Aufgabe 1: Stromrallye Teilaufgabe a)

Teilnahme-Id: 53280

Bearbeiter/-in dieser Aufgabe:
Raphael Gaedtke

12. April 2020

Inhaltsverzeichnis

1 Lösungsidee

1.1 Allgemeines

Definition 1 *Es sei k die Kantenlänge des Spielfeldes, b die Anzahl der Batterien auf dem Feld (inklusive der Bordbatterie des Roboters) und s die Anzahl der Schritte, die der Roboter machen muss, um alle b Batterien zu entladen. Dabei gilt $b, k, s \in \mathbb{N}$.*

Definition 2 *Die einzelnen Batterien werden mit $B_1, B_2, ..., B_n$ mit $n \in \mathbb{N}$ bezeichnet, wobei die Bordbatterie den höchsten Index $n = b$ haben soll. Eine Batterie B_n wird definiert durch ihre Koordinaten x_n und y_n und hat einen (variablen) Ladungswert, lässt sich also durch das Zahlentupel (x_n, y_n, l_n) mit $x_n, y_n, l_n \in \mathbb{N}$ beschreiben.*

Da bei jedem Schritt des Roboters die Ladung einer Batterie (der aktuellen Bordbatterie) um 1 sinkt, muss die Anzahl der Schritte, die der Roboter zum Entladen aller Batterien machen muss, gleich der Summe aller Batterieladungen entsprechen. Es gilt also:

$$s = \sum_{i=1}^{b} l_i \tag{1}$$

1.2 Bestimmen der Lösung

Zur Lösungsfindung werden verschiedene Wege auf dem Spielfeld ausprobiert. Um die Anzahl der zu testenden Wege zu verringern, werden dabei nicht Schrittfolgen berechnet, sondern Wege als eine Liste von besuchten Batterien gedeutet.

Neben der Reihenfolge der besuchten Batterien (wobei eine Batterie auch mehrmals in der Liste vorkommen kann) spielt auch die Anzahl der zwischen diesen Batterien gemachten Schritte eine Rolle (allerdings nicht die genaue Schrittfolge!).
Diese können die Mindestlänge eines Pfades vergrößern oder sogar verhindern, dass ein Pfad zwischen zwei Batterien existiert.

Definition 3 *Im Folgenden steht ein „Weg“ für eine besagte Abfolge von besuchten Batterien, während ein „Pfad“ die Schrittfolge zwischen zwei Batterien meint. Wichtig ist vor allem die „Pfadlänge“, also die Anzahl an Schritten zwischen den Batterien. Dabei kann die Zielbatterie eines Pfades auch die gleiche Batterie wie die Startbatterie des Pfades sein. Dabei bezeichnet $m_{k,l}$ die Mindestpfadlänge zwischen den Batterien B_k und B_l. Ein Pfad zwischen zwei Batterien kann dabei nicht über ein anderes Batteriefeld führen.*

Die Pfadlängen selbst sind variabel, der Roboter kann sie verlängern, indem er beispielsweise einen Schritt vor und wieder zurück macht. Für die Pfadlängen müssen allerdings zwei Regeln gelten:

1. Die Parität der Länge eines Pfades bleibt immer gleich, egal wie stark der Roboter ihn verlängert. Dies kann bewiesen werden, indem das Spielfeld wie ein Schachbrett in schwarze und weiße Felder aufgeteilt wird. Bei jedem Schritt wechselt der Roboter die Farbe, benötigt bei einem Pfad zwischen gleichfarbigen Feldern also immer eine gerade und bei einem Pfad zwischen ungleichfarbigen Feldern immer eine ungerade Anzahl von Schritten.

2. Jeder Pfad muss eine Mindestpfadlänge haben. Fallen Start- und Endpunkt eines Pfades zusammen, ist die Mindestlänge des Pfades 0.

Unter Berücksichtigung dieser Regeln werden alle möglichen Wege ausprobiert, bis entweder eine Lösung gefunden oder eine Lösung als unmöglich erkannt ist.

1.3 Bestimmen der Mindestpfadlängen

Um alle möglichen Längen eines Pfades auszuprobieren, muss die Mindestpfadlänge $m_{k,l}$ jedes Pfades bekannt sein. Diese lässt sich theoretisch wie folgt berechnen:

$$m_{k,l} = |x_k - x_l| + |y_k - y_l| \tag{2}$$

Gleichung 2 vernachlässigt allerdings die anderen auf dem Feld liegenden Batterien, über die der Pfad nicht führen darf. So können andere Batterien die Mindestpfadlänge verlängern oder die Existenz eines Pfades zwischen zwei Batterien gänzlich verhindern.

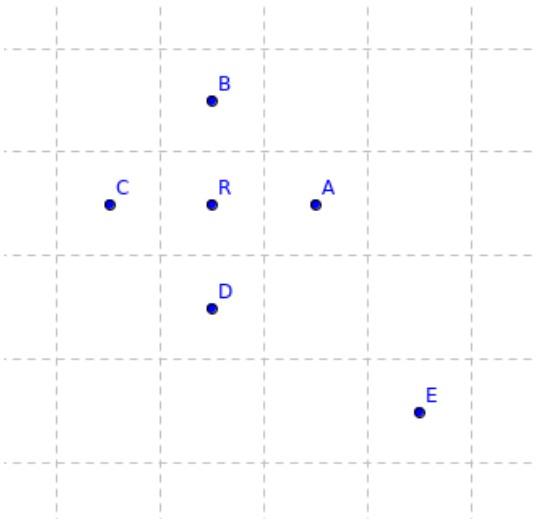

Abbildung 2: Wegen der Batterien A-D existiert kein Pfad zwischen den Batterien R und E.

Existiert kein Pfad zwischen zwei Batterien, wird er im Algorithmus wie ein Pfad der Länge $2s$ behandelt. Diese Eigenschaft verhindert ebenfalls, dass der Roboter ihn benutzen kann.

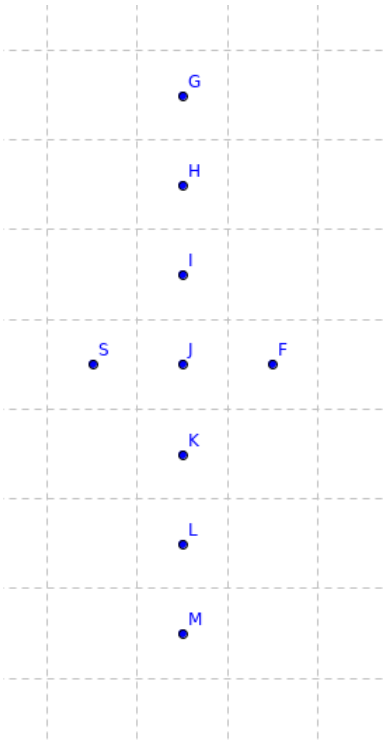

Abbildung 1: Der Pfad zwischen Batterie S und Batterie F wird durch die Batterien G-M verlängert.

Um die Mindestpfadlängen zu bestimmen, geht das Programm durch alle Paare von Batterien. In jedem dieser Paare wird eine Batterie als Start- und eine andere als Zielpunkt eines Pfades verwendet. Dann werden alle Pfade vom Startknoten ausgehend ausprobiert. Dabei gibt es zwei mögliche Fälle:

1. Der Zielpunkt wird mit $s-1$ oder weniger Schritten erreicht. Dann wird diese Zahl als Mindestlänge des Pfades gespeichert.
2. Der Zielpunkt wurde nach $s-1$ Schritten noch nicht erreicht. Dann muss die Mindestpfadlänge auf einen Wert von $2s$ gesetzt werden, weil kein Wert mit einer Länge von s oder größer vom Roboter vollständig begangen werden kann (Die Länge s ist für einen Pfad nicht möglich, weil mindestens eine Ladung von 1 in mindestens einer weiteren Batterie platziert sein muss - nämlich der Zielbatterie).

Nach diesem System werden alle Mindestpfadlängen berechnet.

2 Umsetzung

Die Lösungsidee wird in C++ umgesetzt.

2.1 Modellierung

Das Netz von Batterien kann auch als Graph gesehen werden, bei dem die Batterien die Ecken und die Pfade die Kanten sind.
Davon ausgehend werden die Knoten -also die Batterien- in einem Array des eigens definierten Datentyps „battery“ gespeichert. Dieser als „struct“ definierte Datentyp hat neben Koordinaten und der aktuellen Ladung der Batterie auch ein Array des Datentyps „path“ als Membervariable.
Dieses Array speichert die Mindestlängen der Pfade zu den anderen Batterien. Somit wird der Graph modelliert, in dem jeder Knoten gewissermaßen einen Teil der Inzidenzmatrix und einer Gewichtungsmatrix -nämlich jeweils eine Zeile- speichert.
Ein „Weg“ wird dann als ein String gespeichert, der aus vierstelligen Zahlen besteht. Diese vierstelligen Zahlen repräsentieren die Indizes der Batterien, die in der im String festgelegten Reihenfolge besucht werden. Zwischen diese vierstelligen Zahlen werden dann noch weitere vierstellige Zahlen eingeschoben, die

jeweils angeben, wie viele Schritte zwischen der Batterie vor und nach dieser Zahl gemacht werden sollen. So können alle Wege kodiert werden (größere benötigte Zahlen als 9999 werden allerdings nicht erwartet). Die in diesem String enthaltene Information wird dann auch als Ausgabe genutzt - das Programm gibt also *keine* genau Schrittfolge aus! Dies hat den Grund, dass die Anzahl der ausgegebenen Schritte sonst zu groß wäre und es onehin viele Möglichkeiten für solche Schrittefolgen gibt, die der Nutzer ohne große Mühe aus den besuchten Batterien konstruieren kann.

2.2 Berechnen der Mindestpfadlängen

Zum Berechnen der Mindestpfadlängen wird eine rekursive Funktion namens „min_ length()“ genutzt, die einen Container mit derzeit erreichten Feldern bearbeitet. Für jedes Feld in diesem Container wird überprüft, welche seiner Nachbarfelder betreten werden könne, ohne ein Batteriefeld zu erreichen oder ein Feld mehrmals zu betreten.
Alle Felder, auf die dies zutrifft, werden in einen neuen Container verschoben. Ist eines der Nachbarfelder der Felder in diesem Container das Zielfeld, wird die dadurch bestimmte Pfadlänge ausgegeben. Ist dies nicht der Fall, ruft die Funktion sich selbst auf und übergibt ihrer neuen Instanz den neuen Container von Feldern.
Dies wird durchgeführt, bis entweder das Zielfeld erreicht wurde oder ein Zähler für die Zahl der Funktionsaufrufe die durch s gegeben Schranke überschreitet.
Die Funktion „min_ length()“ berechnet dabei nur die Mindestpfadlänge für ein einzelnes Batteriepaar und wird in der Funktion „eva_ paths()“ für alle Batteriepaare ausgeführt. Der anfänglich übergebene Container beinhaltet dann nur das Startfeld.

2.3 Bestimmung einer Lösung

Auch zur Berechnung eines Weges zum Entleeren aller Batterien wird eine rekursive Funktion namens „solve()“ verwendet. Anders als die oben beschrieben Funktion zur Berechnung von Mindestpfadlängen, speichert diese keine erweiterten Positionen, sondern Strings, die die bisherigen Wege kodieren. Im weiteren Verlauf werden diese Strings dann erweitert.
Abgesehen von dieser Tatsache ist die Funktion „solve()“ eng an die Funktion „min_ length()“ angelehnt.

3 Beispiele

Ausgabe für die Datei stromrallye0.txt:

Für die gegeben Batterieverteilung gibt es eine Lösung!
Von seinem Startfeld aus bewegt sich der Roboter folgendermaßen:
Der Roboter läuft zur Batterie auf dem Feld (5,4) mit 3 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,1) mit 3 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,4) mit 3 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,2) mit 6 Schritten.
Die verbleibende Restladung beträgt 2.

Ausgabe für die Datei stromrallye1.txt:

Für die gegeben Batterieverteilung gibt es eine Lösung!
Von seinem Startfeld aus bewegt sich der Roboter folgendermaßen:
Der Roboter läuft zur Batterie auf dem Feld (1,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (1,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,9) mit 1 Schritten.

Der Roboter läuft zur Batterie auf dem Feld (2,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (2,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (3,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (4,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (5,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (6,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (7,10) mit 1 Schritten.

Der Roboter läuft zur Batterie auf dem Feld (8,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (8,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,1) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (9,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,10) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,9) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,8) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,7) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,6) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,5) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,4) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,3) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,2) mit 1 Schritten.
Der Roboter läuft zur Batterie auf dem Feld (10,1) mit 1 Schritten.
Die verbleibende Restladung beträgt 1.

Ausgabe für die Datei stromrallye3.txt:

Für die gegebene Batterieverteilung gibt es keine Lösung.

Ausgabe für die Datei stromrallye4.txt:

Für die gegeben Batterieverteilung gibt es eine Lösung!
Von seinem Startfeld aus bewegt sich der Roboter folgendermaßen:
Die verbleibende Restladung beträgt 20.

Notiz: Auch für das „Auslaufen“ der Ladung wird keine genaue Schrittfolge vorgegeben.

Ausgabe für die Datei stromrallye5.txt:

Für die gegebene Batterieverteilung gibt es keine Lösung.

4 Quellcode

Die main-Funktion:

```
int main()
{
    //The global variables are declared like in the documentation:
    int b; //Number of batteries on the field (the battery of the robot is included)
    int k; //Length of the board
    int l; //sum of the charges of all batteries

```

```
    battery* batteries;
    //Array for the batteries (the battery of the robot has the highest index
    init_vars(&b, &k, &l, batteries); //Initializes the variables

    vector<vector<bool>> board(k, vector<bool> (k));
    //array standing for the board, every field is true if a battery is lying on it
    init_board(batteries, &board, k, b); //Initialize the board-variable
    eva_paths(batteries, b, board, l, k);
    //evaluate the lengths of the shortest paths between the batteries

    string solution = ""; //This string stores the solution if such a solution exists
    bool found_solution = false; //Do we already have a solution?
    cout<<endl<<"Die Berechnung der Loesung wird gestartet..."<<endl;
    solve(solution, &solution, batteries[b-1].l, b-1, batteries, b, l, &found_solution);
    //Evaluates the solution
    cout<<"Die Loesung wurde berechnet."<<endl;
    cout<<"Geben Sie den Namen der Ausgabedatei an: ";
    string name_output_file;
    cin>>name_output_file;
    ofstream output_file;
    output_file.open(name_output_file, ios_base::out);
    output_to_file(&output_file, found_solution, solution);

    return 0;
}
```

Die Definitionen der benötigten Datentypen:

```
struct path //An object which stores the relation between two batteries
{
    int index; //index of the battery which is the goal of the path
    int min_length; //minimal length
};

struct battery //the batteries
{
    int x;
    int y; //coordinates of the battery
    int l; //charge of the battery
    path* paths; //relations to the other batteries
};

enum direction{Up = 0, Down, Right, Left, Undefined};
//The four possible directions and a state for the first step

struct position //An object storing a position on the board
{
    int x; //Starting with (0,0) up to (b-1, b-1)
    int y;
    direction origin; //There the robot came from
};
```

Die beiden Funktionen zum Berechnen der Mindestpfadlängen:

```
int min_length(position goal, vector<position> positions,
vector<vector<bool>> board, int length, int s, int k)
{
    //This recursive function evaluates the minimal length of a path between two batteries
    if(length == 1 && goal.x == positions[0].x && goal.y == positions[0].y)
    {
        return 0; //The path begins and ends with the same battery
    }
    if(length == 1 && ((goal.x == positions[0].x+1 && goal.y == positions[0].y) ||
           (goal.x == positions[0].x-1 && goal.y == positions[0].y) ||
           (goal.x == positions[0].x && goal.y == positions[0].y+1) ||
           (goal.x == positions[0].x && goal.y == positions[0].y-1)))
           {
               return 1; //The goal is next to the starting position
           }
```

```
    /*for(int n = 0; n < positions.size(); n++)
    {
        cout<<endl<<"Feld Nr. "<<n+1<<": ("<<positions[n].x<<", "<<positions[n].y<<")";
    }
    cin.get();*/
    vector<position> new_positions (0); //Vector for the new positions

    for(int n = 0; n < positions.size(); n++)
    {
        //Adding all reachable fields for the given positions
        if(positions[n].origin != Up &&
        positions[n].y > 1 && !board[positions[n].x-1][positions[n].y-2] )
        {
            position position1;
            position1.x = positions[n].x;
            position1.y = positions[n].y-1;
            board[positions[n].x-1][positions[n].y-2] = true; //Do not visit one field twice
            position1.origin = Down;
            new_positions.push_back(position1);
        }
        if(positions[n].origin != Down &&
        positions[n].y <= k && !board[positions[n].x-1][positions[n].y])
        {
            position position2;
            position2.x = positions[n].x;
            position2.y = positions[n].y+1;
            board[positions[n].x-1][positions[n].y] = true;
            position2.origin = Up;
            new_positions.push_back(position2);
        }
        if(positions[n].origin != Left &&
        positions[n].x > 1 && !board[positions[n].x-2][positions[n].y-1])
        {
            position position3;
            position3.x = positions[n].x-1;
            position3.y = positions[n].y;
            board[positions[n].x-2][positions[n].y-1] = true;
            position3.origin = Right;
            new_positions.push_back(position3);
        }
        if(positions[n].origin != Right &&
         positions[n].x < k && !board[positions[n].x][positions[n].y-1])
        {
            position position4;
            position4.x = positions[n].x+1;
            position4.y = positions[n].y;
            board[positions[n].x][positions[n].y-1] = true;
            position4.origin = Left;
            new_positions.push_back(position4);
        }
    }
    //Now are all fields which can be reached with "length" steps in the vector "new_positions"

    if(new_positions.size() == 0 || length > s)
    {
        //No new field can be reached or the path is too long!
        return 2*s;
    }

    //Do we have reached the goal?
    for(int n = 0; n < new_positions.size(); n++)
    {
        if((goal.x == new_positions[n].x+1 && goal.y == new_positions[n].y) ||
           (goal.x == new_positions[n].x-1 && goal.y == new_positions[n].y) ||
           (goal.x == new_positions[n].x && goal.y == new_positions[n].y+1) ||
           (goal.x == new_positions[n].x && goal.y == new_positions[n].y-1))
        {
            //We have reached the goal!
            return length+1;
        }
    }

    //If we reach this part of the function, the goal is not reached yet
```

```
    return min_length(goal, new_positions, board, length+1, s, k);
}

void eva_paths(battery*& batteries, int b, vector<vector<bool>> board, int s, int k)
//evaluates the minimal lengths of the paths between the batteries
{
    cout<<endl<<"Berechnen der Mindestlaengen zwischen den Pfaden...";
    for(int n = 0; n < b-1; n++) //goes through all batteries except the battery of robot
    {
        batteries[n].paths = new path[b-1];//New path variable
        for(int a = 0; a < b-1; a++)
        {
            batteries[n].paths[a].index = a; //Index of the goal

            position goal; //The battery we want to reach
            goal.x = batteries[a].x;
            goal.y = batteries[a].y;
            goal.origin = Undefined;
            vector<position> positions(0); //Actual position
            position actual_position;
            actual_position.x = batteries[n].x;
            actual_position.y = batteries[n].y;
            actual_position.origin = Undefined;
            positions.push_back(actual_position);

            batteries[n].paths[a].min_length = min_length(goal, positions, board, 1, s, k);
            //length of the path
        }
    }
    //For the battery of the robot:
    batteries[b-1].paths = new path[b-1];
    for(int n = 0; n < b-1; n++) //Evaluates the paths for the robot
    {
        batteries[b-1].paths[n].index = n;

        //Works like the initialization of the other paths above
        position goal;
        goal.x = batteries[n].x;
        goal.y = batteries[n].y;
        goal.origin = Undefined;
        vector<position> positions(0);
        position actual_position;
        actual_position.x = batteries[b-1].x;
        actual_position.y = batteries[b-1].y;
        actual_position.origin = Undefined;
        positions.push_back(actual_position);

        batteries[b-1].paths[n].min_length = min_length(goal, positions, board, 1, s, k);
    }
    //The reason for separating the battery of the robot from the others is the fact,
    //that at the fields of the other batteries will be a battery every time.
    cout<<endl<<"Die Mindestlaengen der Pfade wurden den Batterien sind berechnet."<<endl;
}
```

Die Funktion zum Finden des auszugebenden Weges:

```
void solve(string solution, string* StringSavingSolution, int charge, int index,
battery* batteries, int b, int l, bool* found_solution)
//This recursive function tries to find a solution
{
    /*
    For storing the path of the robot, we encode this path in a string.
    This string stores every visited battery field, represented by the battery lying on it.
    Every field is clearly named after its index in the batteries-array. The index of a battery
    is written as a number with 4 digits, situations with more than
    9999 batteries are not expected. After the index of a battery, a 4-digit-number
    stores the number of steps added to the minimal length of the path between the two batteries.
    The fast 4-digit-number in the string stores the charge of the battery of
    the robot after the last battery exchange.
    If the situation can not be solved, the string remains empty.
    */
```

```
    if(l > charge) //There are still loaded batteries the battery of the robot
    {
        //The functions goes through the possible paths
        for(int n = 0; n < b-1 && !*found_solution; n++)
        {
            //The functions goes through all possible lengths of paths
            for(int a = batteries[index].paths[n].min_length; a <= charge && !*found_solution; a += 2)
            {
                if(a == 0 && charge > 1)
                {
                    a = 2; //Avoid remaining on one field
                }
                else if((a == 0 && charge == 1) || charge == 0)
                {
                    break; //Avoids -1 as charge
                }
                solution += index_to_4digitstring(batteries[index].paths[n].index);
                solution += index_to_4digitstring(a); //adds the necessary data to the string
                int new_charge = batteries[batteries[index].paths[n].index].l;
                // new charge of the battery of the robot
                int l_beforeexchange = new_charge; //Saves the charge of the visited battery
                batteries[batteries[index].paths[n].index].l = charge - a; //Exchange the battery
                //Now the function calls herself:
                solve(solution, StringSavingSolution, new_charge,
                batteries[index].paths[n].index, batteries, b, l-a, found_solution);
                solution = solution.substr(0, solution.length()-8);
                batteries[batteries[index].paths[n].index].l = l_beforeexchange;
                //cancel the changes of the steps before

            }
        }
    }
    else
    {
        //We found a solution!!!
        solution += index_to_4digitstring(charge);
        *found_solution = true;
        *StringSavingSolution = solution;
    }
}
```