Algorithmen I

Tutorium 1 - 10. Sitzung

Dennis Felsing

dennis.felsing@student.kit.edu
www.stud.uni-karlsruhe.de/~ubcqr/algo

2011-06-20

Klausur

Klausuranmeldung jetzt im Studienportal möglich!
Klausur am 19.07.

Überblick

1. **Graphtraversierung**
   - Wiederholung
   - Breitensuche
   - Bipartite Graphen
2. **Kürzeste Pfade**
   - Allgemeines
   - Bellman-Ford-Algorithmus
   - Dijkstra-Algorithmus
3. **Aufgaben**
   - Japanische Leiterspiele

Wiederholung

Tiefensuche

Vorgehen:

Wiederholung

Tiefensuche

Vorgehen:

1. Unbesuchten Knoten auswählen
2. Knoten besuchen und eintragen von wo und wann
3. Rekursiv alle Kind besuchen
4. Eintragen wann abgeschlossen
5. Falls es noch unbesuchte Knoten gibt, zu Schritt 1

Breitensuche

Idee

Kürzeste Wege von Wurzel zu allen anderen Knoten finden

Breitensuche

Idee

Kürzeste Wege von Wurzel zu allen anderen Knoten finden

Noch einfacher als Tiefensuche:

Vorgehen

1. Wurzel entdecken
2. Alle Nachbarn von Wurzel entdecken
3. Deren noch unbesuchte Nachbarn entdecken
4. ...

Beim Entdecken Distanz zu Wurzel und von wo aus entdeckt speichern.

Es entsteht ein **Breitensuchbaum**. Eigenschaften?

Breitensuche

Idee

Kürzeste Wege von Wurzel zu allen anderen Knoten finden

Noch einfacher als Tiefensuche:

Vorgehen

1. Wurzel entdecken
2. Alle Nachbarn von Wurzel entdecken
3. Deren noch unbesuchte Nachbarn entdecken
4. ...

Beim Entdecken Distanz zu Wurzel und von wo aus entdeckt speichern.

Es entsteht ein **Breitensuchbaum**. Eigenschaften?
Enthält nur kürzeste Wege zu Knoten von Wurzel aus

Breitensuche

```
BFS(G, s)
 1  for each u ∈ G.V
 2        u.visited = FALSE
 3        u.distance = ∞
 4        u.predecessor = NIL
 5  s.distance = 0; s.visited = TRUE; Q = ∅; ENQUEUE(Q, s)
 6  while Q ≠ ∅
 7        u = DEQUEUE(Q)
 8        for each v ∈ G.Adj[u]
 9              if v.visited == FALSE
10                    v.visited = TRUE
11                    v.distance = u.distance + 1
12                    v.predecessor = u
13                    ENQUEUE(Q, v)
```

Bipartite Graphen

Definition

Ein Graph ist **bipartit**, wenn sich seine Knoten in zwei Mengen aufteilen lassen, so dass alle Kanten von einer Menge in die andere verlaufen.

Test

Sei G zusammenhängend:

1. Breitensuche bei beliebigem Knoten starten und diesen mit 0 markieren
2. Neu entdeckte Knoten bekommen andere Markierung als Vorgänger (0 oder 1)
3. Wiederentdeckte Knoten überprüfen statt einfach ignorieren

Wie lässt sich der Test auf nicht zusammenhängende Graphen erweitern?

Kürzeste Pfade

Kürzeste Pfade

Definitionen

- **Gewicht eines Pfades** $p =< v_0, v_1, ...; v_k >$ ist $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
- **Gewicht eines kürzesten Pfades** von u nach v:

$$\delta(u, v) = \begin{cases} min\{w(p) : p \text{ Pfad von } u \text{ nach } v\} & \text{falls } p \text{ ex.} \\ \infty & \text{sonst} \end{cases}$$

- **Kürzester Pfad** von u nach v ist p mit $w(p) = \delta(u, v)$

Kürzeste Pfade

Initialisierug

INITIALIZE_SINGLE_SOURCE(G, s)

```
1  for each v ∈ G.V
2      u.distance = ∞
3      u.predecessor = NIL
4  S.distance = 0
```

Relaxation

Verbesserung des aktuell kürzesten Pfades von s nach v

RELAX(u, v, w)

```
1  if v.distance > u.distance + w(u, v)
2      v.distance = u.distance + w(u, v)
3      v.predecessor = u
```

Bellman-Ford-Algorithmus

Allgemeine Lösung, auch mit negativen Kantengewichten

BELLMAN-FORD(G, w, s)

```
1  INITIALIZE_SINGLE_SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3        for each (u, v) ∈ G.E
4              RELAX(u, v, w)
5  for each (u, v) ∈ G.E
6        if v.distance > u.distance + w(u, v)
7              return FALSE
8  return TRUE
```

Laufzeit: $O(|V| \cdot |E|)$

Dijsktra-Algorithmus

Voraussetzung: nichtnegative Kantengewichten

DIJSKTRA(G, w, s)

```
1  INITIALIZE_SINGLE_SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5        u = EXTRACT-MIN(Q)
6        S = S ∪ {u}
7        for each v ∈ G.Adj[u]
8              RELAX(u, v, w)
```

Laufzeit:

- Mit binärem Min-Heap: $O((|V| + |E|) \log |V|)$
- Mit Fibonacci-Heap: $O(|V| \log |V| + |E|)$

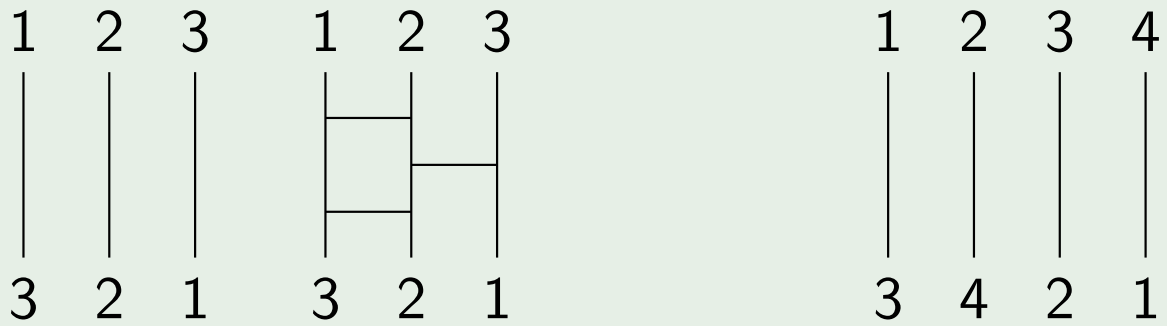
Aufgaben

Japanische Leiterspiele

Spielbeschreibung

Platziere Sprossen zwischen den Stäben, so dass die Nummern oben mit entsprechenden Nummern unten verbunden sind. Jede Nummer wandert dabei von oben nach unten und geht über jede Sprosse, die sie trifft und danach weiter nach unten. Keine zwei Sprossen dürfen sich berühren. Verwende so wenige Sprossen wie möglich.

Beispiel

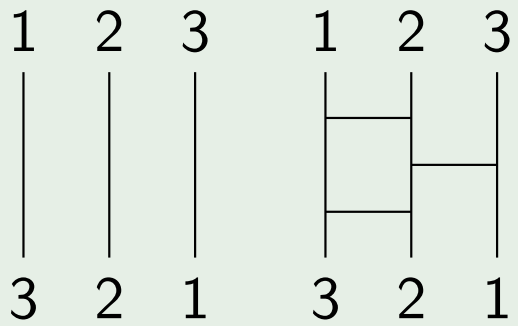


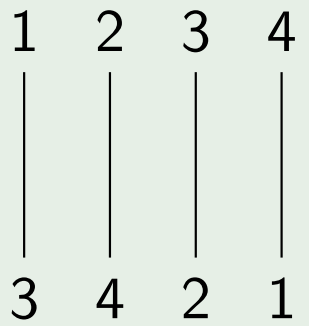
Japanische Leiterspiele

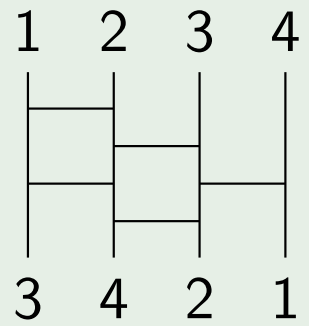
Spielbeschreibung

Platziere Sprossen zwischen den Stäben, so dass die Nummern oben mit entsprechenden Nummern unten verbunden sind. Jede Nummer wandert dabei von oben nach unten und geht über jede Sprosse, die sie trifft und danach weiter nach unten. Keine zwei Sprossen dürfen sich berühren. Verwende so wenige Sprossen wie möglich.

Beispiel





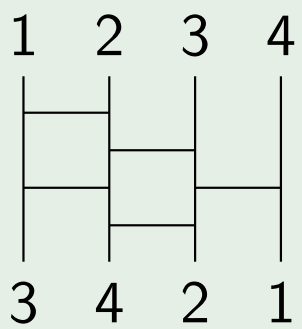


Japanische Leiterspiele

Spielbeschreibung

Platziere Sprossen zwischen den Stäben, so dass die Nummern oben mit entsprechenden Nummern unten verbunden sind. Jede Nummer wandert dabei von oben nach unten und geht über jede Sprosse, die sie trifft und danach weiter nach unten. Keine zwei Sprossen dürfen sich berühren. Verwende so wenige Sprossen wie möglich.

Beispiel



Gedanken

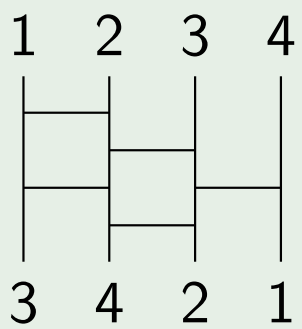
- Gibt es immer eine Lösung?

Japanische Leiterspiele

Spielbeschreibung

Platziere Sprossen zwischen den Stäben, so dass die Nummern oben mit entsprechenden Nummern unten verbunden sind. Jede Nummer wandert dabei von oben nach unten und geht über jede Sprosse, die sie trifft und danach weiter nach unten. Keine zwei Sprossen dürfen sich berühren. Verwende so wenige Sprossen wie möglich.

Beispiel



Gedanken

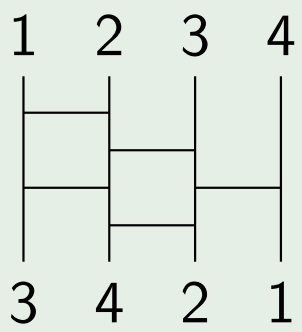
- Gibt es immer eine Lösung?
- Wann gibt es eine Lösung?

Japanische Leiterspiele

Spielbeschreibung

Platziere Sprossen zwischen den Stäben, so dass die Nummern oben mit entsprechenden Nummern unten verbunden sind. Jede Nummer wandert dabei von oben nach unten und geht über jede Sprosse, die sie trifft und danach weiter nach unten. Keine zwei Sprossen dürfen sich berühren. Verwende so wenige Sprossen wie möglich.

Beispiel



Gedanken

- Gibt es immer eine Lösung?
- Wann gibt es eine Lösung?
- Passendes mathematisches Modell?
- Algorithmus? (Korrektheit, Effizienz)

Übersicht

Klausur

Klausuranmeldung jetzt im Studienportal möglich!
Klausur am 19.07.