03-05-H
-709.53

Informatik für Nichtinformatiker (7)

Prof. Dr. Udo Frese
Tobias Hammer

Algorithmen:

Euklidischer Algorithmus

Sieb des Eratosthenes

Selectionsort und Heapsort

Dijkstra

Was bisher geschah

Zusammenfassung

- **Berechnung statistischer Vorgänge durch die wiederholte Simulation einzelner Ereignisse durch Zufallszahlen.**
  - modelliere betrachteten Vorgang als Programm,…
  - …, bei dem der wirkliche Zufall durch Zufallszahlen simuliert wird.
  - simuliere Vorgang mit sehr vielen Versuchen (z.B. Millionen mal)
  - berechne aus den einzelnen Simulationsergebnissen, gesuchte Größe, z.B. Wahrscheinlichkeiten, Erwartungswert, Varianz, etc.
  - ACHTUNG: Ergebnis ist Näherungslösung!
- **Sehr mächtiges Werkzeug**
- **Kann Probleme lösen, die analytisch sehr schwer oder unlösbar sind**
- **Vorsicht: Manchmal in der jeweiligen Fachkultur unüblich**

Algorithmen

- **Muhammad ibn Musa, Abu Dscha'far al-Chwarizmi**
  - Erfinder der Null
  - Werk über das Rechen mit indischen Ziffern
  - durch Abwandlung entstand das Wort Algorithmus
- **Algorithmus: Verfahren zur Berechnung von etwas**
- **Das Prinzip hinter einem Programm, das etwas nichttriviales ausrechnet.**
- **Quelle der Algorithmen: R. Sedgewick, Algorithmen, Addison Wesley, (1992)**

Euklidischer Algorithmus

Euklidischer Algorithmus

- **Gesucht: Größter gemeinsamer Teiler (ggT) zweier Zahlen**
- **Algorithmus von Euklid (360-280 v. Chr.)**
- **Beobachtungen:**
  - addiert man ein Vielfaches eines Teilers auf eine Zahl, bleibt er Teiler
  - wenn i a teilt, dann teilt i auch a+j*i
  - ggT(a, b)=ggT(a, b-a)
  - ggT(a, b)=ggT(b, a%b), besonders für a>b (% = modulo)
  - ggT(a, 0)=a
- **Wie entwickelt man daraus einen Algorithmus?**
  - die vorletzte Aussage erlaubt, das Problem zu reduzieren.
  - von dem ggT zweier Zahlen zum ggT zweier kleinerer Zahlen
  - die letzte Aussage erlaubt, ein hinreichend kleines Problem zu lösen
  - typisches Paradigma der Informatik

Euklidischer Algorithmus

- **<u>Frage an das Auditorium</u>: Formuliert aus dieser Erkenntnis einen Algorithmus**
  - ggT(a, b)=ggT(b, a%b), besonders für a>b (% = modulo)
  - ggT(a, 0)=a

Euklidischer Algorithmus

- **<u>Frage an das Auditorium</u>: Formuliert aus dieser Erkenntnis einen Algorithmus**
  - ggT(a, b) = ggT(b, a%b), besonders für a>b (% = modulo)
  - ggT(a, 0) = a

- **gcd (a, b)**
  - if a<b swap a and b
  - while b>0
    - a, b = b, a%b
  - result a

Euklidischer Algorithmus

- **<u>Frage an das Auditorium</u>: Formuliert aus dieser Erkenntnis einen Algorithmus**
  - ggT(a, b) = ggT(b, a%b), besonders für a>b (% = modulo)
  - ggT(a, 0) = a

- **gcd (a, b)**
  - if a<b swap a and b
  - while b>0
    - a, b = b, a%b
  - result a

```
def gcd (a, b)
   if a<b then a,b = b,a end
  while b>0 do
    a, b = b, a%b
  end
  a
end
```

Euklidischer Algorithmus

- **lcd (a, b)**
  - if a<b swap a and b
  - while b>0
    - a, b = b, a%b
  - result a

```
def gcd (a, b)
  if a<b then a,b = b,a end
  while b>0 do
    a, b = b, a%b
  end
  a
end
```

Beispiel

a	b
24	102
102	24
24	6
6	0

Sieb das Eratosthenes

Sieb das Eratosthenes

- **<u>Frage an das Auditorium</u>: Wie berechne ich die Liste der Primzahlen unter 1000 bzw. n?**

Sieb das Eratosthenes

- **Frage an das Auditorium: Wie berechne ich die Liste der Primzahlen unter 1000 bzw. n?**
- **for i von 2..n:**
  - **for j von 2..i-1:**
    - **if j divides i: no prime**
  - **If loop finished: is prime**

```
def prime_up_to (n)
  primes = []
  (2..n).each do |i|
    is_prime = true
    (2...i).each do |j|
      if (i%j==0) then
        is_prime = false
        break
      end
    end
    if is_prime then
      primes += [i]
    end
  end
end
```

Sieb das Eratosthenes

- **Wie lange rechnet der triviale Algorithmus?**
- **Im Mittel werden n/2 Zahlen als Teiler probiert.**
- **Rechenzeit ≅ const · n·n/2**
- **Betrachten keine Konstanten, weil vom Rechner und der Programmiersprache abhängig**
- **Rechenzeit $\propto n^2$**
- **Geht das schneller?**
- **Ja: Sieb des Eratosthenes (276-195 v.Chr.)**

Sieb das Eratosthenes

Idee:

- **Wenn du eine Primzahl findest, streiche alle Vielfachen als nicht-prim weg.**

Sieb das Eratosthenes

Idee:

- **Wenn du eine Primzahl findest, streiche alle Vielfachen als nicht-prim weg.**

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

Sieb des Eratosthenes

Algorithmus

- set is_prime to n times true
- for i from 2..n:
  - if is_prime[i] then
    - i ist prime
    - j=i*i
    - while j<n:
      - is_prime[j] = false
      - j += i

Sieb des Eratosthenes

Wie schnell ist das Sieb des Eratosthenes?

- **Dominante Operation: Innerste Schleife**
- **Wie oft wird sie ausgeführt?**
- **Für jede Primzahl i:**
- **jedes i-te Element löschen**
- **Summe mathematisch schwierig.**

$$\sum_{i=2, i \text{ prim}}^{n} \frac{n}{i} \propto n \ln \ln n \overset{\text{praktisch}}{\approx\propto} n$$

Selectionsort und Heapsort

Selectionsort und Heapsort

- **Bringe ein Array a von Objekten durch Vergleiche und Vertauschen in eine geordnete Reihenfolge**
- **Frage an das Auditorium: Schlagt einen Algorithmus vor**

Selectionsort und Heapsort

- **Bringe ein Array a von Objekten durch Vergleiche und Vertauschen in eine geordnete Reihenfolge**
- **Frage an das Auditorium: Schlagt einen Algorithmus vor**
- **Einfache Grundstrategie (selection sort):**
  - Suche das Minimum
  - Vertausche es an den Anfang
  - Sortiere den Rest

Selectionsort und Heapsort

- **Bringe ein Array a von Objekten durch Vergleiche und Vertauschen in eine geordnete Reihenfolge**
- **<u>Frage an das Auditorium</u>: Schlagt einen Algorithmus vor**
- **Einfache Grundstrategie (selection sort):**
  - Suche das Minimum
  - Vertausche es an den Anfang
  - Sortiere den Rest

Beispiel

5	1	77	4	9	15
1	5	77	4	9	15
1	4	77	5	9	15
1	4	5	77	9	15
1	4	5	9	77	15
1	4	5	9	15	77

Selectionsort und Heapsort

Selection sort

- for i from 0..a.length-1
  - search index jMin with smallest entry of i..a.length-1
  - swap a[i] and a[jMin]

Selectionsort und Heapsort

Selection sort

- for i from 0..a.length-1
  - jMin = i
  - for j from i..a.length-1
    - if a[j]<a[jMin] then jMin=j
  - swap a[i] and a[jMin]

Selectionsort und Heapsort

Selection sort

```
def sort (a)
   a.length.times do |i|
      jMin = i
      (i...a.length).each do |j|
         if a[j]<a[jMin] then
            jMin = j
         end
      end
      a[i], a[jMin] = a[jMin], a[i]
   end
   a
end
```

Selectionsort und Heapsort

Selection sort

- **Frage an das Auditorium: Wie viele Vergleichsoperationen macht der Algorithmus?**

```
def sort (a)
   a.length.times do |i|
      jMin = i
      (i...a.length).each do |j|
         if a[j]<a[jMin] then
            jMin = j
         end
      end
      a[i], a[jMin] = a[jMin], a[i]
   end
   a
end
```

Selectionsort und Heapsort

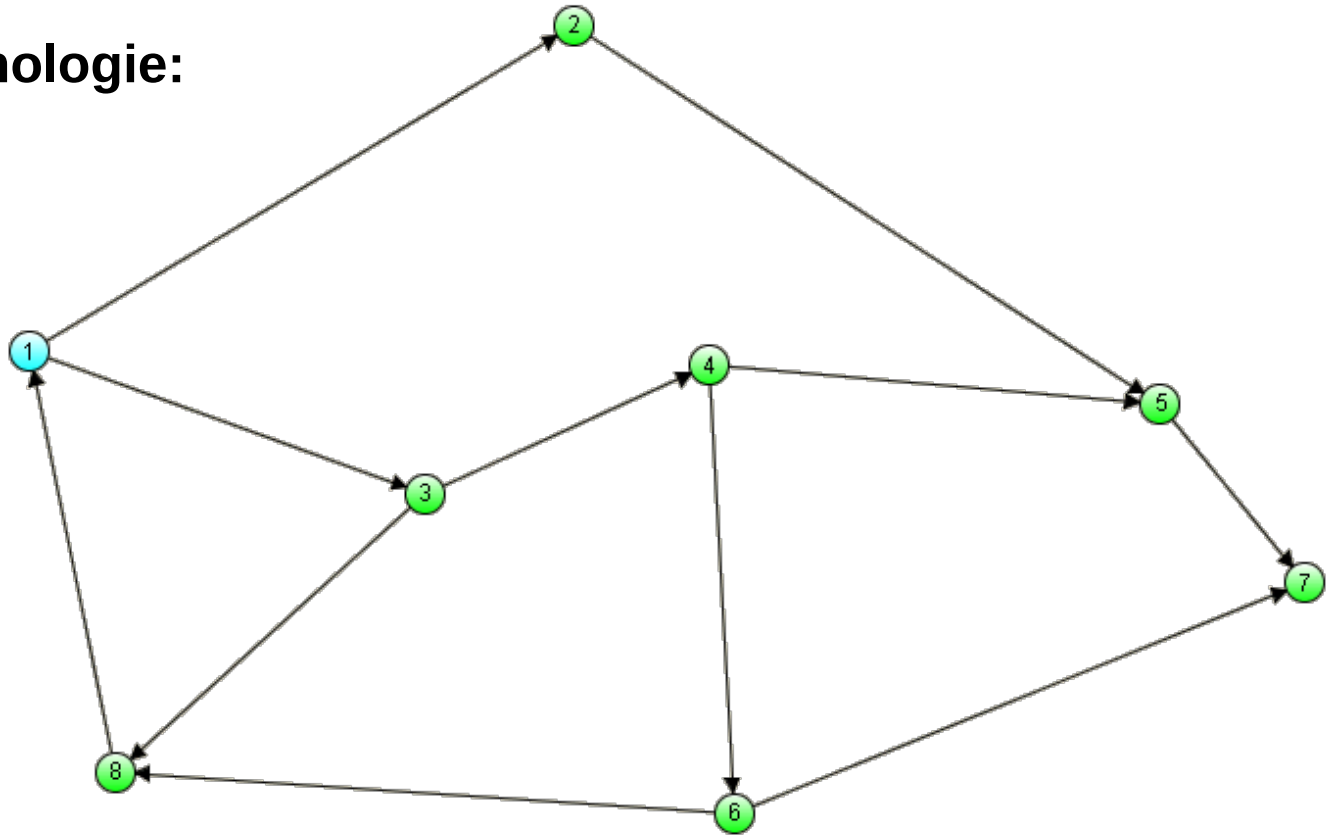
Selection sort

- **Frage an das Auditorium: Wie viele Vergleichsoperationen macht der Algorithmus?**
- **$\propto n^2$, weil n mal im Mittel n/2 Einträge durchsucht werden**
- **schneller mit Hilfe einer raffinierten Methoden zum Suchen des Minimums in ln n Schritten**
- **Dann: Heapsort n ln n**

Dijkstra Algorithmus

Dijkstra Algorithmus

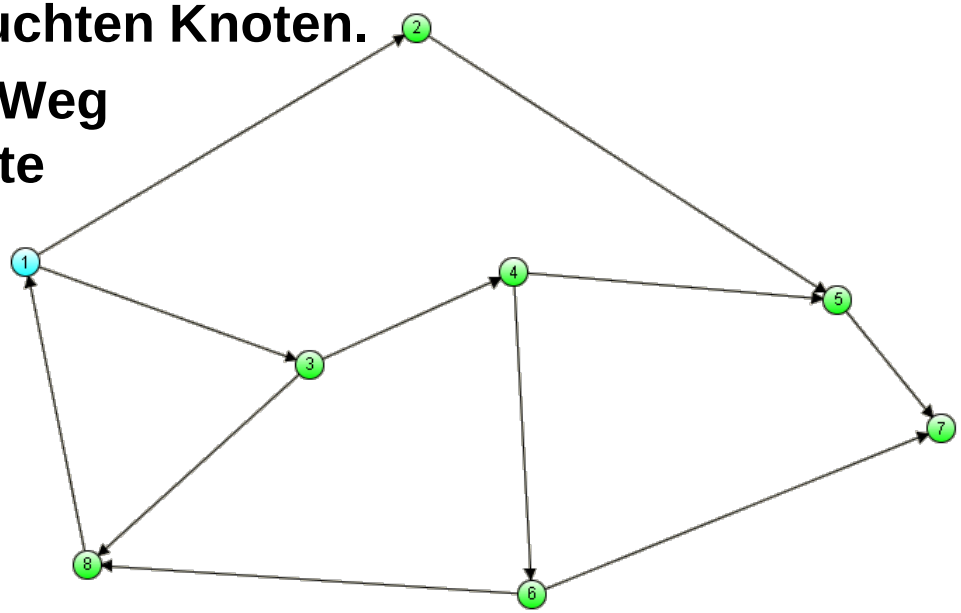
- **Gegeben eine Karte mit Orten und Straßen, außerdem ein Startort und ein Zielort. Gesucht, der kürzeste Weg vom Start zum Ziel.**
- **Abstrakte Terminologie:**
  - Karte ⇒ Graph
  - Ort ⇒ Knoten
  - Straße ⇒ Kante
  - Länge ⇒ Kosten
  - Einbahnstraße⇒ gerichtete Kante



Dijkstra Algorithmus

Beobachtungen:

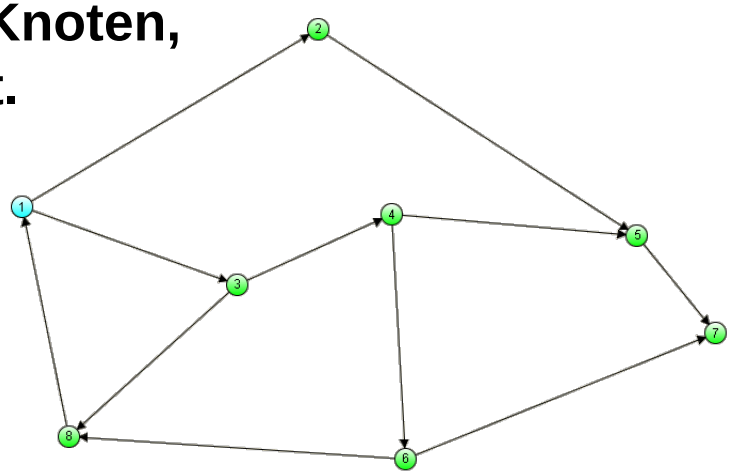
- **Es gibt im Allgemeinen unendlich viele Wege**
- **Es gibt exponentiell viele *einfach* Wege (ohne doppelte Knoten)**
- **$\Rightarrow$ alle durchprobieren ist viel zu langsam.**
- **Der kürzeste Weg zu einem Knoten besteht aus lauter kürzesten Wegen zu den auf dem Weg besuchten Knoten.**
- **Beispiel: Ist 1-3-4-5 der kürzeste Weg nach 5. Dann ist 1-3-4 der kürzeste Weg nach 4 und 1-3 der kürzeste Weg nach 3.**



Dijkstra Algorithmus

Idee des Dijkstra Algorithmus:

- **Konstruiere sukzessiv kürzeste Wege vom Startknoten zu allen Knoten,…**
- **…, durch ergänzen schon gefundener kürzester Wege um eine Kante.**
- **Generiere die Wege in der Reihenfolge aufsteigender Länge**
- **Der erste ist dann der kürzeste.**
- **Generiere nur Kandidatenwege zu einem Knoten, dessen kürzester Weg noch unbekannt ist.**



Demo
(http://www.uweschmidt.org/dijkstravis)

Dijkstra Algorithmus

Idee des Dijkstra Algorithmus (etwas formaler):

- **Halte Array von**
  - offenen Knoten: Knoten, zu denen ein Weg gefunden wurde, aber u.U. noch nicht der kürzere
  - abgeschlossene Knoten: Knoten, zu denen der kürzeste Weg gefunden wurde
- **jeweils Länge des Weges mit abspeichern**
- **solange es offene Knoten gibt**
  - **entferne den Knoten mit der kleinsten Entfernung**
  - **falls noch nicht in abgeschlossene Knoten:**
    - **füge zu abgeschlossenen Knoten hinzu**
    - **füge alle Nachbarknoten (mit addierter Entfernung) zu offenen Knoten hinzu**

Dijkstra Algorithmus

Klasse NodeWithStep (NWS)

- **Attribute:**
  - **node: Der Knoten zu dem dieser Weg führt**
  - **distance: Die Länge des Weges vom Startknoten zu node**

NodeWithPath (NWP)
node, distance

Dijkstra Algorithmus

Dijkstra Algorithmus

- open = [NWS.new(start, 0)],
- closed = [ ]
- while open≠[ ]
  - n = NWS in open with minimum .distance
  - if n.node not in closed
    - for all neighbours n‘ of n.node
      - open += [NWS.new(n‘, n.distance+distance (n, n‘))]
    - closed += [n]
  - open -= [n]

Dijkstra Algorithmus

- open = [NWS.new(start, 0)],
- closed = [ ]
- while open≠[ ]
  - n = NWS in open with minimum .distance
  - if n.node not in closed
    - for all neighbours n‘ of n.node
      - open += [NWS.new(n‘, n.distance+distance (n, n‘))]
    - closed += [n]
  - open -= [n]
- Frage an das Auditorium: Wie kriegt man jetzt den kürzesten Weg heraus?

Dijkstra Algorithmus

- **<u>Frage an das Auditorium</u>: Wie kriegt man jetzt den kürzesten Weg heraus?**
- **Mit den Knoten merken, von dem der Weg gekommen ist. Zum Schluß zurückverfolgen**
- **Attribut previous: Der Knoten von dem aus der Weg verlängert wurde zu einem Weg zu Knoten.**

NodeWithPath (NWP)
node, distance, previous

Dijkstra Algorithmus

Dijkstra Algorithmus mit Wegrückverfolgung

- open = [NWS.new(start, 0, nil)],
- closed = [ ]
- while open≠[ ]
  - n = NWS in open with minimum .distance
  - if n.node not in closed
    - for all neighbours n‘ of n.node
      - open += [NWS.new(n‘, n.distance+distance (n, n‘), n.node)]
    - closed += [n]
  - open -= [n]
- path = [goal]
- while path[0]≠start
  - path = [closed[path[0]].previous] + path

Dijkstra Algorithmus

Rechenzeit

- **Beobachtung: Jede Kante führt einmal dazu, dass ein Knoten in offene_knoten eingefügt wird.**
- **n Kanten**
- **Rechenzeit für das Minimum**
  - einfache Implementierung: $\propto n$
  - verbesserte Implementierung (Heap): $\propto \ln n$
- **Gesamtrechenzeit**
  - einfach n^2
  - Verbesserte Implementierung (Heap): $n \cdot \ln n$

Zusammenfassung

- **Algorithmus: Verfahren zur Berechnung von etwas**
- **Euklidischer Algorithmus: ggT zweier Zahlen**
  - Reduktion: ggT(a, b)=ggT(b, a%b)
  - Initialproblem: ggT(a,0)=a
- **Sieb des Eratosthenes: Liste von Primzahlen**
  - Wenn ein Primzahl gefunden wurde, streiche alle ihre Vielfachen weg
- **Selectionsort bzw. Heapsort: Sortiere eine Liste von Objekten**
  - Suche nach und nach das Minimum der verbliebenen Objekte
- **Dijkstra kürzeste Wege: Finde kürzesten Weg zwischen zwei Knoten in einem Graph**
  - Liste offener und abgeschlossener Knoten
  - schließe jeweils den offenen Knoten mit kürzestem Weg ab
  - wenn der noch nicht abgeschlossen war: füge Nachbarn zu offenen hinzu