Hertentamen Algoritmiek
Maandag 17 juli 2023, 13.00 – 16.00 uur

Wanneer er in een opgave gevraagd wordt om uitleg, toelichting of motivatie van je antwoord, is het belangrijk om die ook te geven.
De aantallen punten die bij het begin van elke opgave vermeld worden, zijn indicatief. Ze kunnen dus nog iets wijzigen.
Veel succes!

---

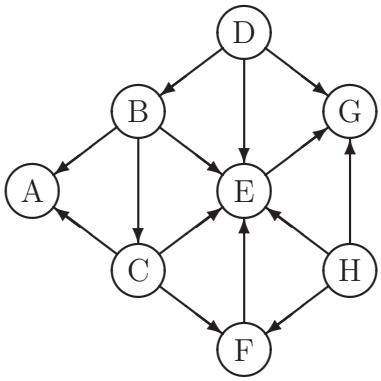
1. [17 pt]

   (a) In een gerichte acyclische graaf (*directed acyclic graph = DAG*) kun je een *topologische ordening* van de knopen bepalen. Wat verstaan we onder zo'n topologische ordening?

   Als je het antwoord op dit onderdeel niet weet, dan kun je het 'kopen' van de docent. Wellicht kun je dan wel de hierna volgende onderdelen maken.

   (b) Beschrijf (in woorden of met (pseudo-)code, maar in ieder geval duidelijk en volledig) een *decrease-by-one-and-conquer* algoritme om in een DAG een topologische ordening te bepalen.

   (c) Pas het algoritme uit onderdeel (b) toe op onderstaande DAG. Geef voor elke stap van het algoritme duidelijk aan, waartussen je moet kiezen en wat je vervolgens doet, bijvoorbeeld in een overzichtelijke tabel. Licht toe hoe je uitwerking gelezen moet worden.



   Wat is de resulterende topologische ordening?

---

2. [10 pt] Bij het muntenprobleem hebben we $m \geq 1$ soorten munten, met waarden $d_1 = 1 < d_2 < \cdots < d_m$. Van elk soort munt hebben we er onbeperkt veel. We willen een bedrag $n \geq 0$ precies betalen (en zonder wisselgeld) met zo weinig mogelijk munten.

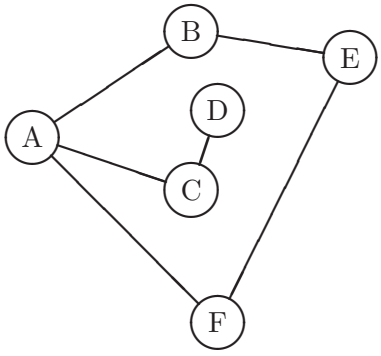
   (a) Beschrijf (in woorden of met (pseudo-)code, maar in ieder geval duidelijk en volledig) een gretig algoritme voor het muntenprobleem.

   (b) Geef een voorbeeld van het muntenprobleem, in de vorm van een reeks muntwaarden $d_1 = 1 < d_2 < \cdots < d_m$ met $m \geq 1$, en een waarde $n \geq 0$, waarvoor het gretige algoritme geen optimale oplossing vindt. Geef ook aan wat de optimale oplossing is, en welke oplossing gevonden wordt door het gretige algoritme. Zorg ervoor dat in je voorbeeld m hoogstens 5 is.

---

3. [21 pt] Deze opgave gaat over depth-first search en breadth-first search. Je hoeft je antwoorden bij deze opgave **niet** toe te lichten.

   (a) Beschouw onderstaande graaf G:



   We gaan een depth-first search en een breadth-first search uitvoeren in graaf G, te beginnen in knoop B. Als we daarbij buren van een knoop willen bezoeken, lopen we die in alfabetische volgorde af. We gaan door met onze graafwandeling totdat alle knopen bezocht zijn.

   i. In welke volgorde bezoeken we de zes knopen van G bij de depth-first search, startend in knoop B?

   ii. In welke volgorde bezoeken we de zes knopen van G bij de breadth-first search, startend in knoop B?

   (b) Bij elk van de volgende drie subonderdelen is steeds een van de twee mogelijkheden correct. Bepaal welke.

   i. Maakt een natuurlijke implementatie van breadth-first search gebruik van een queue of van een stapel?

   ii. Een van de twee zoektochten depth-first search en breadth-first search levert twee volgordes van de knopen in de graaf op. Is dit depth-first search of breadth-first search?

   iii. Laat n het aantal knopen in de graaf zijn, en m het aantal takken. Neem aan dat de graaf gerepresenteerd wordt met behulp van een adjacency list. Is de tijdcomplexiteit van depth-first search dan in $\Theta(n+m)$ of in $\Theta(n^2)$?

   (c) Ga er nu vanuit dat de graaf knopen $0, 1, \ldots, n-1$ heeft, en gerepresenteerd wordt met een boolean adjacency matrix A. Er geldt dat $A[i][j]$ is `true`, dan en slechts dan als er een tak is van knoop i naar knoop j.

   Geef een *recursieve* C++-functie `int bereikbareKnopen (int i, int n, bool visited[n], bool A[n][n])`, die met behulp van een *depth-first search* het aantal knopen bepaalt dat direct of indirect bereikbaar is vanuit knoop `i`. Hierin bevat `visited` bij eerste aanroep van de functie allemaal booleans `false`. Knoop `i` telt ook mee als bereikbare knoop vanuit zichzelf.

   Het is niet de bedoeling dat je bij dit onderdeel gebruik maakt van een globale teller. Doe je dat wel, dan verlies je 3 van de punten die je voor dit onderdeel kunt verdienen.

---

4. [29 pt] We hebben in deze opgave $n \geq 1$ verschillende objecten, genummerd $1, 2, \ldots, n$, met waardes (respectievelijk) $v_1, v_2, \ldots, v_n$ en gewichten $w_1, w_2, \ldots, w_n$. We hebben een knapzak met capaciteit W. Alle waardes, gewichten en de capaciteit zijn positieve integers. Doel is om een deelverzameling van de n objecten in de knapzak te stoppen die aan de volgende drie voorwaarden voldoet:

   - Het gezamenlijke gewicht van de objecten in de knapzak is hoogstens W.
   - Als, voor $i = 1, 2, \ldots, n-1$, object i in de knapzak zit, zit object $i+1$ niet in de knapzak (dus geen opeenvolgende objecten).
   - De gezamenlijke waarde van de objecten in de knapzak is zo groot mogelijk.

   Bij dit tentamen gaan we dit probleem oplossen met dynamisch programmeren. Voor de deelproblemen kijken we naar de eerste k objecten (met $0 \leq k \leq n$) en een knapzak met capaciteit j (met $0 \leq j \leq W$). Laat $F(k, j)$ gedefinieerd zijn als de oplossing van het deelprobleem bestaande uit de eerste k objecten en een knapzak met capaciteit j. Ofwel: de maximale waarde van een deelverzameling van de eerste k objecten, die in een knapzak van capaciteit j past, en waarbij geen opeenvolgende objecten zitten. Uiteindelijk zijn we dus geïnteresseerd in $F(n, W)$.

   (a) Beredeneer dat $F(k, j)$ voldoet aan de volgende recurrente betrekking:

$$F(k,j) = \begin{cases} 0 & \text{als } k = 0 \text{ of } j = 0 \\ 0 & \text{als } k = 1 \text{ en } w_k > j \\ v_k & \text{als } k = 1 \text{ en } w_k \leq j \\ F(k-1, j) & \text{als } k \geq 2 \text{ en } w_k > j \\ \max\{F(k-1, j), v_k + F(k-2, j-w_k)\} & \text{als } k \geq 2 \text{ en } w_k \leq j \end{cases}$$

   Bij de hierna volgende onderdelen mag je ervanuit gaan

   - dat er een globaal array `waarde` beschikbaar is, dat gedeclareerd is als `int waarde[n+1]`, waarbij voor $k = 1, 2, \ldots, n$, `waarde[k]`$= v_k$, en
   - dat er een globaal array `gewicht` beschikbaar is, dat gedeclareerd is als `int gewicht[n+1]`, waarbij voor $k = 1, 2, \ldots, n$, `gewicht[k]`$= w_k$, en
   - dat er een globaal array F beschikbaar is, dat gedeclareerd is als `int F[n+1][W+1]`, en dat al geïnitialiseerd is met allemaal waardes -1.

   (b) Geef een recursieve C++-functie `int losOpTD (int n, int W)` die met behulp van top-down dynamisch programmeren, gebruikmakend van de recurrente betrekking uit onderdeel (a), de waarde $F(n, W)$ berekent (en retourneert).

   (c) Geef een niet-recursieve C++-functie `int losOpBU (int n, int W)` die met behulp van bottom-up dynamisch programmeren, gebruikmakend van de recurrente betrekking uit onderdeel (a), de waarde $F(n, W)$ berekent (en retourneert).

   (d) Wat is de tijdcomplexiteit van je functie `losOpBU` uit onderdeel (c), uitgedrukt in n en W? Motiveer je antwoord door een geschikte basisoperatie aan te wijzen en te berekenen hoe vaak die operatie wordt uitgevoerd.

---

5. [23 pt] Een reisbureau wil een route uitzetten langs $n \geq 1$ steden, genummerd $0, 1, \ldots, n-1$. De steden vormen, samen met de tussengelegen wegen, een gerichte graaf, met gewichten op de takken. Het gewicht op de tak van i naar j stelt de kosten voor om rechtstreeks van i naar j te reizen. We noteren het daarom als `kosten[i][j]`. Bij een aanwezige tak van i naar j is `kosten[i][j]` een niet-negatief geheel getal. Merk op dat kosten 0 ook mogelijk zijn. Als er geen tak van i naar j is, is `kosten[i][j]`$= -1$.

   We zoeken een route

   - die (uiteraard) alleen gebruik maakt van aanwezige takken, en
   - die elke stad precies één keer bezoekt,
   - waarvan de totale kosten zo klein mogelijk zijn (ofwel, die het goedkoopst is).

   Een geldige route is een route die aan de eerste twee eisen voldoet. Een geldige *deel*route is een route die aan de eerste eis voldoet en die elke stad *hoogstens* één keer bezoekt. Anders dan bij het handelsreizigersprobleem is de beginstad van een geldige route niet gelijk aan de eindstad van de route. Er is dan ook geen vaste beginstad.

   (a) Geef een niet-recursieve C++-functie `int bepaalKosten (int n, int kosten[n][n], int route[n])` die controleert of de rij getallen die is opgeslagen in `route[0], route[1], ..., route[n-1]` een geldige route is. Zo ja, dan worden de totale kosten van de route geretourneerd. Zo nee, dan wordt `INT_MAX` geretourneerd. Je mag ervanuitgaan dat alle waardes `route[i]` getallen zijn uit $\{0, 1, \ldots, n-1\}$. De parameters n en `kosten` komen overeen met de beschrijving in de introductie van de opgave.

   Probeer ervoor te zorgen dat de tijdcomplexiteit van je functie lineair is in n. Als dat niet lukt, kun je nog wel het grootste deel van de punten verdienen.

   (b) We zouden de functie `bepaalKosten` uit het vorige onderdeel kunnen gebruiken voor een *exhaustive search* algoritme om het probleem op te lossen. Dat gaan we niet doen. We gaan nu *backtracking* doen.

   Geef een recursieve C++-functie `void bepaalMinKosten (int n, int kosten[n][n], int route[n], bool gehad[n], int i, int deelkosten, int &best)` die met behulp van backtracking de kosten van de goedkoopste geldige route bepaalt. Bij binnenkomst in de functie is er al een deelroute, die is opgeslagen in `route[0], route[1], ..., route[i-1]`. Van deze deelroute is al gecontroleerd dat het een geldige deelroute is, die mogelijk nog uit kan groeien tot de goedkoopste geldige route. De parameter `deelkosten` bevat de kosten van de deelroute. Het boolean array `gehad` houdt bij welke steden al in de deelroute voorkomen: `gehad[k]` is `true`, dan en slechts dan als stad k al in de deelroute voorkomt. Als volgende stap moet dus `route[i]` worden ingevuld. De parameter `best` bevat de kosten van de goedkoopste complete route die tot nu toe in de zoektocht is gevonden. De parameters n en `kosten` komen overeen met de beschrijving in de introductie van de opgave.

   De eerste aanroep van de functie zal van de vorm `bepaalMinKosten (n, kosten, route, gehad, 0, 0, best)` zijn. Je mag ervanuit gaan dat alle booleans in array `gehad` op dat moment `false` zijn, en dat `best` is geïnitialiseerd als `INT_MAX`. Na afloop bevat `best` de totale kosten van de goedkoopste geldige route, tenminste als er een geldige route betaat. Als er geen enkele geldige route bestaat, moet `best` `INT_MAX` blijven.