Übungen zur Vorlesung
Grundlagen der Künstlichen Intelligenz
Sommersemester 2024

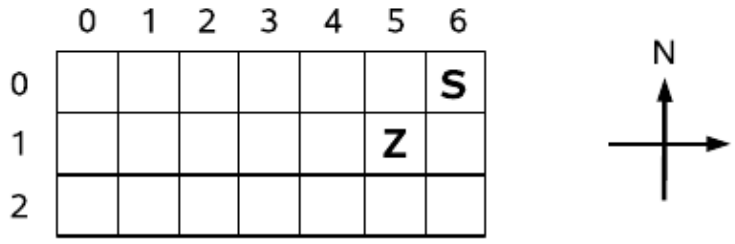
Institut für Informatik 4
Priv.-Doz. Dr. V. Steinhage
Friedrich-Hirzebruch-Allee 8
53115 Bonn
Email: steinhage@cs.uni-bonn.de
WWW: http://net.cs.uni-bonn.de/ivs/

Blatt 2 (10 Punkte)

Abgabe durch Hochladen (nur PDF-Format bzw. Python-Code) auf der eCampus-Seite bis **Sonntag, 21.04.2024, 12:00 Uhr**, in Gruppen von 3 Personen.

Aufgabe 2.1: Uninformierte Suche $(2 + 1 = 3)$

Wenden Sie bei den folgenden Aufgaben den Suchalgorithmus GENERAL-SEARCH (Vorlesung 3, Folie 19) an. Beachten Sie bitte genau, an welcher Stelle des Algorithmus der GOAL-TEST angewendet wird.
Ein Roboter soll in einem Raum einen Weg von seinem ihm bekannten Startzustand S $= (6, 0)$ zu einem ihm unbekannten Zielfeld Z suchen. Er hat keinerlei Sensoren, um die Richtung des Ziels zu bestimmen. Er besitzt lediglich einen Sensor um festzustellen, ob er sich aktuell im Zielfeld befindet (GOAL-TEST). Mit einem Schritt kann der Roboter entweder horizontal oder vertikal in ein angrenzendes Feld wechseln, diagonale Bewegungen sind nicht erlaubt, d.h. seine möglichen OPERATORS sind *Ost, Nord, West, Süd*. Für die Aufgabenlösung haben Sie **genau** diese Reihenfolge zu nutzen! Die Roboterumgebung sei durch folgende Abbildung definiert:



Die Zahlen geben die Feldkoordinaten der Umgebung an. Beachten Sie das *Formulate-Search-Execute-Schema* (Folien 6 - 7, 3. Vorl.). Es handelt sich also um eine Offline-Planung bzw. Suche nach der vollständigen Aktionssequenz. Der Roboter kennt a priori das gesamte Szenario und weiß damit auch, wo sich Start- und Zielfeld sowie die Wände des Raums befinden etc. Ferner wird er nie versuchen, die Umgebung zu verlassen.

a) Suchen Sie in diesem Aufgabenteil das Zielfeld mittels *Breitensuche*. Gehen Sie davon aus, dass der Roboter keine Möglichkeit hat, sich bereits besuchte Felder zu merken.

   i) Beschriften Sie die Felder der Umgebung nach der Reihenfolge ihrer (evtl. mehrfachen) Expansion bis ersichtlich ist, wann das Ziel entdeckt wird.

   ii) Notieren Sie auch die Entwicklung der Warteschlange (QUEUE).

   iii) Wieviele Knoten werden insgesamt expandiert?

   iv) Wieviele Einträge umfasst die Warteschlange maximal?

   v) Zeichnen Sie auch den resultierenden Pfad ein.

   vi) Wie lang ist der resultierende Pfad?

b) Führen Sie dieselbe Aufgabenstellung wie in a) mit *Tiefensuche* durch.

Aufgabe 2.2: Bewertung von Suchstrategien **(1)**

Bewerten Sie anhand der Kriterien *Vollständigkeit, Optimalität, Zeit-* sowie *Platzkomplexität* eine *Bidirektionale Suche*, die für eine Suchrichtung *Breitensuche* und für die andere *Tiefensuche* verwendet. Halten Sie sich bzgl. der Komplexitätsmaße an die Angaben in der Vorlesung.

Aufgabe 2.3: Verwandtschaft von Suchstrategien **(0)**

Diskutieren Sie in der Übungsgruppe:

a) Die *Breitensuche* ist ein Sonderfall der *uniformen Kostensuche*.

b) *Uniforme Kostensuche* ist ein Sonderfall der *A*-Suche*

Beachten Sie, dass von Aufgaben 2.4 und 2.5 nur eine auszusuchen ist, die abgegeben werden kann/muss. Es wird dementsprechend nur eine Aufgabe bepunktet korrigiert. Bei Abgabe beider Aufgaben bitte kenntlich machen, welche bewertet werden soll. Erfolgt keine solche Kennzeichnung, wird Aufg. 2.4 bewertet.

Aufgabe 2.4: Informierte Suche **(entweder hier: $1 + 1 + 2 = 4$)**

Sie programmieren nun die KI für die Gegner in Ihrem Tower-Defense-Spiel. Die Gegner können sich in acht verschiedene Richtungen bewegen, die Kosten jedes Schrittes sehen Sie im Bild unten. Die Gegner sollen nun mittels **A*** den kürzesten Weg vom Start (S) zum Ziel (Z) finden. Ihre Gegner merken sich, welche Knoten im Suchbaum sie schon expandiert haben. Nutzen Sie die Euklidische Distanz[1] als Heuristik. Für bessere Rechenbarkeit runden Sie zudem den Wert der Euklidischen Distanz kaufmännisch auf die erste Nachkommastelle und multiplizieren diesen Wert mit 10.

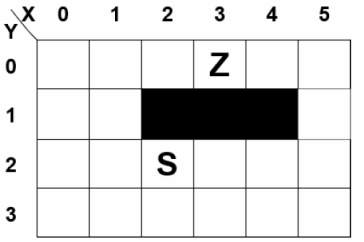


Figure 1: Karte: Start (S), Ziel (Z)

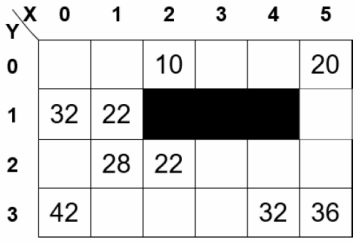


Figure 2: *h(n)*-Kosten

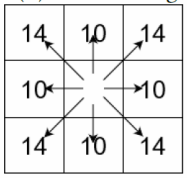


Figure 3: Bewegungsrichtungen und Kosten

[1] $d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

a) Berechnen Sie für jedes Feld, das kein Hindernis darstellt, die *h(n)*-Kosten, indem Sie die Karte der *h(n)*-Kosten vervollständigen.

b) Ist die Euklidische Distanz in diesem Szenario zulässig? Begründen Sie Ihre Antwort. Nennen Sie unabhängig davon eine weitere hier unzulässige und zulässige Heuristik.

c) Führen Sie den **A***-Algorithmus aus. Expandieren Sie die Knoten im Osten startend gegen den Uhrzeigersinn (O, NO, N, ..). Zeichnen Sie dafür den Suchbaum und geben Sie die Reihenfolge der expandierten Knoten an. Zeichnen Sie den Zielpfad in die Karte ein. Planen Sie genug Platz ein und lassen Sie Knoten weg, deren Kosten Sie schon einmal berechnet haben.

Aufgabe 2.5: Programmieraufgabe: Routenplanung mit A* (oder hier: $2 + 2 = 4$)

Vorbereitung. Laden Sie bitte das ZIP-Archiv AIMA-Py-Routenplanung.zip herunter von unserer eCampus-Seite unter Kursunterlagen » Python und AIMA Python » AIMA-Py Routing. Das ZIP-Archiv enthält die beiden Ordner *aima* und *routen* sowie ein Handbuch.
Die gesamte AIMA-GUI wird gestartet mit dem Skript AIMA.py im Ordner *aima*. AIMA.py importiert für verschiedene Zeichenoperationen zeichne.py und für die Bearbeitung von Graphen GRAPH.py, das wiederum Node.py und zeichne.py importiert. Die restlichen Skripte im Ordner *aima* sind noch nicht von Bedeutung.
Im Ordner *routen* bietet Aimaqueue.py verschieden Queue-Klassen an. Die Klasse FiFoQueue wird von breitenqueue.py importiert, das die die Breitensuche umsetzt.
Eine Straßenkarte is vorgegeben in der ASCII-Datei *7Cities12Streets.map*. Für die Aufgabe ist nur diese Karte nötig und die Datei nicht zu bearbeiten.
Wenden Sie zunächst die Breitensuche an, um den kürzesten Weg von Berlin nach Stuttgart zu finden: Start von Skript AIMA.py erzeugt die GUI als neues Fenster. Dort *Select Application » Routefinding » Openmap*. Auswahl von *7Cities12Streets.map*. Die Karte erscheint. Die Zahlen geben die wahren Straßenlängen an. Auswahl *Load Algorithm » Load* » Ordner *routen* » *breitenqueue.py » Run*. Das gesamte GUI-Fenster wird am besten maximiert. Die Suche terminiert mit der Melding *Done*. Das Ergebnis und der Weg dazu stehen im rechten Anzeigeteil der GUI. Für Berlin-Stuttgart wird z.B. eine Route mit 370 km gefunden.

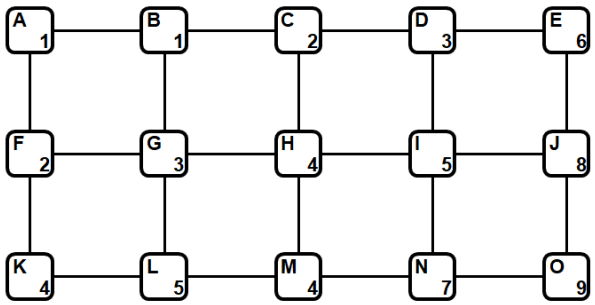
Aufgabe: Implementieren Sie die A*-Suche zur Lösung der Routenplanung.

a) Ergänzen Sie dazu zunächst die Queue-Klasse *SortedQueue* in Aimaqueue.py, welche die zu expandierenden Knoten nach ihren f-Kosten heraussucht (ca. 5 - 6 Codezeilen).

b) Gehen Sie vom Skript *breitenqueue.py* (Breitensuche zur Lösung der Routenplanung) aus. Kopieren Sie dies in ein neues Skript *asternqueue.py* und modifizieren Sie die gegebene Breitensuche nun geeignet zur A*-Suche. Es handelt sich dabei um lediglich ca. 5 - 6 Codezeilen, die zu erstellen bzw. zu ändern sind. Als Heuristik ist die Luftliniendistanz zu verwenden. Diese kann über graph.luftlinie(nameA, nameB) erfragt werden.
Hinweis: Die A*-Suche sollte die Route Berlin-Stuttgart mit 360 km finden.

Aufgabe 2.6: Lokale Suche $(0,5 + 1,5 = 2)$

Eine Problemstellung ergab folgenden Graphen von Zuständen (Knoten) und Aktionen (Kanten). Die Zahlen in den Zuständen geben deren Bewertung wieder.



Übertragen Sie zunächst den Graphen in Ihre Lösungsabgabe.

a) Benennen Sie bitte alle möglichen Endzustände einer lokalen Suche mittels *Hill Climbing*, indem Sie diese im Graphen mit einem Kreis umzeichnen.

b) Für alle übrigen Zustände markieren Sie bitte durch je einen fetten Richtungspfeil die zum Nachfolgezustand führende Aktion.