Aufgabe 5: Das ägyptische Grabmal

Team-ID: 00153

Team-Name: ByteBusters

Bearbeiter/-innen dieser Aufgabe:
Leonard Beinlich

18. November 2024

Inhaltsverzeichnis

1 Lösungsidee **1**

2 Klassen und ihre Funktionen **2**
2.1 Klasse: Block . . . 2
2.2 Klasse: ListBlocks . . . 3
2.3 Klasse: ListeAktionen . . . 3
2.4 Klasse: Aktion . . . 3

3 Umsetzung **4**
3.1 Intervalle der Blöcke einlesen . . . 4
3.2 Finden des schnellsten Wegs . . . 4

4 Beispiele **4**
4.1 Diskussion zu Beispielen . . . 5

5 Quellcode **5**
5.1 Backtracking-Algorithmus . . . 5
5.2 Aktion . . . 5
5.3 Block . . . 6
5.4 ListBlocks . . . 6
5.5 ListeAktionen . . . 7
5.6 DateiEinlesen . . . 7

1 Lösungsidee

Die Lösungsidee basiert auf einer Art Backtracking-Algorithmus. Um den schnellstmöglichen Weg zu Finden würde man zuerst davon ausgehen das man an jedem Steinblock einfach wartet bis sich dieser öffnet jedoch muss man dabei die Option beachten das sich Blöcke auch wieder schließen können. Daraus resultieren jede vergangene Minute drei Szenarien:

1. Der Block vor einem Bleibt geschlossen und der Block über einem Geöffnet
2. Der Block Vor einem Öffnet sich während der Block über einem offen bleibt
3. Der Block Über einem schließt sich während man darunter steht

Diese drei Möglichkeiten lassen sich wie folgt darstellen:

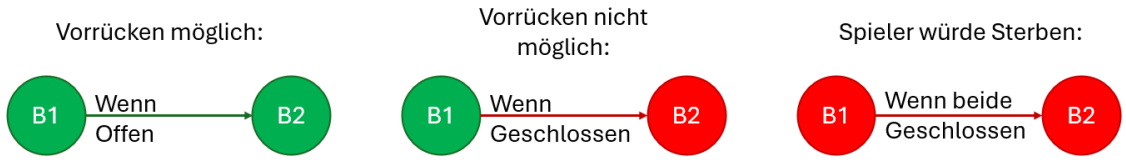

Abbildung 1: Möglichkeiten

Die Resultate für die ersten beiden Szenarien sind weitestgehend selbst erklärend. Während man bei dem Ersten einfach einen Schritt zum Nächsten Block macht wartet man beim Zweiten einfach bis sich eins der andern Beiden Szenarien ergibt. Sollte aber Szenario drei eintreten muss die Zeit auf den Punkt zurück gedreht werden bevor der Block betreten wurde und es muss gewartet werden dass sich nicht nur der Nächste sondern die nächsten beiden bzw. bei mehrfachem zerquetscht werden die nächsten n Blöcke gleichzeitig offen sind.

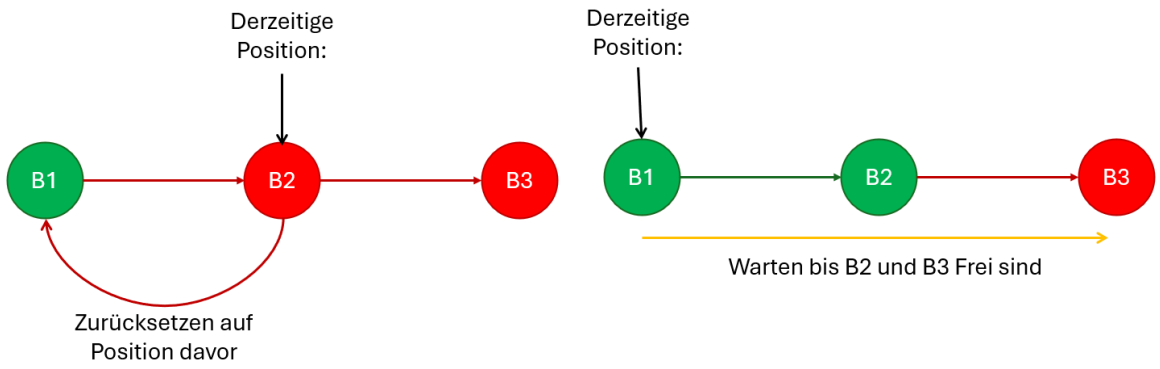

Abbildung 2: Zeit zurückdrehen Beispiel

Das in der Abbildung gezeigt Szenario kann bis zum Start Punkt an welchem kein Block mehr über dem Spieler ist erweitert werden.

2 Klassen und ihre Funktionen

Hier stehen alle für die Ausführung des Programm Essentiellen Methoden und Attribute

2.1 Klasse: Block

Stellt die einzelnen Blöcke mit ihren Zeitintervallen da.

Attribute:

interval: Gibt die Intervalle an in welchen sich der Block öffnet und schließt

closed: Gibt mit einem Wahrheitswert an ob der Block offen oder geschlossen ist (False = Geschlossen, True = offen)

Methoden:

checkStatues(time): Prüft, ob der Block zu der derzeitigen Zeit (time) offen oder geschlossen ist

2.2 Klasse: ListBlocks

Verwaltet die Objekte der Klasse Block

Attribute:

listOfBlocks: Eine Liste in welcher alle Blöcke gespeichert werden

Methoden:

addBlock(interval): Fügt der Liste einen neuen Block mit dem gegebenen Intervall hinzu

checkStatuesFromTo(start,end,time): Prüft ob alle Blöcke in einem Bereich zwischen 'start' und 'end' zu einem Zeitpunkt (time) offen sind gibt einen Wahrheitswert zurück

checkStatuesAt(postion,time): Prüft ob ein Block an einer bestimmten Postion (position) zu einem Zeitpunkt (time) offen oder geschlossen ist

2.3 Klasse: ListeAktionen

Verwaltet eine Liste mit Objekten der Klasse Aktion

Attribute:

list: Eine Liste, welche alle Objekte der Klasse Aktion enthält

Methoden:

extendList(): Fügt der Liste eine weiteres Objekt der Klasse Aktion hinzu und initialisiert dieses mit einer Wartezeit von 1 und der aktuellen Position

popLastElement(): Löscht das letzte Element aus der Liste und gibt dieses zurück

getTime(): gibt die bisher verstrichene Zeit zurück in dem die gespeicherten wartezeiten aller Aktionen in der Liste zusammengezählt werden

getPostione(): Gibt die aktuelle Position zurück welche durch das zusammenzählen aller Bewegungen welche in den Aktionen gespeichert sind ermittelt wird

2.4 Klasse: Aktion

Stellt die einzelnen Aktionen da

Attribute:

warten: Zeit welche gewartet wurde

bewegt: Strecke welche zurückgelegt wurde

zuWarten: Gibt an wie viele Blöcke von der Position der Aktion offen sein müssen bevor eine Bewegung möglich ist

positon: Derzeitige Position der Aktion

Methoden:

move(ownTime): versucht eine Bewegung durchzuführen verändert warten solange bis entweder eine Bewegung möglich ist oder durch die Methode 'checkForKill' festgestellt wurde das ein vorankommen unmöglich ist bei einer erfolgreichen Bewegung wird 'bewegt' zurückgegeben und bei einem Tod wird 'kill' zurückgegeben

checkForKill(time): Überprüft ob die Aktion an der aktuellen Position und zum Aktuellen Zeitpunkt (time) zerquetscht wird. Bei einem Tod wird True und bei keinem False zurückgegben

setBack(): Setzt warten und bewegt auf 0 zurück

3 Umsetzung

3.1 Intervalle der Blöcke einlesen

Zu beginn des Python-Programms wird eine Textdatei eingelesen welche im Explorer ausgewählt wird bei welcher mit Ausnahme der ersten Zeile alle Zeilen getrennt voneinander in einer liste gespeichert Werden dann wird für jedes Element dieser Liste ein Block durch die Methode 'addBlock' der Klasse 'ListBlocks'in einer Liste (ListOfBlocks) gespeichert welche von dem Objekt 'blöcke' welche ein Objekt der Klasse 'ListBlocks' ist gespeichert

3.2 Finden des schnellsten Wegs

Unser Backtracking-Algorithmus wird in der Methode ausführen definiert. Die Methode ausführen ist die Methode welche ausgeführt werden muss um das ganze Programm durchzuführen. Zu beginn wird unserer Liste mit Aktionen eine Aktion (Objekt der Klasse 'Aktion') hinzugefügt. Nun wierd eine Schleife Ausgeführt welche Solange läuft bis die Position (bekommen durch 'getPosition') mit der Anzahl der Blöcke übereinstimmt innerhalb der Schleife wird bei dem Letzten Objekt der Liste mit Aktionen die Methode für die Bewegung 'move' ausgeführt. In der Methode 'move' wird zu beginn Überprüft ob nur ein oder mehrere Blöcke weit bewegt werden muss. Sollte es nur einer sein wird solange die Zeit um eins erhöht bis die Nächste Block offen ist (checkStatuesAt) oder bis der Todesfall durch das schließen des Blocks an der aktuellen Position eintritt (checkForKill) Im Fall eines Todes wird dann 'kill' zurückgegeben und im Fall das ein Vorrücken möglich ist wird 'bewegt' zurückgegbn. Sollte Mehr als ein Block weit gegangen werden müssen wird das selbe gemacht nur das hier die Nächsten Blöcke überprüft werden welche abzuwarten sind (z.B. bei zuWarten = 2 müssen die Nächsten beiden offen sein) auch hier wird nach Abschluss entweder 'kill' oder 'bewegt' zurückgeben. Sollte bei der Bewegung 'kill' zurückgegeben werden wird das letzte Objekt der Aktionsliste gelöscht und das nun letzte Objekt wird zurückgesetzt (setBack) wobei die abzuwartenden Blöcke mit denen des gelöschten Objekts addiert werden. Sollte 'bewegt' zurückgegeben werden wird die Liste um eine Weitere Aktion ergänzt. Sobald dann die Position mit der Menge an Blöcken Übereinstimmt ist der kürzeste Weg gefunden. Und das Ergebnis wird ausgeben

4 Beispiele

grabmal0.txt: Der Weg der Gegangen werden muss lautet
Warte für 5 Minuten. Gehe dann 1 Blöcke weit
Warte für 3 Minuten. Gehe dann 1 Blöcke weit
Warte für 4 Minuten. Gehe dann 1 Blöcke weit

grabmal1.txt: Der Weg der Gegangen werden muss lautet
Warte für 51 Minuten. Gehe dann 5 Blöcke weits

grabmal2.txt: Der Weg der Gegangen werden muss lautet
Warte für 510000 Minuten. Gehe dann 5 Blöcke weit

grabmal3.txt: Der Weg der Gegangen werden muss lautet
Warte für 22 Minuten. Gehe dann 2 Blöcke weit
Warte für 0 Minuten. Gehe dann 1 Blöcke weit
Warte für 0 Minuten. Gehe dann 1 Blöcke weit
Warte für 0 Minuten. Gehe dann 1 Blöcke weit
Warte für 0 Minuten. Gehe dann 1 Blöcke weit
Warte für 13 Minuten. Gehe dann 4 Blöcke weit

grabmal4.txt: Der Weg der Gegangen werden muss lautet
Warte für 5299924 Minuten. Gehe dann 10 Blöcke weit

grabmal5.txt: Es wird nichts Zurückgegeben da die Berechnung zu lange dauert.

4.1 Diskussion zu Beispielen

Der Algorithmus liefert bei leichten Beispielen zumindest augenscheinlich das richtige Ergebnis jedoch gibt es ein paar Probleme:

- Das Problem an den Ergebnissen bei den Beispielen ist das sie nur schwer nachzuprüfen sind daher kann nicht das Ideale Ergebnis nicht einfach überprüft werden.
- Bei zu großen Beispielen wie grabmal5.txt kommt das Programm auf keine Antwort da es zu lange dauert.

Für Zukünftige Verbesserungen:

- Das Programm sollte auf ein regelmäßiges durch Listen durchgehen verzichten da dies zulange dauert

5 Quellcode

5.1 Backtracking-Algorithmus

```
def ausf hren():

    einlesenBl cke()

    liste.extendList()
    #Eigentlicher Algoritmus starte hier
    while(liste.getPosition() != len(bl cke.listOfBlocks)):
        time = liste.getTime()
        t = liste.list[-1].move(time)
        if t == "kill":
            a = liste.list[-1]
            liste.list.pop()
            liste.list[-1].zuWarten = liste.list[-1].zuWarten + a.zuWarten
            liste.list[-1].setBack()

        elif t == "bewegt":
            liste.extendList()

    liste.list.pop()
```

5.2 Aktion

```
from ListeBloecke import ListBlocks

class Aktion():
    def __init__(self,zuWarten,position):
        self.warten = int(0)
        self.bewegt = int(0)
```

```
        self.zuWarten = int(zuWarten)
        self.position = int(position)
        pass

    def move(self,ownTime):

        if self.zuWarten == 1:
            while ListBlocks.checkStatusAt(self.position,ownTime) != True:
                if self.checkForKill(ownTime) == True:

                    return "kill"
                self.warten = self.warten + int(1)
                ownTime = ownTime + 1
        else:
            end = self.zuWarten + self.position
            while ListBlocks.checkStatusFromTo(self.position,end,ownTime) != True :

                if self.checkForKill(ownTime) == True:

                    return "kill"
                self.warten = self.warten + int(1)
                ownTime = ownTime + 1
        self.bewegt = self.zuWarten
        return "bewegt"

#return True wenn der Block zerdr cks und False wenn nicht.
    def checkForKill(self,time):
        if self.position == 0:
            return False
        else:
            if ListBlocks.checkStatusAt(self.position-1,time) == True :
                return False
            else:
                return True

    def setBack(self):
        self.warten = 0
        self.bewegt = 0
```

5.3 Block

```

class Block():
    def __init__(self, interval):
        self.interval = int(interval)
        self.closed = True

#    berprft    ob der Block gerade geschlossen oder offen ist
    def checkStatues(self,time):
        if time == 0:
            return False
        else:
            return ((time)//self.interval)%2 == 1
```

5.4 ListBlocks

```
import Block as bl

class ListBlocks:
    listOfBlocks = []

    def __init__(self) -> None:
        pass
```

```

    def addBlock(self,interval):
        block = bl.Block(interval)
        self.listOfBlocks.append(block)


    @classmethod
    def checkStatusFromTo(self,start,end,time):
        # if time == 0:
        #     return False
        x = end  - start
        for i in range(x):
            if self.listOfBlocks[start + i].checkStatues(int(time)) == False:

                return False
        return True


    @classmethod
    def checkStatusAt(self,position,time):
        return self.listOfBlocks[position].checkStatues(time)


    def printList(self):
        for i in range(len(self.listOfBlocks)):
            print(self.listOfBlocks[i].interval)
```

5.5 ListeAktionen

```

class ListeAktionen:
    def __init__(self):
        self.list = []
        # self.extendList()

    def extendList(self):
        import Aktion as ak
        a = ak.Aktion(int(1),self.getPosition())
        self.list.append(a)

    def popLastElement(self):
        return self.list.pop()

    def getTime(self):
        time = int(0)
        for i in range(len(self.list)):
            time = time + self.list[i].warten
        return int(time)


    def getPosition(self):
        position = int(0)
        for i in range(len(self.list)):
            position = position + self.list[i].bewegt
        return int(position)
```

5.6 DateiEinlesen

```
import tkinter as tk
from tkinter.filedialog import askopenfilename
tk.Tk().withdraw()

```

```
7 def lese_datei_ohne_erste_zeile():
      datei_name = askopenfilename()
9     with open(datei_name, 'r', encoding='utf-8') as datei:
          zeilen = datei.readlines()[1:]
11    return zeilen
```