

Selected Fun Problems of the ACM Programming Contest: **Booby Traps**

Noah Doersing

noah.doersing@student.uni-tuebingen.de

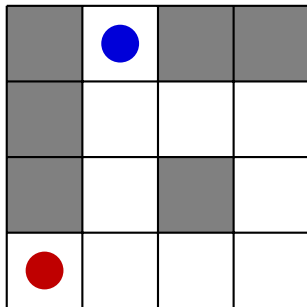
22. Januar 2016



- Chinesische Grabräuber suchen nach einer Strategie, um eine spezielle Art von Labyrinth nach Schätzen zu durchsuchen.
- Dabei soll der **kürzeste Pfad** von einem **Start-** zu einem **End**punkt ermittelt werden (bzw. dessen Länge).

Problembeschreibung

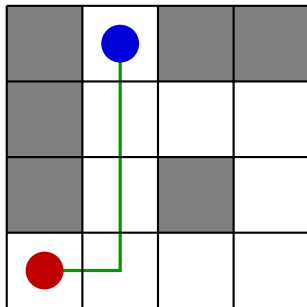
Beispiel



- Beim Labyrinth handelt es sich um ein $w \times h$ -Grid von begehbaren und nicht begehbaren Feldern.
- Start- und Endpunkt.

Problembeschreibung

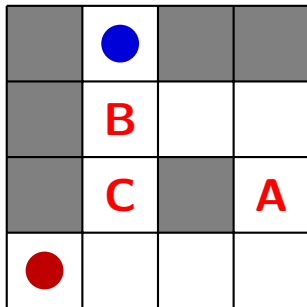
Beispiel



- Beim Labyrinth handelt es sich um ein $w \times h$ -Grid von begehbaren und nicht begehbaren Feldern.
- Länge des kürzesten Pfades: 4.

Problembeschreibung

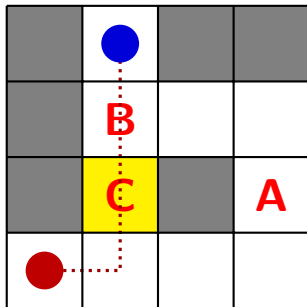
Beispiel



- Im Labyrinth befinden sich **Fallen**, wobei das Auslösen einer Falle α auch **alle Fallen $\leq \alpha$ auslöst**, also **nicht begehbar** macht.
- Hier: $C < B < A$

Problembeschreibung

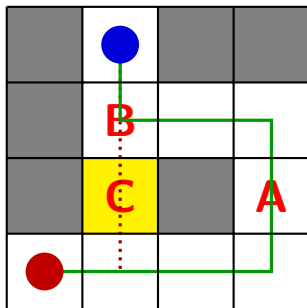
Beispiel



- Im Labyrinth befinden sich **Fallen**, wobei das Auslösen einer Falle α auch **alle** **Fallen** $\leq \alpha$ auslöst, also **nicht begehbar** macht.
- Hier: **C** $<$ **B** $<$ **A**
 - ▶ Pfad durch **C** unmöglich.
 - ▶ **Dynamisches** Labyrinth.

Problembeschreibung



Beispiel



- Im Labyrinth befinden sich **Fallen**, wobei das Auslösen einer Falle α auch **alle Fallen $\leq \alpha$ auslöst**, also **nicht begehbar** macht.
- Hier: $C < B < A$
- Länge des kürzesten Pfades: 8.

Problembeschreibung

Input & Output

(0,0)	(1,0)	(2,0)	(3,0)
			
(0,1)	(1,1)	(2,1)	(3,1)
	B		
(0,2)	(1,2)	(2,2)	(3,2)
	C		A
(0,3)	(1,3)	(2,3)	(3,3)
			

ZYXWVUTSRQPONMLKJIHGFEDCBA

4 4

x o x x

x B o o

x C x A

o o o o

1 0

0 3

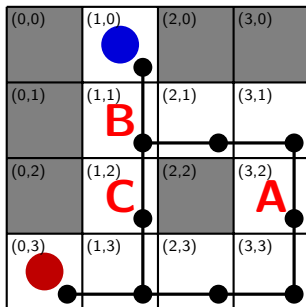
- **Input:** *trap domination order*, Breite w und Höhe h , *map* (Labyrinth), **Start**- und **End**punkt.
- **Output:** Länge des kürzesten Pfades von **Start**- zu **End**punkt oder IMPOSSIBLE.

- Schwache und dynamische Typisierung
 - ▶ ermöglicht schnelles und einfaches Prototyping.
- Listen, Mengen und insbesondere Dictionaries sind nahtlos integriert und einfach zu handhaben.
- Allgemein simple und intuitive Syntax.
- Mehrere Funktionsrückgabewerte.
- **Nachteil:** Performance (verglichen zu kompilierenden Sprachen).

- 1 Die Eingabe wird vom *standard input* gelesen und geparkt.
- 2 Die Map-Darstellung des Labyrinths wird zu einem **Graphen in Adjazenzlistendarstellung** umgewandelt.
- 3 Auf dem Graphen wird der **kürzeste Pfad** vom **Start-** zum **Endpunkt** ermittelt.
- 4 Die Länge des kürzesten Pfades wird ausgegeben, ggf. auch das Labyrinth und der Pfad selbst.

Problemlösung

Umwandlung zu Adjazenzlistendarstellung



Feld \rightarrow Nachbarn

$(1, 0) \rightarrow [(1, 1)]$

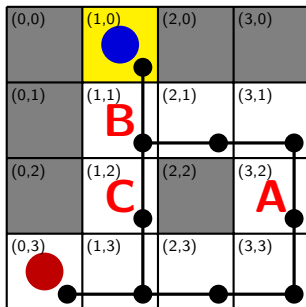
$(1, 1) \rightarrow [(1, 0), (1, 2)]$

$(2, 1) \rightarrow [(1, 1), (3, 1)]$

$\dots \rightarrow \dots$

Problemlösung

Umwandlung zu Adjazenzlistendarstellung



Feld \rightarrow Nachbarn

$(1, 0) \rightarrow [(1, 1)]$

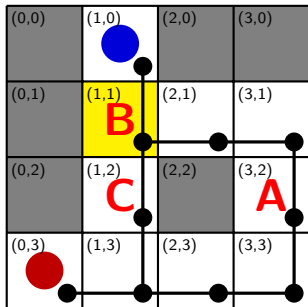
$(1, 1) \rightarrow [(1, 0), (1, 2)]$

$(2, 1) \rightarrow [(1, 1), (3, 1)]$

$\dots \rightarrow \dots$

Problemlösung

Umwandlung zu Adjazenzlistendarstellung



Feld \rightarrow Nachbarn

$(1, 0) \rightarrow [(1, 1)]$

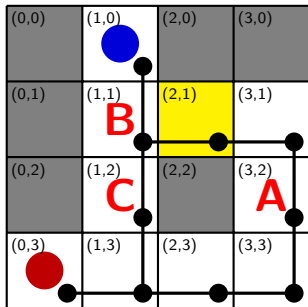
$(1, 1) \rightarrow [(1, 0), (1, 2)]$

$(2, 1) \rightarrow [(1, 1), (3, 1)]$

$\dots \rightarrow \dots$

Problemlösung

Umwandlung zu Adjazenzlistendarstellung



Feld \rightarrow Nachbarn

$(1, 0) \rightarrow [(1, 1)]$

$(1, 1) \rightarrow [(1, 0), (1, 2)]$

$(2, 1) \rightarrow [(1, 1), (3, 1)]$

$\dots \rightarrow \dots$

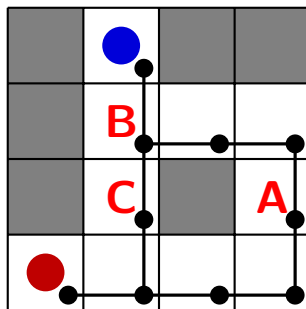
- Im soeben erzeugten Graphen soll **gesucht** werden (n : Knotenanzahl, m : Kantenanzahl).
 - ▶ Breitensuche: $\mathcal{O}(n + m)$
 - ▶ Tiefensuche: $\mathcal{O}(n + m)$
- Genauer: Es soll ein **kürzester Pfad** im Graphen ermittelt werden.
 - ▶ Dijkstra: $\mathcal{O}(n^2 + m)$ bzw. $\mathcal{O}(n \cdot \log n + m)$
 - ▶ A*: $\mathcal{O}(n^2)$ bzw. $\mathcal{O}(n \cdot \log n)$
 - ▶ Bellman-Ford: $\mathcal{O}(n \cdot m)$
 - ▶ ...
- Noch genauer: Nur die **Länge** interessiert.
 - ▶ Manhattan-Distanzen: $\mathcal{O}(1)$

- Im soeben erzeugten Graphen soll **gesucht** werden (n : Knotenanzahl, m : Kantenanzahl).
 - ▶ Breitensuche: $\mathcal{O}(n + m)$
 - ▶ Tiefensuche: $\mathcal{O}(n + m)$
- Genauer: Es soll ein **kürzester Pfad** im Graphen ermittelt werden.
 - ▶ Dijkstra: $\mathcal{O}(n^2 + m)$ bzw. $\mathcal{O}(n \cdot \log n + m)$
 - ▶ A*: $\mathcal{O}(n^2)$ bzw. $\mathcal{O}(n \cdot \log n)$
 - ▶ Bellman-Ford: $\mathcal{O}(n \cdot m)$
 - ▶ ...
- Noch genauer: Nur die **Länge** interessiert.
 - ▶ Manhattan-Distanzen: $\mathcal{O}(1)$

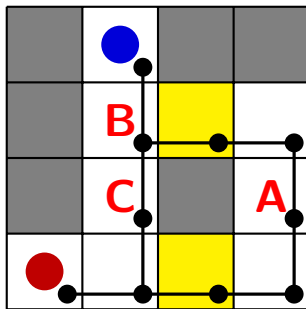
- Im soeben erzeugten Graphen soll **gesucht** werden (n : Knotenanzahl, m : Kantenanzahl).
 - ▶ Breitensuche: $\mathcal{O}(n + m)$
 - ▶ Tiefensuche: $\mathcal{O}(n + m)$
- Genauer: Es soll ein **kürzester Pfad** im Graphen ermittelt werden.
 - ▶ Dijkstra: $\mathcal{O}(n^2 + m)$ bzw. $\mathcal{O}(n \cdot \log n + m)$
 - ▶ A*: $\mathcal{O}(n^2)$ bzw. $\mathcal{O}(n \cdot \log n)$
 - ▶ Bellman-Ford: $\mathcal{O}(n \cdot m)$
 - ▶ ...
- Noch genauer: Nur die **Länge** interessiert.
 - ▶ Manhattan-Distanzen: $\mathcal{O}(1)$

- Im soeben erzeugten Graphen soll **gesucht** werden (n : Knotenanzahl, m : Kantenanzahl).
 - ▶ Breitensuche: $\mathcal{O}(n + m)$
 - ▶ Tiefensuche: $\mathcal{O}(n + m)$
- Genauer: Es soll ein **kürzester Pfad** im Graphen ermittelt werden.
 - ▶ Dijkstra: $\mathcal{O}(n^2 + m)$ bzw. $\mathcal{O}(n \cdot \log n + m)$
 - ▶ A*: $\mathcal{O}(n^2)$ bzw. $\mathcal{O}(n \cdot \log n)$
 - ▶ Bellman-Ford: $\mathcal{O}(n \cdot m)$
 - ▶ ...
- Noch genauer: Nur die **Länge** interessiert.
 - ▶ Manhattan-Distanzen: $\mathcal{O}(1)$

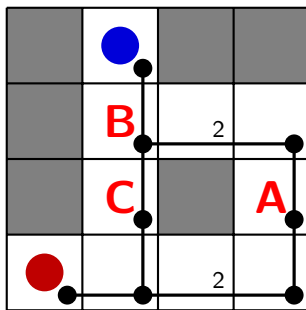
- **Nachteil:** Auf einem ungewichteten Graphen weniger effizient als Breitensuche: bestenfalls $\mathcal{O}(n \cdot \log n + m)$ vs. $\mathcal{O}(n + m)$.
- **Vorteil:** Möglichkeit der Graph-Optimierung vor Ausführung des Algorithmus.



- **Nachteil:** Auf einem ungewichteten Graphen weniger effizient als Breitensuche: bestenfalls $\mathcal{O}(n \cdot \log n + m)$ vs. $\mathcal{O}(n + m)$.
- **Vorteil:** Möglichkeit der Graph-Optimierung vor Ausführung des Algorithmus.

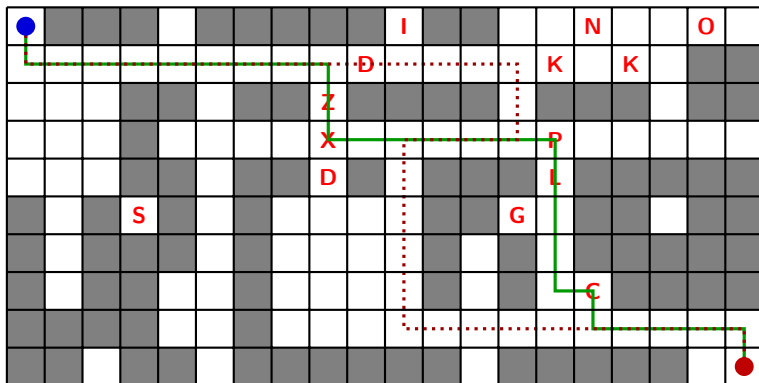


- **Nachteil:** Auf einem ungewichteten Graphen weniger effizient als Breitensuche: bestenfalls $\mathcal{O}(n \cdot \log n + m)$ vs. $\mathcal{O}(n + m)$.
- **Vorteil:** Möglichkeit der Graph-Optimierung vor Ausführung des Algorithmus.



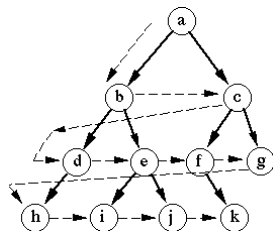
- Fragliche Anpassbarkeit an Problembeschreibung: Wie kann hier die *trap domination order* berücksichtigt werden?
 - „Forking“ des Algorithmus, sobald eine Falle erreicht wird.
 - ▶ $\mathcal{O}((n \cdot \log n + m) \cdot 3^{|T|})$, wobei $|T|$ die Anzahl der Fallen im Labyrinth ist.
 - Setzen der Distanzen zu nicht mehr begehbaren Fallen auf ∞ .
 - In beiden Fällen: Häufiges Kopieren des *algorithm state* nötig.
 - ▶ $\mathcal{O}(\text{teuer})$.

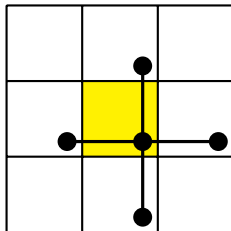
- **Teilerfolg:** Rekursive Implementation mit Backtracking, falls kein Pfad von **Start-** zu **Endpunkt** gefunden werden kann.



Algorithmus

```
q ← Queue(start)
v ← {start}
while not q.empty() do
    c ← q.pop()
    foreach n ∈ adj(c) do
        if c ∉ v then
            if c = end then
                | return True
            q.push(n)
            v ← v ∪ {n}
return False
```





- Fast lineare Komplexität: $\mathcal{O}(n + m)$.
 - Wegen der Grid-Struktur tatsächlich $\mathcal{O}(n + 4n) = \mathcal{O}(n)$.
 - ▶ Wesentlich schneller als Dijkstra.
- Simpel und deswegen flexibel anpassbar.

- In der Warteschlange q soll neben dem zu besuchenden Feld folgendes gespeichert werden:
 - Bisheriger Pfad
 - ▶ weil dessen Länge zurückgegeben werden muss.
 - Maximale bisher ausgelöste Falle α
 - ▶ um schnell prüfen zu können, ob ein Feld, das Falle α' enthält, besucht werden kann.

$$q = [((1, 1), [\dots, (1, 2), (1, 1)], A), \\ ((6, 5), [\dots, (5, 5), (6, 5)], B), \\ ((1, 6), [\dots, (1, 5), (1, 6)], A), \\ \dots]$$

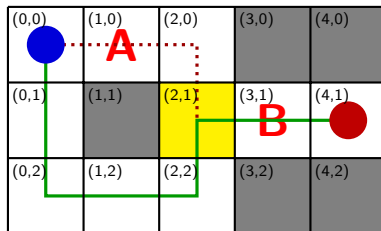
- Beim Verarbeiten von Feldern kann **für jeden Nachbarn** einer von vier Fällen auftreten:
 - 1 Nachbar besucht
 - ▶ nicht bearbeiten.
 - 2 Nachbar nicht besucht, enthält keine Falle
 - ▶ zur Warteschlange hinzufügen.
 - 3 Nachbar nicht besucht, enthält eine Falle $\alpha' \leq$ maximale bisher ausgelöste Falle α
 - ▶ nicht bearbeiten.
 - 4 Nachbar nicht besucht, enthält eine Falle $\alpha' >$ maximale bisher ausgelöste Falle α
 - ▶ mit neuer maximalen ausgelösten Falle α' zur Warteschlange hinzufügen.

$(\dots, [\dots], \alpha')$

Problemlösung

Modifizierte Breitensuche: Bereits besucht?

- Wegen Fallen kann keine einzelne Menge v verwendet werden, um zu speichern, welche Felder bereits besucht wurden:



Problemlösung

Modifizierte Breitensuche: Bereits besucht? (Möglichkeit 1)

- Naive Lösung: verwende den Pfad, der mit dem Feld in der Warteschlange gespeichert ist.
 - ▶ Bei einem nicht lösbaren Labyrinth werden alle Felder in jeder vom Start aus möglichen Reihenfolge besucht.
 - ▶ $\mathcal{O}(n!)$

$$q = [((1, 1), [\dots, (1, 2), (1, 1)], A), \\ ((6, 5), [\dots, (5, 5), (6, 5)], B), \\ ((1, 6), [\dots, (1, 5), (1, 6)], A), \\ \dots]$$

Problemlösung

Modifizierte Breitensuche: Bereits besucht? (Möglichkeit 2)

- Schneller: Verwende für jede Falle $\in T = \{A, B, \dots, Z\}$ eine Menge v_A, v_B, \dots, v_Z , zusätzlich v_0 .
- Arbeite immer mit der Menge v_α für die maximale bisher ausgelöste Falle α .
 - ▶ Jedes Feld wird maximal $|T| + 1 = 26 + 1 = 27$ mal besucht.

$$v_0 = \{(1, 0)\}$$

$$v_A = \{(3, 2)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(\dots, [\dots], \alpha)]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1) B	(2,1)	(3,1)
(0,2)	(1,2) C	(2,2)	(3,2) A
(0,3)	(1,3)	(2,3)	(3,3)

$$v_0 = \{(1, 0)\}$$

$$v_A = \emptyset$$

$$v_B = \emptyset$$

$$v_C = \emptyset$$

$$q = [((1, 0), [(1, 0)], 0)]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

$$c_1 = ((1,0), [(1,0)], 0)$$

$$\Downarrow$$

$$v_0 = \{(1,0)\}$$

$$v_A = \emptyset$$

$$v_B = \{(1,1)\}$$

$$v_C = \emptyset$$

$$q = [(\underline{(1,1)}), [(1,0)], (1,1)], B]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

Diagram illustrating a 4x4 grid with coordinates (x,y) from (0,0) to (3,3). The grid contains a blue circle at (1,0), a red circle at (0,3), and a red circle at (1,2). The cell (1,1) is highlighted in yellow and contains a red letter 'B'. The cell (1,2) contains a red letter 'C'. The cell (3,2) contains a red letter 'A'. Dotted green lines connect the blue circle at (1,0) to the yellow cell (1,1), and the red circle at (1,2) to the yellow cell (1,1).

$$c_2 = ((1, 1), [(1, 0), (1, 1)], B)$$

↓

$$v_0 = \{(1, 0)\}$$

$$v_A = \emptyset$$

$$v_B = \{(1, 1), (2, 1), (1, 0)\}$$

$$v_C = \emptyset$$

$$q = [((2, 1), [(1, 0), (1, 1), (2, 1)], B), \\ ((1, 0), [(1, 0), (1, 1), (1, 0)], B)]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

Diagram illustrating a 4x4 grid with coordinates (x,y) from (0,0) to (3,3). The grid contains a blue circle at (1,0), a red circle at (0,3), and a yellow square at (2,1). A green path is shown starting from the blue circle, moving down to (1,1), then right to (2,1), and finally right to (3,1). The cells (1,1) and (2,1) are labeled with red letters B and C respectively. The cell (3,1) is labeled with a red letter A.

$$c_3 = ((2, 1), [(1, 0), (1, 1), (2, 1)], B)$$

↓

$$v_0 = \{(1, 0)\}$$

$$v_A = \emptyset$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((1, 0), [(1, 0), (1, 1), (1, 0)], B), \\ ((3, 1), [\dots, (2, 1), (3, 1)], B)]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

A 4x4 grid with coordinates (x,y) in the top-left corner of each cell. The grid is divided into four quadrants by a vertical line between x=1 and x=2, and a horizontal line between y=1 and y=2. The top-left quadrant (x=0,1; y=0,1) is shaded gray. The top-right quadrant (x=2,3; y=0,1) is shaded gray. The bottom-left quadrant (x=0,1; y=2,3) is shaded gray. The bottom-right quadrant (x=2,3; y=2,3) is shaded gray. The cell (1,0) is highlighted in yellow and contains a blue circle. The cell (1,1) contains a red letter 'B'. The cell (1,2) contains a red letter 'C'. The cell (3,2) contains a red letter 'A'. The cell (0,3) contains a red circle. A green line connects the blue circle in (1,0) to the red letter 'B' in (1,1).

$$c_4 = ((1, 0), [(1, 0), (1, 1), (1, 0)], B)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

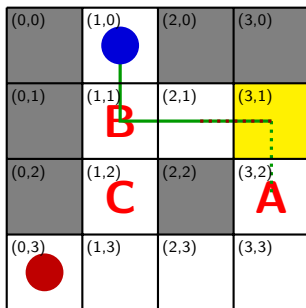
$$v_A = \emptyset$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((3, 1), [\dots, (2, 1), (3, 1)], B)]$$

Beispiel



$$c_5 = ((3, 1), [\dots, (2, 1), (3, 1)], B)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

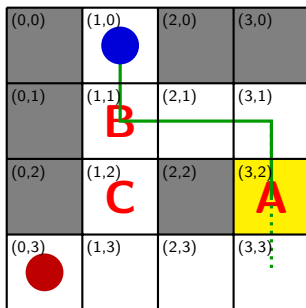
$$v_A = \{(3, 2)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(3, 2), [\dots, (3, 1), (3, 2)], A]$$

Beispiel



$$c_6 = ((3, 2), [\dots, (3, 1), (3, 2)], A)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

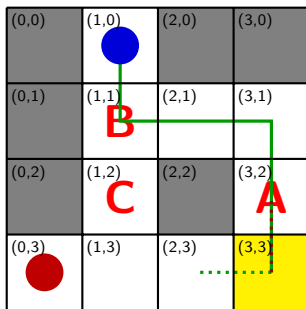
$$v_A = \{(3, 2), (3, 3), (3, 1)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((3, 3), [\dots, (3, 2), (3, 3)], A), \\ ((3, 1), [\dots, (3, 2), (3, 1)], A)]$$

Beispiel



$$c_7 = ((3, 3), [\dots, (3, 2), (3, 3)], A)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

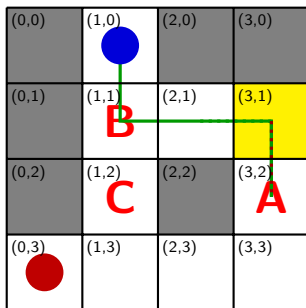
$$v_A = \{(3, 2), (3, 3), (3, 1), (2, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((3, 1), [\dots, (3, 2), (3, 1)], A), \\ ((2, 3), [\dots, (3, 3), (2, 3)], A)]$$

Beispiel



$$c_8 = ((3, 1), [\dots, \underline{(3, 2)}, (3, 1)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

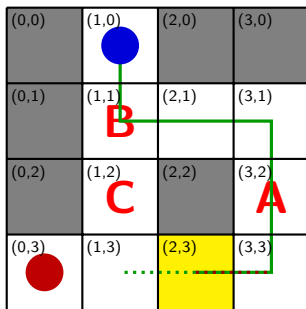
$$v_A = \{\dots, (3, 3), (3, 1), (2, 3), \underline{(2, 1)}\}$$

$$v_B = \{(\underline{1, 1}), (2, 1), \underline{(1, 0)}, (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((2, 3), [\dots, (3, 3), (2, 3)], A), \\ (\underline{(2, 1)}, [\dots, (3, 1), (2, 1)], A)]$$

Beispiel



$$c_g = ((2, 3), [\dots, (3, 3), (2, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

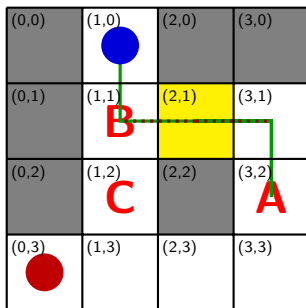
$$v_A = \{\dots, (3, 1), (2, 3), (2, 1), (1, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((2, 1), [\dots, (3, 1), (2, 1)], A), \\ ((1, 3), [\dots, (2, 3), (1, 3)], A)]$$

Beispiel



$$c_{10} = ((2, 1), [\dots, (3, 1), (2, 1)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

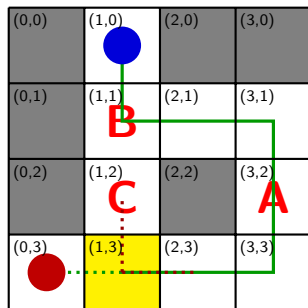
$$v_A = \{\dots, (3, 1), (2, 3), (2, 1), (1, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [((1, 3), [\dots, (2, 3), (1, 3)], A)]$$

Beispiel



$$c_{11} = ((1, 3), [\dots, (2, 3), (1, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

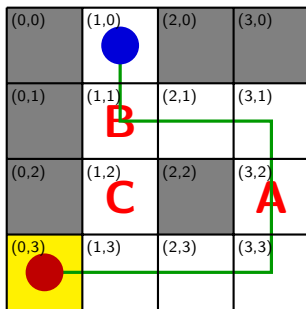
$$v_A = \{\dots, (2, 3), (2, 1), (1, 3), (0, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(0, 3), [\dots, (1, 3), (0, 3)], A]$$

Beispiel



$$c_{12} = ((0, 3), [\dots, (1, 3), (0, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

$$v_A = \{\dots, (2, 3), (2, 1), (1, 3), (0, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (1, 0), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = []$$

Problemlösung

Modifizierte Breitensuche: Bereits besucht? (Möglichkeit $2\frac{1}{2}$)

- Prüfe nicht nur in v_α .
- Erzeuge Menge T' aller Fallen, die auf dem bisherigen Pfad liegen.
- Prüfe in $\bigcup_{t \in T'} v_t$.
 - ▶ Felder werden (hier) nicht mehrfach besucht.
 - ▶ Kürzere Laufzeit (Beispiel: 12 Schritte vs. 9 Schritte).
- Korrektheit nicht bewiesen, aber kein Gegenbeispiel gefunden.

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1) B	(2,1)	(3,1)
(0,2)	(1,2) C	(2,2)	(3,2) A
(0,3)	(1,3)	(2,3)	(3,3)

$$v_0 = \{(1, 0)\}$$

$$v_A = \emptyset$$

$$v_B = \emptyset$$

$$v_C = \emptyset$$

$$q = [((1, 0), [(1, 0)], 0)]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

$$c_1 = ((1,0), [(1,0)], 0)$$

\Downarrow

$$v_0 = \{(1,0)\}$$

$$v_A = \emptyset$$

$$v_B = \{(1,1)\}$$

$$v_C = \emptyset$$

$$q = [(\underline{(1,1)}), [(1,0)], (1,1)], B]$$

Beispiel

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

$$c_2 = ((1, 1), [(1, 0), (1, 1)], B)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

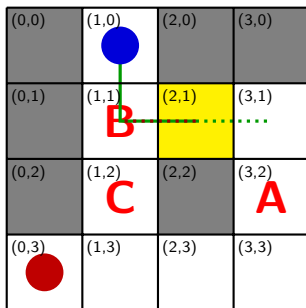
$$v_A = \emptyset$$

$$v_B = \{(1, 1), (2, 1)\}$$

$$v_C = \emptyset$$

$$q = [(2, 1), [(1, 0), (1, 1)], (2, 1)], B]$$

Beispiel



$$c_3 = ((2, 1), [(1, 0), (1, 1), (2, 1)], B)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

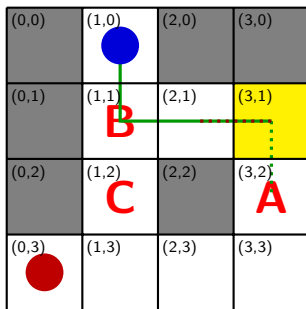
$$v_A = \emptyset$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(3, 1), \dots, (2, 1), (3, 1)], B]$$

Beispiel



$$c_4 = ((3, 1), [\dots, (2, 1), (3, 1)], B)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

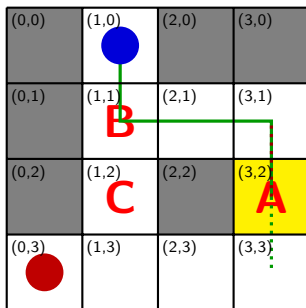
$$v_A = \{(3, 2)\}$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(3, 2), [\dots, (3, 1), (3, 2)], A]$$

Beispiel



$$c_5 = ((3, 2), [\dots, (3, 1), (3, 2)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

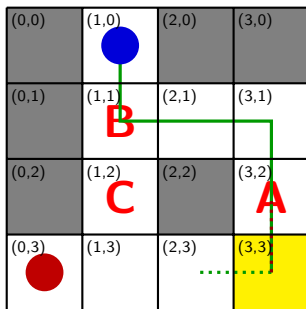
$$v_A = \{(3, 2), (3, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(3, 3), [\dots, (3, 2), (3, 3)], A]$$

Beispiel



$$c_6 = ((3, 3), [\dots, \underline{(3, 2)}, (3, 3)], A)$$

\Downarrow

$$v_0 = \{(1, 0)\}$$

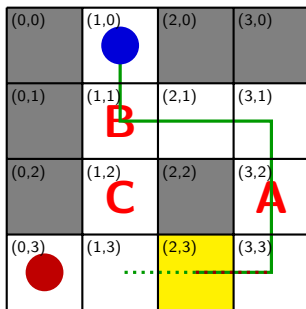
$$v_A = \{(\underline{(3, 2)}, (3, 3), \underline{(2, 3)})\}$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(\underline{(2, 3)}, [\dots, (3, 3), (2, 3)], A)]$$

Beispiel



$$c_7 = ((2, 3), [\dots, (3, 3), (2, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

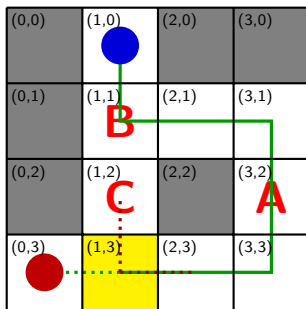
$$v_A = \{(3, 2), (3, 3), (2, 3), (1, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(1, 3), [\dots, (2, 3), (1, 3)], A]$$

Beispiel



$$c_8 = ((1, 3), [\dots, (2, 3), (1, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

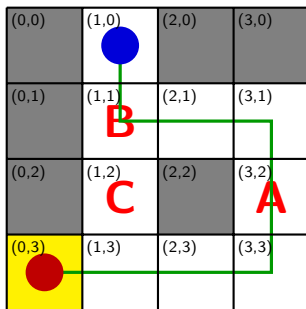
$$v_A = \{\dots, (3, 3), (2, 3), (1, 3), (0, 3)\}$$

$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = [(0, 3), [\dots, (1, 3), (0, 3)], A]$$

Beispiel



$$c_9 = ((0, 3), [\dots, (1, 3), (0, 3)], A)$$

$$\Downarrow$$

$$v_0 = \{(1, 0)\}$$

$$v_A = \{\dots, (3, 3), (2, 3), (1, 3), (0, 3)\}$$

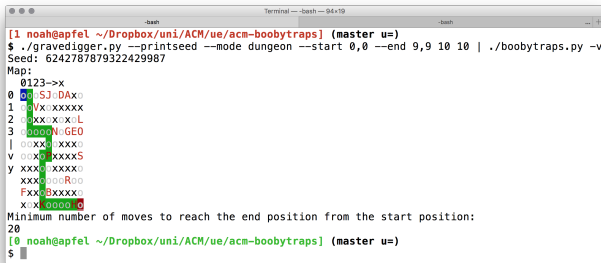
$$v_B = \{(1, 1), (2, 1), (3, 1)\}$$

$$v_C = \emptyset$$

$$q = []$$

Map generator und pretty printing

- Simpler *map generator*, um Performance auf großen Labyrinthen ($w \cdot h \leq 40000$) testen zu können: `gravedigger.py`.
- Außerdem sorgen bei `boobytraps.py` die Flags `-v` und `-v2` für eine übersichtliche Ausgabe des Labyrinths und des Pfades.

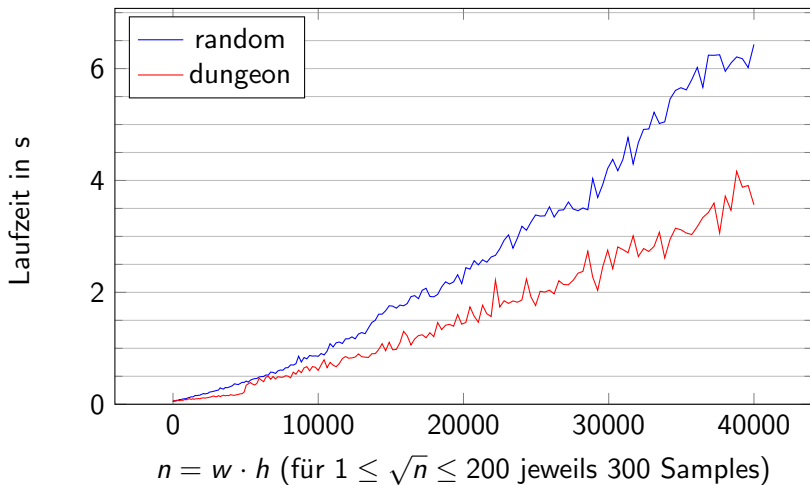


```
noah@apfel ~/Dropbox/uni/ACM/ue/acm-boobytraps [master u=]
$ ./gravedigger.py --printseed --mode dungeon --start 0,0 --end 9,9 10 10 | ./boobytraps.py -v
Seed: 6242787879322429987
Map:
0123->x
0 00pSJoDAx0
1 00Vx0xxxxx
2 00xx0x0x0L
3 0000NoGEO
| 00xx00xxx0
v 00x00xxxxS
y xxx00xxxx0
  xxx000R00
  Fxx0Bxxxx0
  x0x00000
Minimum number of moves to reach the end position from the start position:
20
[0 noah@apfel ~/Dropbox/uni/ACM/ue/acm-boobytraps [master u=]
$
```

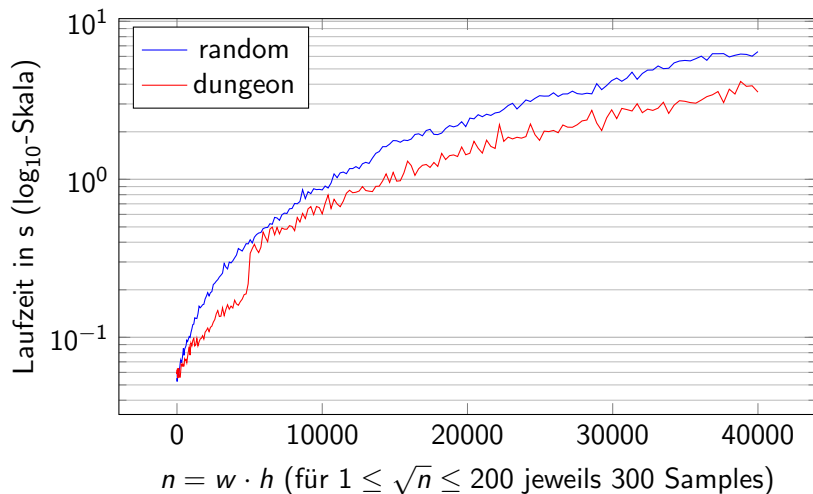
Demo

- Erinnerung: Knotenanzahl n , Kantenanzahl m .
- Komplexität der Breitensuche: $\mathcal{O}(n + m)$.
- Sei $|T''|$ die Anzahl der paarweise verschiedenen (*unique*) Fallen im Labyrinth.
- Komplexität wegen mehrfachem Besuchen von Feldern aufgrund der Fallen: $\mathcal{O}((n + m) \cdot (1 + |T''|))$.
- Es gilt $0 \leq |T''| \leq |T| = 26$ und $1 \leq m \leq 4n$.
- Wähle $|T''| = |T| = 26$ und $m = 4n$.
 - ▶ Komplexität $\mathcal{O}((n + 4n) \cdot (1 + 26)) = \mathcal{O}(n) = \mathcal{O}(w \cdot h)$.

Benchmarks



Benchmarks



Danke für die Aufmerksamkeit!

Quellen

- Grabräuber-Illustration: <https://c418.bandcamp.com/album/catacomb-snatch-original-soundtrack>
- Beschreibung der Breitensuche nach <https://de.wikipedia.org/wiki/Breitensuche>
- Illustration zur Breitensuche: <http://www.cse.unsw.edu.au/~billw/Justsearch1.gif>

Fork me on GitHub: <https://github.com/doersino/acm-boobytraps>