The Study of Invasive Species

2020-2021

Students: Gabor Ioana (11^{th} grade), Păcurar Irina (11^{th} grade), Karaivanoff Eduard-Ivan (10^{th} grade), Vana Marc-Aurelius (10^{th} grade), Marcu Mihai (10^{th} grade), Potfălean Andrei (10^{th} grade), Selegean Victor (11^{th} grade), Baicu Antoniu (10^{th} grade), Boca Ștefan (11^{th} grade)

Institution: Colegiul Naţional "Emil Racoviţă" Cluj-Napoca

Teacher: Ariana Văcăreţu

Contents

1 Statement **2**

2 Introduction **2**
2.1 Distribution Model . . . 2
2.2 The first 6 years . . . 3

3 The Algorithm **3**
3.1 Inputs and Outputs . . . 3
3.2 Polynomials . . . 4
3.3 Algorithm Explanation . . . 4
3.4 Algorithm Analysis . . . 5
3.5 C++ Algorithm . . . 5

4 Special Approach I **8**
4.1 Statement . . . 8
4.2 Visual Representation . . . 8
4.3 Demonstration . . . 8
4.4 Formula . . . 11

5 Special Approach II **11**
5.1 Statement . . . 11
5.2 Development Structure . . . 11
5.3 Proof . . . 12
5.4 Formula . . . 12
5.5 Visual Representation . . . 12
5.6 Structure . . . 13

6 Special Approach III **13**
6.1 Statement . . . 13
6.2 Explanation . . . 14

7 C++ Program for Special Approaches II and III **14**

8 Conclusion **17**

1 Statement

To elaborate the dispersal of the pampas grass in a simple manner, we shall imagine a square shaped garden, divided in 9 identical parcels of land. The reproduction of the plant is made, obviously, through its seeds. Each plant starts reproducing after 3 years. In the 3rd year, the first plant produces $5x$ seeds. x seeds stay in their place, while the other $4x$ seeds are equally distributed to the north, south, east and west of the seeds that remain in their place, in the parcels of land adjacent to them. The newly distributed seeds will grow into new plants and will start reproducing after they reach 3 years of age as well, redistributing their own seeds following the same distribution model.

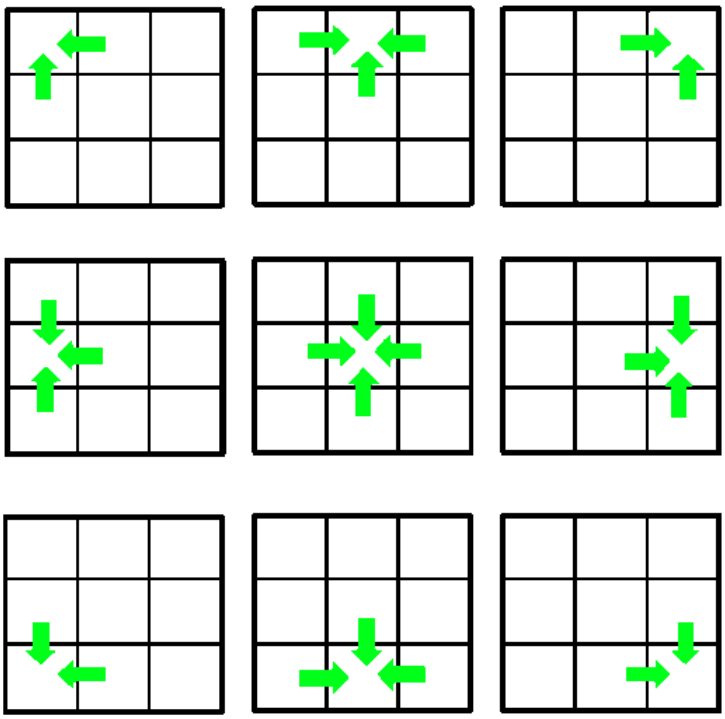
Extend and generalise the process of reproduction by placing the first $5x$ seeds in one of the 9 parcels of land, knowing that each plant dies after it reaches age 12.

Following the analysis of the original statement, we worked on other cases similar to the initial one. In the first one, plants, which start reproducing immediately and die right after laying their seeds, produce only 4 seeds which are equally distributed to their adjacent plots. In the second one, the garden has no fence, hence it is not limited to 9 parcels, while the plants reproduce in the same manner as in the first particular case. And lastly, in the third one, we have multiple gardens on a plain, identical to the garden in the second case (with plants reproducing in the same manner), which stop expanding once they intersect their paths.

2 Introduction

2.1 Distribution Model

To easily understand the way our plants are reproducing, we have elaborated a graphical representation of the manner in which seeds are distributed. For each one of the 9 parcels, the model shows from which directions it receives its seeds.



Picture 1 - *Visual representation of the way seeds get distributed*

As observed in the image, each parcel receives seeds like this:

1. Parcels receiving seeds from 2 directions:
- the top left parcel – from its east and south;
- the bottom left parcel – from its east and north;
- the top right parcel – from its west and south;
- the bottom left parcel – from its west and north.

2. Parcels receiving seeds from 3 directions:
- the upper middle parcel – from its west, east and south;
- the left middle parcel – from its east, north and south;
- the right middle parcel – from its west, north and south;
- the bottom middle parcel – from its east, west and north.

3. Parcels receiving seeds from 4 directions:
- the centre parcel – from its north, south, east and west.

2.2 The first 6 years

This is how our garden will develop in the first 6 years:

							x	
	1			1		x	x+1	x
							x	
	2x			3x		$2x^2$	$2x^2+4x$	$2x^2$
2x	2x+1	2x	3x	3x+1	3x	$2x^2+4x$	$5x^2+4x+1$	$2x^2+4x$
	2x			3x		$2x^2$	$2x^2+4x$	$2x^2$

Picture 2 - *Number of plants in each parcel for the first 6 years*

3 The Algorithm

Now that we have understood the way our garden works, a new question arises. How many plants are in our garden? We've developed a C++ program which shows us how many plants there are in each parcel at any given time.

3.1 Inputs and Outputs

Input:

t - the number of years that have passed

Output:
- a matrix of polynomials representing the number of living plants in each cell after t years have passed
- a polynomial representing the number of living plants after t years have passed
- a polynomial representing the total number of dead plants after t years have passed

3.2 Polynomials

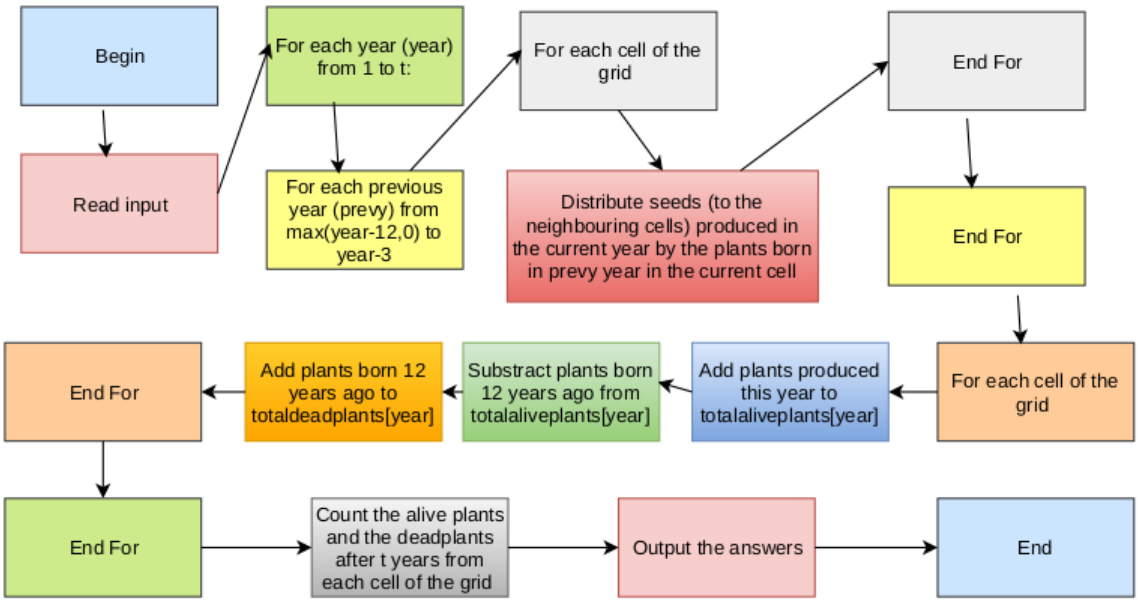
As a result of the number of seeds produced by a plant being possibly as large as one million, we decided to implement all the mathematical operations using polynomials (e.g. addition of two polynomials, multiplying a polynomial by -1 or by x), with the indeterminate x being a fifth of the number of seeds produced by a plant.

3.3 Algorithm Explanation

After defining a polynomial structure and implementing all the necessary mathematical operators, 4 three-dimensional polynomial arrays (named *"newplants"*, *"deadplants"*, *"totaldead"* and *"totalalive"*) were declared, the first dimension of each of them being the current year, and the second and third being the position in the garden.

For example: *newplants*[0][2][2] has the value 1, as in the beginning there is only one seed in the centre of the garden. After twelve years, that plant dies, therefore *deadplants*[12][2][2] is 1. The arrays *totalalive*[i][j][k] and *totaldead*[i][j][k] represent the number of alive plants and the number of dead plants, respectively, after i years, in the cell with coordinates *(j,k)*. Additionally, the polynomial variables *"alive"* and *"dead"* stand for the number of alive plants and the number of dead plants (from all cells), respectively, after t years have passed.

In order to populate the arrays, a for-loop has been used to iterate through all the years, one by one. The distribution of seeds is performed inside of an inner for-loop, that iterates through the past twelve years, and adds the seeds produced by the older plants to the current garden. This addition is completed with the help of two directional arrays that determine in which parcel each seed has to go. As a plant dies after 12 years, *deadplants*[i] would have the same values as *newplants*[i-12]. Moreover, it is needed to subtract the number of new plants from 12 years ago, from the number of total alive plants, and to add them to the number of total dead plants.



Picture 3 - *Flowchart of the algorithm*

Consequently, after populating the three-dimensional arrays, it is only needed to output the two-dimensional array *totalalive*[t] (t being the number of years that have passed), and the variables alive and dead.

3.4 Algorithm Analysis

The time and memory complexities of our algorithm are both *O(t*g)*, t being the number of years and g being the maximum degree of a polynomial, due to the for-loop that iterates through all the years. The inner for-loops are negligible, as they have a constant number of iterations.

3.5 C++ Algorithm

```
#include <bits/stdc++.h>
#define TMAX 50
#define GRADMAX 20

using namespace std;

/// declaration of variables and structures

struct polinom{
    int coef[GRADMAX+1],grm;
};

int t;
polinom add,alive,dead;
polinom newplants[TMAX+5][4][4];
polinom deadplants[TMAX+5][4][4];
polinom totaldead[TMAX+5][4][4];
polinom totalalive[TMAX+5][4][4];
int dx[]={0,1,-1,0};
int dy[]={1,0,0,-1};

/// function for outputting a polynomial

void write_polynomial(polinom a){
    for(int i=a.grm;i>=1;i--){
        cout<<a.coef[i]<<"*x^"<<i<<"+";
    }
    cout<<a.coef[0]<<" ";
}

/// function for addition of two polynomials

polinom add_polynomials(polinom& a,polinom b){
    a.grm=max(a.grm,b.grm);
    for(int i=0;i<=a.grm;i++){
        a.coef[i]=a.coef[i]+b.coef[i];
    }
    while(a.coef[a.grm]==0){
        a.grm--;
    }
    return a;
}
```

```
/// function that changes the sign of a polynomial

polinom minus_polynomial(polinom a){
    for(int i=0;i<=a.grm;i++){
        a.coef[i]=-a.coef[i];
    }
    return a;
}

/// function that multiplies a polynomial by x

polinom shift_polynomial(polinom a){
    for(int i=a.grm;i>=0;i--){
        a.coef[i+1]=a.coef[i];
    }
    a.coef[0]=0;
    a.grm++;
    while(a.coef[a.grm]==0){
        a.grm--;
    }
    return a;
}

/// function that distributes seeds

void add_plants(int year,int ii,int jj){
    add_polynomials(newplants[year][ii][jj],add);
    int x,y;
    for(int d=0;d<4;d++){
        x=ii+dx[d];
        y=jj+dy[d];
        if(x>=1&&x<=3&&y>=1&&y<=3){
            add_polynomials(newplants[year][x][y],add);
        }
    }
}

/// function that initialises the matrices

void init_polynomials(){
    newplants[0][2][2].grm=0;
    newplants[0][2][2].coef[0]=1;
    totalalive[0][2][2].grm=0;
    totalalive[0][2][2].coef[0]=1;
}

/// main body
int main(){
    cout<<"How many years have passed?"<<'\n';
    cin>>t;
    init_polynomials();
```

```
    ///interate through every year and update the current year
    for(int year=1;year<=t;year++){
        for(int prevy=max(year-12,0);prevy<=year-3;prevy++){
            for(int ii=1;ii<=3;ii++){
                for(int jj=1;jj<=3;jj++){
                    add=shift_polynomial(newplants[prevy][ii][jj]);
                    add_plants(year,ii,jj);
                }
            }
        }
        for(int ii=1;ii<=3;ii++){
            for(int jj=1;jj<=3;jj++){
                totalalive[year][ii][jj]=totalalive[year-1][ii][jj];
                totaldead[year][ii][jj]=deadplants[year-1][ii][jj];
                add_polynomials(totalalive[year][ii][jj],newplants[year][ii][jj]);
                if(year-12>=0){
                    add_polynomials(totalalive[year][ii][jj],minus_polynomial(
                                    newplants[year-12][ii][jj]));
                    deadplants[year][ii][jj]=newplants[year-12][ii][jj];
                    add_polynomials(totaldead[year][ii][jj],deadplants[year][ii][jj]);
                }
            }
        }
    }
    for(int i=1;i<=3;i++){
        for(int j=1;j<=3;j++){
            add_polynomials(alive,totalalive[t][i][j]);
            add_polynomials(dead,totaldead[t][i][j]);
        }
    }
    /// output the garden in the given year
    cout<<"A grid showing the number of alive plants in each cell:"<<'\n';
    for(int i=1;i<=3;i++){
        for(int j=1;j<=3;j++){
            write_polynomial(totalalive[t][i][j]);
            cout<<' ';
        }
        cout<<'\n';
    }
    cout<<"Total number of alive plants:";
    write_polynomial(alive);
    cout<<'\n';
    cout<<"Total number of dead plants: ";
    write_polynomial(dead);
    cout<<'\n';
}
```

4 Special Approach I

4.1 Statement

Our plant will produce only 4 seeds, which will be equally distributed to the north, south, east and west of the initial parcel, without leaving any in its current position. The reproduction starts immediately, in the first year of the plant's life and, after reproducing itself, the plant dies.

4.2 Visual Representation

The first five years, including the initial stage will look like in the following picture:

				1		2		2
	1		1		1		4	
				1		2		2
	8		16		16		64	
8		8		32		64		64
	8		16		16		64	

Picture 4 - *The evolution for the first 5 years (Special Approach I)*

4.3 Demonstration

We consider the garden as 3x3 matrix. Let A_k, $k \in \{0, 1, 2, ..., n\}$ be the matrices, where A_0 represents the initial stage, A_1 the one after the first reproduction, A_2 the one after the second reproduction and so on and so forth.

So, $A_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$. Following the rule, $A_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$, $A_2 = \begin{pmatrix} 2 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & 2 \end{pmatrix}$, $A_3 = \begin{pmatrix} 0 & 8 & 0 \\ 8 & 0 & 8 \\ 0 & 8 & 0 \end{pmatrix}$, $A_4 = \begin{pmatrix} 16 & 0 & 16 \\ 0 & 32 & 0 \\ 16 & 0 & 16 \end{pmatrix}$, $A_5 = \begin{pmatrix} 0 & 64 & 0 \\ 64 & 0 & 64 \\ 0 & 64 & 0 \end{pmatrix}$, $A_6 = \begin{pmatrix} 128 & 0 & 128 \\ 0 & 256 & 0 \\ 128 & 0 & 128 \end{pmatrix}$ *etc.*

Rewriting the matrices using the powers of 2, we obtain:

$A_1 = \begin{pmatrix} 0 & 2^0 & 0 \\ 2^0 & 0 & 2^0 \\ 0 & 2^0 & 0 \end{pmatrix}$, $A_2 = \begin{pmatrix} 2^1 & 0 & 2^1 \\ 0 & 2^2 & 0 \\ 2^1 & 0 & 2^1 \end{pmatrix}$, $A_3 = \begin{pmatrix} 0 & 2^3 & 0 \\ 2^3 & 0 & 2^3 \\ 0 & 2^3 & 0 \end{pmatrix}$, $A_4 = \begin{pmatrix} 2^4 & 0 & 2^4 \\ 0 & 2^5 & 0 \\ 2^4 & 0 & 2^4 \end{pmatrix}$, $A_5 = \begin{pmatrix} 0 & 2^6 & 0 \\ 2^6 & 0 & 2^6 \\ 0 & 2^6 & 0 \end{pmatrix}$, $A_6 = \begin{pmatrix} 2^7 & 0 & 2^7 \\ 0 & 2^8 & 0 \\ 2^7 & 0 & 2^7 \end{pmatrix}$ *etc.*

Due to the rules of this approach, the matrices have 2 structures based on the parity of the matrices' indices.

Case I. k is odd

In this case, we have the matrix of the form: $A_k = \begin{pmatrix} 0 & 2^{\mathrm{a}} & 0 \\ 2^{\mathrm{a}} & 0 & 2^{\mathrm{a}} \\ 0 & 2^{\mathrm{a}} & 0 \end{pmatrix}$, $a \in \mathbb{N}$.

The powers of 2 from the matrices' nonzero elements form the following row: 0, 3, 6, 9, ... So, the n^{th} element in this row is *3(n-1)* and the k^{th} matrix has the nonzero elements equal to $2^{\frac{3(k-1)}{2}}$ because the matrices in this case are skip counted by 2.

It follows that $A_k = \begin{pmatrix} 0 & 2^{\frac{3(k-1)}{2}} & 0 \\ 2^{\frac{3(k-1)}{2}} & 0 & 2^{\frac{3(k-1)}{2}} \\ 0 & 2^{\frac{3(k-1)}{2}} & 0 \end{pmatrix}$ *(1)*

Case II. k is even

In this case, we have the matrix of the form: $A_k = \begin{pmatrix} 2^{\mathrm{b}} & 0 & 2^{\mathrm{b}} \\ 0 & 2^{2\mathrm{b}} & 0 \\ 2^{\mathrm{b}} & 0 & 2^{\mathrm{b}} \end{pmatrix}$, $b \in \mathbb{N}$.

The exponents of a_{22} form the following row: 2, 5, 8, 11, ...

So, the n^{th} element in this row is *3n-1* and the k^{th} matrix has the nonzero elements equal to $2^{3\frac{k}{2}-1} = 2^{\frac{3k-2}{2}}$ because the matrices in this case are skip counted by 2.

It follows that $A_k = \begin{pmatrix} 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \\ 0 & 2^{\frac{3k-2}{2}} & 0 \\ 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \end{pmatrix}$ *(2)*

We show that $A_{k+1}=A_k \cdot (A_1)^3$, following the same cases, after the parity of k.

Case I. k is odd

$A_k = \begin{pmatrix} 0 & 2^{\frac{3(k-1)}{2}} & 0 \\ 2^{\frac{3(k-1)}{2}} & 0 & 2^{\frac{3(k-1)}{2}} \\ 0 & 2^{\frac{3(k-1)}{2}} & 0 \end{pmatrix}$ (from *(1)*)

$A_{k+1} = \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix}$ (from the fact that *k+1* is even and *(2)*)

$(A_1)^3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix}$ *(3)*

$\begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix} = \begin{pmatrix} 0 & 2^{\frac{3(k-1)}{2}} & 0 \\ 2^{\frac{3(k-1)}{2}} & 0 & 2^{\frac{3(k-1)}{2}} \\ 0 & 2^{\frac{3(k-1)}{2}} & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} \Leftrightarrow$

$\Leftrightarrow \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix} = \begin{pmatrix} 2^{\frac{3(k-1)}{2}+1} & 0 & 2^{\frac{3(k-1)}{2}+1} \\ 0 & 2^{\frac{3(k-1)}{2}+1} + 2^{\frac{3(k-1)}{2}+1} & 0 \\ 2^{\frac{3(k-1)}{2}+1} & 0 & 2^{\frac{3(k-1)}{2}+1} \end{pmatrix} \Leftrightarrow$

$\Leftrightarrow \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix} = \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3(k-1)}{2}+2} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix} \Leftrightarrow$

$$\Leftrightarrow \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix} = \begin{pmatrix} 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \\ 0 & 2^{\frac{3k+1}{2}} & 0 \\ 2^{\frac{3k-1}{2}} & 0 & 2^{\frac{3k-1}{2}} \end{pmatrix}, \text{ which is true.}$$

$\Rightarrow A_{k+1}=A_k \cdot (A_1)^3$, where k is odd. *(4)*

Case II. k is even

$$A_k = \begin{pmatrix} 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \\ 0 & 2^{\frac{3k-2}{2}} & 0 \\ 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \end{pmatrix} \text{ (From } \textit{(2)}\text{)}$$

$$A_{k+1} = \begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix} \text{ (From the fact that } k+1 \text{ is odd and } \textit{(1)}\text{)}$$

$$(A_1)^3 = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} \text{ (From } \textit{(3)}\text{)}$$

$$\begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix} = \begin{pmatrix} 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \\ 0 & 2^{\frac{3k-2}{2}} & 0 \\ 2^{\frac{3k-4}{2}} & 0 & 2^{\frac{3k-4}{2}} \end{pmatrix} \cdot \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2^{\frac{3k-4}{2}+1} + 2^{\frac{3k-4}{2}+1} & 0 \\ 2^{\frac{3k-2}{2}+1} & 0 & 2^{\frac{3k-2}{2}+1} \\ 0 & 2^{\frac{3k-4}{2}+1} + 2^{\frac{3k-4}{2}+1} & 0 \end{pmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2^{\frac{3k-4}{2}+2} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k-4}{2}+2} & 0 \end{pmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2^{\frac{3k}{2}} & 0 \\ 2^{\frac{3k}{2}} & 0 & 2^{\frac{3k}{2}} \\ 0 & 2^{\frac{3k}{2}} & 0 \end{pmatrix}, \text{ which is true.}$$

$\Rightarrow A_{k+1}=A_k \cdot (A_1)^3$, where k is even. *(5)*

From *(4)* and *(5)*, it follows that $A_{k+1}=A_k \cdot (A_1)^3$, for all integers $k \geq 1$. *(6)*

We prove that $A_n = (A_1)^{1+3(n-1)}$ for all integers $n \geq 1$ using the Mathematical Induction.
P(n): $A_n = (A_1)^{1+3(n-1)}$, $n \geq 1$.

Basis step: We verify if *P(1)* is true.
P(1): $A_1 = (A_1)^{1+3(1-1)} = A_1$, which is true.
P(2): $A_2 = (A_1)^{1+3(2-1)} = (A_1)^4$
$(A_1)^4 = A_1 \cdot (A_1)^3$

From *(3)* $\Rightarrow (A_1)^4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & 2 \end{pmatrix} \Rightarrow P(2)$ is also true.

Induction step: We show that if *P(k):* $A_k = (A_1)^{1+3(k-1)}$ is true for some integer $k \geq 1$, then *P(k+1):* $A_{k+1} = (A_1)^{1+3k}$ is also true.

$A_{k+1} = A_k \cdot (A_1)^3$ (from *(6)*)
From *P(k)* $\Rightarrow A_{k+1} = (A_1)^{1+3(k-1)} \cdot (A_1)^3 = (A_1)^{1+3k-3+3} = (A_1)^{1+3k} \Rightarrow$ *P(k+1)* is true.
Therefore, *P(n)* is true for all integers $n \geq 1$.

$\Rightarrow A_n = (A_1)^{1+3(n-1)}$

4.4 Formula

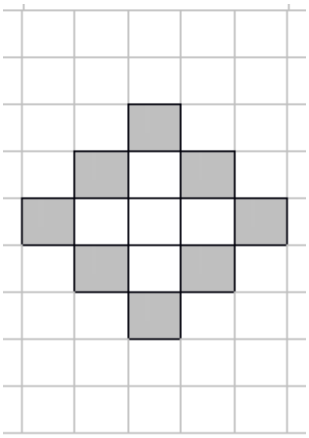
We proved the formula is $A_n = (A_1)^{1+3(n-1)}$ for all integers $n \geq 1$.

5 Special Approach II

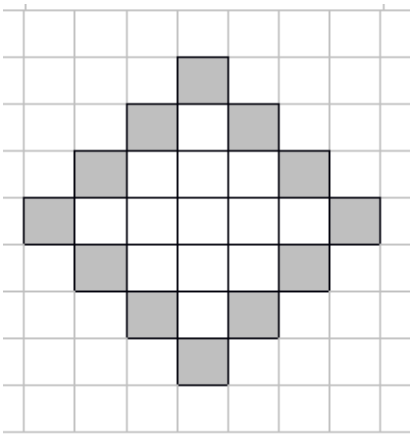
5.1 Statement

But what if the garden had no fence? Suppose we had an infinite number of parcels in our garden. The plants follow the same reproducing pattern as in the first special approach, meaning that they will produce 4 seeds each, which they will equally distribute to their north, south, east and west. They will start reproducing immediately and will die right after laying their seeds, meaning that each plant will live 1 year.

5.2 Development Structure



Picture 5 - *Garden after 2 years of reproduction (Special Approach II)*



Picture 6 - *Garden after 3 years of reproduction (Special Approach II)*

Analysing the structure, we observe that in each year the garden takes the shape of a rhombus with the number of squares it's composed of being equal to the sum of the numbers of an arithmetic progression with the ratio 4.

5.3 Proof

Each year that passes by, 2 new plants get added on each row, one on the left side and one on the right side of the respective row. Besides, another plant is added on top of the northernmost parcel of the garden and another one on the southernmost parcel. In the n^{th} year, the rhombus that represents the garden is composed of *2n-1* rows and columns, which means that each year *2(2n-1)+2=4n* parcels will be added to the garden. We obtain an arithmetic progression with the ratio 4.

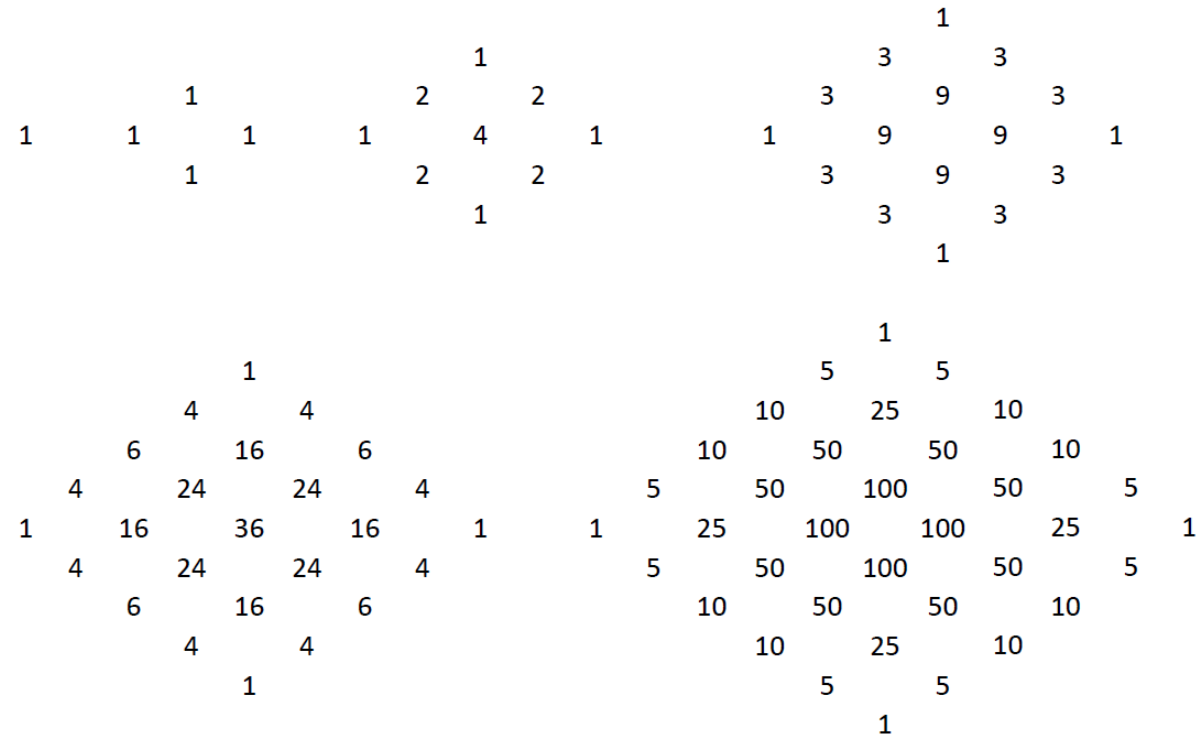
5.4 Formula

This formula calculates how many parcels the rhombus is made of, k representing the given year.

$1 + 4 + 8 + 12 + 16 + ... + 4(k-1) =$
$= 1 + 4(1 + 2 + 3 + 4 + ... + k - 1) =$
$= 1 + 4k(k-1)/2 =$
$= 1 + 2k(k - 1)$

5.5 Visual Representation

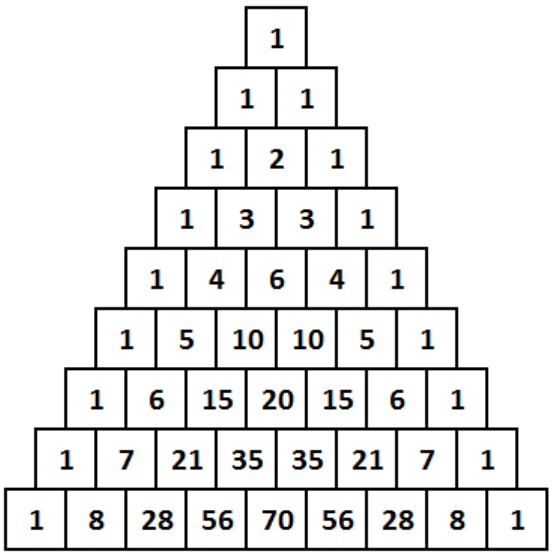
Let's now focus on the number of plants we'll have on each parcel in each year. The first five years, including the initial stage will look like the following picture:



Picture 7 - *The first 5 years (Special Approach II)*

5.6 Structure

Starting with one plant in the middle and continuing with the reproduction as in the first approach, the numbers from rhombuses' margins are added following the same rules as the numbers from Pascal's Triangle. So, the margins of n^{th} year represent the n^{th} line from Pascal's Triangle.

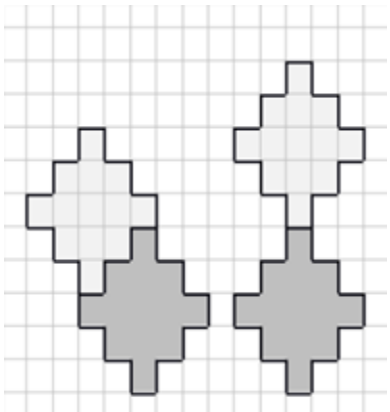


Picture 8 - *Pascal's Triangle*

6 Special Approach III

6.1 Statement

But what if we had more gardens on a plain, each garden expanding itself in the same manner as the garden at approach II? Find out in how many years all the gardens will stop expanding, knowing that each expansion stops when it meets another expansion.



Picture 9 - *Gardens stop expanding when they meet each other (Special Approach III)*

6.2 Explanation

1. We build a matrix with the distance between every 2 plants equal to:

$$(|x_A - x_B| + |y_A - y_B|)/2$$

because each plant expands (x_A and y_A are the coordinates of plant A, while x_B and y_B are the coordinates of plant B, both plants being chosen arbitrarily). We also build an array which represents the values of the distances between each plant and the closest one to it.

2. We find the minimum value in this array (this will correspond to the plant that first stops expanding). We change the values in the matrix with *2 * distance – minimum* on the lines and columns which contain the initial minimum value. The program's complexity is $O(n^2)$ which means we can manage a parcel with 1000 plants in 0.1 seconds.

7 C++ Program for Special Approaches II and III

```
#include <bits/stdc++.h>
using namespace std;
int p,n,nx,ny;
int x[1005],y[1005];
int dxyMin[1005];
int dMin[1005];
int difMin[1005][1005];
int difMin2[1005][1005];
int lMin[1005];
int jlMin[1005];
unsigned long long t,ct[1005];
int main()
{
    cin>>p>>nx>>ny>>t>>n;
    /// the formula for the 2nd approach
    if(p==1)
        cout<<2*t*(t-1)+1;
    else
    {
        /// 3rd approach
        /// we read the coordinates of the plants
        for(int i=1;i<=n;i++)
            cin>>x[i]>>y[i];
        /// we initialise the minimum distance between
        /// each garden with a big number
        for(int i=1;i<=n;i++)
        {
            int dxmin=min(nx-x[i]+1,x[i]);
            int dymin=min(ny-y[i]+1,y[i]);
            dxyMin[i]=min(dxmin,dymin);
        }
        /// we create a matrix containing the distances between
        /// all the gardens (e.g. for every garden i and garden j
        /// we memorise the distance between them)
        for(int i=1;i<=n;i++)
```

```
{
    lMin[i]=nx+ny;
    for(int j=1;j<=n;j++)
    {
        difMin[i][j]=abs(x[i]-x[j])+abs(y[i]-y[j])+1;
        difMin2[i][j]=difMin[i][j]/2;
        if(difMin2[i][j]<lMin[i] && i!=j)
        {
            lMin[i]=difMin2[i][j];
            jlMin[i]=j;
        }
    }
}
int nr=0;
/// we update the minimum distance for every garden
while(nr<n)
{
    int mMin=nx+ny,iMin,jMin,im=0;
    for(int i=1;i<=n;i++)
    {
        if(mMin>lMin[i] && !dMin[i])
        {
            mMin=lMin[i];
            iMin=i;
            jMin=jlMin[i];
            im=0;
        }
        if(mMin>dxyMin[i] && !dMin[i])
        {
            mMin=dxyMin[i];
            im=i;
        }
    }
    int i=iMin,j=jMin;
    if(im!=0)
    {
        i=im;
        dMin[i]=dxyMin[i];
    }
    else
    {
        if(dMin[i])
        {
            i=jMin;
            j=iMin;
        }
        dMin[i]=difMin2[i][j];
    }
    nr++;
    lMin[i]=nx+ny;
    /// we check in the matrix when each garden stops expanding
```

```
            for(int k=1;k<=n;k++)
                if(!dMin[k] || !dMin[i])
                {
                    if(difMin2[k][i]!=dMin[i])
                    {
                        difMin2[k][i]=difMin2[i][k]=difMin[k][i]-dMin[i];
                        if(difMin2[k][i]<lMin[i] && k!=i && (!dMin[k] || !dMin[i]))
                            {lMin[i]=difMin2[k][i];jlMin[i]=k;}
                    }
                    if(jlMin[k]==i)
                    {
                        lMin[k]=nx+ny;
                        /// for each garden, we update j once they stop growing
                        /// as a result of a collision with the garden i
                        for(int ik=1;ik<=n;ik++)
                        {
                            if(difMin2[k][ik]<lMin[k] && k!=ik)
                            {
                                lMin[k]=difMin2[k][ik];
                                jlMin[k]=ik;
                            }
                        }
                    }
                }
        }
        for(int i=1;i<=n;i++)
        {
            t=dMin[i];
            ct[i]=1+2*t*(t-1);
        }

        unsigned long long tot=0;
        int dmax=0;

        /// we take the biggest result of every expansion
        /// (when they stop growing)
        for(int i=1;i<=n;i++)
        {
            tot+=ct[i];
            if(dmax<dMin[i])
                dmax=dMin[i];
        }
        cout<<(unsigned long long)nx*ny-tot<<"\n";
        /// we output the result
        cout<<dmax;
    }
    return 0;
}
```

8 Conclusion

Having presented all types of solving methods and approaches to our readers, our article regarding the study of invasive species comes to an end. We conclude that the best way of solving the initial problem is using the tools informatics offer us, whereas the other approaches are easily solvable using mathematical discoveries and properties such as matrices, arithmetical progressions and Pascal's triangle.

Our team sincerely thanks all readers for their attention!