

datastructuren en algoritmen
2010 10 10
college 11

tijdscomplexiteit

- $f(n) \in \mathcal{O}(g(n))$
 f is op den duur kleiner dan $g(n)$ (op constante factor na)
worst-case quick-sort in $\mathcal{O}(n^2)$
- $f(n) \in \Omega(g(n))$
 f is op den duur groter dan $g(n)$ (op constante factor na)
worst-case comparison-based sorteren in $\Omega(n \log(n))$
- $f(n) \in \Theta(g(n))$
 f is op den duur groter en kleiner dan $g(n)$ (op constanten na)
worst-case merge-sort in $\Theta(n \log(n))$

totnutoe

- **datastructuren: lineair en hierarchisch**
stacks, queues, vectoren, lijsten, rijtjes, priority queues
bomen, binaire bomen, binaire zoekbomen, AVL-bomen, heaps
grafen
- **sorteer-algoritmen**
heap-sort, selection sort, insertion sort, merge sort, quick sort,
quick select
- **hashing**
- **verdeel en heers:** merge, maxSubArray,
dynamic programming: geld wisselen, hout hakken, maxSubArray,
greedy algoritmen: geld wisselen, fractional knapsack, scheduling,

maar

- **verwachte tijdscomplexiteit van randomized quick-sort in $\mathcal{O}(n \log(n))$**
quick-sort is in de praktijk beter dan $\mathcal{O}(n^2)$
- **tijdscomplexiteit van 01knapsack in $\mathcal{O}(nW)$**
running time is geen functie van de grootte van de input
pseudo-polynomiale-tijd algoritme
- **amortized (uitgesmeerde) complexiteit**

P en NP

(boek hoofdstuk 13, geen tentamenstof)

- met een preciezer berekeningsmodel
(bijv grootte van de input: aantal bits nodig voor codering)
- complexiteitsklasse P
bevat alle beslissingsproblemen met worst-case polynomiale tijd
- complexiteitsklasse NP
bevat alle beslissingsproblemen met worst-case polynomiale tijd met nondeterministische keuze
- $P \subseteq NP$
maar onbekend: $P = NP?$

NP -compleet

van veel problemen in NP wordt geloofd dat er geen (echt) polynomiaal algoritme voor is

NP -compleet:

in NP EN

elk probleem in NP kan er in polynomiale tijd naar **gereduceerd** worden

NP : voorbeelden

intuïtief:

elke instantie van het probleem heeft eindig veel oplossingen die in polynomiale tijd te checken zijn

- is de Boolese formule satisfiable?
- heeft de graaf een Hamiltonian cycle?
- heeft de graaf een vertex-cover met ten hoogste k vertices?

NP -compleet: voorbeelden

- satisfiability
- Hamiltonian cycle
- traveling salesman
- knapsack01

schema

- recap en intro
- amortized complexity
- grafen
- alle kortste paden
- materiaal

tijdscomplexiteit

worst-case analyse van een rijtje van n operaties

- in $\mathcal{O}(n^2)$ want n operaties die elk in $\mathcal{O}(n)$ zijn
- correct, maar kan scherper!

voorbeeld: clearable table

- **clearable table met twee operaties:**
add(e): voeg e toe op eerste vrije plaats
clear(): verwijder alle elementen
- **implementatie:**
 n elementen in array ter grootte N
- **worst-case analyse**
add(e) $\in \mathcal{O}(1)$
clear() $\in \mathcal{O}(n)$ (zelfs in $\Theta(n)$)

clearable table

bekijk rijtje van n operaties $a_0 \dots a_{n-1}$
schrijf alle k clear() operaties expliciet:

$$\dots c_{i_0} \dots c_{i_1} \dots c_{i_{k-1}} \dots$$

voor c_j : table bevat ten hoogste $i_j - i_{j-1} - 1$ elementen
voor alle clear() operaties:

$$\begin{aligned} \sum_{j=0}^{k-1} i_j - i_{j-1} - 1 &= \\ i_0 - i_{-1} + i_1 - i_0 + i_2 - i_1 + \dots + i_{k-1} - i_{k-2} &= \\ i_{k-1} - i_{-1} \end{aligned}$$

de add-operaties zijn in $\mathcal{O}(1)$
alle n operaties in $\mathcal{O}(n)$
één operatie in $\mathcal{O}(1)$ gemiddeld

amortized running time

- bekijk worst-case tijdscomplexiteit van rijtje operaties
deel door aantal operaties voor gemiddelde tijdscomplexiteit per operatie
- dus:
gemiddelde tijdscomplexiteit van elke operatie in worst-case
- geen probability!

stack uitgebreid met multipop

- push(x): toevoegen in $\mathcal{O}(1)$
- pop(): verwijderen in $\mathcal{O}(1)$
- multipop(k): verwijder k elementen voor zover mogelijk

Algorithm multipop(k):
 while not isEmpty() **and** $k \geq 0$ **do**
 pop()
 $k := k - 1$

amortized complexity: technieken

- aggregate analysis
- accounting method
- potential method

tijdscomplexiteit multipop

- aantal iteraties in de while-loop:
 $\min(s, k)$ voor stack met s elementen
- in rijtje van n operaties:
 worst-case tijdscomplexiteit van een multipop: in $\mathcal{O}(n)$
 worst-case tijdscomplexiteit van hele rijtje: in $\mathcal{O}(n^2)$

stack met multipop: aggregate analysis

we bekijken een rijtje van n operaties

- elk element kan maar ten hoogste één keer van de stack gehaald dus totaal aantal pop() (inclusief die in multipop): n
- dus worst-case tijdscomplexiteit van rijtje ter lengte n in $\mathcal{O}(n)$
- de amortized cost per operatie is de average cost, dus in $\mathcal{O}(1)$

stack met multipop: amortized cost

wat zijn de costs?

- push: 1
- pop: 1
- multipop: $\min(k, s)$
(k weghalen van stack met s elementen)

we definiëren de charges oftewel amortized costs:

- push: 2
- pop: 0
- multipop: 0
(k weghalen van stack met s elementen)

intuïtie: met push betalen we voor alle latere pop

accounting method

per operatie:

- **charge**: wat rekenen we voor de operatie?
- **cost**: wat kost de operatie echt?
cost is kleiner-gelijk charge
- **amortized cost**: precies de charge

binary counter

voeg 1 toe aan k -bit binair getal
 $A.length = k$

Algorithm increment(A):

$i := 0$

while $i < A.length$ **and** $A[i] = 1$ **do**

$A[i] := 0$

$i := i + 1$

if $i < A.length$ **then**

$A[i] := 1$

binary counter: voorbeeld

waarde	A[7,6,5,4,3,2,1,0]	kosten
0	00000000	0
1	00000001	1
2	00000010	3
3	00000011	4
4	00000100	7

Dijkstra's kortste pad algoritme

- **input:**

$G = (V, E)$
 $w : E \rightarrow \mathbb{R}^+$
 $v \in V$

- **output:**

$d(u, v)$ voor alle $u \in V$

grafen

- $G = (V, E)$
- gewogen $w : E \rightarrow \mathbb{R}^+$
- afstand $d(u, v)$ lengte kortste pad van u naar v

kortste pad: correctheid

B: als u aan cloud wordt toegevoegd dan $D[u] = d(v, u)$

bewijs:

Stel niet. Bekijk de eerste u met $D[u] > d(v, u)$. Dan: er is een kortste pad $P : v \rightarrow^* y \rightarrow z \rightarrow^* u$ met $|P| = d(v, u)$, en z de eerste knoop niet in de cloud.

We gaan een tegenspraak afleiden. We gebruiken:

$D[z] \leq D[y] + w(y, z) = d(v, y) + w(y, z) = (!)d(v, z)$.

$$\begin{aligned} D[u] &\leq D[z] && \text{anders was } z \text{ gekozen} \\ &= d(v, z) && \text{zie boven} \\ &\leq d(v, u) \\ &< D[u] \end{aligned}$$

Tegenspraak. Dus aanname 'stel niet' onwaar. Dus B waar.

kortste pad: complexiteit

- n knopen en m kanten
- **init:**
 n keer in $\log(n)$ dus in $\mathcal{O}(n \log n)$
- **while:**
 n keer verwijder kleinste in $\log n$ dus in $\mathcal{O}(n \log n)$

geef buren in $\mathcal{O}(\deg(u))$ en update z in $\mathcal{O}(\log n)$
dus in $\sum \deg(u_i) \log n$

NB: $\sum \deg(u_i) = 2m$ (Thm 6.6)
dus while in $\mathcal{O}((n + m) \log n)$
samenhangend dus $m \geq n - 1$ dus $\mathcal{O}(m \log n)$

Bellman-Ford kortste pad algortime

- **aanpassing van Dijkstra's algoritme**
- **slechtere complexiteit maar ook voor negatieve gewichten**
- **idee:**
bekijk $n - 1$ keer alle pijlen (n is aantal elementen van V)
voer update uit als nodig
de i -de iteratie vindt kortste paden ter lengte i

negatieve gewichten

- **ongerichte graaf problematisch:**
 $d(u, v) = -\infty$ als gewicht van (u, v) is -1 ?
- **dus: gerichte graaf, $e \in E$ is een pijltje**

Dijkstra's algoritme werkt niet voor gerichte grafen met negatieve gewichten
voorbeeld

Bellman-Ford kortste pad algortime

- **input:**
 $G = (V, E)$ gericht $n = \#V$
 $w : E \rightarrow \mathbb{R}$
 $v \in V$
- **init:**
 $D[v] = 0$
 $D[u] = \infty$ voor alle $u \neq v$ in V
- **iter:**

```
for  $i := 1$  to  $n - 1$  do
  for  $e = (u, z) \in E$  do
    if  $D[u] + w(u, z) < D[z]$  then
       $D[z] := D[u] + w(u, z)$ 
```

correctheid

wanneer werkt het niet?
(denk aan voorbeeld ongericht)
bij cykel met negatief gewicht

schema

- recap en intro
- amortized complexity
- grafen
- alle kortste paden
- materiaal

tijdscomplexiteit

- **init:**
 n keer insert in $\mathcal{O}(1)$ dus totaal in $\mathcal{O}(n)$
- **iter:**
update in $\mathcal{O}(1)$ en dan n keer m keer dus totaal in $\mathcal{O}(nm)$
dus slechter dan Dijkstra's algoritme maar voor meer inputs

alle kortste paden

- geef afstand korste pad tussen elk tweetal knopen
- doe n keer Dijkstra's algoritme dus $\mathcal{O}(nm \log n)$
- doe n keer Bellman-Ford's algoritme dus $\mathcal{O}(n^2 m)$
- of doe dynamic programming
- idee: laat het aantal mogelijkheden voor een tussenstap steeds toenemen

alle kortste paden: algoritme

- **input:**
 $G = (V, E)$ gericht
 $w : E \rightarrow \mathbb{R}$ geen negatieve cykels
 $V = \{v_1, \dots, v_n\}$

- **init:**

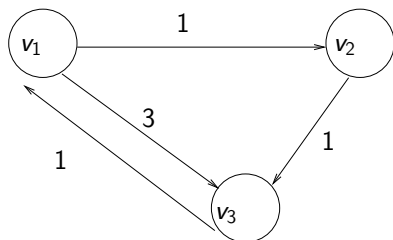
$$D_{ij}^0 = \begin{cases} 0 & \text{als } i = j \\ w(e) & \text{als } e = (v_i, v_j) \in E \\ \infty & \text{anders} \end{cases}$$

- **iter:**

```
for k := 1 to n do
  for i := 1 to n do
    for j := 1 to n do
       $D_{ij}^k := \min(D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1})$ 
```

D_{ij}^k : bovengrens op afstand v_i en v_j
met tussenstops in $\{v_1, \dots, v_k\}$

alle kortste paden: voorbeeld



complexiteit

$n \times n$ matrix moet je n keer updaten
dus totaal in $\mathcal{O}(n^3)$

schema

- recap en intro
- amortized complexity
- grafen
- alle kortste paden
- materiaal

materiaal

- boek 1.5
- boek 7.1, 7.2

extra materiaal

- wiki amortized complexity
- wiki P is NP probleem