35-1-S

Daniel Culliver

November 11, 2022

Úkol 1 – Robot na Marsu [5b]

Jsou dány 2 vektory v prostoru $\mathbb{R}^2$: $(x_1, y_1)^T$ a $(x_2, y_2)^T$.
Jestliže se chceme pohybovat v jednom směru, stačí přičíst a odečíst vektory, dokud x nebo y bude 0. Toto dokážeme velmi jednoduše. Vezmeme tedy x_1 a x_2:

Algorithm 1 Vytvoř jednotkový vektor

Vstup: (x_1, y_1), (x_2, y_2)

1: $m_1 \leftarrow lcm(x_1, x_2)/x_1$
2: $m_2 \leftarrow lcm(x_1, x_2)/x_2$
3: $(x, y) \leftarrow m_1 * (x_1, y_1) - m_2 * (x_2, y_2)$
4: $(x, y) \leftarrow (x, y)/y$

Výstup: (x, y), kde buď x je 0 a y je buď 1 nebo -1

Tento algoritmus je velmi jednoduchý. Zjištění nejmenšího společného násobku je potřeba k tomu, Následným dělením zjistíme kolikrát musíme x_1 a x_2 násobit, abychom tohoto násobku docílili. Tento krok samozřejmě lze optimalizovat, ale vzhledem k tomu, že zadání chce jen popis algoritmu, nebudu se zaměřovat na komplexitu a efektivita algoritmu, ale jenom na principu.

Když tedy známe m_1 a m_2, můžeme vektory od sebe odečíst a získame tak vektor, kde $x = 0$. Zbývá nám jenom y, které může být libovolné číslo. Stačí jen vydělit tento vektor y a máme výlsedný vektor, který určuje pohyb o 1 metr v jednom směru.

To dělení můžeme také pochopit tak, že všechny kroky s původními vektory vlastně dělíme y. Tedy, robot by spočítal y a následně by se pohyboval m_1 krát směrem $(x_1, y_1)/y$ a poté $-m_2$ krát $(x_2, y_2)/y$.

Pro pohyb opačným směrem stačí změnit pořadí vektorů v odečítání a stejný postup můžeme provést pro vynulování y a pohyb ve směru x.

Toto je samozřejmě celý s předpokladem, že vectory (x_1, y_1) a (x_2, y_2) nejsou lineárně závislé. Pro případ, že jsou lineárně závislé, úlohu nelze vyřešit.

Úkol 2 – Nezávislost a soustavy [3b]

Na řešení této úlohy můžeme odkázat na úkol 1. Tam jsme uvedli, že úkol 1 má řešení právě tehdy když vectory x_1 a x_2 nejsou lineárně závislý. V takovém případě by totiž vyšel vzniklý vector jako (0, 0). Stejný princip můžeme uplatnit u soustavy lineárních rovnic. Tedy, jedna rovnice je lineárně závislá na druhou, v případě že ji můžeme zcela vynulovat pomocí kroků 1-3 z algoritmu v úkolu 1.

Podobný princip jako z úkolu 1 využívá Gaussova eliminace, která je součástí úkolu 3.

Úkol 3 – Algoritmus eliminace [7b]

Lehčí varianta (za 4 body) (Gauss-Easy.cpp): Až po nějaké práci na těžší variantu si uvědomil, že toto řešení funguje jenom pro celočíselné kořeny, protože všude používám int místo float. Bohužel ale už nemám čas toto opravit, ale jediná potřebná změna která mě napadá je, že místo $gcm()$ a $lcm()$, které zrovna nefungují s floaty kvůli modula, bych mohl použít ten algoritmus, který jsem popsal v úkolu 1.

Pokus o těžší variantě byl, ale s blížícím školním čtvrtletí jsem musel pokus opustit za upřednostnění školí práce.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

int parNum;
vector<vector<int>> leftMatrix;
```

```
vector<int> rightMatrix;

int gcd(int a,int b)
{
  if (b == 0)
    return a;
  return gcd(b, a % b);
}

int lcm(int a, int b)
{
    return (a * b / gcd(a, b));
}

int reorderMatrixRows(int ind) {
    int reordered = 0;
    for (int i = parNum-1; i > -1; i--) {
        bool cond = true;
        for (int j = 0; j <= ind; j++) {
            if (leftMatrix[i][j] != 0) {
                cond = false;
                break;
            }
        }
        if (cond) {
            swap(leftMatrix[i], leftMatrix[parNum-1-reordered]);
            swap(rightMatrix[i], rightMatrix[parNum-1-reordered]);
            reordered++;
        }
    }
    return reordered;
}

// int upTo is not inclusive!!
void nullifyMatrixColumn(int ind, int upTo) {
    for (int i = ind+1; i < upTo; i++) {
        int multiple1 = lcm(leftMatrix[i][ind],
            leftMatrix[ind][ind])/leftMatrix[ind][ind];
        int multiple2 = lcm(leftMatrix[i][ind],
            leftMatrix[ind][ind])/leftMatrix[i][ind];
```

```
        for (int j = ind; j < parNum; j++) {
            leftMatrix[i][j] = leftMatrix[i][j]*multiple2 -
                leftMatrix[ind][j]*multiple1;
        }
        rightMatrix[i] = rightMatrix[i]*multiple2 -
            rightMatrix[ind]*multiple1;
    }
}

int main() {
    string fileDir = "./inOut/test";
    ifstream inputFile;
    ofstream outputFile;

    string answer;

    inputFile.open(fileDir + ".in", ios::in);

    inputFile >> parNum;

    for (int i = 0; i < parNum; i++) {
        vector<int> tempParamHolder;
        int tempInp;
        for (int j = 0; j < parNum; j++) {
            inputFile >> tempInp;
            tempParamHolder.push_back(tempInp);
        }
        leftMatrix.push_back(tempParamHolder);
        inputFile >> tempInp;
        rightMatrix.push_back(tempInp);
    }
    cout << parNum << endl;

    for (int i = 0; i < parNum; i++) {
        for (int j = 0; j < parNum; j++) {
            for (int x = 0; x < parNum; x++) {
                cout << leftMatrix[j][x] << " ";
            }
            cout << rightMatrix[j] << endl;
        }
```

```
    cout << endl;
    int reordered = reorderMatrixRows(i);
    nullifyMatrixColumn(i, parNum-reordered);
}

//If this is true, the matrix can either be parameterized or is
    unsolveable
if (leftMatrix[parNum-1][parNum-1] == 0) {
    answer = "N";
} else {
    rightMatrix[parNum-1] /= leftMatrix[parNum-1][parNum-1];
    leftMatrix[parNum-1][parNum-1] = 1;
    for (int i = parNum-2; i > -1; i--) {
        for (int j = parNum-1; j > i; j--) {
            rightMatrix[i] -= leftMatrix[i][j]*rightMatrix[j];
            leftMatrix[i][j] = 0;
        }
        rightMatrix[i] /= leftMatrix[i][i];
        leftMatrix[i][i] = 1;
    }

    for (int j = 0; j < parNum; j++) {
        for (int x = 0; x < parNum; x++) {
            cout << leftMatrix[j][x] << " ";
        }
        cout << rightMatrix[j] << endl;
    }
    cout << endl;

    answer = "J\n";
    for (int i = 0; i < parNum; i++) {
        answer += to_string(rightMatrix[i]) + " ";
    }
}


outputFile.open(fileDir + ".out", ios::out);
outputFile << answer;
outputFile.close();
```

```
}
```