



Stof

- slides college 11
- boek 1.5
- boek 7.1, 7.2, 7.3

Opgaven

1. Een rijtje van operaties $\text{multipush}(k, e)$ gevolgd door $\text{multipop}(k)$, totaal n operaties, heeft een tijdscomplexiteit van orde nk (waarbij k niet perse klein is ten opzichte van n). De tijdscomplexiteit van één operatie is dan van orde k , waarbij k niet begrensd wordt door een constante. De amortized complexiteit is dus niet in $\mathcal{O}(1)$.

2. We bekijken een array A ter lengte k , dus met indexen $0, \dots, k-1$. In het array staat het binaire getal x met $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$.

We bekijken een rijtje van **increment** operaties. Het bit $A[0]$ wisselt bij elke stap. Het bit $A[1]$ wisselt om de stap (elke tweede stap). Het bit $A[2]$ wisselt elke vierde stap. Dus in een rijtje van n **increment**-operaties wisselt het bit $A[0]$ totaal n keer, het bit $A[1]$ totaal $\lfloor \frac{n}{2} \rfloor$ keer, het bit $A[2]$ totaal $\lfloor \frac{n}{4} \rfloor$ keer. In het algemeen: Bij startwaarde 0, in een rijtje van n toepassingen van **increment**, geldt voor $i = 0, \dots, k-1$ dat bit $A[i]$ (in de array-weergave) $\lfloor \frac{n}{2^i} \rfloor$ keer wisselt. Het totale aantal wisselingen van bits is dus

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \cdot \sum_{i=1}^{\infty} \frac{1}{2^i} = 2n$$

(We gebruiken hier of ad hoc inzicht, of $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ voor $|x| < 1$.) Dat betekent dat de worst-case tijdscomplexiteit voor een rijtje van n **increment** operaties in $\mathcal{O}(n)$ is. Dus de amortized tijdscomplexiteit van één operatie is in $\mathcal{O}(1)$.

3. Geef de amortized complexiteit van de **increment** operatie van de binary counter, gebruikmakend van de accounting methode. (Dus: geef costs en geef charges.)

De **cost** van een **increment**-operatie is het aantal bits dat wisselt. We nemen als **charge**: 2 om een bit op 1 te zetten, en 0 om een bit op 0 te zetten. Van de 2 om het bit op 1 te zetten wordt 1 gebruikt om dit daadwerkelijk te doen, en het restant van 1 wordt bewaard om later te betalen voor het weer op 0 zetten.

waarde	cost	charge	over totaal
1	1	2	1
2	2	2	1
3	1	2	2
4	3	2	1
5	1	2	2

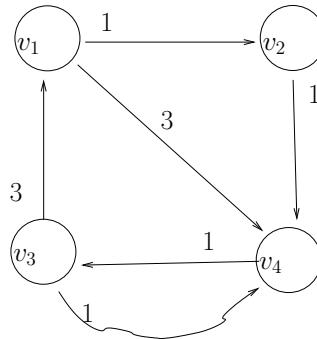
Na elke stap is het deel over totaal positief; dit omdat het aantal len in de counter niet negatief wordt. Dus de charges vormen een bovengrens voor de costs. De amortized tijdscomplexiteit voor één **increment** operatie is 2, dus in $\mathcal{O}(1)$, en de amortized tijdscomplexiteit van een rijtje van n operaties is in $\mathcal{O}(n)$.

4. We bekijken de implementatie van een queue gebruikmakend van twee stacks. Eerst een naïeve analyse van de tijdscomplexiteit: de worst-case tijdscomplexiteit van **enqueue** is in $\mathcal{O}(1)$ omdat **enqueue** is geïmplementeerd met **push**. De worst-case tijdscomplexiteit van **dequeue** is in $\mathcal{O}(n)$, met n het aantal elementen van de stack, omdat we in het slechtste geval alle n elementen van de ene stack moeten halen, op de andere stack moeten zetten, en daar dan weer het bovenste element afhalen ($2n + 1$ operaties).

We kunnen met een amortized analyse een scherpere afschatting geven. Daartoe bekijken we een rijtje van n operaties (een operatie is **enqueue** of **dequeue**). Een element dat aan de stack wordt toegevoegd wordt in het slechtste geval twee keer gepusht en één keer gepopt (en staat dan klaar om desgewenst verwijderd te worden). We rekenen dus 3 voor elke **enqueue** operatie, en 1 voor elke **dequeue** operatie. Dat levert een amortized cost van $4n$, dus in $\mathcal{O}(n)$ voor een rijtje van n operaties, en een amortized cost in $\mathcal{O}(1)$ voor één operatie.

De aanpak met de accounting methode lijkt hier erg op. Voor **enqueue** geldt $\text{cost} = 1$, en we nemen $\text{charge} = 3$. Voor **dequeue** geldt dat de cost variabel is, en we nemen $\text{charge} = 1$. Hebben we op deze manier steeds een positief bedrag over? Ja, want 1 van de charge van totaal 3 voor **enqueue** wordt gebruikt voor het uitvoeren van **push**. De overige 2 komen ten goede aan de spaarpot, waarmee voor elk toegevoegd element voldoende tegoed klaarstaat voor het overhevelen naar de andere stack. Met de charge van 1 voor **dequeue** kunnen de de laatste **pop** stap betalen. Dan: de charges vormen een bovengrens voor costs. De amortized tijdscomplexiteit van zowel **enqueue** als **dequeue** is in $\mathcal{O}(1)$, en de amortized tijdscomplexiteit van het hele rijtje is in $\mathcal{O}(n)$.

5. Pas het dynamic programming algoritme om alle kortste paden te vinden toe op de volgende graaf:



De initialisatie (nergens een tussenstop) geeft:

	v_1	v_2	v_3	v_4
v_1	0	1	∞	3
v_2	∞	0	∞	1
v_3	3	∞	0	1
v_4	∞	∞	1	0

De eerste iteratie (eventueel tussenstop in punten uit $\{v_1\}$) geeft:

	v_1	v_2	v_3	v_4
v_1	0	1	∞	3
v_2	∞	0	∞	1
v_3	3	4	0	1
v_4	∞	∞	1	0

De tweede iteratie (eventueel tussenstop in punten uit $\{v_1, v_2\}$) geeft:

	v_1	v_2	v_3	v_4
v_1	0	1	∞	2
v_2	∞	0	∞	1
v_3	3	4	0	1
v_4	∞	∞	1	0

De derde iteratie (eventueel tussenstop in punten uit $\{v_1, v_2, v_3\}$) geeft:

	v_1	v_2	v_3	v_4
v_1	0	1	∞	2
v_2	∞	0	∞	1
v_3	3	4	0	1
v_4	4	5	1	0

De vierde iteratie (eventueel tussenstop in punten uit $\{v_1, v_2, v_3, v_4\}$) geeft:

	v_1	v_2	v_3	v_4
v_1	0	1	3	2
v_2	5	0	2	1
v_3	3	4	0	1
v_4	4	5	1	0

6. Stel we voeren een rijtje van n operaties uit met de volgende kosten: operatie i kost i als i een 2-macht, en operatie i kost 1 anders. Bepaal de amortized cost per operatie, zowel met de aggregate methode als met de accounting methode.

Eerst met de aggregate methode. We hebben:

operatie	kosten
1	1
2	2
3	1
4	4
5	1
6	1
7	1
8	8
9	1
10	1
\vdots	

Dan hebben we voor de som van de kosten c_i per operatie:

$$\begin{aligned} \sum_{i=1}^n \text{cost}_i &\leq n + \sum_{j=0}^{\log(n)} 2^j \\ &= n + (2n - 1) \\ &< 3n \end{aligned}$$

De gemiddelde kosten voor één operatie zijn dus < 3 . Dan geldt: de amortized cost voor één operatie is in $\mathcal{O}(1)$.

Nu met de accounting methode. We namen als *charge* voor elke operatie 3. We hebben:

operatie	cost	charge	over totaal
1	1	3	2
2	2	3	3
3	1	3	5
4	4	3	4
5	1	3	6
6	1	3	8
7	1	3	10
8	8	3	5
\vdots	\vdots	\vdots	\vdots

Er geldt $\sum_{i=1}^n \text{charge}_i = 3n$. Uit de vorige analyse weten we: $\sum_{i=1}^n \text{cost}_i < 3n$. Dus wat er totaal over is, wordt niet negatief. Dus de amortized cost is de charge, dus in $\mathcal{O}(1)$.

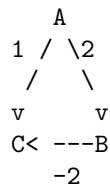
7. Gegeven een simpele samenhangende graaf met n knopen en m kanten. Omdat de graaf samenhangend is geldt $n - 1 \leq m$. Omdat de graaf simpel is geldt $m \leq \frac{1}{2}(n^2 - 1) \leq n^2$. Dus $n - 1 \leq m \leq n^2$ dus $\log(n - 1) \leq \log(m) \leq 2 \log(n)$. Daaruit volgt $\mathcal{O}(\log(m))$ is $\mathcal{O}(\log(n))$.

Met iets meer detail: Stel $f \in \mathcal{O}(\log(m))$. Dan vanaf zekere x_0 : $f(x) \leq c \log(m)$ voor zekere c . Omdat $\log(m) \leq 2 \log(n)$ geldt dan ook $f(x) \leq 2c \log(n)$, dus $f \in \mathcal{O}(\log(n))$. Stel $f \in \mathcal{O}(\log(n))$. Dan vanaf zekere x_1 : $f(x) \leq d \log(n)$ voor zekere d . (Hier x_0 en d gebruiken ipv x_0 en c hoeft eigenlijk niet.) Omdat $\log(n - 1) \leq \log(m)$ is er een constante d' met $f(x) \leq d' \log(m)$, dus $f \in \mathcal{O}(\log(m))$.

8. (nog toevoegen !)

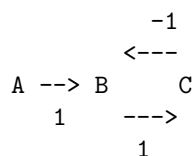
Houd een current knoop bij die initieel de start-knoop v is. Voeg bij een update een tak toe aan de boom.

9. Een gewogen gerichte graaf met negatieve gewichten waarvoor het algoritme van Dijkstra niet correct werkt



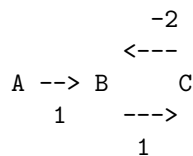
Het kortste pad van A naar C heeft totaalgewicht 0, maar met het algoritme van Dijkstra vind je het pad met totaalgewicht 1.

10. Bij het Bellman-Ford algoritme zoals in het boek wordt een cykel met gewicht 0 niet gevonden. Bekijk bijvoorbeeld de graaf



Bij start in A vinden we als waarden voor D in A, B, C in opeenvolgende iteraties $0, \infty, \infty$, dan $0, 1, \infty$, dan $0, 1, 2$. Het algoritme checkt vervolgens of er bij een volgende iteratie nog een lagere waarde wordt gevonden, hetgeen niet het geval is.

Bij bijna dezelfde graaf maar nu met een negatief-gewicht-cykel:



geldt dat we na dezelfde iteraties $0, \infty, \infty$, dan $0, 1, \infty$, dan $0, 1, 2$ bij de check aan het eind wel een lagere waarde vinden, namelijk 0 voor de afstand van A tot B . Dan wordt de waarschuwing dat de graaf een negatief-gewicht cykel bevat teruggegeven.

11. Laat zien dat het aantal cykelvrije paden van knoop u naar knoop v exponentieel kan zijn in het aantal knopen van de graaf.

Neem als knopen de natuurlijke getallen $\{0, \dots, n + 1\}$, en als pijlen de $<$. We kunnen dan op 2^n manieren van 0 naar $n + 1$, omdat we voor elke knoop $1, \dots, n$ kunnen kiezen of we er wel of niet doorheen gaan. (Voorbeelden: $0 < n + 1$, $0 < 1 < 2 < \dots < n + 1$.) De graaf is cykelvrij want de natuurlijke getallen met $<$ bevat geen cykel.

12. (misschien toevoegen)