Runde 2 A1: Laubmaschen

Imanuel Fehse

Teilnahme-ID: 69274

06.04.2024

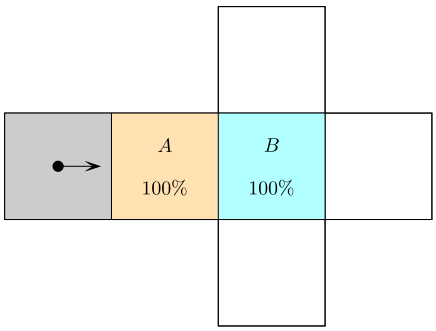
Inhaltsverzeichnis

1. Aufgabe

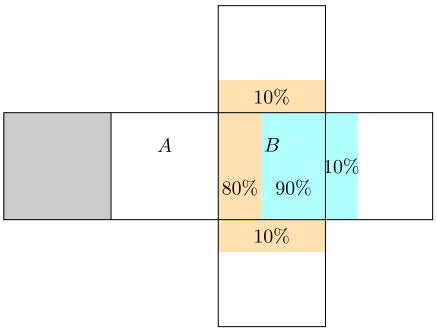
Die Aufgabenstellung wurde dem von BWINF zur Verfügung gestellten Aufgabenblatt entnommen und auf die notwenigen Teile gelürzt.

1.1. Regeln

Der Hausmeister möchte sich immer in der Mitte eines Planquadrats aufstellen und in Richtung des Nachbarquadrats A nördlich, südlich, östlich oder westlich davon blasen. Am liebsten hätte er, wenn das gesamte Laub von Quadrat A auf das dahinter liegende Quadrat B geblasen würde; aber in Wirklichkeit landen jeweils ungefähr 10% davon auf den Quadraten rechts und links von B. Und nicht genug damit: Ungefähr 10% des Laubes, das vorher auf B lag, werden auf das Quadrat noch hinter B weitergeblasen.



[Abb. 1] Visualisierung der Regeln – Vorher



[Abb. 2] Visualisierung der Regeln – Nachher

1.2. Teil 1

Schreibe zuerst ein Programm, mit dem man den Prozess des Laubblasens simulieren kann. Beachte dabei, dass die oben genannten Regeln für die Wirkung des Blasevorganges am Rand und in den Ecken des Schulhofes nicht unverändert anwendbar sind. Überlege dir für diese Fälle sinnvolle Ergänzungen der Regeln und verwende diese in deiner Simulation.

1.3. Teil 2

Entwickle eine Strategie für den Hausmeister: Die Strategie entscheidet vor jedem Blasevorgang, auf welches Feld er sich stellen und in welche Richtung er von dort jeweils blasen soll, damit schließlich auf einem gegebenen Zielquadrat, das kein Rand- oder Eckquadrat ist, möglichst viel Laub versammelt ist.
Lasse die Simulation mit deiner Strategie unter anderem für einen Schulhof aus 5 × 5 Planquadraten laufen, mit anfangs 100 Blättern auf jedem Planquadrat.

2. Lösungsidee

Um die Aufgabe zu lösen, wird das Problem in zwei Schritte aufgeteilt:

1. Entwicklung des Algorithmus zur Simulation des Blasvorganges entsprechend der in der Aufgabenstellung gegeben Regeln.
2. Entwicklung des Algorithmus der Strategie, die verfolgt wird, um so viele Blätter wie möglich auf einem Feld zu Konzentrieren.

2.1. Schritt 1

Die in der Aufgabenstellung gegeben Regeln geben vor, dass bei einem Blasvorgang von Feld A in Richtung von Feld B jeweils 10% der Blätter links und rechts auf den Felder neben Feld B landen (Blasvorgang Schritt1) und dass 10% der Blätter, die zuvor auf Feld B gelegen haben, nun auf dem Feld hinter B liegen (Blasvorgang Schritt 2).

Um dies in einen effizienten Algorithmus umzusetzen, wird die Reihenfolgedieser Schritte umgekehrt, sodass zuerst der Blasvorgang Schritt 2 ausgeführt wird und danach Schritt 2.

Für jeden dieser Schritte wird überprüft, ob sich in die entsprechende Richtung eine Wand befindet. Ist dies der Fall, so verbleiben die Blätter, die eigentlich in diese Richtung geblasen worden wären, auf Feld B sofern Feld B keine Wand ist. Ist Feld B eine Wand, so werden je 10% der Blätter auf die Felder links und rechts von Feld A in Blasrichtung übertragen.

2.2. Schritt 2

Da es sich bei dem Pausenhof um ein Quadrat handelt, kann ein einfaches Prinzip verfolgt werden, um so viele Blätter wie möglich auf einem Feld zu konzentrieren:

1. Ermittle ein finales Feld, dass den Regeln entspricht (Kein Rand- oder Eckenfeld) und so weit wie möglich im Süden des Pausenhofes ist, auf dem die Felder konzentriert werden sollen.
2. Blase alle Blätter in Richtung Süden des Pausenhofes und konzentriere sie in der letzten Zeile, die von Westen nach Osten verläuft. Da der Pausenhof ein Quadrat ist, ist garantiert, dass sich unter dieser Zeile ausschließlich Randfelder befinden, welche beim Konzentrieren der Blätter helfen.
3. Kontentriere alle Blätter in dieser Zeile unter dem finalen Feld und blase diese im letzten Schritt auf das finale Feld.

Auf diese Weise wird erreicht, dass sich möglichst viele Blätter auf dem finalen Feld befinden. Aufgrund der Regeln, wie sich das Laub bei einem Blasvorgang verhält, ist es unmöglich alle

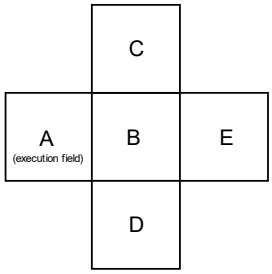
Blätter, die sich auf dem Pausenhofe befinden, auf einem einzigen Feld zu konzentrieren, dass kein Rand- oder Eckenfeld ist.

3. Umsetzung

Zur Umsetzung des Lösungsansatzes, wird zunächst der Algorithmus zur Simulation des Blasvorganges entwickelt und anschließend die Strategie.

3.1. Blasvorgang

Um den Blasvorgang des Hausmeisters simulieren zu können, muss zunächst ein Feld erstellt werden, dass dem Pausenhof entspricht. Hierfür wir eine zweidimensionale Liste verwendet, in der sich die Anzahl der Blätter, die sich auf dem entsprechenden Feld befinden, gespeichert wird. Beim initialisieren des Algorithmus werden auf jedem Feld 100 Blätter plaziert. Zuletzt wird ein beliebiges Feld, sowie die Richtung des Blasvorgangs ausgewählt, woraufhin die Simulation nach dem folgenden Pseudocode ausgeführt wird. Um den Pseudocode leserlich zu machen, wird die folgende Grafik als Referenz für die Felder in eine beliebige Richtung des Blasvorgangs verwendet.



[Abb. 3] Referenzgrafik für Pseudocode; Blasvorgang von Feld A in Richtung B

```
FUNCTION Blow (A, direction):
    IF field B != wall:
        IF field E != wall:
            Transfer 10% of the leaves from field B to field E
        IF field C != wall
            Transfer 10% of the leaves from field A to field C
        IF field D != wall
            Transfer 10% of the leaves that have been on field A to field B
        Transfer all leaves that are left on field A to field B
    Add Blow operation to global list of operations
```

Auf diese Weise wird garantiert, dass die Blätter entsprechend der in der Aufgabenstellung gegebenen Regeln auf die Felder verteilt wird. Ist Feld B eine Wand, so wird kein Blasvorgang ausgeführt, da dieser nicht möglich ist. Ist Feld E eine Wand, so verbleiben die 10% der Blätter,

die von Feld B auf Feld E übertragen worden wären auf Feld B. Ist Feld C oder Feld D eine Wand, so landen die 10% der Blätter von Feld A, die auf Feld C oder D gelandet wären auf Feld B. Als letzter Schritt wird der Blasvorgang zu einer Liste hinzugefügt, in der sich alle Blasvorgänge mit der Position von der sie ausgeführt werden und die Blasrichtung befinden. Diese wird später dazu verwendet um dem Hausmeister eine Anleitung zu geben.

3.2. Strategie

Nachdem der Algorithmus zur Simulation des Blasvorganges von einem beliebigen Feld A in Richtung eines Feldes B vervollständigt ist, kann nun der Algorithmus entwickelt werden, der so viele Blätter wie möglich auf einem Feld konzentrieren soll. Die Funktionsweise dieses Algorithmus wird durch den folgenden Pseudocode beschrieben. Hierbei wird eine Zeile als eine Reihe von Feldern von Westen nach Osten als Zeile verstanden.

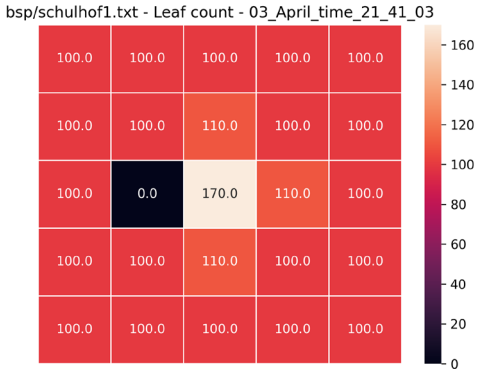
```
FUNCTION Strategy():
    Final_field = get_final_field()
    FOR line in school_yard - 1:        # north to south except most south line
        FOR field in line:              # west to east
            Blow(field, south)
    FOR field in school_yard:           # west to east
        Blow(field,  west)
    Blow(most south-east field, north)
    Blow(field north of most sout-east field, west)
    RETURN the list of operations and result
```

Dieser Algorithmus konzentriert zuerst alle Blätter in der südlichsten Zeile des Pausenhofes, bläst diese anschließend alle in die Südostecke des Pausenhofes und zuletzt auf das finale Feld selbst. Das Finale Feld befindet sich ein Feld entfernt von der östlichen Kante und ein Feld entfernt von der südlichen Kante des Pausenhofes, womit das finale Feld den in der Aufgabenstellung gegebenen Anforderungen entspricht. Als letztes wird eine Liste ausgegeben, in der sich alle Blasvorgänge in der korrekten Reihenfolge befinden, welche dem Hausmeister als Anleitung dient.

Mit diesem Algorithmus wird ein möglichst großer Anteil der gesamten Blätter auf dem Pausenhof auf einem Feld konzentriert, während die Anzahl der der Blasvorgänge minimal gehalten wird, sodass der Hausmeister ebenfalls so wenig Zeit wie möglich mit dem Laubblasen verbringt.

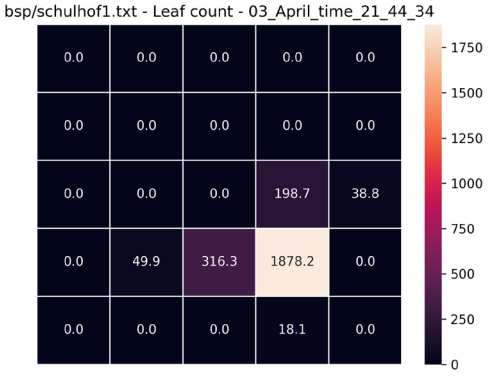
3.3. Ergebnisse

Die Folgende Grafik zeigt die Simulation des Blasvorganges anhand eines 5x5 Pausenhofes, bei dem ein einziger Blasvorgang von Feld (2,3) in Richtung Osten ausgeführt wird.



[Abb. 4] Visualisierung des Ergebnisses des Blasvorgangs

Wird nun der die Strategiealgorithmus ausgeführt, erhält man das folgende Ergebnis für einen 5x5 Pausenhof.



[Abb. 5] Visualisierung des Ergebnisses des Strategiealgorithmus für einen 5x5 Pausenhof

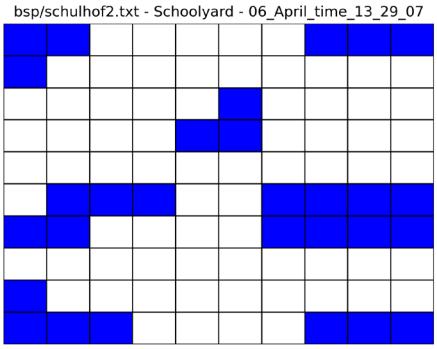
Von den 2500 Blättern, die sich insgesamt auf dem Pausenhof befanden (25 Felder, je 100 Blätter), wurden 1878,2 (≈75,1288%) auf einem einzigen Feld konzentriert.
Die dafür nötigen Blasvorgänge sind in der zugehörigen Datei im Anhang enthalten.

4. Weiterentwicklung

Der Pausenhof, der in der Aufgabenstellung beschrieben wird ist quadratisch, wodurch der Algorithmus sehr einfach ist, um das Problem zu lösen. Jedoch sind die meisten Pausenhöfe an Schule nicht quadratisch, sondern haben komplexere Formen und oft auch Objekte auf dem Pausenhof. Um auch eine Lösung für diese Pausenhöfe zu finden, kann der Strategie-Algorithmus weiterentwickelt werden. Zunächst muss allerdings der Algorithmus der den Pausenhof erstell, welcher später für den Strategiealgorithmus verwendet wird, modifiziert werden.

4.1. Erstellen des Pausenhofs

Um einen Pausenhof mit einer komplexen Form zu erstellen, wird eine Datei eingelesen, welche die Umrisse des Pausenhofes sowie die Position von Objekten auf dem Pausenhof enthält. Hierbei wird zwischen freien und belegten Feldern unterschieden. Ist neben einem freien Feld ein Objekt oder der Umriss des Pausenhofes, so wird angenommen, zwischen diesen eine undurchlässige Barriere befindet, welche als Wand betrachtet werden kann.
Der so modifizierte Algorithmus erstellt als letztes eine Grafik, die den Pausenhof darstellt. Ein Beispiel für einen komplexen Pausenhof wäre der folgende (weiß = freies Feld, blau = belegtes Feld).



[Abb. 6] Visualisierung eines komplexen Pausenhofes

Nachdem der Pausenhof erfolgreich erstellt wurde, so kann nun ein finales Feld ermittelt werden, auf dem die Blätter konzentriert werden sollen.

4.2. Ermitteln des finalen Feldes

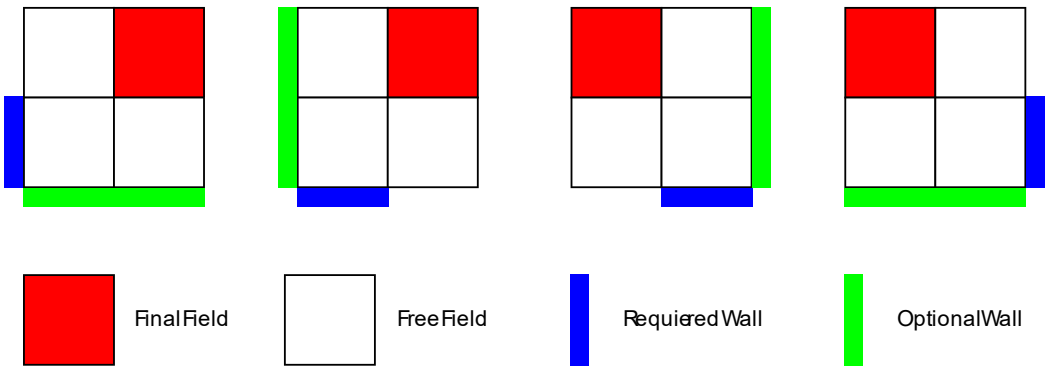
Da der Pausenhof zuvor immer quadratisch war und keine Objekte in diesem Gebiet hatte, war die Ermittlung eines finalen Feldes sehr einfach, da immer dieselbe relative Position zum Umriss des Pausenhofes genommen werden kann. Hierfür wurde bisher immer das Feld genommen, dass sich ein Feld von der südlichen und ein Feld von der östlichen Kante des Pausenhofes entfernt befindet. Da es sich nun allerdings nicht mehr um einen quadratischen Pausenhof

handelt, muss das finale Feld für jeden Pausenhof neu ermittelt werden, sodass dieses weiterhin den Anforderungen entspricht.

Um möglichst viele Blätter auf einem Feld zu konzentrieren zu können, ist es sehr hilfreich, wenn sich Wände an den umliegenden Feldern (nicht das finale Feld selbst, sondern die Felder um das finale Feld) befinden, da so ein größerer Anteil der Blätter auf dem vorgesehenen Feld landen. Aus diesem Grund wird bei der Suche nach dem finalen Feld nicht nur nach einem Feld gesucht, dass kein Rand- Eckenfeld ist, sondern ebenfalls möglichst viele Wände an den umliegenden Feldern hat.

Da der Algorithmus zum Konzentrieren der Blätter auf dem finalen Feld von Norden nach Süden arbeitet, wird zusätzlich nach dem südlichsten geeigneten finalen Feld gesucht, dass möglich ist.

Um diese Prinzipien in einen Algorithmus umzusetzen, werden Musterkonstellationen genutzt, nach denen der Pausenhof abgesucht wird bei denen sich die Blätter gut auf einem Feld konzentrieren lassen. Die folgende Grafik zeigt diese Konstellationen.



[Abb. 7] Konstellationen zur Auswahl des finalen Feldes

Die Funktionsweise des Algorithmus zur Auswahl eines finalen Feldes wird durch den folgenden Pseudocode dargestellt.

```
FUNCTION Find_Final_Field(schoolyard, constellations):
    Possible_zones = []
    FOR line in schoolyard – 1:
        FOR field in line – 1:
            Select 2x2 zone where current field is the top left field of the zone
            IF zone is one of the constellations:
                Add current zone to possible_zones
    Final_zone = possible_zones[0]
    FOR possible_zone in possible_zones:
        IF final_field of possible_zone is more south than the one of the current final zone
            Final_zone =  possible_zone
    RETURN final_zone and final_field
```

Um ein möglichst geeignetes finales Feld zu finden, das sich in einer der Konstellationen befindet, durchläuft der Algorithmus den Schulhof und speichert jede gefundene Konstellation in

einer Liste. Anschließend wird diese Liste durchlaufen und nach der Konstellation gesucht, die so weit südlich wie möglich ist. Ist das finale Feld erfolgreich ermittelt, gibt der Algorithmus dieses sowie die Konstellation, in welcher sich dieses Feld befindet, zurück, sodass diese zur Konzentration der Blätter auf dem finalen Feld genutzt werden können.

4.3. Weiterentwicklung der Strategie

Nachdem der komplexe Pausenhof erstellt ist und das finale Feld ermittelt wurde, muss nun auch die Strategie, mit der die Blätter auf einem Feld konzentriert werden, angepasst werden, um eine Lösung finden zu können.

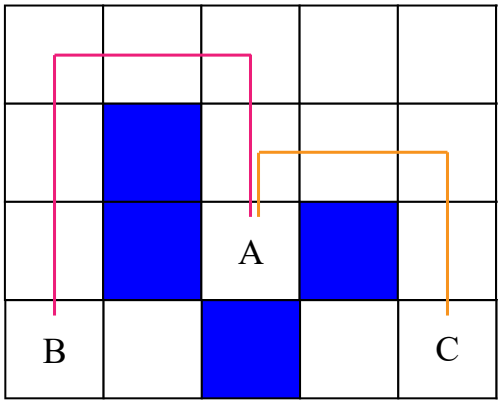
4.3.1. Lösen von komplexen Pausenhöfen

Mit einem komplexen Layout des Pausenhofes kommen einige neue Probleme zum Lösen des Problems hinzu. Die Blätter können nichtmehr einfach in eine Richtung geblasen werden, da sich dort eine Wand befinden könnte was die möglichen Blasvorgänge einschränkt. Zusätzlich macht dies das Problem deutlich schwerer, da eventuell der einzige von einem Feld aus Richtung norden ist, was entgegengesetzt der grundsätzlichen Blasrichtung der Blätter ist, um diese im Süden zu versammeln.
Dieses Problem wird gelöst, indem ein alternativer Weg für den Blasvorgang von einem solchen Feld aus gesucht wird. Die Funktionsweise des modifizierten Algorithmus wird durch diesen Pseudocode dargestellt.

```
FUNCTION Strategy(y_start, y_end):
    FOR line in school_yard[y_start, y_end]:
        FOR field in line:
            IF field != wall and leaves on the field != 0:
                IF there is no south of field:
                    Blow(field, south)
                ELSE:
                    Search for alternative way to the right
                    Search for alternative way to the left
                    Select shorter alternative way
                    Execute shorter alternative way
```

Dieser Algorithmus durchläuft den Schulhof von einem gegebenen Startpunkt bis zu einem Endpunkt und überprüft zunächst, ob das aktuelle Feld, an dem er sich befindet, eine Wand ist, oder ob schon alle Blätter von diesem Feld entfernt sind. Ist dies der Fall, geht der Algorithmus einfach zum nächsten Feld. Ansonsten, sofern sich keine Wand südlich von dem Feld aus befindet, werden die Blätter Richtung Süden geblasen. Befindet sich dort aber eine Wand, so wird nach einem alternativen Weg gesucht. Ein alternativer Weg ist ein Weg, der zu einem anderen Feld führt, welches sich in der Zeile unter dem aktuellen Feld befindet, also südlicher ist. Somit

ist garantiert, dass sich die Blätter von dem aktuellen Feld in der richtigen Zeile landen, um den Algorithmus weiterhin erfolgreich ausführen zu können. Ein solcher Weg wird in westliche und östliche Richtung gesucht, woraufhin der kürzere Weg ausgewählt und ausgeführt wird. Anschließend fährt der Algorithmus mit demselben Prozess dann am nächsten Feld fort.
Die folgende Grafik zeigt eine Darstellung des Verhaltens des Algorithmus an einem Beispiel.



[Abb. 9] Veranschaulichung des modifizierten Strategie-Algorithmus

Bei diesem Beilspiel befindet sich der Algorithmus an Feld A. Da dieses Feld keine Wand ist, sich allerdings eine Wand südlich von dem Feld befindet, wird nach einem alternativen Weg in westliche und östliche Richtung gesucht. Der alternative Weg in westliche Richtung zu Feld B wird durch die pinke Linie dargestellt und der alternative Weg in östliche Richtung zu Feld C durch die orangene. Feld B und C befinden sich beide in der Zeile unter Feld A, weshalb beide Wege gültig sind. Da der Weg zu Feld C jedoch kürzer ist als der Weg zu Feld B, wird der alternative Weg in westliche Richtung ausgeführt.

4.3.2. Zeilen unter dem finalen Feld

Zuvor hat sich das finale Feld immer in der vorletzten Zeile des Pausenhofes, also ein Feld über dem südlichen Ende des rechteckigen Pausenhofes befunden. Aufgrund der Tatsache, dass sich in dem rechteckigen Pausenhof nichts anderes als eine freie Fläche befunden hat, war somit auch garantiert, dass alle Felder in der Zeile unter dem finalen Feld eine Wand in südliche Richtung haben, was das konzentrieren der Blätter sehr einfach gemacht hat.
Da sich bei einem komplexen Pausenhof das finale Feld allerdings nicht zwingend in der vorletzten Zeile befindet, sondern sich gegebenenfalls auch in einer höheren (= nördlicheren) Zeile befinden kann. Ist dies der Fall, so befinden sich mehr als eine Zeile unter dem finalen Feld. Würden nun alle Blätter in der letzten Zeile des Pausenhofes gesammelt, so müssten diese wieder nach oben geblasen werden, um auf dem finalen Feld konzentriert werden zu können, was in einem ineffizienten Blasverfahren resultiert.

Aus diesem Grund wird, bevor der Algorithmus beginnt die Blätter in Richtung Süden zu blasen, überprüft in welcher Zeile sich das finale Feld befindet. Abhängig von dieser Position, sollen alle Blätter in der Zeile unter dem finalen Feld konzentriert werden. Dies bedeutet, dass die Blätter, die sich in einer tieferen Zeile befinden in nördliche Richtung geblasen werden müssen, bis sie auf der entsprechenden Zeile sind. Hierfür wird der Algorithmus, welcher die Blätter von Norden nach Süden bläst in dem Bereich unter dem finalen Feld und der nicht umgekehrte Algorithmus in dem Bereich über dem finalen Feld angewendet.
Der folgende Pseudocode implementiert dieses Prinzip in der Strategie.

```
IF final_field[y] + 1 < schoolyard_size[y]:
        Execute Strategy() from y = 0 to y = final_field[y]
        Execute Strategy_reversed() from schoolyard_size[y] to final_field[y]+2
ELSE:
        Execute Strategy() from y = 0 to schoolyard_size[y]
```

Mithilfe dieser Modifikation der Strategie wird immer garantiert, dass die Blätter in der Zeile unter dem finalen Feld gesammelt werden, um die Blätter so gut wie möglich zu konzentrieren, während die Schritte des Hausmeisters zusätzlich minimal gehalten werden.

4.3.3. Konzentration der Blätter auf finalem Feld

Nachdem alle Blätter erfolgreich (wie zuvor bei dem originalen Algorithmus) in der Zeile unter dem finalen Feld versammelt wurden, können diese nun alle auf dem finalen Feld konzentriert werden. Durch die Konstellationen in der sich das finale Feld befinden muss ist garantiert, dass sich eine Wand entweder am rechten oder linken Ende der Zeile unter dem finalen Feld befindet (siehe Konstellationen: hier). Abhängig von der Position des finalen Feldes, werden alle Blätter gegen die vorausgesetzte Wand in der Konstellation geblasen und anschließend auf das finale Feld.
Nach dem Ausführen dieser Aktion, wird der Algorithmus beendet und die Lösung ausgegeben.

4.4. Ergebnis, Anwende Möglichkeiten und Voraussetzungen.

Mithilfe des fortentwickelten Algorithmus können nun auch sehr komplexe Pausenhöfe bearbeitet und gelöst werden. Hierbei hat der Algorithmus jedoch eine einzige Voraussetzung, um korrekt zu funktionieren: Die freie Fläche des Pausenhofes muss vollständig miteinander verbunden sein. Das heißt, dass es keine abgekapselten freien Felder geben darf oder zwei voneinander abgetrennte Teile des Pausenhofes.
Handelt es sich jedoch um einen Pausenhof, der in zwei abgetrennte Teile aufgeteilt ist, so können diese auch als separate Pausenhöfe in den Algorithmus eingegeben werden, woraufhin man eine Lösung für den jeweiligen Teil des Pausenhofes erhält.

5. Laufzeitanalyse

Die Laufzeitanalyse des Algorithmus vor der Weiterentwicklung hat eine Laufzeit von O(n), wobei n die Anzahl der Felder auf dem Pausenhofes ist. Diese Laufzeit kommt dadurch zustande, dass für alle n Felder dieselbe Operation ausgeführt wird welche immer gleich viele Berechnungen benötigt.

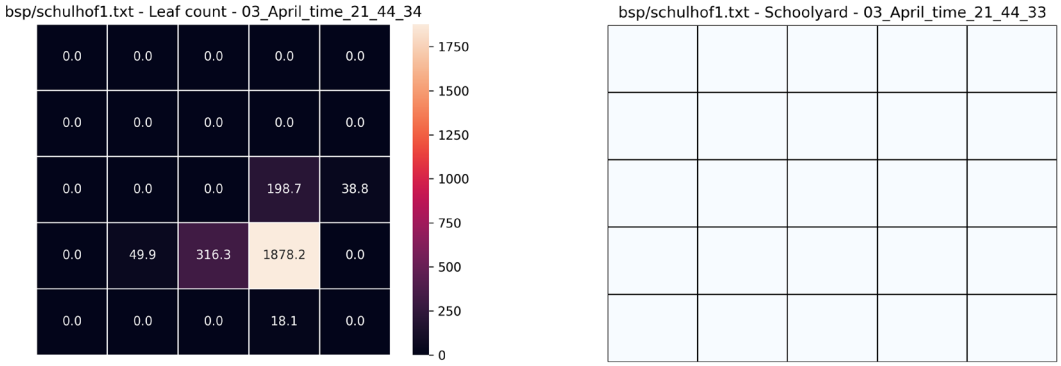
Die theoretische maximale Laufzeit des weiterentwickelten Algorithmus ist $O(n^2)$. Diese Laufzeit kommt durch die zusätzliche Suche nach alternativen Wegen zustande, falls diese nötig sind. Hierbei muss jedoch beachtet werden, dass dieser Vorgang in den meisten Fällen nur für sehr wenige Felder ausgeführt werden muss, weshalb die tatsächliche Laufzeit des weiterentwickelten Algorithmus sehr viel näher an O(n) liegt (basierend auf Anzahl von der von dem Algorithmus ausgeführten Operationen für komplexe Pausenhöfe mit unterschiedlicher Anzahl von Feldern). Dies zeigt, dass der Algorithmus erfolgreich für deutlich komplexere Probleme weiterentwickelt wurde und die Laufzeit nur minimal beeinflusst ist.

6. Beispiele

Die hier gelisteten Ergebnisse von Beispieleingaben enthalten nur die erstellten Grafen der Lösung sowie ausgewählte Daten. Hierbei zeigt die erste Grafik die Verteilung der Blätter auf dem Pausenhof, nachdem der Hausmeister seine Arbeit beendet hat und die zweite Grafik zeigt den Umriss des Pausenhofes (weiß = freies Feld, blau = Wand/belegtes Feld). Bitte beachten Sie, dass das Koordinatensystem für die Grafiken für x und y bei 0 beginnt und y in der ersten Zeile startet und die positive Achse nach unten verläuft. Die Prozentzahl der Blätter, die sich auf dem finalen Feld befinden ist dezimal angegeben.

Die Liste der Schritte, die nötig sind um diese Ergebnisse zu erhalten können genauso wie die Eingabedatei im Anhang als Ausgabedatei gefunden werden.

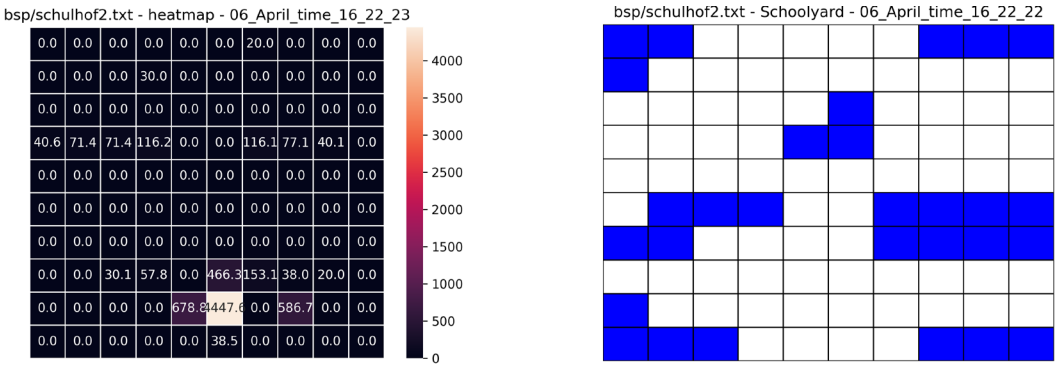
Schulhof 1



final field:[3, 3] final percentage:0.7512882196873933

[R1] Ergebnis der Eingabedatei Schulhof1.txt

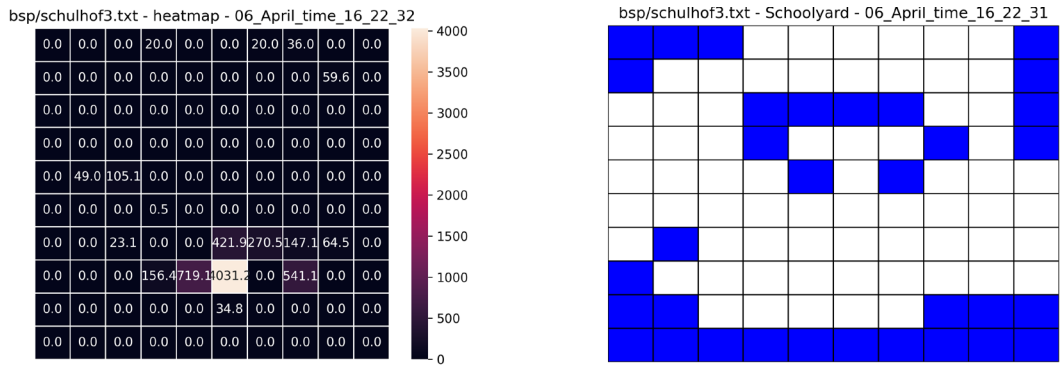
Schulhof 2



final field:[5, 8] final percentage:0.6264265972409349

[R2] Ergebnis der Eingabedatei Schulhof2.txt

Schulhof 3



final field:[5, 7] final percentage:0.6016722165029089

[R3] Ergebnis der Eingabedatei Schulhof3.txt

7. Wichtige Teile des Codes

```
def blow(self, x, y, direction):
    # upwards
    if direction == 0:
        walls_around_target = self.schoolyard[y - 1][x][2][1]

        # BLOW STAGE ONE
        # if there is a wall above target
        if walls_around_target[0] == 1:
            pass
        # if there is no wall above target
        else:
            self.movements.append([x, y, "up"])
            leafs_target_field = self.schoolyard[y-1][x][1]
            percentage_above_target = 0.1
            percentage_target_field = 0.9

            # transfer leafs from blow stage 1
            self.schoolyard[y-1][x][1] = leafs_target_field * percent-
age_target_field
            self.schoolyard[y-2][x][1] += leafs_target_field * percent-
age_above_target

        # BLOW STAGE TWO
        percentage_target_field = 0.8
        percentage_right_field = 0.1
        percentage_left_field = 0.1
        leafs_c_field = self.schoolyard[y][x][1]
        # if target has a wall left
        if walls_around_target[3] == 1:
            percentage_left_field = 0
            percentage_target_field += 0.1
        # if target has a wall right
        if walls_around_target[1] == 1:
            percentage_right_field = 0
            percentage_target_field += 0.1

        # transfer leafs from blow stage 2
        self.schoolyard[y][x][1] = 0
        self.schoolyard[y - 1][x][1] += leafs_c_field * percentage_tar-
get_field
        if walls_around_target[3] == 0:
            self.schoolyard[y - 1][x - 1][1] += leafs_c_field * percent-
age_left_field
        if walls_around_target[1] == 0:
            self.schoolyard[y - 1][x + 1][1] += leafs_c_field * percent-
age_right_field
```

[C1] Implementierung des Blasvorgangs in nördliche Richtung. Alle Richtungen verfolgen dasselbe Prinzip

```
def find_final_field(self):
    def get_final_field(c_x, c_y, final_arrangement):
        if final_arrangement == 0 or final_arrangement == 1:
            return [c_x + 1, c_y]
        elif final_arrangement == 2 or final_arrangement == 3:
            return [c_x, c_y]
        elif final_arrangement == 4 or final_arrangement == 5:
            return [c_x, c_y + 1]
        elif final_arrangement == 6 or final_arrangement == 7:
            return [c_x + 1, c_y + 1]
        else:
            print("invalid input")
    # constellations of possible arrangements
    possible_arrangements = [
        # 0
        [
            [1, ["?", 0, 0, "?"]], [1, [0, 0, 0, 0]],
            [1, [0, 0, "?", 1]], [1, [0, "?", "?", 0]]
        ],
        # 1
        [
            [1, ["?", 0, 0, "?"]], [1, [0, 0, 0, 0]],
            [1, [0, 0, 1, "?"]], [1, [0, "?", "?", 0]]
        ],
        # 2
        [
            [1, [0, 0, 0, 0]], [1, ["?", "?", 0, 0]],
            [1, [0, 0, "?", "?"]], [1, [0, 1, "?", 0]]
        ],
        # 3
        [
            [1, [0, 0, 0, 0]], [1, ["?", "?", 0, 0]],
            [1, [0, 0, "?", "?"]], [1, [0, "?", 1, 0]]
        ],

    ]

    # get all 2x2 areas
    areas = []
    for y in range(self.dimensions[1]-1):
        for x in range(self.dimensions[0]-1):
            # select 2x2 area
            c_area = []
            for a_y in range(2):
                for a_x in range(2):
                    c_area.append(self.schoolyard[y+a_y][x+a_x])
            areas.append([x, y, c_area])

    # check if area is a possible arrangement
    possible_areas = []
    for raw_area in areas:
        sublist = raw_area[2]
        area = []
```

```
        for field in sublist:
            area.append([field[0], field[2][1]])

        # check if area is a possible arrangement
        for arrangement in possible_arrangements:
            valid_arrangement = True
            for field in range(4):
                # if type of field of area is equal to the type of field of
arrangement
                if arrangement[field][0] != area[field][0]:
                    valid_arrangement = False
                    break
                # check if walls are at the same positions
                for wall in range(4):
                    # if wall can be anything
                    if arrangement[field][1][wall] == "?":
                        pass
                    # if wall is not at the same position
                    elif arrangement[field][1][wall] !=
area[field][1][wall]:
                        valid_arrangement = False
                        break
            # if one arrangement was found
            if valid_arrangement:
                c_arrangement = possible_arrangements.index(arrangement)
                possible_areas.append([raw_area, c_arrangement, get_fi-
nal_field(raw_area[0], raw_area[1],

c_arrangement)])
    # select area with final field in the lowest line and leftmost column
    self.final_area = possible_areas[0]
    for possible_area in possible_areas:
        # if current area with final field is in a lower line or more left,
set new final values
        if self.final_area[2][0] > possible_area[2][0] or self.fi-
nal_area[2][1] < possible_area[2][1]:
            self.final_area = possible_area

        self.final_field = possible_area[2]
        self.final_arrangement = possible_area[1]
```

[C2] Implementierung des Algorithmus zur Ermittlung des finalen Feldes in einer Konstellation

```
for y in range(self.dimensions[1]-(1 + y_skip)):
    for x in range(self.dimensions[0]):
        # if field is part of the schoolyard
        if self.schoolyard[y][x][0] == 1:
            # check if field has a wall downwards
            if self.schoolyard[y][x][2][1][2] == 1:
                # check if there is a way to the right
                found_way_right = False
                found_way_left = False
                # while no way was found
                move_right = 0
                move_left = 0
                move_up_right = 0
                move_up_left = 0
                move_list_right = []
                move_list_left = []
                fail_right = False
                fail_left = False

                # check for way to the right
                while not found_way_right:
                    # check if there is no wall downwards and last movement
wasn't upwards
                    if
self.schoolyard[y+move_up_right][x+move_right][2][1][2] == 0 and
move_list_right[-1] != "up":
                        # as long as you can move down
                        while
self.schoolyard[y+move_up_right][x+move_right][2][1][2] == 0:
                            move_up_right += 1
                            move_list_right.append("down")
                            # check if move ended up lower as starting
field
                            if move_up_right > 0:
                                found_way_right = True
                                break
                    # check if there is no wall to the right
                    elif
self.schoolyard[y+move_up_right][x+move_right][2][1][1] == 0:
                        move_right += 1
                        move_list_right.append("right")
                    # if there is no wall upwards
                    elif
self.schoolyard[y+move_up_right][x+move_right][2][1][0] == 0:
                        move_up_right -= 1
                        move_list_right.append("up")
                    else:
                        fail_right = True
                        break
                    if fail_right:
                        break

                # check for way to the left
```

```
                while not found_way_left:
                    # check if there is no wall downwards and last movement
wasn't upwards
                    if
self.schoolyard[y+move_up_left][x+move_left][2][1][2] == 0 and
move_list_left[-1] != "up":
                        # as long as you can move down
                        while
self.schoolyard[y+move_up_left][x+move_left][2][1][2] == 0:
                            move_up_left += 1
                            move_list_left.append("down")
                            # check if move ended up lower as starting
field
                            if move_up_left > 0:
                                found_way_left = True
                                break
                    # check if there is no wall to the left
                    elif
self.schoolyard[y+move_up_left][x+move_left][2][1][3] == 0:
                        move_left -= 1
                        move_list_left.append("left")
                    # if there is no wall upwards
                    elif
self.schoolyard[y+move_up_left][x+move_left][2][1][0] == 0:
                        move_up_left -= 1
                        move_list_left.append("up")
                    else:
                        failLleft = True
                        break
                    if fail_left:
                        break


                """EXECUTION"""
                # if a way to the right was found and a way to the left was
found
                if found_way_right and found_way_left:
                    # check which one is shorter
                    if len(move_list_right) <= len(move_list_left):
                        execute_right()
                    else:
                        execute_left()
                # if only a way to the right was found
                elif found_way_right:
                    execute_right()
                # if only a way to the left was found
                elif found_way_left:
                    execute_left()
                # if no way was found
                else:
                    break

            # if field has no wall downwards, blow leafs downwards
```

```
            else:
                self.blow(x, y, 2)
```

[C3] Implementierung der modifizierten Strategie

8. Quellen

- **[Abb. 1]** Visualisierung der Regeln – Vorher
  https://bwinf.de/fileadmin/bundeswettbewerb/42/aufgaben422.pdf
  01.04.2023 | 23:20 Uhr
- **[Abb. 2]** Visualisierung der Regeln – Nachher
  https://bwinf.de/fileadmin/bundeswettbewerb/42/aufgaben422.pdf
  01.04.2023 | 23:20 Uhr
- **[Abb. 3]** Referenzgrafik für Pseudocode
  Imanuel Fehse
  03.04.2023 | 21:00 Uhr
- **[Abb. 4]** Visualisierung des Ergebnisses des Blasvorgangs
  Imanuel Fehse
  03.04.2023 | 21:50 Uhr
- **[Abb. 5]** Visualisierung des Ergebnisses des Strategiealgorithmus für einen 5x5 Pausenhof
  Imanuel Fehse
  03.04.2023 | 21:55 Uhr
- **[Abb. 6]** Visualisierung eines komplexen Pausenhofes
  Imanuel Fehse
  06.04.2023 | 13:35 Uhr
- **[Abb. 7]** Konstellationen zur Auswahl des finalen Feldes
  Imanuel Fehse
  04.04.2023 | 22:15 Uhr
- **[Abb. 8]** Veranschaulichung des modifizierten Strategie-Algorithmus
  Imanuel Fehse
  06.04.2023 | 14:35 Uhr
- **[C1]** Implementierung des Blasvorgangs in nördliche Richtung
  Imanuel Fehse
  07.04.2023 | 19:20 Uhr
- **[C2]** Implementierung des Algorithmus zur Ermittlung des finalen Feldes in einer Konstellation
  Imanuel Fehse
  07.04.2023 | 19:25 Uhr

- **[C3]** Implementierung der modifizierten Strategie
  Imanuel Fehse
  07.04.2023 | 19:30 Uhr
- **[R1]** Ergebnis der Eingabedatei Schulhof1.txt
  Imanuel Fehse
  06.04.2023 | 16:10 Uhr
- **[R2]** Ergebnis der Eingabedatei Schulhof2.txt
  Imanuel Fehse
  06.04.2023 | 16:20 Uhr
- **[R3]** Ergebnis der Eingabedatei Schulhof3.txt
  Imanuel Fehse
  06.04.2023 | 16:25 Uhr