

Město ze vstupního souboru načítám do dynamicky alokovaného pole, v němž je každý čtverec kamení nebo písku reprezentován jako číslo, v případě kamení nulou, v případě písku kladným číslem, které je pro každou oblast čtverců písku jdoucích za sebou na řádku stejné. Počet takovýchto polí si ukládám do proměnné. Poté ve svém algoritmu procházím toto pole čtverec po čtverci a v případě, že aktuální čtverec neobsahuje kamení, zjišťuji jestli se ve čtverci pod ním nenachází také písek. Pokud ano, a má jiné číslo (resp. toto číslo musí být větší, aby se předešlo dvojímu započtení místnosti v případě že má tvar “U”) místnosti než aktuální čtverec, projdu řádek ve kterém se toto pole nachází (tedy řádek pod aktuálním řádkem) a všem čtvercům, které mají stejné číslo jako čtverec nacházející se pod aktuálním polem změním jejich číslo na číslo aktuálního pole. Tím jsou tyto čtverce sloučeny do jedné místnosti, dekrementuji tedy počítadlo místností. Tímto postupem ovšem není možno správně sloučit například místnosti ve tvaru “U”, proto je z aktuálního čtverce ještě zkontrolováno číslo čtverce nad ním a pokud tam je místnost s neodpovídajícím číslem, jsou tato čísla sloučena v celém poli a je dekrementováno počítadlo místností. Po projití celého pole bude počítadlo místností obsahovat počet místností v zasypaném městě.

Časová složitost bude záviset na velikosti čtvercové sítě zadané na vstupu (počet řádků M , sloupců N), v nejlepším případě $O(M*N)$, v nejhorším teoretickém případě s komplikovanými tvary místností $O((M*N)^2)$.

Paměťová složitost bude lineárně záviset na počtu čtverců zadaných na vstupu, což můžeme vyjádřit jako $O(M*N)$.

```

#include <iostream>
#include <fstream>

#define ROCK 0

using namespace std;

class city {
private:
    unsigned *square;
    unsigned width, height;
    unsigned nrooms;
    void merge_all(unsigned from, unsigned to);
    void merge_line(unsigned line, unsigned from, unsigned to);
public:
    void feed(unsigned sand, unsigned rocks);
    unsigned rooms(void);
    city(unsigned w, unsigned h);
    ~city() { delete [] square; };
};

void city::feed(unsigned sand, unsigned rocks)
{
    static unsigned *feeder = square;
    static unsigned nextnum = 1;
    if(sand){
        while(sand--){
            *feeder++ = nextnum;
        }
        nextnum++;
        nrooms++;
    }
    while(rocks--){
        *feeder++ = ROCK;
    }
}

```

```

unsigned city::rooms(void)
{
    //pro kazdy ctverec ...
    for(unsigned i=0; i<width*height; i++){
        //pokud je o radek nize vetsi cislo, prepiseme tento radek
        if(square[i]!=ROCK && (i+width)<height*width
            && square[i+width]!=ROCK && square[i]<square[i+width])
        {
            merge_line((i/width)+1, square[i+width],square[i]);
            nrooms--;
        }
        //pokud je o radek vyse vyssi cislo, prepiseme cele pole
        if(square[i]!=ROCK && (signed)(i-width)>=0
            && square[i-width]!=ROCK && square[i]<square[i-width])
        {
            merge_all(square[i-width], square[i]);
            nrooms--;
        }
    }
    return nrooms;
}

```

```

void city::merge_line(unsigned line, unsigned from, unsigned to)
{
    for(unsigned i=(line*height); i<(line*height)+width; i++){
        if(square[i]==from){
            square[i]=to;
        }
    }
}

```

```

void city::merge_all(unsigned from, unsigned to)
{
    for(unsigned i=0; i<width*height; i++){
        if(square[i]==from){
            square[i]=to;
        }
    }
}

```

```

city::city(unsigned w, unsigned h)
{
    square = new unsigned[w*h];
    width = w;
    height = h;
    nrooms = 0;
}

int main(void)
{
    ifstream infile("mesto.in");
    ofstream outfile("mesto.out");

    unsigned height, width, lines;

    if(!(infile >> height >> width >> lines)){
        cerr << "Neplatny vstup\n";
        exit(EXIT_FAILURE);
    }

    city my_city(width,height);

    for(unsigned i=0; i<lines; i++){
        unsigned sand, rock;
        if(!(infile >> sand >> rock)){
            cerr << "Neplatny vstup\n";
            exit(EXIT_FAILURE);
        }
        my_city.feed(sand,rock);
    }

    outfile << my_city.rooms() << '\n';

    return EXIT_SUCCESS;
}

```