Technische Universität München
Fakultät für Informatik
Prof. Dr. M. Broy

SS 2003
Übungsblatt 9
20. Juni 2003

Übungen zur Vorlesung Einführung in die Informatik IV

Aufgabe 37 Nimm-Spiele

Gegeben ist ein Stapel mit s Streichhölzern, von dem 2 Spieler abwechselnd mindestens 1 und höchstens n Streichhölzer wegnehmen. Wer die letzten Streichhölzer nimmt, hat gewonnen.

(a) Skizzieren Sie für $s = 5$ und $n = 3$ den Baum aller möglichen Spielverläufe.

(b) Geben Sie ein Prädikat an, das für eine gegebene Spielsituation ermittelt, ob es sich für den ziehenden Spieler um eine Gewinnstellung handelt, d.h. um eine Stellung, aus der heraus er das Spiel für sich entscheiden kann, unabhängig von der Vorgehensweise des Gegners.

(c) Ermitteln Sie durch Breitensuche, ob die obige Ausgangssituation eine Gewinnstellung ist.

(d) Die Bestimmung, ob eine Spieler eine Gewinnstellung hat, lässt sich besonders einfach durch ein Tiefensuchverfahren mit Backtracking realisieren. Geben Sie Pseudocode für ein Tiefensuchverfahren mit Backtracking an, das für beliebige *s* und *n* als Eingabe bestimmt, ob der beginnende Spieler eine Gewinnstellung hat.

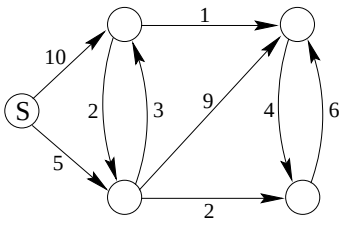
Aufgabe 38 Algorithmus von Dijkstra

Gegeben sei ein (gerichteter oder ungerichteter) Graph mit Knotenmenge `V` und positiven Kantengewichten `d: V × V → ℕ ∪ {∞}`. Der Algorithmus von Dijkstra zur Berechnung der Länge `l(v)` der kürzesten Pfade von einem Startknoten `s` zu jedem beliebigen anderen Knoten `v` lässt sich wie folgt formulieren:

```
fct dijkstra = (set node V, node s):
   forall v ∈ m do l(v) := ∞ od
   l(s) := 0;
   W := V;
   while W ≠ ∅ do
         Finde einen Knoten w ∈ W mit l(w) minimal
         W := W - {w};
         forall n ∈ N(w), n ∈ W do
               if l(w) + d(w, n) < l(n)
               then l(n) := l(w) + d(w,n) fi
         od
   od
```

dabei bezeichne `N(w)={v ∈ V: d(v,w) < ∞}` die Menge der Nachbarknoten von `w` im Graph.

(a) Bestimmen Sie mit dem Algorithmus von Dijkstra für den folgenden Graphen die Länge der kürzesten Verbindung vom Knoten *s* zu allen anderen Knoten.



(b) Schätzen Sie die Zeitkomplexität des Algorithmus ab.

(c) Konstruieren Sie einen Graphen mit ganzzahligen (nicht notwendigerweise positiven) Kantengewichten, für den der Algorithmus von Dijkstra ein falsches Ergebnis liefert.

Aufgabe 39 (P) Rundreiseproblem, Greedy-Heuristik

Aufgrund des hohen Aufwands zur Bestimmung einer exakten Lösung werden in der Praxis häufig Näherungsverfahren zur Lösung des Rundreiseproblems eingesetzt. In dieser Aufgabe soll die bestehende Lösung aus Aufgabe 36 um ein Verfahren erweitert werden, dass eine möglichst kurze Rundreise in einem beliebigen Graphen durch Anwendung einer sog. Greedy-Strategie (*greedy*: engl. gierig) ermittelt. Hierbei wird ausgehend von einem zufällig gewählten Startpunkt der jeweils nächste besuchte Knoten nach der kürzesten Entfernung zum vorhergehenden Knoten gewählt (vgl. Aufgabe 34b).

(a) **(T)** Diskutieren Sie Vor- und Nachteile der oben angegebenen Heuristik. Geben Sie ein charakteristisches Beispiel an, bei dem dieses Verfahren vergleichsweise schlechte Lösungen ermittelt. Wie könnte die Heuristik in dieser Hinsicht verbessert werden?

(b) Erweitern Sie die Klasse `TSP` aus Aufgabe 36 um eine Methode `shortestPathGreedy()`, die mittels der oben angegebenen Heuristik eine möglichst kurze Rundreise berechnet.

(c) Um Güte und Effizienz einer Heuristik zu überprüfen, ist es gewöhnlich aufschlussreich, eine Menge unterschiedlicher Graphen mit wachsender Anzahl von Knoten zu untersuchen und die erhaltene Lösung mit einer zufällig bestimmten Lösung zu vergleichen. Entwickeln Sie daher eine Methode `generateRandomGraph()` der Klasse `TSP`, die einen neuen, zufällig gewählten Graphen mit vorgegebener Anzahl von Knoten erzeugt, sowie eine Methode `shortestPathRandom()`, welche eine zufällig gewählte Rundreise ermittelt.

(d) Analysieren Sie das in Teilaufgabe a) entwickelte Näherungsverfahren durch Anwendung auf verschiedene, zufällig gewählte Graphen mit steigender Knotenzahl. Vergleichen Sie das erhaltene Ergebnis hinsichtlich Genauigkeit und benötigtem Aufwand mit der exakten sowie zufällig bestimmten Lösung. Dokumentieren Sie ihre Erkenntnisse durch ein übersichtlich formatiertes Ausgabeprotokoll.

Hinweise: Die Funktion `Math.random()` liefert weitgehend zufällige Gleitpunktzahlen im Bereich zwischen 0 und 1. Die benötigte Rechenzeit einer Java-Funktion kann zweckmäßig durch geeignete Aufrufe von `System.currentTimeMillis()` in Millisekunden ermittelt werden.