ICPC North American 2019

Great NY Regional Contest

A

题意

> 水题，读入一个七位数字，判断前三位数字是否为 **555** ， 是则输出**YES**，否则输出**NO**
> 显然 **N/10000** 即为前三位数字

代码

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    cout << (((n/10000) == 555) ? "YES" : "NO") << endl;
    return 0;
}
```

B

题意：

以数独规则完成以下任务：

给定两个方格共同边，每条边邻接的两个格子的值a,b
(a+b)%min(a,b) 等于边上给出的数值，以此完成数独

大体思路:

把每一组位置能用的数字组合记录，用二进制状压，记录优化 。先用约束缩小搜索范围，然后改变搜索的顺序

搜索用二进制优化，极其考验码力。

```
#include <bits/stdc++.h>

typedef unsigned short WORD;
typedef unsigned char BYTE;
```

```
char inbuf[256];

int initCnt;
int constraints[15][9];
WORD valid_masks[10] = {0, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x100};
#define ALL_MASK    0x1ff

int dbcnt = 0;

BYTE bit_cnts[32] = {
    0, 1, 1, 2, 1, 2, 2, 3,
    1, 2, 2, 3, 2, 3, 3, 4,
    1, 2, 2, 3, 2, 3, 3, 4,
    2, 3, 3, 4, 3, 4, 4, 5
};

// number of bits on (in low 10 bits) in mask
int get_bitcnt(WORD mask)
{
    return (bit_cnts[mask & 0x1f] + bit_cnts[(mask >> 5) & 0x1f]);
}

// return first on bit in mask (in use should be the only one)
int get_bit_index(WORD mask)
{
    int i;
    WORD test = 0x01;
    for(i = 1; i <= 9 ; i++, test <<= 1) {
        if(mask & test) {
            return i;
        }
    }
    return 0;
}

/*
 * depth first search stack struct for sudoku
 */
typedef struct _search_state_
{
    WORD avail_mask[9][9];  // which values (valid_masks) can still be used in
thsi box
    BYTE row_avail_counts[9][9];    // for each row, counts of how many boxes in
the row have each value available
    BYTE col_avail_counts[9][9];    // for each col, counts of how many boxes in
the col have each value available
    BYTE box_avail_counts[3][3][9]; // for each 3x3 box, counts of how many
boxes in the 3x3 box have each value available
    BYTE val_set[9][9]; // if non-zero, this box is set to the current value in
the current search
} SEARCH_STATE;

SEARCH_STATE states[81];

/*
 * intialize the search stack to no values chosen and all values available
 * in all boxes
```

```
 */
void search_init()
{
    SEARCH_STATE *pss = &(states[0]);
    int i, j, k;
    for(i = 0; i < 9; i++) {
        for(j = 0; j < 9 ; j++) {
            pss->avail_mask[i][j] = ALL_MASK;
            pss->val_set[i][j] = 0;
            pss->row_avail_counts[i][j] = 9;
            pss->col_avail_counts[i][j] = 9;
        }
    }
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3 ; j++) {
            for(k = 0; k < 9 ; k++){
                pss->box_avail_counts[i][j][k] = 9;
            }
        }
    }
}

// read constraints from stdin
int scan_constraints()
{
    int i, j;
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            if(fgets(&(inbuf[0]), 255, stdin) == NULL)
            {
                fprintf(stderr, "Read of first line of region %d block %d
failed\n", j, i);
                return -21;
            }
            if(sscanf(&(inbuf[0]), "%d %d %d %d %d %d",
                &(constraints[5*i+2*j][0]), &(constraints[5*i+2*j][1]), &
(constraints[5*i+2*j][2]),
                &(constraints[5*i+2*j][3]), &(constraints[5*i+2*j][4]), &
(constraints[5*i+2*j][5])) != 6) {
                fprintf(stderr, "Scan of first line of region %d block %d
failed\n", j, i);
                return -22;
            }
            if(j < 2) {
                if(fgets(&(inbuf[0]), 255, stdin) == NULL)
                {
                    fprintf(stderr, "Read of second line of region %d block %d
failed\n", j, i);
                    return -23;
                }
                if(sscanf(&(inbuf[0]), "%d %d %d %d %d %d %d %d %d",
                    &(constraints[5*i+2*j+1][0]), &(constraints[5*i+2*j+1][1]),
&(constraints[5*i+2*j+1][2]),
                    &(constraints[5*i+2*j+1][3]), &(constraints[5*i+2*j+1][4]),
&(constraints[5*i+2*j+1][5]),
                    &(constraints[5*i+2*j+1][6]), &(constraints[5*i+2*j+1][7]),
&(constraints[5*i+2*j+1][8])) != 9) {
```

```
                    fprintf(stderr, "Scan of second line of region %d block %d
failed\n", j, i);
                    return -24;
                }
            }
        }
    }
    return 0;
}

int MaskInit(SEARCH_STATE *pss)
{
    int i, con, row, col, baseConsRow, baseConsCol, con_cnts[10], ncons, mincon,
maxconcnt;
    WORD remMask, OKmask;
    for(row = 0, baseConsRow = 0; row < 9 ; row++) {
        for(col = 0, baseConsCol = 0; col < 9 ; col++) {
            for(i = 0; i < 10 ; i++) con_cnts[i] = 0;
            ncons = maxconcnt = 0;
            mincon = 9;
            if((row == 0) && (col == 2)) {
                dbcnt++;
            }
            remMask = 0;
            if((col %3) != 0) { // if not first col in 3x3 block, check
constraint with box to left
                con = constraints[baseConsRow][baseConsCol-1];
                con_cnts[con]++;
                ncons++;
                if(mincon > con) mincon = con;
                if((con < 9) && (con_cnts[con] > maxconcnt)) maxconcnt =
con_cnts[con];
            }
            if((col %3) != 2) { // if not last col in 3x3 block, check
constraint with box to right
                con = constraints[baseConsRow][baseConsCol];
                con_cnts[con]++;
                ncons++;
                if(mincon > con) mincon = con;
                if((con < 9) && (con_cnts[con] > maxconcnt)) maxconcnt =
con_cnts[con];
            }
            if((row %3) != 0) { // if not top row in 3x3 block, check constraint
with box above
                con = constraints[baseConsRow-1][col];
                con_cnts[con]++;
                ncons++;
                if(mincon > con) mincon = con;
                if((con < 9) && (con_cnts[con] > maxconcnt)) maxconcnt =
con_cnts[con];
            }
            if((row %3) != 2) { // if not bottom row in 3x3 block, check
constraint with box below
                con = constraints[baseConsRow+1][col];
                con_cnts[con]++;
                ncons++;
                if(mincon > con) mincon = con;
```

```
                if((con < 9) && (con_cnts[con] > maxconcnt)) maxconcnt =
con_cnts[con];
            }
            if(mincon != 9) { // constraints
                if(con_cnts[4] > 0) {   // one si 5 and the other 9
                    OKmask = valid_masks[5] | valid_masks[9];
                    remMask |= ~OKmask;
                }
                if(con_cnts[3] > 0) { // either 4 & 7 or 5 & 8 or 6 & 9
                    remMask |= valid_masks[1] | valid_masks[2] | valid_masks[3];
                }
                if(con_cnts[2] > 0) { // cannot be 1 or 2
                    remMask |= valid_masks[1] | valid_masks[2];
                }
                if(con_cnts[1] > 0) { // cannot be 1
                    remMask |= valid_masks[1];
                }
                if(remMask != 0) {  // adjust counts for boxes in same row, col
or 3x3 box
                    pss->avail_mask[row][col] &= ~remMask;
                    for(i = 0; i < 9 ; i++) {
                        if(valid_masks[i] & remMask) {
                            pss->col_avail_counts[col][i-1]--;
                            pss->row_avail_counts[row][i-1]--;
                            pss->box_avail_counts[row/3][col/3][i-1]--;
                        }
                    }
                }
            }
            if((col %3) != 2) { // advance to next row constraint if not last
col in 3x3 box
                baseConsCol++;
            }
        }
        if((row % 3) != 2) {    // if not last row of 3x3 box used 2 rows of
constrsints, else 1
            baseConsRow += 2;
        } else {
            baseConsRow++;
        }
    }
    return 0;
}

WORD checkConstraint(int constraint, WORD baseMask, WORD chkMask)
{
    WORD OKmask, result;
    int i, j;
    OKmask = 0;
    if(constraint == 9) { // no restricition
        return 0;
    }
    // if value in chkMask is the larger, values in base can be in range
    // (chkval/(constraint+1)) + 1. to chkval/constraint (could be empty set)
    for(i = constraint+1; i <= 9 ; i++) {
        if(chkMask & valid_masks[i]) {
            for(j = constraint+1; j <= 9; j++) {
```

```
                if((j != i) && (((j%i) == constraint) || ((i % j) ==
constraint))) OKmask |= valid_masks[j];
            }
        }
    }
    result = baseMask & ~OKmask;
    return result;
}

int check_constraints(SEARCH_STATE *pss)
{
    int i, row, col, baseConsRow, baseConsCol, scan_count, change_count = 1;
    WORD baseMask, chkMask, resultMask, totResult;
    scan_count = 0;
    // sacn all constraints until no more changes
    while(change_count > 0) {
        scan_count++;
        change_count = 0;
        for(row = 0, baseConsRow = 0; row < 9 ; row++) {
            for(col = 0, baseConsCol = 0; col < 9 ; col++) {
                if((row == 2) && (col == 8)) {
                    dbcnt++;
                }
                if(pss->val_set[row][col] == 0) {   // if we have not already
set this box in search see if it can change
                    baseMask = pss->avail_mask[row][col];
                    totResult = 0;  // all changes to baseMask
                    if((col %3) != 0) { // if not first col in 3x3 block, check
constraint with box to left
                        chkMask = pss->avail_mask[row][col-1];
                        resultMask = checkConstraint(constraints[baseConsRow]
[baseConsCol-1], baseMask, chkMask);
                        if(resultMask != 0) {
                            baseMask &= ~resultMask;
                            change_count++;
                            totResult |= resultMask;
                        }
                    }
                    if((col %3) != 2) { // if not last col in 3x3 block, check
constraint with box to right
                        chkMask = pss->avail_mask[row][col+1];
                        resultMask = checkConstraint(constraints[baseConsRow]
[baseConsCol], baseMask, chkMask);
                        if(resultMask != 0) {
                            baseMask &= ~resultMask;
                            change_count++;
                            totResult |= resultMask;
                        }
                    }
                    if((row %3) != 0) { // if not top row in 3x3 block, check
constraint with box above
                        chkMask = pss->avail_mask[row-1][col];
                        resultMask = checkConstraint(constraints[baseConsRow-1]
[col], baseMask, chkMask);
                        if(resultMask != 0) {
                            baseMask &= ~resultMask;
                            change_count++;
                            totResult |= resultMask;
```

```
                        }
                    }
                    if((row %3) != 2) { // if not bottom row in 3x3 block, check
constraint with box below
                        chkMask = pss->avail_mask[row+1][col];
                        resultMask = checkConstraint(constraints[baseConsRow+1]
[col], baseMask, chkMask);
                        if(resultMask != 0) {
                            baseMask &= ~resultMask;
                            change_count++;
                            totResult |= resultMask;
                        }
                    }
                    if(baseMask == 0) { // not already set and no values
available for box

                        return -1;
                    }
                    pss->avail_mask[row][col] = baseMask;
                    if(totResult != 0) {    // adjust counts for boxes in same
row, col or 3x3 box
                        for(i = 0; i < 9 ; i++) {
                            if(valid_masks[i] & totResult) {
                                pss->col_avail_counts[col][i-1]--;
                                pss->row_avail_counts[row][i-1]--;
                                pss->box_avail_counts[row/3][col/3][i-1]--;
                            }
                        }
                    }
                }
                if((col %3) != 2) { // advance to next row constraint if not
last col in 3x3 box
                    baseConsCol++;
                }
            }
            if((row % 3) != 2) {    // if not last row of 3x3 box used 2 rows of
constrsints, else 1
                baseConsRow += 2;
            } else {
                baseConsRow++;
            }
        }
    }
    return 0;
}

#define STYP_ROW    1   // depth first search on boxes with a value available in
a given row
#define STYP_COL    2   // depth first search on boxes with a value available in
a given col
#define STYP_BOX    3   // depth first search on boxes with a value available in
a given 3x3 box
#define STYP_VAL    4   // depth first search on values available in a given box
(row & col)
typedef struct _solve_data_ {
    int solve_type;
    int solve_val;
    int solve_row;
```

```
    int solve_col;
    int solve_cnt;
    int solve_index;
    int test_row;
    int test_col;
} SOLVE_DATA;

int GetSolveStep(SEARCH_STATE *pss, SOLVE_DATA *psd)
{
    int i, j, k;
    psd->solve_cnt = 10;
    for(i = 0; i < 9; i++) {
        for(j = 0; j < 9; j++) {
            if(pss->val_set[i][j] == 0) {
                k = get_bitcnt(pss->avail_mask[i][j]);
                if(k < psd->solve_cnt) {
                    psd->solve_cnt = k;
                    psd->solve_type = STYP_VAL;
                    psd->solve_row = i;
                    psd->solve_col = j;
                    psd->solve_val = get_bit_index(pss->avail_mask[i][j]);
                }
            }
        }
    }
    for(i = 0; i < 9; i++) {
        for(j = 0; j < 9; j++) {
            if(pss->row_avail_counts[i][j] < psd->solve_cnt) {
                psd->solve_cnt = pss->row_avail_counts[i][j];
                psd->solve_type = STYP_ROW;
                psd->solve_row = i;
                psd->solve_val = j+1;
            }
        }
    }
    for(i = 0; i < 9; i++) {
        for(j = 0; j < 9; j++) {
            if(pss->col_avail_counts[i][j] < psd->solve_cnt) {
                psd->solve_cnt = pss->col_avail_counts[i][j];
                psd->solve_type = STYP_COL;
                psd->solve_col = i;
                psd->solve_val = j+1;
            }
        }
    }
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            for(k = 0; k < 9 ; k++) {
                if(pss->box_avail_counts[i][j][k] < psd->solve_cnt) {
                    psd->solve_cnt = pss->box_avail_counts[i][j][k];
                    psd->solve_type = STYP_BOX;
                    psd->solve_row = i;
                    psd->solve_col = j;
                    psd->solve_val = k+1;
                }
            }
        }
    }
```

```
    if(psd->solve_cnt == 0) {   // if some value has no choices left, fail
        return -1;
    } else {
        return 0;
    }
}

int FindNextTest(SEARCH_STATE *pss, SOLVE_DATA *psd)
{
    int i, j, starti, startj;
    WORD mask = valid_masks[psd->solve_val];
    if(psd->solve_index >= psd->solve_cnt) {
        return -1;
    }
    switch(psd->solve_type) {
    case STYP_VAL:
        if(psd->solve_index == 0) {
            j = 0;
        } else {
            j = psd->solve_val+1;
        }
        psd->test_col = psd->solve_col;
        psd->test_row = psd->solve_row;
        mask = pss->avail_mask[psd->solve_row][psd->solve_col];
        for(i = j ; i < 10 ; i++) {
            if(mask & valid_masks[i]) {
                psd->solve_val = i;
                psd->solve_index++;
                return 0;
            }
        }
        return -1;
    case STYP_ROW:
        if(psd->solve_index == 0) {
            startj = 0;
        } else {
            startj = psd->test_col+1;
        }
        i = psd->solve_row;
        for(j = startj ; j < 9 ; j++) {
            if(pss->avail_mask[i][j] & mask) {
                psd->test_col = j;
                psd->test_row = i;
                psd->solve_index++;
                return 0;
            }
        }
        return -1;
    case STYP_COL:
        if(psd->solve_index == 0) {
            starti = 0;
        } else {
            starti = psd->test_row+1;
        }
        j = psd->solve_col;
        for(i = starti ; i < 9 ; i++) {
            if(pss->avail_mask[i][j] & mask) {
                psd->test_col = j;
```

```
                psd->test_row = i;
                psd->solve_index++;
                return 0;
            }
        }
        return -1;
    case STYP_BOX:
        if(psd->solve_index == 0) {
            starti = 0;
            startj = 0;
        } else {
            starti = psd->test_row - 3*psd->solve_row;
            startj = psd->test_col+1 - 3*psd->solve_col;
        }
        for(i = starti; i < 3 ; i++) {
            for(j = startj ; j < 3; j++) {
                if(pss->avail_mask[i + 3*psd->solve_row][j + 3*psd->solve_col] &
mask) {
                    psd->test_col = j + 3*psd->solve_col;
                    psd->test_row = i + 3*psd->solve_row;
                    psd->solve_index++;
                    return 0;
                }
            }
        }
        return -1;
    default:
        fprintf(stderr, "bad solve type %d\n", psd->solve_type);
        return -1;
    }
}

int ApplyChoice(SEARCH_STATE *pss, int row, int col, int val)
{
    int i, j, boxr, boxc;
    WORD mask = valid_masks[val];
    if(pss->val_set[row][col] != 0) {
        fprintf(stderr, "ApplyChoice: row %d col %d val %d already set to %d\n",
row, col, val, pss->val_set[row][col]);
        return -1;
    }
    pss->val_set[row][col] = val;
    // remove val from other avails in row, col and box
    boxr = row/3;
    boxc = col/3;
    for(j = 0; j < 9 ; j++) {
        if(pss->avail_mask[row][j] & mask) {    // reduce counts
            pss->box_avail_counts[boxr][j/3][val-1]--;
            pss->col_avail_counts[j][val-1]--;
        }
        pss->avail_mask[row][j] &= ~mask;
    }
    for(i = 0; i < 9 ; i++) {
        if(pss->avail_mask[i][col] & mask) {
            pss->box_avail_counts[i/3][boxc][val-1]--;
            pss->row_avail_counts[i][val-1]--;
        }
        pss->avail_mask[i][col] &= ~mask;
```

```
    }
    boxr = row/3;
    boxc = col/3;
    for(i = 3*boxr; i < 3*(boxr+1); i++) {
        for(j = 3*boxc; j < 3*(boxc+1); j++) {
            if(pss->avail_mask[i][j] & mask) {
                pss->col_avail_counts[j][val-1]--;
                pss->row_avail_counts[i][val-1]--;
            }
            pss->avail_mask[i][j] &= ~mask;
        }
    }
    //for each other value at row/col, decrement ist counts
    for(i = 1; i <= 9 ; i++) {
        if((i != val) && ((pss->avail_mask[row][col] & valid_masks[i]) != 0)){
            pss->box_avail_counts[row/3][col/3][i-1]--;
            pss->col_avail_counts[col][i-1]--;
            pss->row_avail_counts[row][i-1]--;
        }
    }
    pss->avail_mask[row][col] = mask;   // no one can use it but we need to tell
neighbors that other vals not avail
    pss->row_avail_counts[row][val-1] = 32; // never choose the row and val
again
    pss->col_avail_counts[col][val-1] = 32; // never choose the col and val
again
    pss->box_avail_counts[boxr][boxc][val-1]= 32;   // never choose the box and
val again
    return 0;
}

int solnCnt = 0;

void PrintSoln(SEARCH_STATE *pss)
{
    int i, j;
    printf("%d\n", solnCnt);
    for(i = 0; i < 9 ; i++) {
        for(j = 0; j < 9 ; j++) {
            printf("%d ", pss->val_set[i][j]);
        }
        printf("\n");
    }
}

int Solve(int level)
{
    SEARCH_STATE *pssnxt, *pss = &(states[level]);
    SOLVE_DATA sd;
    int i, j;
    if(GetSolveStep(pss, &sd) != 0) {   // find row, col or 3x3 box) + value to
scan
        return -1;
    }
    sd.solve_index = 0;
    while(FindNextTest(pss, &sd) == 0) {    // for each candidate in chosen row,
col or 3x3 box, get row& col to set
```

```
        if(level == 80 - initCnt) { // if this is the last box to fill in we are
done, set it and return 0 (success)
            pss->val_set[sd.test_row][sd.test_col] = sd.solve_val;
#ifdef FIND_ALL
            solnCnt++;
            PrintSoln(pss);
            return -1;
#else
            return 0;
#endif
        } else {    // else copy current to next and try each possiblility
            pssnxt = &(states[level+1]);
            *pssnxt = *pss;
            if(ApplyChoice(pssnxt, sd.test_row, sd.test_col, sd.solve_val) == 0)
{ // set this value choice
                if(check_constraints(pssnxt) == 0) {    // if not killed by
constraints
                    if(Solve(level+1) == 0) {   // call solve recursively, if
success, copy val_set
                        for(i = 0; i < 9; i++) {
                            for(j = 0; j < 9 ; j++) {
                                pss->val_set[i][j] = pssnxt->val_set[i][j];
                            }
                        }
                        return 0;
                    }
                }
            }
        }
    }
    return -1;
}

int main()
{
    int ret, i, row, col, val;
#ifndef FIND_ALL
    int j;
#endif
#ifdef ONE_SHOT
    int nprob, curprob, index;
    if(fgets(&(inbuf[0]), 255, stdin) == NULL)
    {
        fprintf(stderr, "Read failed on problem count\n");
        return -1;
    }
    if(sscanf(&(inbuf[0]), "%d", &nprob) != 1)
    {
        fprintf(stderr, "Scan failed on problem count\n");
        return -2;
    }
    for(curprob = 1; curprob <= nprob ; curprob++)
    {
        if(fgets(&(inbuf[0]), 255, stdin) == NULL)
        {
            fprintf(stderr, "Read failed on problem %d header\n", curprob);
            return -3;
        }
```

```
        // get prob num degree
        if(sscanf(&(inbuf[0]), "%d", &index) != 1)
        {
            fprintf(stderr, "scan failed on problem header problem index %d\n",
                curprob);
            return -6;
        }
        if(index != curprob)
        {
            fprintf(stderr, "problem index %d not = expected problem %d\n",
                index, curprob);
            return -7;
        }
#endif
        // get num init vals
        if(fgets(&(inbuf[0]), 255, stdin) == NULL)
        {
            fprintf(stderr, "Read failed on initCnt\n");
            return -6;
        }
        if(sscanf(&(inbuf[0]), "%d", &initCnt) != 1)
        {
            fprintf(stderr, "scan failed on initCnt\n");
            return -7;
        }
        if((initCnt < 0) || (initCnt > 81)) {
            fprintf(stderr, "init val cnt %d not in range 0 .. 81\n",
                initCnt);
            return -8;
        }
        search_init();  // init first atate
        if((ret = scan_constraints()) != 0) {   // read constriants
            return ret;
        }
        MaskInit(&(states[0]));
        if(check_constraints(&(states[0])) != 0) { // apply constraints to first
state
            return -8;
        }
        for(i = 0; i < initCnt ; i++) {
            if(fgets(&(inbuf[0]), 255, stdin) == NULL)
            {
                fprintf(stderr, "Read failed on init val %d\n", i+1);
                return -9;
            }
            // get prob num degree
            if(sscanf(&(inbuf[0]), "%d %d %d", &row, &col, &val) != 3)
            {
                fprintf(stderr, "scan failed on init val %d\n", i+1);
                return -10;
            }
            if((row < 1) || (row > 9) || (col < 1) || (col > 9) || (val < 0) ||
(val > 9)) {
                fprintf(stderr, "init val %d row %d col %d or val %d not in
range 1 .. 9\n",
                    i+1, row, col, val);
                return -11;
            }
```

```
            if(ApplyChoice(&(states[0]), row-1, col-1, val) != 0) {
                fprintf(stderr, "init val %d error adding row %d col %d or val
%d\n",
                    i+1, row, col, val);
                return -12;
            }
            if(check_constraints(&(states[0])) != 0) { // apply constraints to
first state
                fprintf(stderr, "check constraints failed after init val %d\n",
                    i+1);
                return -13;
            }
        }
    return 0;
}
```

C

题意：

给定一张无向图，从任意一个节点开始，每次选中该节点本身的概率为1/(d+1) *d为节点的度* ，选中与其相邻的节点的概率皆为1/(d+1)，若选中非自身节点，重复上述过程，知道在某个节点选中这个节点本身。求每个节点被选中的概率。

注意：题目所求解的获胜情况并不是有限的。例如：在图一中，当发牌给1号时，若求解1号获胜的概率，则可以考虑以这些情况：1→1（结束），1→2→1→1（结束），1→2→3→2→1→1（结束），1→2→3→4→3→2→1→1（结束），1→2→3→4→5→4→3→2→1→1（结束）。

题解：

下面使用数学语言描述这一过程：记节点 V 的度为 $d(V)$。定义二元有序对

$$N=(V,pV)$$

其中 V 为节点， pV 为我们预先定义该节点的的初始概率。

定义任意二元有序对的函数

$$f(N)=\{(V',pV')|V'为V的所有邻接节点,pV'=\frac{pV}{d(V)+1}\}$$

先给出周游前的集合 $NS(0)=\{(Start,pStart=1)\}$。接着在每一部中依此方式求得 $NS(n)$，即：

$$NS(n)=\bigcup f(K),K\in NS(n-1)$$

其中，每次运算结束后，合并 $NS(n)$ 中所有节点相同的有序对，方法如下：

$$\bigcup\{(V,pV_i)\}=\{(V,\sum pV_i)\}$$

特殊地，若在某周游步骤中 $NS(n-1)$ 包含一节点为终点的有序对 $(End,pEnd)$，则在总胜率中加上 $\frac{pEnd}{d(End)+1}$ 以统计总胜率。在某次递增的胜率小于1e-7（实验得到）时终止计算，输出计算结果。

标程:

```
#include<bits/stdc++.h>
using namespace std;

bool graph[21][21] = { false };
int degree[21] = { 0 };
int N, P;

double ProbabilityWin(int start, int end) {
    double ret = 0.0, curp[21], nextp[21], roundp;
    int i, j;
    for (i = 0; i < 21; i++) curp[i] = nextp[i] = 0.0;
    roundp = 0.0;
    curp[start] = 1.0;
    do {
        ret += roundp;
        roundp = 0.0;
        for (i = 0; i <= N; i++) nextp[i] = 0.0;
        for (i = 1; i <= N; i++)
            if (curp[i] > 0.0)
                for (j = 1; j <= N; j++)
                    if (graph[i][j]) {
                        if (i == j) {
                            if (i == end) roundp += curp[i] / (double)degree[i];
                        }
                        else nextp[j] += curp[i] / (double)degree[i];
                    }
        for (i = 0; i < 21; i++) curp[i] = nextp[i];
    } while (roundp == 0.0 || roundp > 0.0000001);
    return ret;
}

int main() {
    int i, j, k, t, index, start, end;
    cin >> N >> P;
    for (i = 1; i <= N; i++) {
        cin >> j;
        for (k = 0; k < j; k++) {
            cin >> t;
            graph[i][t] = graph[t][i] = true;
        }
        graph[i][i] = true;
    }
    for (i = 1; i <= N; i++) {
        for (j = 1; j <= N; j++) {
            if (graph[i][j]) degree[i]++;
        }
    }
    for (i = 1; i <= P; i++) {
        cin >> index >> start >> end;
        printf("%d %.5lf\n", index, ProbabilityWin(start, end));
    }
    return 0;
}
```

D

题意

给定一个多边形，以多边形的质心为圆心作一个圆，其中圆的面积与多边形面积相同。

求：该圆与多边形重合面积与多边形面积的比值

分析

本题主要四个任务：

> 1.多边形的质心
>
> 2.多边形的面积
>
> 3.求圆与线段的交点
>
> 4.求圆与三角形重合面积

首先明确，对于多边形的问题，全部转换成三角形的问题来求。

定义

首先对于点、线段、圆、多边形、向量进行定义

```
struct point { //点
    double x, y;
    point() {}
    point(double _x,double _y) {
        x = _x, y = _y;
    }
};

struct circle {//圆
    point center;
    double r;
    circle(){}
    circle(point c,double _r){
        center = c;
        r = _r;
    }
};

struct segment { //线段
    point start, end;
    segment(){}
    segment(point a,point b) {
        start = a;
        end = b;
    }
};
```

```
typedef vector<point> polygon;//多边形

point vec(point a,point b) {//向量
    return point(a.x - b.x, a.y - b.y);
}
```

多边形的质心

三角形的质心：$(x,y)=(\frac{x_a+x_b+x_c}{3},\frac{y_a+y_b+y_c}{3})$

多边形的质心：将多边形分割为若干个三角形，多边形的质心为三角形的质心以三角形面积为权值求出的加权平均值（有向）

则$(x,y)=(\frac{\sum(x_0+x_i+x_{i+1})*S_{PQ_iQ_{i+1}}}{3\sum SQ_iQ_{i+1}},\frac{\sum(y_0+y_i+y_{i+1})*S_{PQ_iQ_{i+1}}}{3\sum SQ_iQ_{i+1}})$

代码如下：

```
point CentroidOfPolygon(polygon p) { //多边形的质心
    double x = 0, y = 0, m = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for (int i = 0; i < sz-1;i++) {
        double pm = crossmultiply(p[i], p[i + 1]) * 0.5;
        x += (p[i].x + p[i + 1].x) * pm;
        y += (p[i].y + p[i + 1].y) * pm;
        m += pm;
    }
    p.pop_back();
    return point((x / m) / 3.0, (y / m) / 3.0);
}
```

多边形的面积

求多边形的面积，同样是对多边形进行三角分割

将多边形拆分成多个以(0,0)点为其中一个顶点的三角形，将三角形的有向面积叠加即为多边形面积

$S_{\Delta_{OAB}}=\frac{\vec{OA}\times\vec{OB}}{2}$

则代码如下：

```
double AreaOfPolygon(polygon p) { //多边形的面积
    double ret = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for (int i = 0; i < sz - 1; i++) {
        ret += crossmultiply(p[i], p[i + 1]) * 0.5;
    }
    p.pop_back();
    return abs(ret);
}
```

圆与线段的交点

圆与线段的交点考虑两种情况：

斜率不存在：

$x = C(C为常数)$

斜率存在：

将圆与直线方程联立求解

> 直线方程组 $y = kx + b$

代入线段起点坐标(s_x, s_y)

则 $y - s_y = k(x - s_x)$

$b = s_y - ks_x$

> 圆:$(x - c.x)^2 + (y - c.y)^2 = r^2$

将直线方程代入

$(1 + k^2)x^2 + (2 * k * (b - c.y) - 2 * c.x)x = r * r - c.x * c.x - (b - c.y)^2$

> $Ax^2 + Bx = C$

$\Delta = B * B + 4 * A * C$

$x_1 = \sqrt{\frac{C}{A} + \frac{B^2}{4*A^2}} - \frac{B}{2*A}$

$x_2 = -\sqrt{\frac{C}{A} + \frac{B^2}{4*A^2}} - \frac{B}{2*A}$

$A = 1 + k^2$

$B = (2 * k * (b - c.y) - 2 * c.x)$

$C = r * r - c.x * c.x - (b - c.y)^2$

最后判断求出的点是否在给定线段上即可

代码如下：

```
vector<point> intersection(segment l,circle c) {//线段与圆的交点
    vector<point> ret;
    point s = l.start, e = l.end, cn = c.center;
    double r = c.r;
    if(s.x==e.x) { //斜率不存在
        double dx = abs(cn.x - s.x);
        if(dx<r){
            double dy = sqrt(r * r - dx * dx);
            if(cn.x-r<s.x||cn.x+r>s.x){
                if(s.y>cn.y||e.y>cn.y) {
                    ret.push_back(point(s.x, cn.y + dy));
                }
                if(s.y<cn.y||e.y<cn.y) {
```

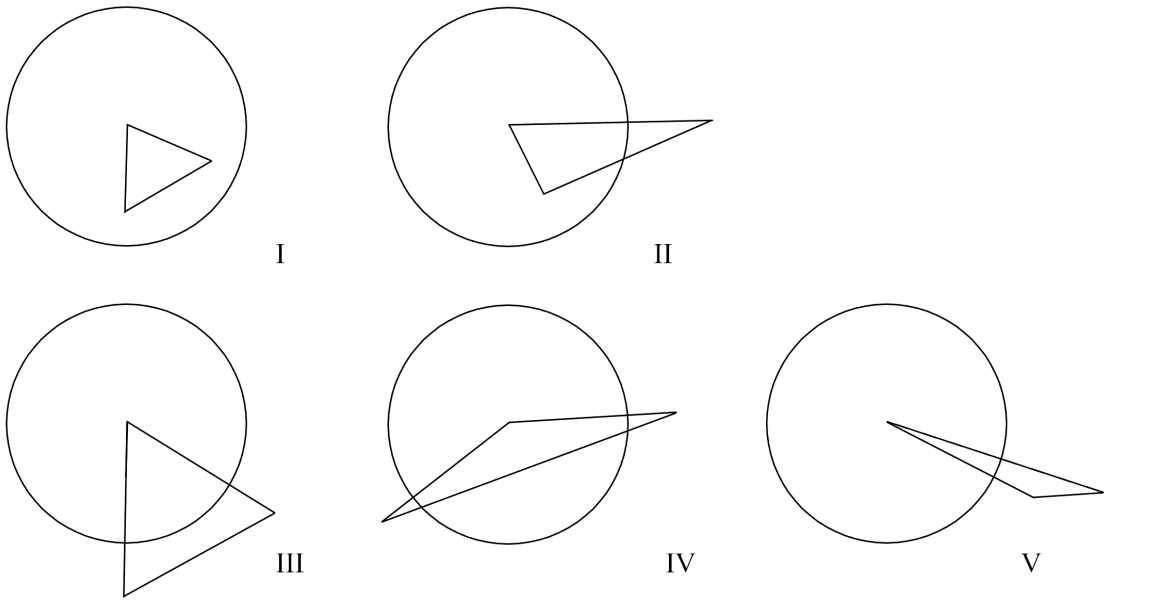
```
                    ret.push_back(point(s.x, cn.y - dy));
                }
            }
        }
    }
    else { //斜率存在
        double k = (s.y - e.y) / (s.x - e.x);
        double A = 1 + k * k;
        double b = s.y - k * s.x;
        double d = b - cn.y;
        double B = 2 * k * d - 2 * cn.x;
        double C = r * r - cn.x * cn.x - d * d;
        double delta = B * B + 4 * A * C;
        if(delta>0) {
            double x = sqrt(C / A + (B * B) / (4 * A * A)) - B / (2 * A);
            double y = k * x + b;
            double sx = s.x;
            double ex = e.x;
            if(sx>ex)
                swap(sx, ex);
            if(cn.x<x&&x<ex&&x>sx) {
                ret.push_back(point(x, y));
            }
            x = -sqrt(C / A + (B * B) / (4 * A * A)) - B / (2 * A);
            y = k * x + b;
            if(sx<x&&x<cn.x&&x<ex){
                ret.push_back(point(x, y));
            }
        }
    }
    return ret;
}
```

圆与三角形重合面积

分为四种情况

借用网上的图

其中第3种与第5种合并



代码如下：

```
double Area(segment l,circle c) { //圆与三角形重叠的面积
    double sign, area;
    sign = crossmultiply(vec(l.start, c.center), vec(l.end, c.center)) > 0 ? 1.0
: -1.0;//保持方向
    if(dis(l.start,c.center)>dis(l.end,c.center)) {//保证起点与原点中心的边为短边，减
少讨论
        point t = l.start;
        l.start = l.end;
        l.end = t;
    }
    //情况1:
    if(dis(l.end,c.center)<=c.r) {
        area = AreaOfTriangle(l.start, l.end, c.center);
    }
    //情况2:
    else if(dis(l.start,c.center)<=c.r) {
        point a = intersection(segment(c.center,l.end), c)[0];
        point b = intersection(l, c)[0];
        area = AreaOfSector(c, a, b) + AreaOfTriangle(c.center, l.start, b);
    }
    //情况3,5;
    else if(intersection(l,c).size()==0) {
        point a = intersection(segment(c.center, l.start), c)[0];
        point b = intersection(segment(c.center, l.end), c)[0];
        area = AreaOfSector(c, a, b);
    }
    //情况4:
    else {
        point a = intersection(segment(c.center, l.start), c)[0];
        point b = intersection(segment(c.center, l.end), c)[0];
        point x = intersection(l, c)[0];
        point y = intersection(l, c)[1];
```

```
        area = AreaOfSector(c, a, b) - AreaOfSector(c, x, y) +
AreaOfTriangle(c.center, x, y);
    }
    return sign * area;
}
```

Solve函数

将多边形分为若干个三角形，求出这些三角形与圆重叠的有向面积，叠加得到多边形与圆叠加的总面积

代码如下：

```
double Solve(polygon p,circle c) {
    double area = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for(int i=0;i<sz-1;i++) {
        area += Area(segment(p[i], p[i + 1]), c);
    }
    p.pop_back();
    return abs(area);
}
```

最终代码：

```
#include<bits/stdc++.h>
using namespace std;
const double pi = acos(-1.0);

struct point { //点
    double x, y;
    point() {}
    point(double _x,double _y) {
        x = _x, y = _y;
    }
};

struct circle {//圆
    point center;
    double r;
    circle() {}
    circle(point c,double _r){
        center = c;
        r = _r;
    }
};

struct segment { //线段
    point start, end;
    segment() {}
```

```
    segment(point a,point b) {
        start = a;
        end = b;
    }
};

typedef vector<point> polygon;//多边形

point vec(point a,point b) {//向量
    return point(a.x - b.x, a.y - b.y);
}

double crossmultiply(point a,point b) {//向量叉乘
    return (a.x * b.y- a.y * b.x);
}

double dis(point a,point b) { //两点间距离
    double x = a.x - b.x;
    double y = a.y - b.y;
    return sqrt(x * x + y * y);
}

double AreaOfPolygon(polygon p) { //多边形的面积
    double ret = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for (int i = 0; i < sz - 1; i++) {
        ret += crossmultiply(p[i], p[i + 1]) * 0.5;
    }
    p.pop_back();
    return abs(ret);
}

point CentroidOfPolygon(polygon p) { //多边形的质心
    double x = 0, y = 0, m = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for (int i = 0; i < sz-1;i++) {
        double pm = crossmultiply(p[i], p[i + 1]) * 0.5;
        x += (p[i].x + p[i + 1].x) * pm;
        y += (p[i].y + p[i + 1].y) * pm;
        m += pm;
    }
    p.pop_back();
    return point((x / m) / 3.0, (y / m) / 3.0);
}

double AreaOfTriangle(point a,point b,point c) { //三角形面积
    double ret = 0.0;
    point x = vec(a, b);
    point y = vec(a, c);
    ret = 0.5 * crossmultiply(x, y);
    return abs(ret);
}

double AreaOfSector(circle c,point a,point b) {//扇形面积
    double r = c.r;
    double x = dis(a, b);
```

```
    double d = (r * r * 2 - x * x) / (2 * r * r);
    d = acos(d);
    double area = 0.5 * d * r * r;
    return abs(area);
}

vector<point> intersection(segment l,circle c) {//线段与圆的交点
    vector<point> ret;
    point s = l.start, e = l.end, cn = c.center;
    double r = c.r;
    if(s.x==e.x) {
        double dx = abs(cn.x - s.x);
        if(dx<r){
            double dy = sqrt(r * r - dx * dx);
            if(cn.x-r<s.x||cn.x+r>s.x){
                if(s.y>cn.y||e.y>cn.y) {
                    ret.push_back(point(s.x, cn.y + dy));
                }
                if(s.y<cn.y||e.y<cn.y) {
                    ret.push_back(point(s.x, cn.y - dy));
                }
            }
        }
    }
    else {
        double k = (s.y - e.y) / (s.x - e.x);
        double A = 1 + k * k;
        double b = s.y - k * s.x;
        double d = b - cn.y;
        double B = 2 * k * d - 2 * cn.x;
        double C = r * r - cn.x * cn.x - d * d;
        double delta = B * B + 4 * A * C;
        if(delta>0) {
            double x = sqrt(C / A + (B * B) / (4 * A * A)) - B / (2 * A);
            double y = k * x + b;
            double sx = s.x;
            double ex = e.x;
            if(sx>ex)
                swap(sx, ex);
            if(cn.x<x&&x<ex&&x>sx) {
                ret.push_back(point(x, y));
            }
            x = -sqrt(C / A + (B * B) / (4 * A * A)) - B / (2 * A);
            y = k * x + b;
            if(sx<x&&x<cn.x&&x<ex){
                ret.push_back(point(x, y));
            }
        }
    }
    return ret;
}

double Area(segment l,circle c) { //圆与三角形重叠的面积
    double sign, area;
    sign = crossmultiply(vec(l.start, c.center), vec(l.end, c.center)) > 0 ? 1.0
: -1.0;//保持方向
    if(dis(l.start,c.center)>dis(l.end,c.center)) {
        point t = l.start;
```

```
        l.start = l.end;
        l.end = t;
    }
    //情况1:
    if(dis(l.end,c.center)<=c.r) {
        area = AreaOfTriangle(l.start, l.end, c.center);
    }
    //情况2:
    else if(dis(l.start,c.center)<=c.r) {
        point a = intersection(segment(c.center,l.end), c)[0];
        point b = intersection(l, c)[0];
        area = AreaOfSector(c, a, b) + AreaOfTriangle(c.center, l.start, b);
    }
    //情况3,5;
    else if(intersection(l,c).size()==0) {
        point a = intersection(segment(c.center, l.start), c)[0];
        point b = intersection(segment(c.center, l.end), c)[0];
        area = AreaOfSector(c, a, b);
    }
    //情况4:
    else {
        point a = intersection(segment(c.center, l.start), c)[0];
        point b = intersection(segment(c.center, l.end), c)[0];
        point x = intersection(l, c)[0];
        point y = intersection(l, c)[1];
        area = AreaOfSector(c, a, b) - AreaOfSector(c, x, y) +
AreaOfTriangle(c.center, x, y);
    }
    return sign * area;
}

double Solve(polygon p,circle c) {
    double area = 0;
    p.push_back(p[0]);
    int sz = p.size();
    for(int i=0;i<sz-1;i++) {
        area += Area(segment(p[i], p[i + 1]), c);
    }
    p.pop_back();
    return abs(area);
}

int main() {
    int n;
    scanf("%d", &n);
    polygon p;
    for (int i = 1; i <= n;i++) {
        double x, y;
        scanf("%lf%lf", &x, &y);
        p.push_back(point(x, y));
    }
    double ans = 0;
    point cn = CentroidOfPolygon(p);
    double area = AreaOfPolygon(p);
    circle c = circle(cn, sqrt(area / pi));
    printf("%.4lf\n", Solve(p, c) / area);
}
```

E

题意

给定n条已知长度的线段，判断能否构造出如下的圆，使得其满足以下条件：

（1）n条线段首尾相连构成一个多边形，且顶点都在圆上；

（2）该外接圆的圆心在多边形内部；

（3）圆的半径不超过120英寸

题解：

首先，要分别判断能否满足题目中描述的三个条件，构造核心函数int FindRadius(void)；其四个返回值的含义分别为：

-1：无法满足条件（1），不共圆；

-2：满足条件（1）但无法满足条件（2），圆心落入多边形外；

-3：满足条件（1）（2）但无法满足条件（3），圆的尺寸超过120英寸；

0：有半径，半径存储在circ_radius里；

（1）返回值为-1

固定圆的半径，则定长线段对应的圆心角固定，想象圆的半径从无穷逐渐减小，则只有最大边小于其他边之和时无解。

代码：

```
double minrad, minang, maxrad, maxang, currad, curang, dadr, diff;
int cnt = 50;
if(2.0*maxseg > segsum) {//segsum是预处理的线段和
         return -1;
}
```

（2）返回值为-2

假设最长半弦即为半径，求出其他弦对应的圆心角之和，与π作比较，若比π小说明圆心落在多边形外。

代码：

```
double FanAngle(double radius)//radius的值为最长半弦
{
     int i;
     double ang, angsum, ovdenom;
     if(radius < (0.5*maxseg)) {
         return -1.0;
     }

     ovdenom = 1.0/(2.0*radius);
     for(i = 0, angsum = 0.0; i < nseg; i++) {
```

```
            ang = asin(segs[i]*ovdenom);
            angsum += ang;

        }
       return angsum;

}
```

（3）返回值为-3

当最长边超过允许半径的两倍，则可直接认为圆的尺寸大了。更加一般的，当以120英寸为半径时，若圆心角之和还是大于π，说明圆的半径不止120英寸。

代码：

```
        minrad = 0.5*maxseg;
        if(minrad > MAX_RAD) {
            return -3;
        }
        maxrad = MAX_RAD;//maxrad是120
        maxang = FanAngle(maxrad);
        if(fabs(maxang - D_PI) < EPS) {
            circ_radius = maxrad;
            return 0;
        }
        if(maxang > D_PI) {
            return -3;
        }
```

（4）返回值为0

其余情况下返回值均为0。用二分法计算满足条件的圆的半径即可，终止条件是迭代50次，尽量使得所有弦的圆心角之和等于2π。

代码：

```
minrad = 0.5*maxseg;
        minang = FanAngle(minrad);
        if(fabs(minang - D_PI) < EPS) {//最长弦正好是半径时，直接赋值即可
            circ_radius = minrad;
            return 0;
        }
        maxrad = MAX_RAD;//maxrad是120
        maxang = FanAngle(maxrad);
        if(fabs(maxang - D_PI) < EPS) {
            circ_radius = maxrad;
            return 0;
        }
        currad = 0.5*(minrad + maxrad);
        curang = FanAngle(currad);
        if(fabs(curang - D_PI) < EPS) {
            circ_radius = currad;
```

```
            return 0;
        }
        if(curang < D_PI) {
            maxang = curang;
            maxrad = currad;
        } else {
            minang = curang;
            minrad = currad;
        }
        while(cnt>0){
            cnt--;
            diff = maxrad - minrad;
            if(diff > 0.01) {
                dadr = 100.0*(FanAngle(currad + 0.01) - curang);
            } else {
                dadr = (FanAngle(currad + diff) - curang)/diff;
            }
            currad = currad - (curang - D_PI)/dadr;
            if(currad < minrad) {
                currad = 0.75*minrad + 0.25*maxrad;
            } else if(currad > maxrad) {
                currad = 0.25*minrad + 0.75*maxrad;
            }
            curang = FanAngle(currad);
            if(fabs(curang - D_PI) < EPS) {
                circ_radius = currad;
                return 0;
            }
            if(curang < D_PI) {
                maxang = curang;
                maxrad = currad;
            } else {
                minang = curang;
                minrad = currad;
            }
        }
```

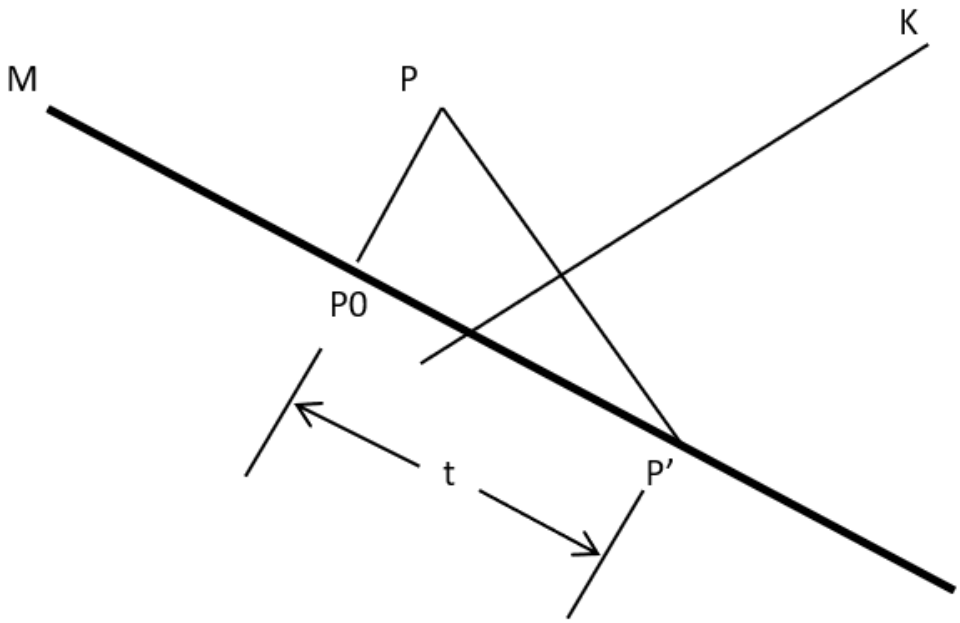
综上，分析复杂度，知计算次数约为50*n。其实对长度为120的区间二分，要求精度1e-4，只需迭代21次，这里在时间允许的情况下迭代50次也没有问题。

E

解法

首先要明确假若P'的位置确定了，那么K以及相应的Q'也确定了，于是我们可以考虑搜索P'，用P'求出K和Q'，并验证Q'是否在N上。

由于P'在M上，出于对称性我们考虑P'在M上的投影点$P0$,那么如果确定了P'与$P0$之间的距离t（分正负）也就确定了P'的位置，如图所示：



下面考虑如何找到这个t的值，根据t我们可以求出K,进而确定Q',那么将Q'代入N的方程，若$abs(N(Q')) < eps$则可以认为Q'在N上。对于所有的t我们都可以计算这样一个函数：$f(t) = N(Q') = err$，显然这个函数在t从负无穷到正无穷变化过程中是连续变化的， 于是我们可以用牛顿迭代法来近似求$f(t) = 0$的解。

另外，由于err的值肯定是从正连续变化到负或者从负连续变化到正，可以利用这一性质检验err值从而调整迭代点使其更快收敛。

代码

1.求$P0$

```
    double d = ma*ma + mb*mb;
    d = 1.0/sqrt(d);
    ma *= d; mb *= d; mc *= d;          //归一化

    d = px*ma + py*mb + mc;
    mx0 = px - d*ma;
    my0 = py - d*mb;                    //求出P0x

    scale = max(fabs(mx0),fabs(my0));
```

对$P0$的y坐标同样处理

2. 求$f(t)$

```
double FoldErr(double t)
{
    double px1, py1, qx1, qy1, dot, kdot, ret;
    px1 = mx0 - t*mb;
    py1 = my0 + t*ma;
    ka = px - px1;
    kb = py - py1;
    kc = -0.5*(ka*(px + px1) + kb*(py + py1));          //根据t得到K
```

```
    kdot = ka*ka + kb*kb;
    dot = (qx)*ka + (qy)*kb + kc;
    qx1 = qx - 2.0*dot*ka/kdot;
    qy1 = qy - 2.0*dot*kb/kdot;                    //根据K得到Q'并归一化
    ret = na*qx1 + nb*qy1 + nc;                    //得到f(t)
    return ret;
}
```

3.牛顿迭代法找 t

辅助以上下界来限制迭代点的位置

lft和rt是左右界，$errlft$和$errrt$是左右界的$f(t)$值。

```
int IntervalSearch(double lft, double rt, double errlft, double errrt)
{
    double left, right, cen, err, lefterr, righterr, dedt;
    lefterr = errlft;
    righterr = errrt;
    left = lft;
    right = rt;
    cen = 0.5*(lft + rt);
    err = FoldErr(cen);                            //取中点作为初始点
    while((fabs(err) >= EPS)) {
        if(err*lefterr > 0.0) {                    //err与下界同号
            lefterr = err;
            left = cen;
        } else {
            righterr = err;
            right = cen;
        }
        dedt = (FoldErr(cen + DEPS) - err)/DEPS;    //DESP是迭代法的增量
        if((fabs(dedt) < EPS) ||((cen = cen - err/dedt) <= left) || (cen >=
right)) {
            cen = (0.5*(left + right));
        }                                          //如果迭代点在界外，那么直接取中
点
        err = FoldErr(cen);
    }
    return 0;
}
```

将ka, kb, kc设为全局变量就可以得到最终的直线K.

G

题意

操作：对奇数加一，对偶数除二。对于一个32无符号位数，输出经过多少操作变为1。

题解

循环计数。水题。

H

题意&题解

显然如果两个公匙a,b有公因数那么则对应三个质因数

$$gcd(a,b),a/gcd(a,b),b/gcd(a,b)$$

然后按题意处理即可

I

题意

进制转换，基数变为一个由两个互质的整数相除所得的分数

题解

进制转换类问题的变式，解法不难，普通的进制转换只要对基数不断进行取模和除操作即可得到答案，这题的变化就是基数变为一个由两个互质的整数相除所得的分数，解题方法其实和普通的进制转换一样，还是取模求系数，除法消基数的次数，只不过每次在除完了p之后，还要乘一个q，才能进行下一次操作。

J

题意&题解

递推+结论+矩阵快速幂

对于一个这样的图，设外圈点数为n，Unicycles的数量为n*S（n）

S（n）为不同的图的数目，乘n是旋转后的结果

对S（n+1），相当于在外圈多加了一个点，

对于圈的形成，可分为三种情况：

1只有中心点不在圈上

2.k个点在圈上，中心点也在圈上，k<=n

设M(i)为第二种情况下，有i个点不在圈上的图的数目

显然M(0)=1,M(1)=3

我们在圈的旁边添加一个新点

这个点有三种添加方式：，连接中心点，连接圈上的点(外圈)，连接圈以外的点

每种方式分别是M(i)种图

其中第二三种方式有重复，即存在这个点同时与两边相连的情况，即为M(i-1)

综上，M(i+1)=3*M(i)-M(i-1)。

S(n)=sum(1,M(0),...,M(n-2));

由于求和的需要，将其转化成斐波那契数列M(n)=fib(2*n-1);

运用斐波那契数列奇数项求和公式得S(n)=fib(2*n-1)

求得时候使用矩阵快速幂加速

K

题意：

给定一行字符，判断以 '-' 分割的前半段和后半段长度是否分别在(1,8]和[1,24]的区间内，是则YES否则NO

tips:空格算作一个字符，若给定串中没有‘-’，则也输出NO

题解：

签到，整行读入后直接找到‘-’判断即可

使用

```
scanf("%[^\n]", &str);
```

读入整行

%[ ],这个参数的意义是读入一个字符集合。[ ]是个集合的标志,因此%[ ]特指读入此集合所限定的那些字符,比如%[A-Z]是输入大写字母,一旦遇到不在此集合的字符便停止。如果集合的第一个字符是"^", 这说明读取不在"^"后面集合的字符，遇到"^" 后面集合的字符便停止。

标程

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    char a[10001];
    scanf("%[^\n]",a);
    int l=strlen(a),pos=-1;
    for(int i=0;i<l;i++)if(a[i]=='-')pos=i;
    if(pos==-1){cout<<"NO"<<endl;return 0;}
    if(!(pos>1 && pos<=7) && !(l-pos>2 && l-pos<24)){cout<<"NO"<<endl;return 0;}
    cout<<"YES"<<endl;
return 0;
}
```