Werkcollegebundel Algoritmiek

Voorjaar 2015, Gerard Tel, Utrecht.

Deze bundel bevat een collectie van vragen bij het vak Algoritmiek, die de laatste jaren zijn gebruikt op toetsen, in huiswerk of op werkcolleges. De inhoud van Algoritmiek varieert van jaar tot jaar; maak je dus niet direct zorgen als een opgave je onbekend voorkomt.
Let op: Op deze bundel geldt auteursrecht. Deze bundel is alleen voor deelnemers aan Algoritmiek Voorjaar 2015. Je mag deze bundel niet aan andere personen geven. Op een toets zijn meestal vragen met een totaal van 17 tot 19 punten. Cijfer is totaal, plus een voet van 1 tot 3, gedeeld door twee.

1 Divide and Conquer

1. **Gemiddelde met Divide-and-Conquer:** Gegeven is een array A van n doubles. Je moet het gemiddelde berekenen met een Divide-and-Conquer algoritme; dit algoritme berekent het gemiddelde van alle getallen uit de gemiddelden van de linker- en rechterdeelrij.
   (a) Geef zo'n algoritme.
   (b) Bewijs dat de complexiteit lineair is.
   (c) Gegeven is een tweede array W waarbij $W[i]$ het *gewicht* van waarde $A[i]$ bevat. Geef een Divide-and-Conquer algoritme dat het *gewogen gemiddelde* berekent in lineaire tijd.

2. **De ontbreker:** Gegeven is een ongesorteerde lijst A van n *verschillende* getallen in het bereik $0 \ldots n$. Er is dus één getal x, $0 \leq x \leq n$, dat niet in A voorkomt en je moet dat getal vinden, door naar slechts $O(n)$ *bits* te kijken. Je kunt van het i^{de} getal de j^{de} bit vragen als $A[i][j]$. Geef een algoritme dat het ontbrekende getal vindt en bewijs dat het maar $O(n)$ van de invoerbits bekijkt.

3. **Divide and Conquer: Majority:** In een lijst A van n positieve getallen is waarde w een *majority* (meerderheid) als hij vaker dan $n/2$ keer voorkomt. Een lijst heeft hoogstens één majority.
   Om een majority te vinden, verdeel je A in paren. Als n oneven is blijft 1 element a over, check of a een majority is, zoja return a. Zonee, als de waarden in een paar gelijk zijn, stop je 1 ervan in hulplijst B, bij ongelijkheid geen. Hulplijst B heeft lengte hoogstens $n/2$ en *als* A een majority w heeft, *dan* heeft B ook majority w. Ga in recursie op B en, als B majority v heeft, controleer of v een majority in A is.
   (a) Wat zijn de basisgevallen van de recursie?
   (b) Analyseer de looptijd van het algoritme.

4. **Divide and Conquer: Majority:** In een lijst A van n positieve getallen is waarde w een *majority* (meerderheid) als hij vaker dan $n/2$ keer voorkomt. Een lijst heeft hoogstens één majority.
   Om een majority te vinden, verdeel je A in paren. Als n oneven is blijft 1 restelement over, check of dit een majority is. Als de waarden in een paar gelijk zijn, stop je 1 ervan in hulplijst B, bij ongelijkheid geen. Ga in recursie op B en, als B majority v heeft, controleer of v een majority in A is.

(a) Bewijs dat: *Als* A een majority heeft, *dan* heeft B die majority ook.
(b) Analyseer de looptijd van dit algoritme op invoerlijsten waarin elk element uniek is.

5. **Divide and Conquer: Majority:** In een lijst A van n positieve getallen is waarde w een *majority* (meerderheid) als hij vaker dan $n/2$ keer voorkomt. Een lijst heeft hoogstens één majority.
   Om een majority te vinden, verdeel je A in paren. Als n oneven is blijft 1 restelement over, check of dit een majority is. Als de waarden in een paar gelijk zijn, stop je 1 ervan in hulplijst B, bij ongelijkheid geen. *Als* A een majority heeft, *dan* heeft B die majority ook. Ga in recursie op B en, als B majority v heeft, controleer of v een majority in A is.
   (a) Wat is de (maximale) omvang van B?
   (b) Analyseer de looptijd van het algoritme, ingeval de oorspronkelijke invoer A als lengte een tweemacht $n > 1$ heeft.

6. **Subsitututie-methode:** Bewijs met de Substitutiemethode dat de oplossing van $T(n) = T(n/2) + d.n$ voldoet aan $T(n) = O(n)$.

7. **Substitutie-methode:** Bewijs met de Substitutie-methode dat de oplossing van $T(n) = 2.T(n/2) + O(n)$ voldoet aan $T(n) = O(n. \lg n)$.

8. **Substitutie-methode:** Bewijs met de substitutie-methode dat de oplossing van $T(n) = T(n/2) + T(n/4) + O(n)$ voldoet aan $T(n) = O(n)$.

9. **Substitutie:** Gegeven is de recurrente betrekking $T(n) = T(n-2) + O(n)$. Bewijs met de substitutiemethode dat $T(n) = O(n^2)$.

10. **Substitutie-methode:** Bewijs met de Substitutie-methode dat de oplossing van $T(n) = T(n-1) + O(n)$ voldoet aan $T(n) = O(n^2)$.

11. **Substitutie-methode:** Bewijs met de substitutie-methode dat de oplossing van $T(n) = T(n/2) + T(n/3) + O(n)$ voldoet aan $T(n) = O(n)$.

12. **Master Theorem:** Geef de (asymptotische) oplossing voor de recurrentie $T(n) = T(n/4) + O(1)$.

13. **Master Theorem:** Wat is de oplossing van de recurrentie $T(n) = 2.T(n/4) + n$?

14. **Master Theorem:** Geef de asymptotische oplossing voor de recurrentie $T(n) = 4.T(n/2) + n$.

15. **Master Theorem:** Los deze reccurrente betrekking op met de master theorem: $T(n) = 2.T(n/4) + \sqrt{n}$. Wat zijn de a en de b?

16. **Master Theorem:** Los op met de Master Theorem: $T(n) = 5.T(n/3) + O(n)$.

17. **Master Theorem:** Bereken met de Master Theorem de (asymptotische) oplossing voor de recurrentie $T(n) = T(n/2) + O(n \lg n)$.

18. **Substitutie-methode:** Bewijs met de substitutie-methode dat de oplossing van $T(n) = T(n/2) + T(n/4) + d.n$ voldoet aan $T(n) = O(n)$.
En bewijs dat de oplossing van $T(n) = T(\alpha.n) + T(\beta.n) + T(\gamma.n) + d.n$ *altijd* lineair is als $\alpha + \beta + \gamma < 1$.

19. **Incompleet:** Een spel wordt gespeeld met een set van n kaarten, genummerd 0 t/m $n-1$. Oeps, je merkt dat het spel incompleet is: je (geschudde) pak bevat maar $n-1$ kaarten! Beschrijf, hoe je snel kunt ontdekken welke ontbreekt. Je hebt geen plek om alle kaarten uit te leggen, hoogstens drie stapeltjes kun je op je buro maken.

20. **Integer Vermenigvuldigen:** Je kunt n-bits integers A en B vermenigvuldigen door (1) ze te verdelen in $n/2$-bits getallen A_1A_0 en B_1B_0, (2) de deelresultaten $R_0 = A_0 \cdot B_0$ en $R_1 = A_1 \cdot B_0 + A_0 \cdot B_1$ en $R_2 = A_1 \cdot B_1$ te berekenen, en (3) de deelresultaten met goede verschuiving weer op te tellen.
(a) Laat zien hoe je de drie benodigde deelresultaten kunt berekenen met *drie* $n/2$-bits vermenigvuldigingen.
(b) Analyseer de asymptotische rekentijd $V(n)$.
(c) Je overweegt of het nog beter kan, door de getallen in *drie* stukjes $A = A_2A_1A_0$ te verdelen, en deelresultaten te vinden door een aantal $n/3$-bits vermenigvuldigingen te doen. Hoeveel vermenigvuldigingen kun je je veroorloven om toch een beter algoritme te krijgen dan met verdelen in twee?

2 Asserties en Binary Search

21. **Het Gat:** Gegeven is een gesorteerde array A van integers, waarbij $A[n-1] - A[0] \geq n$. Geef de invariant voor een algoritme dat een getal tussen $A[0]$ en $A[n-1]$ vindt, dat *niet* voorkomt in A. Geef ook het algoritme (het moet logaritmische tijd gebruiken).

22. **Het Duplicaat:** Gegeven is een gesorteerde array A van integers, waarbij $A[n-1]-A[0] < n-1$. Geef een algoritme dat een getal tussen $A[0]$ en $A[n-1]$ vindt, dat *meervoudig* voorkomt in A. Geef ook de invariant van je algoritme.

23. **Split:** Voor QuickSort en QuickSelect heb je deze methode `Split` nodig. `A.Split(p,q,piv)` levert integer r. Bij aanroep is $p \leq piv < q$, de waarde $A[piv]$ is de Pivot. De elementen worden herordend zo dat na afloop de elementen van p *tot* r kleinergelijk Pivot zijn, element r gelijk aan Pivot, elementen *vanaf* r *tot* q zijn grotergelijk Pivot. Bedenk een invariant voor `Split` en geef code. Wat is de variant? (Hint: In het begin heb je een groot onbekend stuk, na afloop een "kleiner" deel en een "groter" deel. Je invariant generaliseert dit en beschrijft waar Pivot staat, en welke drie delen "klein", "groot" en "onbekend" zijn.)

24. **Interpolation Search:** Volgens Wikipedia heeft Interpolation Search een verwachte complexiteit van $O(\lg \lg n)$ stappen. Kun je dit bewijzen of experimenteel bevestigen?

25. **Aliens:** Van een straat met n huizen genummerd van 0 t/m $n-1$ is bekend dat er op nr 0 een Slowaak woont en op nr $n-1$ een Tsjech. Geef een algoritme dat, na het vragen van hoogstens $\lceil \lg(n-1) \rceil$ nationaliteiten, twee buren (personen met huisnummerverschil

1) vindt van verschillende nationaliteit. Geef een invariant van je algoritme en leg met een variant uit, waarom je de complexiteit haalt.

26. **Sommatie:** Je kunt n getallen in A zo optellen:
```
s=0; i=0; while(i<n) { s += A[i]; i += 1; }
```
Geef invariant en variant van deze lus.

27. **Machtsverheffen:** Om, gegeven x en (int) a, de waarde $m = x^a$ te berekenen gebruiken we een variabele R en de invariant **inv:** $M = R \cdot x^a$.
(a) Geef een bijpassende initialisatie *en* een loop-conditie die na afloop de conclusie $M = R$ rechtvaardigt.
(b) Geef een Body die **inv** respecteert en een variant reduceert. Het kan in $\lg a$ slagen!

28. **Kriebels op de Krim:** In Simferopol is een straat met n huizen (aan één kant, 0 t/m $n-1$) waar op nr 0 een Rus woont en op nr $n-1$ een Oekraiener; ertussenin kunnen allerlei nationaliteiten wonen. Vind na het vragen van hoogstens $\lceil \lg n \rceil$ nationaliteiten twee buren (personen met opeenvolgende huisnummers) van verschillende nationaliteit. Geef een variant en invariant van je algoritme.

29. **Invariant: Pizza:** Een berg van k pizza's van omvang p_0 t/m p_{k-1} wordt verdeeld onder n programmeurs. Elk krijgt (i) een stuk uit één pizza (ii) van integer grootte, (iii) ieders stuk is even groot en (iv) ze hebben honger dus ze willen zo veel mogelijk eten.
Voorbeeld: Als $n = 3$ en er zijn twee pizzas van omvang 7 en 8, is de uitkomst 4.
Geef een algoritme dat de maximaal mogelijke portie-omvang berekent met binair zoeken. Geef een variant en invariant van je algoritme. Is de rekentijd polynomiaal?

3 Selectie

30. **Extremen:** Invoer is een *ongesorteerde* rij S van n getallen. Geef een algoritme dat met minder dan $\frac{3}{2}n$ vergelijkingen het grootste *en* het kleinste getal bepaalt. Vertel waarom je het aantal vergelijkingen haalt.

31. **Lineaire selectie:** Beschrijf (kort, hoogstens 12 regels) hoe je in worst case lineaire tijd in een array de waarde met rang k kunt vinden.

32. **Minimum vinden:** Hoeveel vergelijkingen zijn nodig om in een (ongesorteerde) rij getallen de kleinste te vinden? Waarom dit aantal?

33. **Selectie:** Wat is het probleem van *Selectie*? Geef een specificatie.
Legt je specificatie de gevraagde uitkomst uniek vast?

34. **Armoede:** Gegeven is een rij records, elk over 1 inwoner van een groot land; record i vermeldt s_i, het inkomen van persoon i; de records zijn niet gesorteerd op s. De sociale dienst wil weten wie er behoort tot de armste helft, het armste kwart, het armste achtste, etc. Elk record moet worden voorzien van een inkomensklasse k_i lopend van 0 tot $\lg n$, namelijk de grootste k waarvoor de rang van s_i kleiner is dan $n/2^k$.
Voorbeeld: de inkomens zijn 41, 42, 13, 56, 25, 99, 21, 5, 73, 20, 50, 80, 44, 22, 49, 57. Het

inkomen 5 heeft rang 0 dus krijgt klasse 4 en inkomen 13 krijgt klasse 3 (rang kleiner dan 16/8). De andere inkomens met rang kleiner dan 16/4 zijn 21 en 20, die krijgen $k = 2$. De vier inkomens 41, 42, 25, 22 hebben een rang kleiner dan $8 = n/2^1$ dus krijgen $k = 1$, de overige acht inkomens vallen in klasse 0.
Beschrijf een lineair algoritme dat deze klasse voor elk record bepaalt. Waarom is het lineair?

35. **Naar de Maan:** Intergalactic Fleet Command (IFC) beschikt over een vloot ruimteschepen, waarbij schip i een laadvermogen van L_i heeft. De Maanbasis moet worden bevoorraad met S kilo voedsel en IFC wil dit met zo weinig mogelijk schepen gaan brengen. De database is niet gesorteerd op laadvermogen. Geef een algoritme dat een zo klein mogelijke set schepen berekent met totaal laadvermogen S of meer. Zeg ook waarom jouw algoritme correct is.
Het kan in *lineaire tijd*!

36. **Records met Begrensde Som:** Je hebt (ongesorteerde) records over zieke kinderen, waarbij kind i kan worden genezen met een investering k_i. Je hebt D euro aan donaties ontvangen, waarmee je zoveel mogelijk kinderen wilt genezen. Beschrijf een algoritme dat een zo groot mogelijke set kinderen kiest zodat de totale k is begrensd door D. Het kan in *lineaire tijd*!

37. **De Top Twee:** Laat zien hoe je met $\frac{3}{2}n - 2$ vergelijkingen het grootste en het op een na grootste element in een ongesorteerde rij van n elementen kunt vinden, waar n een macht van 2 is.

38. **Mediaan-der-Medianen:** Bij de analyse van het Mediaan-der-Medianen algoritme voor Selectie vonden we deze recurrente betrekking voor de looptijd:

$$W(n) = W(\frac{7}{10}n) + W(\frac{1}{5}n) + \frac{13}{5}n.$$

(a) Waar komt elk van de drie termen van de rechterkant vandaan?
(b) Kun je de oplossing van deze vergelijking vinden met het Master Theorem? Waarom wel/niet?
(c) Bewijs dat voor de oplossing geldt $W(n) \leq 26n$.

39. **De Top-T:** Gegeven: een ongesorteerde rij A van n getallen, en een integer $T \leq n$. Gevraagd: de T kleinste getallen uit A, in oplopende volgorde. Geef een zo snel mogelijk algoritme dat de gevraagde getallen bepaalt. Wat is de tijdcomplexiteit van je algoritme?

40. **De Top Twee:** Gegeven is een ongesorteerde rij integers A van *even* lengte $n > 0$. Laat zien hoe je het *grootste* en *tweede* element van A kunt vinden met $\frac{3}{2}n - 2$ vergelijkingen.

4 Dynamisch Programmeren

41. **Dynamisch Programmeren: Matrix keten:** Gegeven is een rijtje p_0 t/m p_n van n getallen en een rij matrices A_1 t/m A_n, waarbij matrix A_i afmeting $p_{i-1} \times p_i$ heeft.

Gevraagd wordt het minimale aantal vermenigvuldigingen (van getallen) waarmee het matrixproduct $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ kan worden berekend.

Geef een recurrente betrekking waarmee $m[i, j]$, het aantal vermenigvuldigingen voor het berekenen van $A_i \cdot \ldots \cdot A_j$, wordt uitgedrukt in de kosten van vermenigvuldigingen van minder matrices.

42. **Dynamisch Programmeren: Segmentering:** Gegeven een rij A van n getallen en een getal k. Een *k-Segmentering* van A is een rijtje van k indices i_1 t/m i_k; De segmentering verdeelt de rij in $k + 1$ segmenten, namelijk $A[0..i_1)$, $A[i_j..i_{j+1})$ voor $j = 1 \ldots k - 1$, en $A[j_k..n)$. De *prijs* van deze segmentering is de hoogste som van getallen in een van de segmenten; gevraagd een segmentering van minimale prijs.

    Geef een recurrente betrekking waarmee je de beste prijs kunt berekenen met Dynamisch Programmeren.

43. **Van A naar B:** Gegeven integers A en B, die voldoen aan $0 \leq A \leq B$. Gevraagd wordt het kleinste aantal S van stappen, elk *Increment* (I) of *Double* (D), die A veranderen in B (Voorbeeld: $S(8,\ 38) = 4$, met reeks IDID).

    (a) Geef de recursieve karakterisering van S die je krijgt wanneer je als topkeuze stelt: wat is de *eerste stap* van de reeks.
    (b) Is er een voordeel om als topkeuze te kiezen voor *de laatste stap* van de reeks in plaats van de eerste?

44. **Tellen van A naar B:** Een (A, B)-rijtje is een rij tekens I (voor Increment) en D (voor Dubbel) zo, dat als je de operaties toepast op A, er B uitkomt. Voorbeeld: IDII is een (5, 12)-rijtje. Er bestaan drie (5, 12)-rijtjes, namelijk IIIIIII, DII, ID.
    Geef een algoritme dat voor positieve A en B, in $O(B)$ tijd het *aantal* (A, B)-rijtjes uitrekent; licht de looptijd toe.

45. **Verdelen over regels:** Een tekstverwerker (zoals Word of LaTeX) moet de woorden van een alinea verdelen over tekstregels. Stel de alinea heeft n woorden met breedte b_0 t/m b_{n-1} (we negeren de spaties) en de paginabreedte staat op T. Een regel bevat een aantal opeenvolgende woorden, die samen breedte $<= T$ moeten hebben. De *Lelijkheid* van de regel is het kwadraat van de overgebleven ruimte. De Lelijkheid van de alinea is de som van de Lelijkheden van alle regels behalve de laatste.
    Geef een polynomiaal algoritme dat een regelverdeling van minimale Lelijkheid berekent.

46. **Stokzagen met knoesten:** Een stokjesfabriek verzaagt een lat van (integer) lengte n in een of meer stokjes, waarbij een stokje van lengte i wordt verkocht voor p_i. De lat kan op elke integer-positie s worden gezaagd, maar er zitten onregelmatige knoesten in, waardoor zagen op positie s, zaagkosten z_s kost. De *revenu* van de lat is de opbrengst van de stokjes min de totale zaagkosten.

    (a) Geef een recursieve uitdrukking voor de revenu van een lat, waarbij je als topkeuze hanteert: wat is de positie van de hoogste zaagsnede.
    (b) Wat is de looptijd van het resulterende DP-algoritme?

47. **Stokzagen met Zaagbeperking:** Een stokjesfabriek verzaagt een lat van integer lengte n in een of meer stokjes van integer lengte, waarbij een stokje van lengte i wordt verkocht

voor p_i. De lat kan op elke integer-positie s worden gezaagd, maar de zaagbaas heeft bepaald dat er maximaal k keer gezaagd mag worden. De *revenu* van de lat is de opbrengst van de stokjes.

(a) Geef een recurrente uitdrukking voor de revenu van een lat, waarbij je als topkeuze hanteert: de positie van de hoogste zaagsnede (dwz., de zaagsnede met de hoogste positie).
(b) Wat is de looptijd van het resulterende DP-algoritme?

48. **Stokzagen met knoesten en Zaagbeperking:** Een stokjesfabriek verzaagt een lat van (integer) lengte n in een of meer stokjes, waarbij een stokje van lengte i wordt verkocht voor p_i. De lat kan op elke integer-positie s worden gezaagd, maar (1) er zitten onregelmatige knoesten in, waardoor zagen op positie s, zaagkosten z_s kost en (2) van de zaagbaas mag je maximaal k keer zagen.
De *revenu* van de lat is de opbrengst van de stokjes min de totale zaagkosten.

(a) Geef een recursieve uitdrukking voor de revenu van een lat, waarbij je als topkeuze hanteert: wat is de positie van de hoogste zaagsnede.
(b) Wat is de looptijd van het resulterende DP-algoritme?

49. **Chicken:** Een restaurantketen heeft langs een weg n locaties waar een kiprestaurant kan worden gebouwd. Locatie i (voor $i = 0 \ldots n-1$) ligt m_i km vanaf het begin en de winst die men er met een restaurant kan halen is p_i. De directie vindt dat twee restaurants minstens K kilometer van elkaar moeten liggen.

(a) Geef een recursieve karakterisering van de maximale winst.
(b) Wat is de looptijd van het resulterende DP-algoritme?

50. **Snapshots:** Een museum heeft n attracties A_1 t/m A_n, en je bent bij de ingang A_0. De tijd om van A_i naar A_j te reizen is bekend, $d[i, j]$ en een foto van A_i heeft waarde g_i. Het museum sluit over T tijd. Gebruik Dynamisch Programmeren om een reeks foto's te maken met maximale waarde, die je binnen T tijd kunt maken.

51. **Dorst:** Een kamelenhandelaar reist door de Sahara maar zijn water is op. Hij heeft een kaart met de locaties van n oases (1 t/m n), maar het is de droge tijd en veel oases staan droog. De kans dat oase i water bevat, is p_i (dit is onafhankelijk van de andere oases). De reistijd van oase i naar oase j is d_{ij}, en van zijn huidige locatie naar oase i is d_{0i}. De kamelenhandelaar wil een route langs de oases die de verwachte tijd tot het vinden van water minimaliseert.
Help de kamelenhandelaar met Dynamisch Programmeren: geef de recursieve karakterisering en de looptijd van het algoritme.

52. **Topkeuze bij Knapsack:** Bij het Knapsack (Rugzak) probleem zijn gegeven: n objecten van gewicht w_i en opbrengst p_i, en een gewichtslimiet M. De uitvoer is de maximale opbrengst van een deelverzameling van de objecten met totaalgewicht begrensd door M.
Geef een recursieve karakterisering van de te behalen opbrengst, waarbij je als topkeuze neemt: wat is de index van het hoogste gekozen object.

53. **Integer Knapsack:** Geef een recursieve karakterisering voor de waarde van Integer-Knapsack, waar je elk van de beschikbare objecten meerdere malen kunt kiezen.

Input: M (maximum gewicht), p_0 t/m p_{n-1} (opbrengsten van n objecten), w_0 t/m w_{n-1}

(gewichten van n objecten).
Gevraagd: een vector x_0 t/m x_{n-1} van (niet-negatieve) integers, waarbij $\sum_i x_i.w_i \leq M$ en $\sum_i x_i.p_i$ MAXimaal.
Beschrijf de gemaakte topkeuze en alternatieven, geef de resulterende recursieve karakterisering van het optimum, en schat de looptijd van het algoritme. **NB:** Je hoeft geen code te geven.

54. **Fractionele knapsack:** Bedenk een algoritme voor de Fractionele Knapsack, waarbij je elk van de beschikbare objecten kunt doorsnijden en deels meenemen. Input: M, maximum gewicht, p_0 t/m p_{n-1}, winst van object i, w_0 tm $wn-1$, gewicht van i.
Gevraagd: een vector x_0 tm x_{n-1} van reële getallen in [0..1]. Waarbij $\sum_i xi.wi \leq M$ en $\sum_i xi.pi$ zo groot mogelijk.

55. **Disk Controller:** Waarom is de afstands-matrix van het TSP probleem asymmetrisch als je TSP gebruikt om een verzameling pagina's van een disk te halen?

56. **Beladingen tellen:** Een smokkelaar heeft een rugzak van omvang M, keuze uit n objecten 0 t/m $n-1$ van gewicht w_0 t/m w_{n-1} en winst p_0 t/m p_{n-1}. De smokkelaar wil weten op hoeveel manieren hij zijn rugzak kan beladen. Laat zien hoe je het *aantal* deelverzamelingen met gewicht $\leq M$ kunt bepalen in $O(n.M)$ tijd.

57. **Totale Extra Leeropbrengst:** Een informaticadocent moet n algoritmen $A_0, \ldots, A_{n-1}$ uitleggen aan zijn studenten. Hij heeft ontdekt dat er sprake kan zijn van een Extra Leeropbrengst van een onderwerp A_j als onderwerp A_i eerst is behandeld (door didactische synergie). Op de manier waarop deze leeropbrengst wordt gequantificeerd gaan we hier niet in, maar we volstaan te stellen dat er aan het gehele vak een Extra Leeropbrengst $M[i,j]$ wordt TOEGEVOEGD als A_i eerder dan A_j wordt behandeld.
(a) Geef een DP algoritme dat de volgorde met hoogste TEL bepaalt.
(b) Wat is de rekentijd van je algoritme.
(c) Kun je je algoritme verbeteren indien de meeropbrengst SYMMETRISCH is, dwz., $M[i,j] = M[j,i]$?

58. **Fietsverhuur:** Een verhuurder heeft n fietsen; de lengte van fiets i is f_i. Hij kan fietsen verhuren aan een gezelschap van $m \leq n$ personen; de lengte van persoon i is p_i. Omdat iedereen het liefst op een passende fiets rijdt, rekenen we een *mismatch* van $|f_j - p_i|$ wanneer p_i rijdt op f_j.
Geef een DP algoritme dat de fietsen met minimale mismatch verdeelt over de personen en analyseer de rekentijd.

5 Greedy

59. **De Lift:** Een schooljuf komt met n kinderen, met gewicht a_0 t/m a_{n-1}, bij een lift met laadvermogen L. Juf wil *zo veel mogelijk* kinderen in de lift zetten, en met de andere kinderen de trap nemen. Ze komt op het idee, de kinderen van *licht naar zwaar* in de lift te laten lopen tot het volgende kind er niet meer bij kan.

    Bewijs dat de juf op deze manier een zo groot mogelijk aantal kinderen in de lift krijgt.

60. **Typesetten op weinig regels:** Een tekstverwerker verdeelt een alinea, met n woorden met lengte w_0 t/m w_{n-1}, over regels met breedte B. Geen regel mag woorden met totale lengte groter dan B bevatten (de spatie is al in w_i meegerekend).
De alinea wordt over $k+1$ regels verdeeld door het invoegen van k regelovergangen, voor de woorden s_1 t/m s_k. Dat regel r binnen de breedte past wordt uitgedrukt door $\sum_{i=s_r}^{s_{r+1}-1} w_i \leq B$.
De wens van programmeur Poorten is: zo *weinig mogelijk* regels gebruiken. Het algoritme van Poorten begint, met zoveel mogelijk woorden op de bovenste regel te zetten, dus kiest de *hoogste* s_1 waarvoor geldt: $\sum_{i=0}^{s_1-1} w_i \leq B$. De woorden vanaf s_1 worden weer als nieuwe alinea beschouwd; als alles op een regel past, is geen verdere splitsing nodig.
Formuleer en bewijs de eigenschap, dat deze strategie een verdeling in een minimaal aantal regels geeft.

61. **Fibonacci-Huffman:** Geef een optimale Huffman-code voor $\{a, b, c, d, e, f, g, h\}$, waarbij de relatieve frequenties zich verhouden als de Fibonacci-getallen:

σ	a	b	c	d	e	f	g	h
$f(\sigma)$	1	1	2	3	5	8	13	21

62. **Gewogen Activiteiten:** Een zaalverhuurder heeft n verzoeken voor zijn zaal, waarbij verzoek i de zaal wil gebruiken van tijd s_i tot f_i. De toegestane verzoeken mogen niet overlappen, dus mogelijk kan slechts een deel worden toegewezen.
(a) Beschrijf een toewijzings-algoritme dat garandeert, dat *zoveel mogelijk* verzoeken worden toegewezen.
(b) De zaal wordt per tijdseenheid betaald; toewijzen van verzoek i levert dus $f_i - s_i$ op. Geeft je algoritme uit (a) altijd de meest winstgevende toewijzing? Bewijs of geef een tegenvoorbeeld.

63. **Greedy Plaatsing van Lantaarns:** Een lange weg heeft n huizen, met huis i op afstand s_i meter van het begin (dit is gesorteerd dus $i < j \Rightarrow s_i \leq s_j$). Het gemeentebestuur gaat lantaarns plaatsen en heeft besloten dat elk huis een lantaarn moet hebben binnen M meter. **Voorbeeld:** Als de s zijn: 340, 670, 1200, 1600, 2400, 2710 en $M = 500$ dan kan het met drie lantaarns, bv. op posities 500, 1500 en 2400.
Geef een algoritme dat in lineaire tijd het minimaal aantal benodigde lantaarns berekent. Wat is de uitvoer van je algoritme in het voorbeeld? Waarom is je algoritme correct (geef GCP)?

64. **Fietsverhuur:** Een verhuurder heeft n fietsen; de lengte van fiets i is f_i. Hij kan alle fietsen verhuren aan een gezelschap van n personen; de lengte van persoon i is p_i. Omdat iedereen het liefst op een passende fiets rijdt, rekenen we een *mismatch* van $|f_j - p_i|$ wanneer p_i rijdt op f_j.
Geef een Greedy algoritme dat de fietsen met minimale mismatch verdeelt over de personen en analyseer de rekentijd.

6 Union-Find

65. **Union-Find Implementatie:** De *woud-representatie* is een manier om verzamelingen (sets) bij te houden die je kunt samenvoegen. Welke twee verbeteringen moet je toevoegen aan de woud-representatie om te komen tot de snelst bekende implementatie van dit probleem? Noem ze en geef van elk in één zin aan hoe ze werken.

66. **Gewogen Union:** Een Union-Find structuur representeert elke verzameling als een boom en gebruikt gewogen union: bij een union wordt de wortel van de lichtste boom een kind van de wortel van de andere boom.
Toon aan, dat de lengte van het pad van een element naar de wortel van zijn verzameling nooit meer dan $\lg n$ kan zijn (n is het aantal elementen).

7 Amortiseren

67. **Amortisering:** Op een datastructuur wordt een reeks stappen 1 t/m n uitgevoerd. Stap i kost 1 als i geen tweemacht is, en $i + 1$ is als i wel een tweemacht is.
Bewijs dat de geamortiseerde kosten hoogstens 3 per stap zijn.

68. **Geamortiseerde indexering:** Gegeven zijn twee gesorteerde rijen getallen, A en B, van lengte n. We willen een *A-index* voor B maken, dwz., een rij L die de plek in A aangeeft waar $B[i]$ past: dus $A[L[i]] \leq B[i] < A[L[i] + 1]$. De waarde van $L[i]$ is -1 als $B[i]$ kleiner is dan $A[0]$ en $n - 1$ als $B[i]$ groter is dan $A[n - 1]$.

    Je kunt de index berekenen door elke $B[i]$ apart (bv. met Binair Zoeken) te zoeken in A, maar dat kost totaal $\Theta(n \lg n)$. Schrijf een algoritme dat $L[i]$ bepaalt door lineair zoeken vanaf de vorige gevonden positie in A. Laat zien dat voor één waarde van i soms $\Theta(n)$ stappen nodig zijn, maar de totale kosten toch $O(n)$ zijn.

69. **Binaire Counter:** Een array van tien bits wordt gebruikt om een counter bij te houden. In de beginsituatie is de waarde 33, dus binair 0000100001. Met 57 Increments wordt de waarde opgehoogd tot 90.

    (a) Wat is het hoogste aantal bit-wijzigingen dat in een enkele Increment in deze reeks gebeurt, en welke stap is dat?
    (b) Wat is het totale aantal bit-wijzigingen in de 57 Increments? (Beredeneren, niet tellen!)
    (c) Bewijs dat: als de waarde K is, en $K < 2^9$, en je doet dan K Increments om de waarde tot $2K$ op te hogen, dan is het gemiddelde aantal bit-wijzigingen per Increment exact 2.

70. **Amortizering: Binaire Counter:** Een array van 12 bits wordt gebruikt om een counter bij te houden. In de beginsituatie is de waarde 30, binair 000000011110. Met 160 Increments wordt de waarde opgehoogd tot 190 (000010111110).

    (a) Wat is het hoogste aantal bit-wijzigingen dat in een enkele Increment in deze reeks gebeurt, en welke stap is dat?
    (b) Wat is het totale aantal bit-wijzigingen in de 160 Increments?

71. **Complexiteiten:** Leg (kort!) het verschil uit tussen een verwachte, een gemiddelde, en een geamortiseerde complexiteit.

72. **Gemiddelden:** De *gemiddelde* (average), *expected* (verwachte) en *geamortiseerde* amortized complexity berekenen alledrie een gemiddelde over looptijden.
Waarover wordt gemiddeld bij het berekenen van een verwachte, een gemiddelde, en een geamortiseerde complexiteit?

73. **Amortisering: Queue backup:** Van een queue, waar nooit meer dan k elementen tegelijk in staan, wordt na elke k^{e} operatie een backup kopie gemaakt.
Laat zien dat de geamortiseerde complexiteit van de queue operaties nog steeds O(1) is.

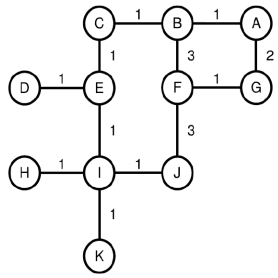
74. **Queue van twee stacks:** Een programmeurtje heeft in zijn programma een queue nodig maar is te lui om in de C# documentatie op te zoeken hoe die heet. Gelukkig weet hij nog wel hoe een stack heet en kent hij dit truukje: Gebruik twee stacks A en B, initieel leeg. Een `enq` vertaalt naar een `push` op de in-stack B. Een `deq` vertaalt naar een `pop` van de uit-stack A. Maar als A leeg is worden eerst alle elementen van B overgestapeld naar A. Als A dan nog steeds leeg is, volgt de empty queue exception.

    Toon aan dat de prijs van elke queue operatie geamortiseerd $O(1)$ is.

8 Breadth First Search

75. **Graaf-representatie:** Geef een algoritme dat een Adjacency-list representatie van een graaf G omrekent naar een matrix-representatie.

76. **BFS Volgorde:**
    Op dit netwerk wordt een Breadth First Search uitgevoerd met startpunt A en elke knoop exploreert zijn buren in alfabetische volgorde. In welke volgorde worden de knopen ontdekt?



77. **De Kliek:** Een *Kliek* is een ongerichte graaf die tussen elk tweetal knopen een kant heeft. Hoeveel kanten heeft een kliek met n knopen?

78. **De In-graad:** Een gerichte graaf G (met n knopen en m kanten) is gegeven in de vorm van een adjacency list: lijst $L[u]$ bevat v als $uv \in E$. Hoe kun je, voor een gegeven knoop w, de *in-graad* van w (het aantal kanten met eindpunt w) bepalen? Hoeveel tijd kost dit?

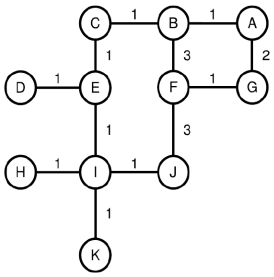
79. **Geared:** In het puzzelspel Geared (zie Youtube) moet je tandwielen verplaatsen, totdat alle blauwe wielen tegelijk draaien, aangedreven vanuit een draaiend geel tandwiel. Elk tandwiel dat raakt aan een draaiend wiel draait ook, maar uiteraard andersom. Als een tandwiel in twee richtingen wordt aangedreven, blokkeert het aandrijvende gele wiel!

    Na elke verplaatsing van een wiel door de speler zijn de tandwielcontacten weergegeven als per tandwiel een lijst van tandwielen die het raakt. Geef een (efficient) algoritme dat bepaalt of alle blauwe wielen draaien. (Het kan in $O(n)$ voor n wielen.)

80. **De Uit-graad:** Een gerichte graaf G (met n knopen en m kanten) is gegeven in de vorm van een adjacency list: lijst $L[u]$ bevat v als $uv \in E$. Hoe kun je, voor een gegeven knoop w, de *uit-graad* van w (het aantal kanten met beginpunt w) bepalen? Hoeveel tijd kost dit?

81. **Eccentricity met BFS:**

    De *eccentricity* van een knoop is de afstand (aantal kanten) tot de verst weg gelegen knoop; bv de eccentricity van E in het graafje hiernaast is 4 (afstand tot G). Geef de BFS procedure, uitgebreid om eccentricity van het startpunt te berekenen, en geef de rekentijd.



82. **De Boom:** Een *Boom* is een ongerichte graaf die samenhangend is en geen cykel bevat. Hoeveel kanten heeft een boom met n knopen?

83. **Dungeons:** Een level van zeker dungeon-game is gemodelleerd als een graaf $G = (V, E)$ (grotten en de gangen ertussen). In dit level kan de speler kiezen uit a knopen s_0 t/m s_{a-1} om het level binnen te komen, en uit b knopen t_0 t/m t_{b-1} om het level te verlaten. Je wilt het level in zo weinig mogelijk stappen doorlopen (elke gang is 1 stap). Geef een algoritme dat in lineaire tijd (dwz., $O(n + m)$) een kortst pad bepaalt van *een* ingangsknoop naar *een* uitgangsknoop.

84. **BFS-bomen:** Wat is een BFS boom (met wortel s)? Geef een voorbeeld van een graaf met startpunt, en een BFS boom, die niet door het BFS algoritme gevonden kan worden.

85. **Bereikbare knopen tellen:** Gegeven een gerichte graaf $G = (V, E)$. Voor knoop s in V, is B_s de verzameling knopen bereikbaar uit s, dus $B_s = \{x \in V \mid \text{er bestaat een pad van } s \text{ naar } x\}$.
    Geef een uitbreiding van het Breadth-First Search algoritme dat telt hoeveel knopen bereikbaar zijn, maw., $\#B_s$ bepaalt.

86. **Routeplannen met BFS:** Een programmeur maakt een routeplanner voor in de auto en wil voor het zoeken van routes van A naar B een BFS doen met B als startpunt. De schil waar A in komt, de waarde van `d[A]`, geeft immers aan hoe ver A afligt van B, en de route is na afloop te vinden met de π-pointers in elke knoop. Waarom is dit geen goed idee? Hoe moet het wel?

87. **Samenhangscomponenten:** Gegeven een ongerichte graaf $G = (V, E)$, gerepresenteerd als Adjacency List. Beschrijf een algoritme dat *in lineaire tijd* $O(n + m)$ telt, uit hoeveel samenhangscomponenten G bestaat.

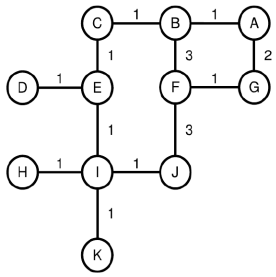
88. **Non-determinisme in BFS:** Laat door een voorbeeld zien dat bij BFS de opgebouwde boom afhankelijk is van de volgorde waarin knopen in de Adjacency-lijsten staan. Laat zien dat de waarde $d[u]$ die in knoop u berekend wordt (namelijk, wanneer u wordt ontdekt vanuit v, als $d[v] + 1$), niet van de ordening van de Adjacency-lijsten afhangt.

9 Depth First Search

89. **Kanten in een Woud:** Een ongerichte graaf is een *woud* als hij geen cykel bevat. De graaf hoeft niet samenhangend te zijn, hij kan dus uit meerdere samenhangscomponenten bestaan. Een *boom* is een speciaal geval van een woud, waarbij er precies 1 component is. Bewijs dat als G een woud is van s samenhangscomponenten, dan heeft G precies $n - s$ kanten.

90. **Bereikbare knopen tellen met DFS:** Gegeven een gerichte graaf $G = (V, E)$. Voor knoop s in V, is $B_s = \{x \in V \mid \text{er bestaat een pad van } s \text{ naar } x\}$, de knopen bereikbaar uit s.
Beschrijf hoe je het Depth-First Search algoritme kunt uitbreiden, zodat het telt hoeveel knopen bereikbaar zijn, maw., $|B_s|$ bepaalt. Het moet werken binnen $O(n + m)$ tijd.

91. **DFS Volgorde:**
Op dit netwerk wordt een Depth First Search uitgevoerd met startpunt A en elke knoop exploreert zijn buren in alfabetische volgorde. In welke volgorde worden de knopen ontdekt?



92. **Discovery en Finishing times:** Laat d_u en f_u de *discovery* en *finishing* time van knoop u zijn in een DFS van een graaf. Kan het, voor buren u en v, voorkomen dat $d_u < d_v < f_u < f_v$? Kan het voor willekeurige knopen voorkomen?

93. **Cykels in Ongerichte Graaf:** Geef een algoritme dat test of een ongerichte graaf met n knopen een cykel bevat, dan wel een woud is. Zorg dat het algoritme loopt in $O(n)$ tijd, dus onafhankelijk van het aantal kanten.

94. **DFS: Klassificatie van kanten:** In welke vier categorieën worden de kanten van een gerichte graaf verdeeld op grond van een DFS exploratie? Geef van elke categorie de definitie.

95. **DFS: Cykels en Back-edges:** Bewijs deze bewering: de graaf G bevat een cykel dan en slechts dan als er bij DFS een back-edge ontstaat.

96. **Pad zoeken met DFS:** Gegeven zijn een gerichte graaf $G = (V, E)$ en twee knopen s en t in de graaf. Geef een algoritme (pseudocode) dat in $O(n + m)$ tijd bepaalt of er een pad van s naar t bestaat, en zo ja, zo'n pad oplevert. Waarom is je algoritme correct en waarom voldoet het aan de tijdgrens?

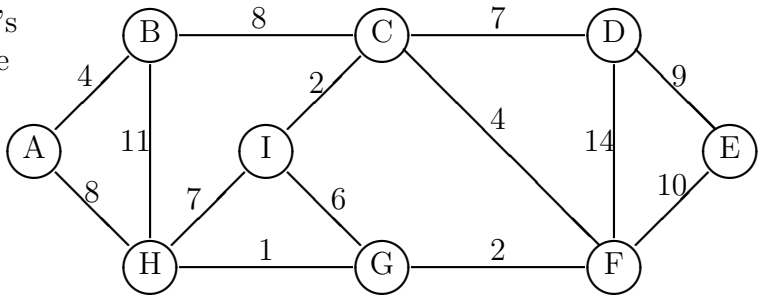
97. **DFS op acyclische graaf:** Welke vier typen kanten onderscheiden we na een Depth First Search op een gerichte graaf? Welk van de types *ontbreekt* als de graaf acyclisch is?

98. **DFS en Cross edges:** (a) Wat verstaan we bij DFS onder een cross edge?
(b) Waarom komen cross edges niet voor als een DFS wordt gedaan in een ongerichte graaf?
(c) Is deze bewering *Elke uitgaande kant van de startknoop bij DFS wordt een tree-edge* WAAR of ONWAAR? (Motiveer!)

10 Minimum Spanning Tree

99. **MST: Prim:**

Laat stap voor stap zien hoe Prim's algoritme een minimale spannende boom berekent op deze graaf, beginnend in punt E (dus niet in A!).



100. **MST: Kruskal:** Laat stap voor stap zien hoe Kruskals algoritme een minimale spannende boom berekent op deze graaf. Zeg expliciet in welke volgorde je de kanten bekijkt.

101. **MST en kant-toevoeging:** Gegeven een gewogen graaf $G = (V, E)$ en een minimum spanning tree T van G. Aan G wordt een kant toegevoegd: dwz., we vormen $E' = E \cup \{h\}$, waar h twee knopen van G verbindt. Beschrijf een algoritme dat vanuit T in $O(n)$ tijd een nieuwe boom T' berekent, die een MST van $G' = (V, E')$ is. (Je hoeft geen correctheids-argument te geven.)

102. **MST en kant-verwijdering:** Gegeven een gewogen graaf G en een Minimum Spanning Tree T van G. Eén van de kanten van T wordt verwijderd uit G, dwz., we vormen de graaf G' die een kant minder heeft. Geef een algoritme (geen code maar beschrijving in tekst) dat een nieuwe MST van de graaf G' berekent in $O(n + m)$ tijd.

103. **Min-Edge in Minimal Spanning Tree:** Gegeven is een gewogen graaf $G = (V, E, w)$. Een *MinKant* is een kant van minimaal gewicht in de graaf, dwz., e is een MinKant als $\forall f \in E : w(f) \geq w(e)$. Een MaxKant is een kant van maximaal gewicht.
(a) Bevat een MST altijd *alle* MinKanten?
(b) Kan een MaxKant in een Minimal Spanning Tree zitten?

104. **MST met negatieve kanten:** Je beschikt over een erg goeie implementatie van Prims Minimum Spanning Tree algoritme, die helaas vooronderstelt dat kantgewichten positief zijn (programmeur heeft nl. `uint` gebruikt). Je wilt hiermee een MST uitrekenen voor een graaf waarin ook negatieve kantgewichten voorkomen.
(a) Hoeveel tijd kost Prims algoritme?
(b) Beschrijf hoe je de implementatie kunt gebruiken voor een graaf met negatieve kantgewichten, zonder dat de asymptotische tijd toeneemt.
(c) Zeg waarom het correct is en waarom de O-tijd gelijk blijft.

105. **Kant kiezen bij MST:** In het Dijkstra/Prim algoritme voor Minimum Spanning Tree moet je telkens een *lichtste uitgaande* kant kiezen en die toevoegen aan je tree. Wanneer er twee uitgaande kanten van hetzelfde laagste gewicht zijn, die samen geen cykel introduceren, mag je er toch maar één daarvan toevoegen.
(a) Geef hiervan een voorbeeld: een graaf met vijf knopen, waarbij er een *foute* MST ontstaat bij Dijkstra/Prim wanneer je toch beide lichtste kanten zou toevoegen.
(b) Kun je bij Kruskal's algoritme wel meerdere kanten tegelijk toevoegen als die een gelijk minimaal gewicht hebben en geen cykel introduceren?

11 Kortste Pad

106. **Kortste Pad Algoritmen:** Wat is de worst-case complexiteit van Dijkstra's algoritme en van het Bellman-Ford algoritme voor single-source shortest path (op n knopen en m kanten)? Waardoor kun je in veel toepassingen, zoals routeplanners op een wegen-, spoorweg of communicatienetwerk, de daadwerkelijke performance van het Bellman-Ford algoritme veel beter krijgen? Onder welke omstandigheden kun je beter Dijkstra, en onder welke omstandigheden beter Bellman-Ford gebruiken?

107. **Relaxaties voor Kortste Pad:** Geef de pseudocode voor het initialiseren van een kortstepad-algoritme met relaxaties, en de code voor het relaxeren van kant uv.

108. **Negatieve Cykel:** Deze vraag gaat over berekening van kortste paden, gebruik makend van relaxaties. Bewijs dat, als er ooit een relaxatie plaatsvindt waarbij de afstandsschatting d van de startknoop s verandert, dan bevat de graaf een negatieve cykel.

109. **Kortste Paden met begrensd Kantgewicht:** Gegeven is een gewogen graaf $G = (V, E, \omega)$, waarbij alle kantgewichten integers zijn in het bereik 1..16. Laat zien, dat je voor deze situatie het Single-Source Single-Destination kortstepad-probleem kunt oplossen in *lineaire* tijd, namelijk $O(|V| + |E|)$.

110. **Kortst Pad met drie kantlengten:** Een level van zeker Dungeon game is een gewogen graaf $G = (V, E, w)$ (n grotten, m gangen, kosten per gang). Het aantal goudstukken dat je betaalt in gang uv is $w(uv)$ en dit is altijd 1, 2 of 3. Ingang s en uitgang t zijn gegeven. Geef een manier om, in $O(n + m)$ tijd, een goedkoopste route van s naar t te vinden.

111. **Routeplanning:** Een routeplanner berekent kortste paden in een wegennetwerk. Wat is het voordeel van het gebruik van een Fibonacci-heap in Dijkstra's algoritme voor een routeplanner?

12 Max Flow

112. **Max Flow berekeningen:** Zij G een flow netwerk (V, E, c, s, t) en f een flow in G.
(a) Waarvoor staan V, E, c, s en t?
(b) Hoe zijn de restcapaciteit (residual capacity) en het restnetwerk (residual network) gedefinieerd?
(c) Gegeven is dat het restnetwerk *geen* pad van s naar t bevat. Bewijs dat f een maxflow is.

113. **Augmenterende paden:** De Ford-Fulkerson methode voor Flow laat steeds de flow toenemen over een pad van s naar t. Als je zo'n pad zoekt, kan je dan het best (a) Breadth-First search, (b) Depth-First search, (c) een "breedst" pad (van maximale restcapaciteit), gebruiken? *Zeg ook waarom.*

114. **Augmented Flow:** Het Ford Fulkerson algoritme zoekt een st-pad P in het restnetwerk, en verhoogt de flow f op de kanten van P, met een waarde r die de kleinste restcapaciteit op het pad is.
(a) Waarom is dit een verbetering, m.a.w., waarom is r positief?

(b) Bewijs dat de nieuwe flow f' inderdaad een flow is (drie eigenschappen, of twee volgens CLRS).

115. **Flow met drie capaciteiten:** Koen gaat een flow bepalen in netwerk $G = (V, E, c, s, t)$, waar de capaciteit van een kant alleen maar 0, 1 of 2 kan zijn.

(a) Bespreek, waarom Koen voor het zoeken van augmenterende paden wel of geen Breadth First Search zou gebruiken in dit geval.
(b) Koens opdracht is: de kanten van een lichtste snede rapporteren, en hij besluit om, na het berekenen van de flow, alle verzadigde kanten te rapporteren. Geef een voorbeeld van een graaf met 5 knopen en een flow, waarin de verzameling van verzadigde kanten *niet* het gewicht van een lichtste (s, t)-snede heeft.

116. **Max Flow voor matching:** Je schrijft een programma voor een projectbureau dat moet bepalen of n programmeurs één-op-één kunnen worden gekoppeld aan n projecten. Elke programmeur geeft een lijst van welke projecten (minstens 1) hij eventueel kan doen; de som van lijstlengten noemen we L. **Voorbeeld** (met $n = 5$ en $L = 12$): P1 kan AB, P2 kan ABC, P3 kan ABD, P4 kan DE, P5 kan DE.
Dit Matching-probleem kun je oplossen met MaxFlow.

(a) Teken het Flow netwerk G dat je krijgt voor het voorbeeld. Wat is in het algemeen het aantal knopen en kanten van G?
(b) Moet je de augmenterende paden zoeken met BFS? Motiveer!
(c) Wat is de looptijd (in O)?

117. **Max Flow in Polynomiale tijd:** Als je Ford-Fulkerson gebruikt met willekeurige paden (dus niet BFS) is de rekentijd begrensd door $O(m.|f^*|)$.

(a) Waarom geldt dit *niet* als polynomiale tijd?
(b) Waarom is het algoritme *wel* polynomiaal als je paden met BFS zoekt?

13 NP-Volledigheid

118. **NP-Volledigheid:** Wanneer is een probleem *NP-Volledig*?
Is deze definitie toepasbaar op Beslissingsproblemen? Is deze definitie toepasbaar op Optimaliseringsproblemen? Is deze definitie toepasbaar op Constructieproblemen?

119. **NP-Volledige Uitdaging:** Je baas vraagt je om een snel algoritme voor een optimaliseringsprobleem R, maar je collega zegt dat R NP-volledig is. Wat kun je doen om het probleem aan te pakken? (Geef minstens vier suggesties.)

120. **NP-Volledige Problemen:** Noem zes NP-volledige problemen.

121. **Polynomiale tijd:** Je kunt het *Knapsack* probleem (met n objecten van gewicht w_i en winst p_i en rugzakgrootte M) oplossen met Dynamisch Programmeren. Je maakt dan een tabel van $(n + 1) * (M + 1)$ deelproblemen die elk in $O(1)$ tijd berekend worden.

Is dit algoritme *polynomiaal*? Waarom?

122. **NP-Volledigheid:** (a) Wanneer is een probleem NP-Volledig?
(b) Wat moet je doen om te bewijzen dat een probleem NP-Volledig is?

123. **NP:** Sommige mensen denken dat NP: *Niet-Polynomiaal* betekent, maar dat is niet zo.
Wat betekent NP wel? Waarom is het geen goed idee om dit Niet-Polynomiaal te noemen?

14 String Matching

124. **Radix conversie:** De radix-r representatie van een getal K is een rijtje van m digits $d_{m-1} \ldots d_1 d_0$, elke digit $0 \leq d_i < r$, zo dat $K = \sum_i d_i \cdot r^i$.
(a) Geef een algoritme dat, gegeven de representatie, K berekent in $O(m)$ tijd.
(b) Geef een algoritme dat, gegeven $K < r^m$, de radix-r representatie van K berekent in $O(m)$ tijd.
(c) **Bonus:** Geef een algoritme dat, gegeven K en een positie i, in $O(1)$ tijd d_i bepaalt. Je mag bij (c) veronderstellen dat machten van r zijn voorberekend en opgeslagen.

125. **Rabin-Karp:** We zoeken een string met Rabin-Karp, zonder de reductie modulo een priemgetal. Het alfabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$.
(a) Op welke getalswaarden worden het patroon `baca` en de begin-shift van de tekst `abacadabca` afgebeeld?
(b) Hoe wordt in Rabin-Karp in constante tijd (dus onafhankelijk van de lengte van het patroon) de getalswaarde van een volgende shift berekend? Geef de formule en laat als voorbeeld de berekening van de tweede shift zien bij dezelfde invoer als (a).

126. **KMP Prefix Functie:** Knuth, Morris en Pratt gebruiken in hun String Matcher een *prefix functie* van het patroon P.
(a) Hoe is de prefix functie gedefinieerd?
(b) Geef de prefix functie van patroon `abacabadabacaba`.

15 Diversen

127. **Elgamal encryptie:** Bij Elgamal encryptie heeft elke gebruiker een *private key* a en een *public key* b. De encryptie van boodschap x is het paar $(u, v) = (g^k, x.b^k)$.
(a) Wat is de relatie tussen de public en de private key?
(b) Hoe wordt de boodschap ontsleuteld?

128. **Handelsreiziger benadering:** De afstandsmatrix in een instantie van TSP (Handelsreiziger) *kan* de eigenschappen **Symmetrie** en/of **Driehoeksongelijkheid** hebben.
(a) Wat betekenen deze eigenschappen (formule!)?
(b) Welke approximatie-ratio is mogelijk (met een polynomiaal algoritme) wanneer beide eigenschappen *niet* gelden?
(c) Welke approximatie-ratio is mogelijk (met een polynomiaal algoritme) wanneer beide eigenschappen *wel* gelden?

129. **Approximatie-Algoritme:** Geef de definitie van een *Approximatie-Algoritme*. Wat is het verschil met een *heuristiek*?

130. **TSP met Closest Point:** Closest Point is een simpel algoritme voor TSP met driehoeksongelijkheid en symmetrie. Een cykel van steden wordt steeds uitgebreid. Begin met de triviale cykel die alleen het startpunt heeft. Telkens kies je het onbezochte punt dat het dichtst ligt bij een punt op de cykel. Als v het meest nabije nieuwe punt is, en u het cykelpunt waar v dichtbij ligt, stop dan v in de cykel na u. Herhaal tot alle punten in de cykel zitten.

    (a) Wat is het verschil tussen een *greedy algoritme*, een *heuristiek* en een *approximatie-algoritme*?
    (b) Welk van deze types is het Closest Point algoritme? Zeg kort waarom.

131. **Randomisering: Waarden:** Gegeven is dat array A van lengte n slechts twee verschillende waarden bevat, elk $n/2$ keer. Geef een randomiserend algoritme dat die twee waarden bepaalt. Wat is de slaagkans, de terminatiekans en de verwachte complexiteit? Is het een Monte Carlo, Las Vegas of Sherwood algoritme?

132. **Discodrukte:** Een disco is op een avond bezocht door n personen, waarbij persoon i arriveerde op tijd $a[i]$ en vertrok op tijd $v[i]$. Beschrijf hoe je in $O(n \lg n)$ tijd kunt uitrekenen, hoeveel personen maximaal tegelijk aanwezig waren. Je hoeft geen code te geven.