isin(k+1)) { w[j][k][i] += w[j+2][k+1][i-1]/posmoves(j+2,k+1); }
                if(isin(j+2) && isin(k-1)) { w[j][k][i] += w[j+2][k-1][i-1]/posmoves(j+2,k-1); }
                if(isin(j-1) && isin(k+2)) { w[j][k][i] += w[j-1][k+2][i-1]/posmoves(j-1,k+2); }
                if(isin(j-1) && isin(k-2)) { w[j][k][i] += w[j-1][k-2][i-1]/posmoves(j-1,k-2); }
                if(isin(j-2) && isin(k+1)) { w[j][k][i] += w[j-2][k+1][i-1]/posmoves(j-2,k+1); }
                if(isin(j-2) && isin(k-1)) { w[j][k][i] += w[j-2][k-1][i-1]/posmoves(j-2,k-1); }

                if(isin(j+1) && isin(k+2)) { b[j][k][i] += b[j+1][k+2][i-1]/posmoves(j+1,k+2); }
                if(isin(j+1) && isin(k-2)) { b[j][k][i] += b[j+1][k-2][i-1]/posmoves(j+1,k-2); }
                if(isin(j+2) && isin(k+1)) { b[j][k][i] += b[j+2][k+1][i-1]/posmoves(j+2,k+1); }
                if(isin(j+2) && isin(k-1)) { b[j][k][i] += b[j+2][k-1][i-1]/posmoves(j+2,k-1); }
                if(isin(j-1) && isin(k+2)) { b[j][k][i] += b[j-1][k+2][i-1]/posmoves(j-1,k+2); }
                if(isin(j-1) && isin(k-2)) { b[j][k][i] += b[j-1][k-2][i-1]/posmoves(j-1,k-2); }
                if(isin(j-2) && isin(k+1)) { b[j][k][i] += b[j-2][k+1][i-1]/posmoves(j-2,k+1); }
                if(isin(j-2) && isin(k-1)) { b[j][k][i] += b[j-2][k-1][i-1]/posmoves(j-2,k-1); }
            }
        }
    }
    long double wwin=0, bwin=0;
    for(int j = 0; j<8; j++) {
        for(int k = 0; k<8; k++) {
            for(int i = 1; i<n; i++) {
                bwin+= w[j][k][i] * b[j][k][i];
                wwin+= w[j][k][i] * b[j][k][i-1];
            }
        }
    }
    if(fabs(wwin-bwin)<0.000001) {
        cout << "draw";
    }
    else if(wwin > bwin) {
        cout << "white";
    }
    else {
        cout << "black";
    }
    cout << endl;
}
```

Horsemeat - alternative solution

Horsemeat - alternative solution

```python
l = [2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2

i1, j1 = [int(x)-1 for x in input().split()]
i2, j2 = [int(x)-1 for x in input().split()]


res = l[(i1+j1*8)*64+(i2+j2*8)]

if res == 0:
    print("draw")
elif res == 1:
    print("white")
else:
    print("black")
```

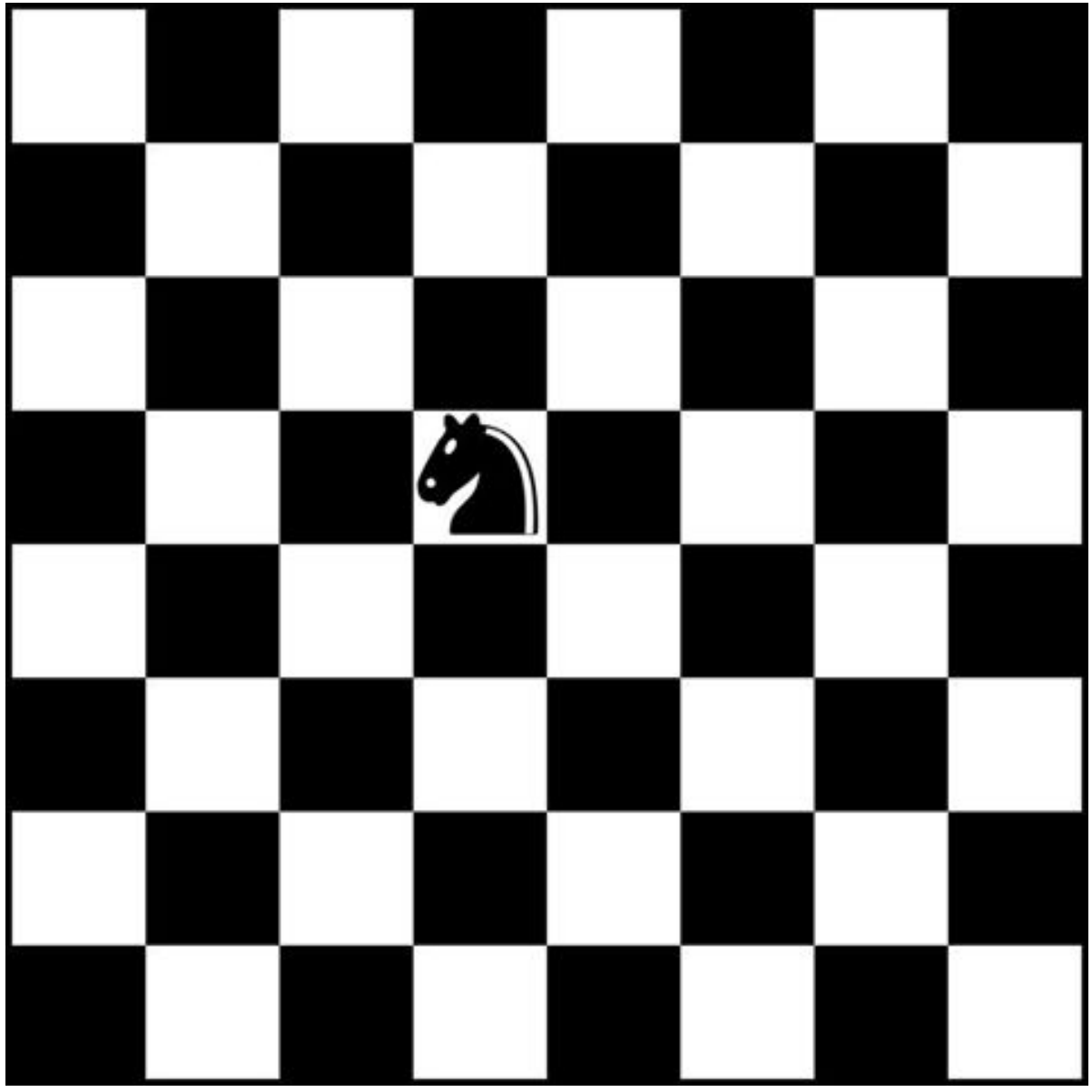
Horsemeat - alternative solution

```python
l = [2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2

i1, j1 = [int(x)-1 for x in input().split()]
i2, j2 = [int(x)-1 for x in input().split()]


res = l[(i1+j1*8)*64+(i2+j2*8)]

if res == 0:
    print("draw")
elif res == 1:
    print("white")
else:
    print("black")

```

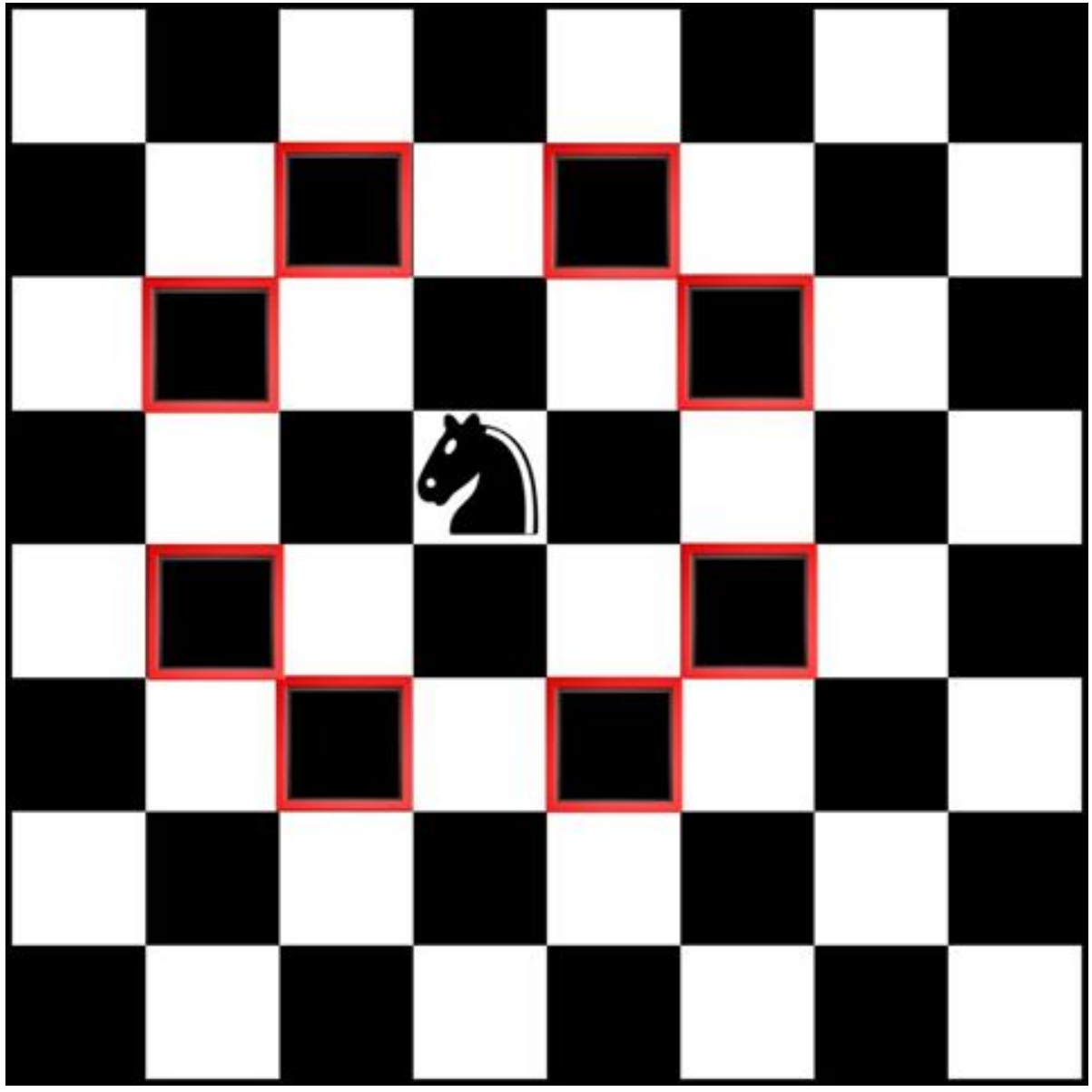
```python
l = [2, 1, 2, 1, 2, 1, 2, 1,
     1, 2, 1, 2, 1, 2, 1, 2,
     2, 1, 2, 1, 2, 1, 2, 1,
     1, 2, 1, 2, 1, 2, 1, 2,
     2, 1, 2, 1, 2, 1, 2, 1,
     1, 2, 1, 2, 1, 2, 1, 2,
     2, 1, 2, 1, 2, 1, 2, 1,
     1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2,

i1, j1 = [int(x)-1 for x in input().split()]
i2, j2 = [int(x)-1 for x in input().split()]


res = l[(i1+j1*8)*64+(i2+j2*8)]

if res == 0:
    print("draw")
elif res == 1:
    print("white")
else:
    print("black")

```

Observation

Horsemeat



Horsemeat



Security Guards

Security Guards

- Fill the 5000×5000 grid with the distance to the nearest guard.
- Use BFS with multiple starts.
- Query in constant time

Security Guards

- Fill the 5000×5000 grid with the distance to the nearest guard.
- Use BFS with multiple starts.
- Query in constant time

Complexity $O(N^2 + Q)$

Security Guards - alternative solution

- Fill the 5000×5000 grid with 1 on guard positions and 0 otherwise.
- Create 2D prefix sum.
- For each query use binary search by answer.

Security Guards - alternative solution

- Fill the 5000×5000 grid with 1 on guard positions and 0 otherwise.
- Create 2D prefix sum.
- For each query use binary search by answer.

Complexity $O(N^2 + Q \log N)$

Lightning

Lightning

- Dynamic Programming
- Parameters:
  - position in bit string
  - number of ones so far
  - carry
- Expand each state by adding 1 or 0

Lightning

- Dynamic Programming
- Parameters:
  - position in bit string
  - number of ones so far
  - carry
- Expand each state by adding 1 or 0

Complexity $O(N\,K)$

Split Game

Split Game

- Each impartial game is equivalent to NIM.
- Sum of games has nimber equivalent to XOR of nimbers of individual games.
- Note that adding two same piles do not change the outcome.
- We precompute nimbers up to piles of size 2000 (DP).
- XOR nimbers of games on the input.
- If XOR is zero second player wins, otherwise first player wins.

Split Game

- Each impartial game is equivalent to NIM.
- Sum of games has nimber equivalent to XOR of nimbers of individual games.
- Note that adding two same piles do not change the outcome.
- We precompute nimbers up to piles of size 2000 (DP).
- XOR nimbers of games on the input.
- If XOR is zero second player wins, otherwise first player wins.

Complexity $O(N^2 \log N)$

Split Game

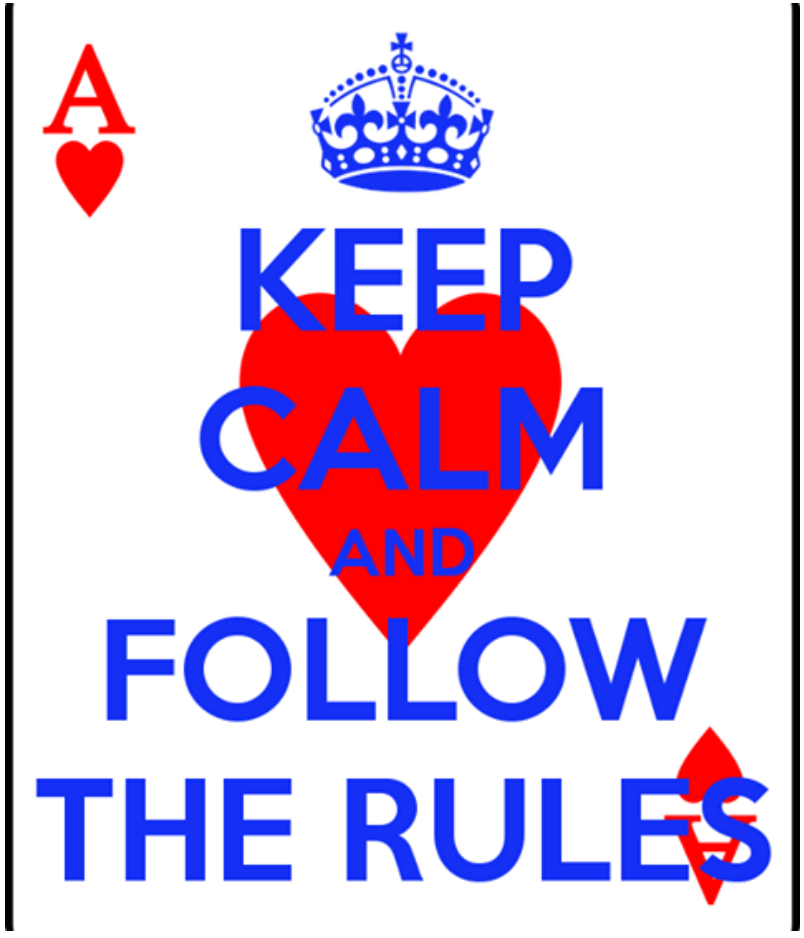
- Each impartial game is equivalent to NIM.
- Sum of games has nimber equivalent to XOR of nimbers of individual games.
- Note that adding two same piles do not change the outcome.
- We precompute nimbers up to piles of size 2000 (DP).
- XOR nimbers of games on the input.
- If XOR is zero second player wins, otherwise first player wins.

Complexity $O(N^2 \log N)$

If you try hard, you can achieve $O(N^2)$

Rulette

Rulette



Rulette

1. If you have at least 4 cards in your hand then add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. If you have at least 2 cards of the same suit in your hand then multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. If the count of black (Clubs and Spades) and the count of red (Hearts and Diamonds) cards in your hand differ then add the absolute difference of the counts to the value.
5. If the value is currently even then add all positive integer divisors of the value (including 1 and the value itself) to the value.
6. If there are exactly 4 cards of rank 7 in your hand then subtract 11^2 from the value.
7. If the value is currently non-negative then add the score of the lowest score card in your hand to the value.
8. If the value is currently negative then multiply the value by -1.
9. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
10. If there is a **straight** in your hand then add five times the number of A's in your hand to the value.
11. If the value was modified by the rules more than 8 times so far then add the number of 1's bits in the binary representation of the value to the value.
12. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
13. If there is at least one card of rank 2 in your hand then add the product of all distinct superfactors of the value to the value. A superfactor divides the value evenly and it is the highest power of a prime factor of the value.
14. If the value is 674 you win!

Rulette

1. If you have at least 4 cards in your hand then add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. If you have at least 2 cards of the same suit in your hand then multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. If the count of black (Clubs and Spades) and the count of red (Hearts and Diamonds) cards in your hand differ then add the absolute difference of the counts to the value.
5. If the value is currently even then add all positive integer divisors of the value (including 1 and the value itself) to the value.
6. If there are exactly 4 cards of rank 7 in your hand then subtract 11^2 from the value.
7. If the value is currently non-negative then add the score of the lowest score card in your hand to the value.
8. If the value is currently negative then multiply the value by -1.
9. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
10. If there is a **straight** in your hand then add five times the number of A's in your hand to the value.
11. If the value was modified by the rules more than 8 times so far then add the number of 1's bits in the binary representation of the value to the value.
12. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
13. If there is at least one card of rank 2 in your hand then add the product of all distinct superfactors of the value to the value. A superfactor divides the value evenly and it is the highest power of a prime factor of the value.
14. If the value is 674 you win!

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. If there are exactly 4 cards of rank 7 in your hand then subtract 11^2 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. If there is a straight in your hand then add five times the number of A's in your hand to the value.
9. If the value was modified by the rules more than 8 times so far then add the number of 1's bits in the binary representation of the value to the value.
10. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
11. If there is at least one card of rank 2 in your hand then add the product of all distinct superfactors of the value to the value. A superfactor divides the value evenly and it is the highest power of a prime factor of the value.

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. A: If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. B: If there are exactly 4 cards of rank 7 in your hand then subtract 11^2 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. $\neg A$: If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. $\neg B$: If there is a straight in your hand then add five times the number of A's in your hand to the value.
9. If the value was modified by the rules more than 8 times so far then add the number of 1's bits in the binary representation of the value to the value.
10. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
11. If there is at least one card of rank 2 in your hand then add the product of all distinct superfactors of the value to the value. A superfactor divides the value evenly and it is the highest power of a prime factor of the value.

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. If there are exactly 4 cards of rank 7 in your hand then subtract 11^2 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. If there is a straight in your hand then add five times the number of A's in your hand to the value.
9. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
10. If there is at least one card of rank 2 in your hand then add the product of all distinct superfactors of the value to the value. A superfactor divides the value evenly and it is the highest power of a prime factor of the value.

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. If there are exactly 4 cards of rank 7 in your hand then subtract 121 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. If there is a straight ending in A, add five to the value.
9. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
10. If there is at least one card of rank 2 in your hand then multiply value by 2.

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. If there are exactly 4 cards of rank 7 in your hand then subtract 121 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. If there is a straight ending in A, add five to the value.
9. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
10. If there is at least one card of rank 2 in your hand then multiply value by 2.

Simple implementation . . .

Rulette

1. Add 1 to the value. Also add to the value the multiple of the number of J's in your hand and the score of the first card in your hand.
2. Multiply the value by 2.
3. If you have at least one card of each suit in your hand then multiply the value by 2.
4. Add the absolute difference between red and black suites to the value.
5. If there are exactly 4 cards of rank 7 in your hand then subtract 121 from the value.
6. Add the score of the lowest score card in your hand to the value.
7. If there are at least 3 cards of Diamond suit in your hand then add 1 to the value and swap ranks of all 6's to 9's, all 9's to 6's, all 2's to 5's, and all 5's to 2's in your hand at once.
8. If there is a straight ending in A, add five to the value.
9. If there is at least one card of rank 2 in your hand then apply once again the last rule which changed the score (after that continue with rule **??**).
10. If there is at least one card of rank 2 in your hand then multiply value by 2.

Simple implementation . . . with high probability of a mistake

Escalators

Escalators

- Split the problem into 20 instances by bits.
- In i-th iteration consider only nodes which have 1 on the i-th bit.
- Find sizes of components of ones (DFS).
- Each component adds $\binom{size}{2} \cdot 2^i$.

Escalators

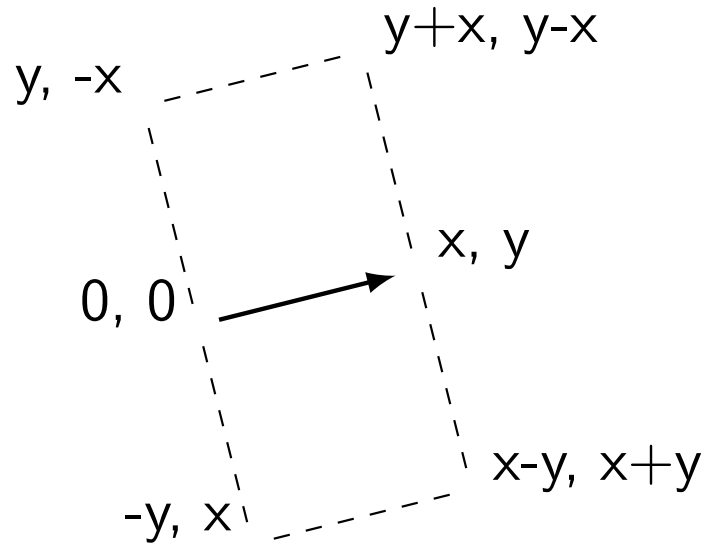
- Split the problem into 20 instances by bits.
- In i-th iteration consider only nodes which have 1 on the i-th bit.
- Find sizes of components of ones (DFS).
- Each component adds $\binom{size}{2} \cdot 2^i$.

Complexity $O(N \log W)$

Moving Furniture

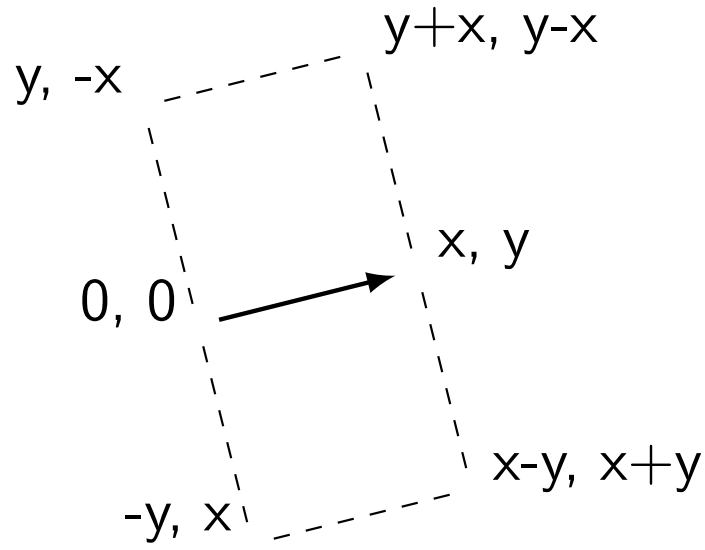
Moving Furniture

- Basic operation: complete square from 2 given points



Moving Furniture

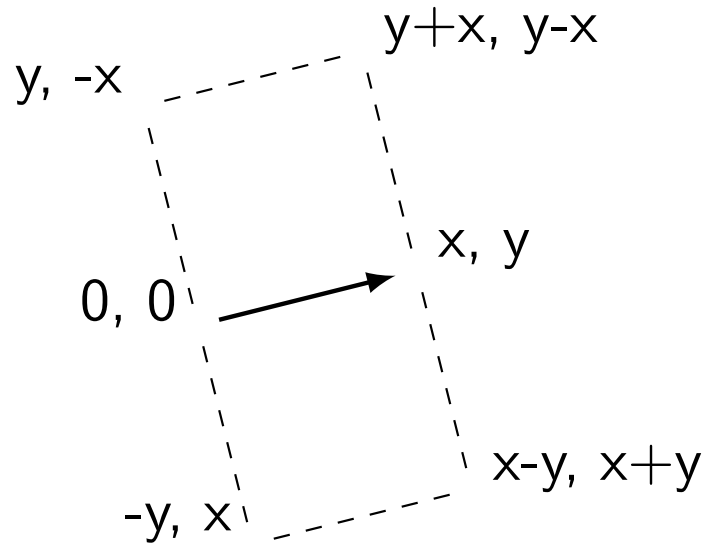
- Basic operation: complete square from 2 given points



- Repeat
  - Find lowest leftmost point
  - Try to create square with each of the remaining points
  - Choose the smallest square
  - Erase points of the chosen square

Moving Furniture

- Basic operation: complete square from 2 given points



- Repeat
  - Find lowest leftmost point
  - Try to create square with each of the remaining points
  - Choose the smallest square
  - Erase points of the chosen square

Complexity $O(N^2 \log N)$

Locker Room

Locker Room

- Create Suffix array over $s + s$.

Locker Room

- Create Suffix array over $s + s$.
- Binary search by answer
  - Scratch out every segment with lesser rank.
  - Scratching can be done with $O(N)$ sweep.
  - Total complexity $O(N \log N)$.

Locker Room

- Create Suffix array over $s + s$.
- Binary search by answer
  - Scratch out every segment with lesser rank.
  - Scratching can be done with $O(N)$ sweep.
  - Total complexity $O(N \log N)$.
- Keep track of non-scratched segments
  - Traverse SA and remove scratched segments.
  - Segments in Set: $O(\log N)$
  - Total complexity $O(N \log N)$.

Locker Room

- Create Suffix array over $s + s$.
- Binary search by answer
  - Scratch out every segment with lesser rank.
  - Scratching can be done with $O(N)$ sweep.
  - Total complexity $O(N \log N)$.
- Keep track of non-scratched segments
  - Traverse SA and remove scratched segments.
  - Segments in Set: $O(\log N)$
  - Total complexity $O(N \log N)$.
- Note that complexity depends on SA construction. $O(N \log^2 N)$ suffices (if implemented reasonably).

Numbers Generator

Numbers Generator

- Keep track of current state with Finite automaton.

Numbers Generator

- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.

Numbers Generator

- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.
- We create matrix A with probabilities of transitions.
- We can achieve this by giving each edge in Aho-Corasick $\frac{1}{2}$ probability of traversal, and creating edges with probability 1 from all final states to one common final state.

Numbers Generator

- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.
- We create matrix A with probabilities of transitions.
- We can achieve this by giving each edge in Aho-Corasick $\frac{1}{2}$ probability of traversal, and creating edges with probability 1 from all final states to one common final state.
- Now find the expected number of steps before ending in absorbing state.

Numbers Generator

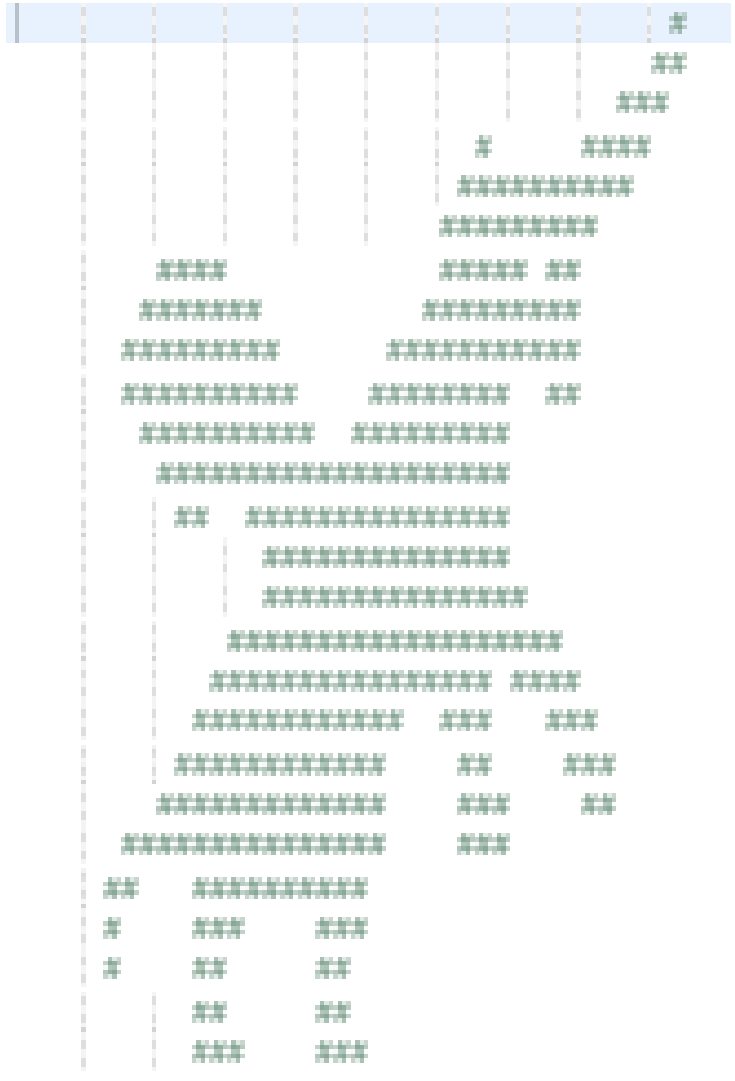
- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.
- We create matrix A with probabilities of transitions.
- We can achieve this by giving each edge in Aho-Corasick $\frac{1}{2}$ probability of traversal, and creating edges with probability 1 from all final states to one common final state.
- Now find the expected number of steps before ending in absorbing state.
- After solving $(E - A)x = 1$ the answer can be found in x_0.

Numbers Generator

- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.
- We create matrix A with probabilities of transitions.
- We can achieve this by giving each edge in Aho-Corasick $\frac{1}{2}$ probability of traversal, and creating edges with probability 1 from all final states to one common final state.
- Now find the expected number of steps before ending in absorbing state.
- After solving $(E - A)x = 1$ the answer can be found in x_0.
- To solve this equation we can use Gaussian Elimination.

Numbers Generator

- Keep track of current state with Finite automaton.
- We can use Aho-Corasick algorithm to achieve this.
- We create matrix A with probabilities of transitions.
- We can achieve this by giving each edge in Aho-Corasick $\frac{1}{2}$ probability of traversal, and creating edges with probability 1 from all final states to one common final state.
- Now find the expected number of steps before ending in absorbing state.
- After solving $(E - A)x = 1$ the answer can be found in x_0.
- To solve this equation we can use Gaussian Elimination.
- Complexity $O((W\,B)^3)$



Thank you for your attention!