Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

22. Dezember 2016

Datenstrukturen & Algorithmen Lösungen zu Blatt 13 HS 16

Lösung 13.1 *Pfadplanung in Labyrinthen.*

a) Wir definieren einen gerichteten Graphen, der einen Knoten für jeden möglichen Zustand des Systems enthält. Ein Zustand wird vollständig durch drei Parameter festgelegt:

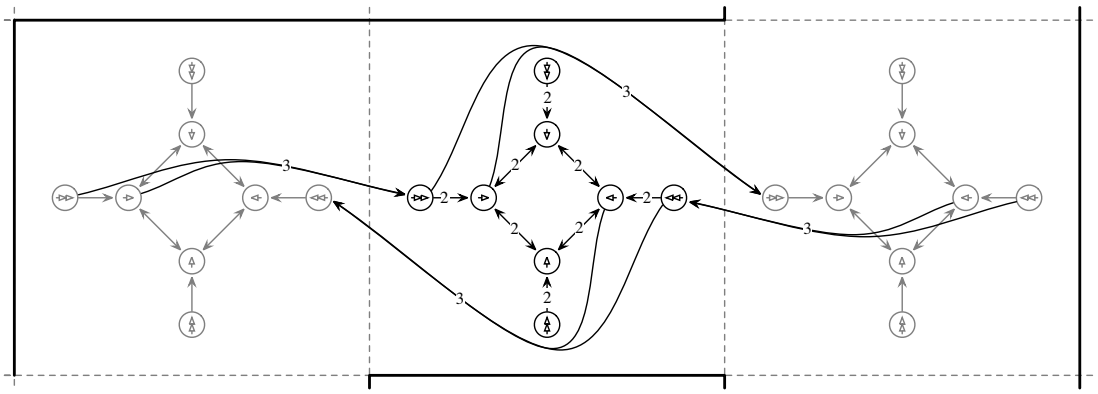
1) Das Feld auf dem der Roboter steht,

2) die Blickrichtung des Roboters und

3) ob der Roboter stillsteht oder in Bewegung ist.

Für jedes Feld, auf dem der Roboter stehen kann, haben wir also 8 Zustände (4 Blickrichtungen multipliziert mit 2 Möglichkeiten ob der Roboter steht oder sich bewegt). Wir konstruieren unseren Graphen entsprechend mit 8 Knoten pro Feld des Labyrinths. Ausserdem brauchen wir noch einen Knoten, der dem Zielzustand entspricht, in dem der Roboter entkommen ist. Wir fügen Kanten für Bewegung, Rotation und Stehenbleiben ein und gewichten diese entsprechend der Zeit, die die entsprechende Operation benötigt. Der kürzeste Pfad im Graphen vom Startzustand des Roboters zum Zielzustand entspricht einer Folge von Operationen des Roboters, die ihn schnellstmöglich entkommen lässt.

Das folgende Bild illustriert unsere Konstruktion an einem kleinen Ausschnitt eines Labyrinths. Die Beschriftung der Knoten zeigt die Blickrichtung des Roboters; Doppelpfeile bedeuten, dass er in Bewegung ist. Wir könnten natürlich Zustände entfernen, die gar nicht vorkommen können.



b) Da der konstruierte Graph keine negativen Kantengewichte enthält, kann ein kürzester Pfad mithilfe des Algorithmus von Dijkstra gefunden werden.

c) Für jedes Feld des Labyrinths wird eine konstante Anzahl Knoten (nämlich 8) und eine konstante Anzahl Kanten (höchstens 20) verwendet. Ist n die Anzahl der Felder im Labyrinth, dann sind $|V| \in \mathcal{O}(n)$ und $|E| \in \mathcal{O}(n)$. Die Laufzeit des Algorithmus von Dijkstra

ist also durch $\mathcal{O}(n \log n)$ beschränkt.

Anmerkung: Für einen gerichteten, ungewichteten Graphen kann ein kürzester Pfad auch mit einer Breitensuche gefunden werden. Da unser Graph nur ganzzahlige Kantengewichte hat, die alle größer als 0 und durch eine Konstante nach oben beschränkt sind, können wir einen erweiterten, ungewichteten Graphen konstruieren, indem wir eine Kante mit Gewicht $w > 1$ durch w Kanten mit je Gewicht 1 ersetzen. Dazu fügen wir $w-1$ Hilfsknoten ein. Hat diese Kante in unserem ursprünglichen Graphen einen Knoten x, der mit einem Knoten y verbunden ist, so sind diese Knoten im erweiterten Graphen durch einen Pfad bestehend aus w Kanten verbunden. Somit haben im erweiterten Graphen alle Kanten Gewicht 1 und wir können die Breitensuche verwenden, um einen kürzesten Pfad zu finden. Da die Kantengewichte durch eine Konstante nach oben beschränkt sind, werden insgesamt nur $\mathcal{O}(n)$ Knoten und Kanten hinzugefügt und die Laufzeit ist in $\mathcal{O}(n)$.