Tutorial 2: Search Spaces

Finn-Lasse Jörgensen, Frederik Wille, Tronje Krabbe

October 31, 2016

Exercise 2.3: (Search Space Construction 2)

Das Spielfeld kann als gerichteter Graph repräsentiert werden, wobei Knoten, auf denen sich ein Spieler befindet, besonders markiert sind.

Eine Möglichkeit, das Problem zu lösen, wäre alle möglichen Zielpunkte von Mister X dahingehend zu überprüfen, ob die Kriminalbeamten es dort hin schaffen. Dies könnte z.B. mithilfe von Dijkstras Algorithmus geschehen. Wenn die Position aller Detektive zu einem Zielpunkt von Mister X eine größere Minimaldistanz hat, als Schritte erlaubt sind, so ist dieser ein guter Zug.

Diese Strategie ist wohl aber wahrscheinlich recht langsam. Eine andere Herangehensweise wäre, alle Knoten, die von den Kriminalbeamten erreichbar sind, zu markieren, z.B. per einer Variation der Breitensuche, die alle Knoten, die sie abgeht, markiert, und abbricht, wenn die Distanz größer ist, als die erlaubte Anzahl Schritte.

Jetzt sucht man für Mister X, z.B. per Tiefensuche, einen Knoten der nicht markiert, und natürlich auch nicht zu weit weg ist.

Excercise 2.4: (Search Space Construction 3)

Placing furniture in a flat

Der Suchraum wird durch alle möglichen Platzierungen der Möbel gebildet. Der Start der Suche ist ohne ein platziertes Möbelstück und Kanten zu anderen Zuständen entstehen durch Platzieren eines weiteren Mobelstückes. Ziel der Suche ist eine (von mehreren) für den Bewohner optimale Platzierung. Wenn man davon ausgeht, dass nur eine begrenzte Anzahl an Möbeln zur Verfügung steht, ist der Suchraum begrenzt. Dadurch, dass wir nur Möbel in die Wohnung platzieren, entsteht ein gerichteter Graph ohne Zyklen. Da

wir nur eine von mehreren möglichen optimalen Lösungen suchen und einen endlichen, zyklen-freien Graphen haben, bietet sich eine Tiefensuche an.

Construction Site planning

Am einfachsten wäre es, die einzelnen Konstruktions-Schritte als Knoten eines Baumes darzustellen. Dies ist leider in den meisten Fällen nicht möglich, da einige Schritte mehrere Abhängigkeiten haben könnten. Es wird also ein gerichteter Graph benötigt, wobei dieser wahrscheinlich einem Baum ähnlich sehen würde. Er wird außerdem keine Zyklen enthalten, da wohl kaum ganze Schritte rückgängig gemacht werden.

Damit das Haus fertig gebaut ist, müssen alle Schritte abgeschlossen sein. Also muss man eine Knotenfolge finden, die vom Startknoten zum Endknoten verläuft, und in der die Knoten in einer Reihenfolge angeordnet sind, die keine Konflikte enthält.

Dies kann bewerkstelligt werden, indem man den Startknoten zu einer Queue hinzufügt, und von dort aus eine Breitensuche macht. Jeder Knoten, der gefunden wird, und noch nicht in der Struktur enthalten ist, wird genau dann hinzugefügt, wenn alle seine Eltern-Knoten bereits enthalten sind. Erreicht die Breitensuche einen Status, in dem es nicht mehr weiter geht, muss von vorne angefangen werden; so lange, bis alle Knoten in der Queue eingeordnet sind. Danach kann das Haus entsprechend der Reihenfolge der Schritte in der Queue gebaut werden.

Elevator

Es gibt ganze Bücher darüber, wie Fahrstühle gut implementiert werden können [1].

Ein simpler Algorithmus, der auch in der Informatik als Algorithmus für Festplatten-Scheduling genutzt wird, funktioniert wiefolgt: Solange es noch Anfragen in der Richtung, in der sich momentan bewegt wird, sind, fahre weiter in diese Richtung. Sind keine Anfragen in der Richtung mehr vorhanden, halte an und gehe in einen Warte-Zustand, oder fahre in die andere Richtung, wenn da Anfragen vorhanden sind. Anfragen können in dieser Darstellung sowohl von außerhalb des Fahrstuhls kommen, oder von innerhalb. Dem Fahrstuhl ist es egal, ob eine Passagier in der angefragten Etage ein- oder aussteigen will. Die genutzte Datenstruktur für Anfragen könnte eine Priority-Queue sein, wobei Anfragen, die in Fahrtrichtung liegen, eine höhere Priorität haben, als Anfragen in die andere Richtung.

[1] `https://books.google.de/books?id=GteIiGQT1S4C&pg=PA278&redir_esc=y&hl=en`