Документация на тема за проект
към курса “Структури от данни и програмиране”
Том и Джери

Проектът се състои от четири класа - Coordinate, Matrix, LList, Graph.
Класът Coordinate се използва при създването на картата на стаята, защото там от файл се четат дойка кординати.
За създаването на картата на стаята се използва класа Matrix.
Класът LList е релизация на линеен списък, представен чрез една връзка.
Класът Graph е реализацията на граф чрез списък на инцидентностите.

В main() функция:

Създаваме променлива от тип Matrix m,отговаряща на данните подадени в конкретния файл, която представлява разположението на обектите в стаята. Извеждаме я на екрана. Създаваме граф g - creategraph(g, m); по данните от матрицата. Във функцията creategraph(g, m); добавяме всички елементи на матрицата като върхове в графа. След това добавяме ребра между върховете в графа. Добавянето на ребра между върховете става във for цикъла, като за всеки текущ елемент се проверяват стоиностите на неговите съседи в матрицата (горен, долен, ляв, десен), и ако някоя от тях е ‘F’ (тоест това означава наличие на мебел) между текущият елемент и този съсед ребро не се поставя. В променливата vector<LList<int>> allPaths; се записват всички възможни пътища в графа. Пътищата в графа се намират с помощта на функцията vector<LList<T>> allways(const T& a, const T& b ,Graph<T>& g,LList<T>& l), която намира всички пътища от върха a до върха b на графа g. Ако allPaths е празен, то тогава няма път от Том до Джери, в противен случай програмата изкарва всички пътища от Том до Джери, като всеки елемент на allPaths се преобразува с помощта на функцията string pathWithDirection(LList<T> l,int degree,Matrix& m); в желания формат (желания формат е например: T->N->N->E->P->E->N->J). След това програмата извежда най-късите пътища от Том до Джери отново в указания формат. Най-късите пътища се намират с помощта на фунцкията int findMinSizePath(const vector<string>& vec);, която намира и връща дължината на най-късия/най-късите път/пътища в подадения като параметър вектор. В променливата vector<LList<int>> saveMinPaths; се добавят най-късия/най-късите път/пътища, само че сега са записани като елементи на вектор от тип списък от int , защото за намирането на подробна информация ще ги използвам в този формат. Програмата очаква въвеждане на стойност от потребите, за да покаже подробна информация, за указания от потребителя най-къс път. Подробната информация за някои от най-късите пътища се намира по следния начин:

- Първо се извеждат командите, които Том трябва да въведе на дрона си. Използва се функцията printCommand(saveMinPaths[i],m); като тук командите се извеждат с пълните им имента N=North, S=South, W=West, E=East, P=Paint.
- След това се извежда информация за това колко боя е разлята по този път. Тук се използва функцията findNumberOfPaintInPath(saveMinPaths[i], m).
- Извежда се броят на завоите направени по този път, с помощта на фукнцията numberOfTurnsInPath(saveMinPaths[i],m).
- И накрая се извежда колко е дължината на пътя.

Приключва извеждането на добробна информация за най-късия път.
Програмата извежда, тези пътища от Том до Джери, които са с максимално разлята боя, но заедно с това и при тях са направени най-малко завои. Функцията

findPathsMaxPaintMinTurns(allPaths, m) ги намира, като тя изплозва vector<LList<T>>
findThisPathsMaxPaint(vector<LList<T>>& q,Matrix& m) - намира всички пътища, при които има най-много разлята боя. След това от тези пътища с най-много разлята боя се взимат и се извеждат само тези, които са с минимални завои.
За оправлението на двата дрона съм реализирала две фунцкии, тъй като не съм сигурна коя от тях точно трябва да се извежда.
Функцията void twoDrone(vector<LList<T>>& q,Matrix& m); намира измежду всички най-къси пътища, два различни пътя, такива че по тях е разлята най-много боя, което смятам че отговаря на условието в задачата, тъй като там пише двата дрона да стигнат възможно най-бързо(=> взимаме само най-късите пътища). Тя намира за всеки два най-къси пътища, колко максимално боя може да бъде разлята, и след това извежда тези два различни пътя.
А функцията void otherTwoDrone(vector<LList<T>>& q,Matrix& m); от всички пътища, по които може да се разлее максимално боя, взима ако има два такива с равна дължина, които ще стигнат най-бързо до Джери, което отново смятам че отговаря на условите на задачата.

Описание на класовете.

Coordinate.hpp

```
public:
// Конструктур по подразбиране.
    Coordinate();

// Конструктор с параметри.
    Coordinate(int,int);

// Създава двойка координати x,y от подаден като параметър string. Нужно ни е тъй като когато четем от файла информацията, четем ред по ред string елементи.
    Coordinate(string coordinates);

// Селектори - член-функции, които позволяват преглед на член-данните.
    int getX() const;
    int getY() const;

// Селектори за извеждане.
    void print() const;

private:
    int x; //
    int y;

// Функция, която преобразува дадено число от тип char към тип int
    int counvertFromCharToInt(const char*) const;
```

Matrix.hpp

```
public:
// Канонично представяне.
// Конструктор по подразниране.
    Matrix();
```

```
// Създава матрица по данните от подадения като параметър файл.
    Matrix(char* pathFile);

// Деструктор
    ~Matrix();

// Селектори за достъп до данните.
    Coordinate getSize() const;
    Coordinate getPositionTom()const;
    Coordinate getPositionJerry()const;
    char** getMatrix() const;

// Селектор за извеждане.
    void print() const;

private:
    char** matrix;
// size е променлива от тип Coordinate която се използва за размерността на
матрицата.
    Coordinate size;
// Координатите на Том.
    Coordinate tom;
// Координатите на Джери.
    Coordinate jerry;

Структура node. Елемтите на един свързан списък.
template <typename T>
struct node{
    T inf; // Информационна част.
    node<T>* link; // Адресна част.
};
```

LList.hpp

```
public:
// Конструктор по подразбиране.
    LList();

// Дестурктор.
    ~LList();

// Копиращ конструктор.
    LList(const LList&);

//Операция за присвояване.
    LList& operator=(const LList&);

// Инициализиране на итератор.
    void iterStart(node<T>* = nullptr);

// Установяване на итератора в следващата позиция.
    node<T>* iter();
```

```
// Добавя елемент в края на списъка.
    void toEnd(const T& x);

// Изтрива елемент от списъка след указан елемент.
    void deleteAfter(node<T>* p,T& x);

 // Изтрива указан елемент.
    void deleteElem(node<T>* p,T& x);

// Извежда съдържанието на списъка.
    void print() const;

// Намира дължината на списък.
    int findLen() const;

// Проверява дали елемент участва в списъка.
    bool member(const T& x,LList<T>& l);

// Изтрива последния елемент от списък.
    void deleteLast(LList<T>& l);

private:
    node<T>* start;
    node<T>* end;
    node<T>* current;

// Копиране на списък.
    void copyList(const LList& other);

// Изтриване на списък.
    void deleteList();
};
```

Graph.hpp

```
public:
// Включва елемента a като връх на неявен граф.
    void addTop(const T& a);

// Изключва върха a на неявен граф.
    void deleteTop(const T& a);

// Добавя ребро от върха a към върха b на неявен граф.
    void addRib(const T& a, const T& b);

// Изтрива ребро от върха a към върха b.
    void deleteRib(const T& a,const T& b);

// Проверява дали a е връг в графа.
    bool top(const T& a);

// Проверява дали има ребро между a и b в графа.
    bool rib(const T& a,const T& b);
```

```
// Проверява дали неявния граф е празен.
    bool empty() const;

// Намира указател към върха a на графа.
    node<T>* point(const T& a);

// Връща списък от върховете на неявения граф.
    LList<T> vertexes();

// Извежда графа.
    void print();

private:
    LList< LList<T> > g;
```