# Systematische Suche

Daniel Horbach, Richard Hettmann

27. April 2010

# Inhaltsverzeichnis

# Rucksackprobleme

- nichtleere Menge von $n$ Gegenständen
- Gewichte $g$
- zugehörige Werte $w$
- zulässiges Maximalgewicht bzw. Aufnahmekapazität $K$

# Was ist das Problem?

Fülle den Rucksack so, dass:

- die Wertsumme der Gegenstände maximal ist
- die Kapazitätsgrenze eingehalten wird

# Rucksackprobleme

## 0/1 Rucksackproblem

- vollständiges Einpacken eines Gegenstandes (1)
- Gegenstand wird nicht eingepackt (0)

## Bruchteil-Rucksackproblem

- Einpacken eines beliebigen Teiles eines jeden Gegenstandes
- $0\% \leq t \leq 100\%$

# 0/1 Rucksackproblem

## Mathematische Formel

$$\sum_{i=0}^{n} x_i w_i \rightarrow max, \ \ wobei \sum_{i=0}^{n} x_i g_i \leq K, mit \ x_i \in \{0, 1\}$$

# Rucksackprobleme

Viele Variationen:

- Kofferpacken vor der Urlaubsreise
- Der Dieb der soviel wie möglich entwenden möchte
- Der Bergsteiger

# Teilmengen-Summen-Problem

- Spezialfall des Rucksackproblems
- ergibt sich für $g = w$
- gegeben ist $M = \{a_i \mid 1 \leq i \leq n \text{ und } a_i, n \in N\}$
- gesucht wird eine Teilmenge von $M$, dessen Elementensumme möglichst nah an $K$ herankommt
- Beispiele Transportunternehmen, Hausneubau/Modernisierung

# Lösungen des Rucksackproblems

## Intuitiv

- Füllen des Rucksacks mit zufällig ausgewählten Gegenständen
- Anschliessend Austausch von Gegenständen um Wert zu erhöhen

# Lösungen des Rucksackproblems

## Intuitiv

- Füllen des Rucksacks mit zufällig ausgewählten Gegenständen
- Anschliessend Austausch von Gegenständen um Wert zu erhöhen

## Nachteil

- Nicht systematisch
- Keine Sicherheit ob das Wertmaximum erreicht wurde

# Lösungen des Rucksackproblems

## Intuitiv

- Füllen des Rucksacks mit zufällig ausgewählten Gegenständen
- Anschliessend Austausch von Gegenständen um Wert zu erhöhen

## Nachteil

- Nicht systematisch
- Keine Sicherheit ob das Wertmaximum erreicht wurde

## Besser

- Konsequent zielführende Suchstrategien
- Tiefensuche und Breitensuche

# Graphenaufbau

- besteht aus Knoten (Vertex) welche durch Kanten (Edge) verbunden sind
- unterschiedliche Varianten von Graphen durch Art und Vielfalt der Knoten und Kanten
- beliebte Form ist der Binärbaum
- jeder Knoten hat maximal 2 Kanten
- Knoten die nicht mehr expandiert werden können sind Blätter

# Graphendurchlauf

- Beginn beim Wurzelknoten
- jeder Knoten wird bei einem Durchlauf mehrmals Berührt
- erste Berührung Prewalk
- zweite Berührung Centralwalk
- dritte Berührung Postwalk

# Bewertung

- **Speicherplatzbedarf**: Wieviel Speicherplatz wird benötigt
- **Zeitaufwand**: Wieviel Zeit wird für einen Durchlauf benötigt
- **Vollständigkeit**: Werden alle Lösungen gefunden wenn es mehrere gibt
- **Optimalität**: Wird die beste Lösung gefunden wenn es mehrere gibt

# Algorithmus der Tiefensuche

- Bestimmung des Startknotens
- Expansion des Knotens und Speicherung aller Nachfolger in einem Stack
- Rekursiver Aufruf von Tiefensuche für alle Nachfolger im Stack bis:
  - Leerer Stack → Abbruch, ergebnislose Suche
  - Element gefunden → Abbruch, erfolgreiche Suche, 1 Ergebnis

# Eigenschaften

- Speicherplatzbedarf: linear
- Zeitaufwand: $\mathcal{O}(|V| + |E|)$
- Vollständigkeit: unvollständig
- Optimalität: nicht optimal

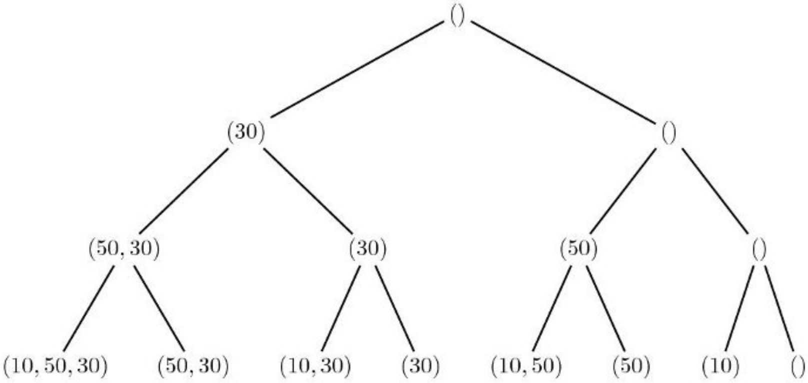
# Anwendung

- Topologisches Sortieren
- Expertensysteme
- Indirekt an vielen komplexeren Algorithmen für Graphen beteiligt

# Bezug auf 0/1 Rucksackproblem

- aktueller Rucksackinhalt ( )
- Gegenstandsliste (30, 50, 10)

# (30, 50, 10)()



# Dynamischer Baum

- Generierung des Baumes als eine Art Protokoll
- Einpacken aller Gegenstände
- Herausnahme des letzten Gegenstandes
- Zurückgehen zum letzten Entscheidungspunkt durch Backtracking
- Alternativen Weg gehen
- Verfahren kann auch auf vorgegebene statische Binärbäume angewandt werden

# Eigenschaften 0/1 Rucksackproblem

- Für $n$ Gegenstände ergeben sich im Allgemeinen $2^n$ Blätter

$$\sum_{k=0}^{n} \binom{n}{k} z^k = \binom{n}{0} z^0 + \binom{n}{1} z^1 + \ldots + \binom{n}{n} z^n = (1+z)^n, f\ddot{u}r\ z = 1$$

- Exponentieller Aufwand $\mathcal{O}(2^n)$
- Die Anzahl der Knoten ergibt sich aus:

$$\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$$

# Algorithmus der Breitensuche

- Bestimmung des Startknotens und Speicherung dieses Knotens in einer Warteschlange
- Entnahme des ersten Knotens der Warteschlange und dessen Markierung
  - Gefundenes Element → Abbruch nach Durchsuchen der Ebene
  - Anhängen aller bisher unmarkierten Nachfolger dieses Knotens ans Ende der Warteschlange
- Wiederholung von Schritt 2
- Abbruch bei leerer Warteschlange → kein Ergebnis

# Eigenschaften

- Speicherplatzbedarf: exponentiell
- Zeitaufwand: $\mathcal{O}(|V| + |E|)$
- Vollständigkeit: vollständig
- Optimalität: optimal

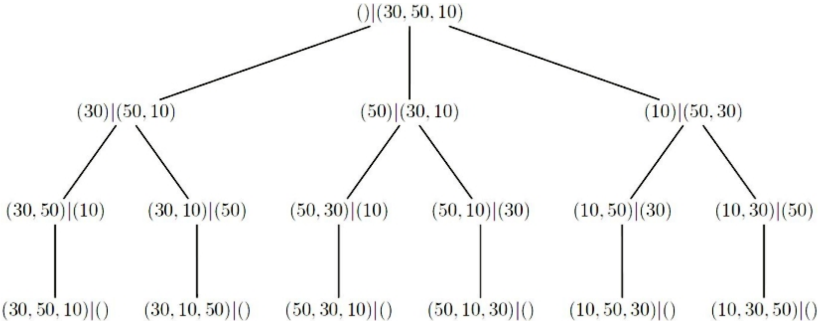
# Anwendung

- Finden aller Zusammenhangskomponenten in einem Graph
- Finden aller Knoten innerhalb einer Zusammenhangskomponente
- Finden des kürzesten Pfades zwischen zwei Knoten
- Kürzeste-Kreise-Problem

# Bezug auf 0/1 Rucksackproblem

- Notation: Rucksackliste | Gegenstandsliste
- Rucksackliste ( )
- Gegenstandsliste (30, 50, 10)

# ( ) | (30, 50, 10)


()|(30, 50, 10)
(30)|(50, 10)
(50)|(30, 10)
(10)|(50, 30)
(30, 50)|(10)
(30, 10)|(50)
(50, 30)|(10)
(50, 10)|(30)
(10, 50)|(30)
(10, 30)|(50)
(30, 50, 10)|()
(30, 10, 50)|()
(50, 30, 10)|()
(50, 10, 30)|()
(10, 50, 30)|()
(10, 30, 50)|()

# Eigenschaften 0/1 Rucksackproblem

Die Anzahl der Knoten ergibt sich aus:

$$1+n+n(n-1)+n(n-1)(n-2)+\ldots+n(n-1)(n-2)(n-3)\ldots(n-(n-1))$$

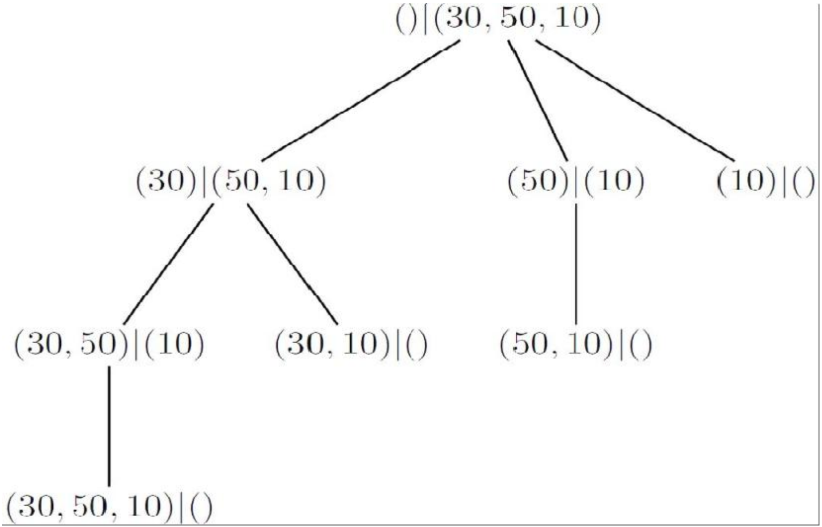
$$= \frac{n!}{0!} + \frac{n!}{1!} + \frac{n!}{2!} + \ldots + \frac{n!}{n!}$$

$$= \sum_{k=0}^{n} \frac{n!}{k!}$$

# Reduzierte Breitensuche

- Einige Knoten sind überflüssig, da die Reihenfolge des Einpackens unwichtig ist
- (30, 50, 10) und (30, 10, 50) beispielsweise beschreiben denselben Packzustand
- Reduzierung des Breitensuchbaumes um Elemente gleicher Bedeutung

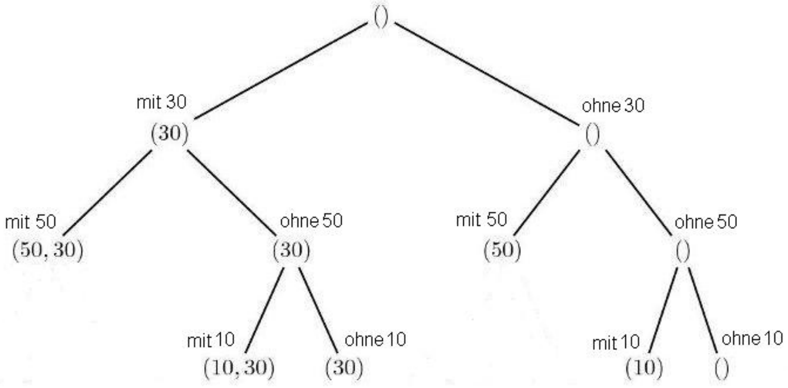
# ( ) | (30, 50, 10)



# Beschränkte Tiefensuche

- Hinzufügen der Kapazitätsbeschränkung
- Veränderter Suchbaum der Tiefensuche
- Gegenstandsliste (30, 50, 10) K = 42

# (30, 50, 10) () mit K = 42



# Eigenschaften

- Speicherplatzbedarf: linear
- Zeitaufwand $\mathcal{O}(|V| + |E|)$
- Vollständigkeit: unvollständig
- Optimalität: nicht optimal

# Algorithmus der Iterativen Tiefensuche

- Bestimmung des Startknotens
- Aufruf von Beschränkter Tiefensuche mit aktueller Suchtiefe
- Erhöhung der Suchtiefe um 1
- Wiederholung von Schritt 2

# Eigenschaften

- Speicherplatzbedarf: linear
- Zeitaufwand $\mathcal{O}(|V| + |E|)$
- Vollständigkeit: vollständig
- Optimalität: optimal

# Vergleich

| Suche | Zeit | Speicher | Vollständig | Optimal |
|---|---|---|---|---|
| Breitensuche | $\mathcal{O}(v^{d+1})$ | exponentiell | Ja | Ja |
| Tiefensuche | $\mathcal{O}(v^d)$ | linear | Nein | Nein |
| Beschr. Tiefensuche | $\mathcal{O}(v^d)$ | linear | Nein | Nein |
| Iterative Tiefensuche | $\mathcal{O}(v^d)$ | linear | Ja | Ja |

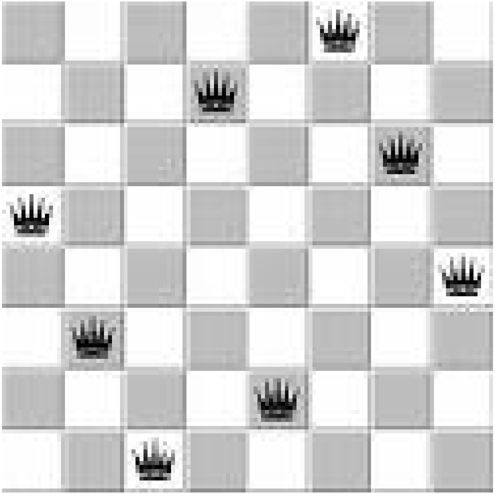
# 8-Damenproblem

- Max Friedrich Wilhelm Bezzel
- (1824 - 1871)
- Rechtsrat der Stadt Ansbach
- Gilt als der älteste bayrische Schachmeister
- Veröffentlichung 1848 in Berliner Schachzeitung

# 8-Damenproblem

- Schachmathematische Aufgabe
- Positionieren von 8 Damen auf einem Schachbrett von 8x8 Feldern
- Keine Dame darf eine andere bedrohen
- Frage nach der Anzahl der möglichen Lösungen

# 8-Damenproblem



# Verallgemeinerung auf n-Damenproblem

| n | eindeutige Möglichkeiten | alle Möglichkeiten |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 1 | 2 |
| 8 | 12 | 92 |
| 10 | 92 | 724 |
| 15 | 285.053 | 2.279.184 |
| 20 | 4.878.666.808 | 39.029.188.884 |
| 26 | 2.789.712.466.510.289 | 22.317.699.616.364.044 |

# Nichtdeterminismus

- Bäume werden nicht als Datenstruktur repräsentiert
- Entkopplung der Formulierung des Suchzieles von der Angabe des zugehörigen Suchprozesses (deskriptive Programmierung)
- Programm soll das richtige Resultat erahnen

# Vorteile

- Verbesserung der Kognitiven Effizienz durch Abstraktion, vereinfachte Erfassung und Implementierung des Algorithmus
- keine Verbesserung der Zeiteffizienz

# Nichtdeterminismus in Scheme

- ambiguous.ss laden
- Nichtdeterminismus wird deterministisch simuliert
- Tiefensuche im Hintergrund

# Neue Sprachelemente

- **choice** gibt das "*richtige*" Element einer Liste zurück
- **bag-of** gibt eine Liste aller "*richtigen*" Elemente einer Liste zurück
- **initialize-amb-fail** nullstellige Prozedur zur Initialisierung des Ambiguous-Systems
- **multichoice** nimmt eine Liste und eine natürliche Zahl $n$ und gibt die $n$ "*richtigen*" Elemente zurück
- **randomchoice** nimmt eine natürliche Zahl $n$ und berechnet choice

# Ende

Vielen Dank für Eure Aufmerksamkeit