Tentamen Algoritmiek
Maandag 12 juni 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. [31 pt] We hebben in deze opgave $n \geq 1$ verschillende objecten, genummerd $0, 1, \ldots, n-1$, met waardes (respectievelijk) $v_0, v_1, \ldots, v_{n-1}$ en gewichten $w_0, w_1, \ldots, w_{n-1}$. 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 $i = 0, 1, \ldots, n-2$, 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.

   (a) Geef een niet-recursieve functie `bool voorwaardenOK (int n, int W, bool inDeelverz[], int totGewicht)` die controleert of de deelverzameling die gerepresenteerd wordt door het boolean array `inDeelverz` voldoet aan de eerste twee voorwaarden hierboven (over gezamenlijk gewicht en opeenvolgende objecten). Zo ja, dan wordt `true` geretourneerd. Zo nee, dan `false`. Er geldt dat `inDeelverz[i]` is `true`, dan en slechts dan als object `i` in de de deelverzameling zit. De parameter `totGewicht` bevat het gezamenlijke gewicht van de objecten in de deelverzameling.

   (b) Geef een recursieve functie `int maxWaarde (int n, int i, int W, bool inDeelverz[], int totWaarde, int totGewicht, int waarde[], int gewicht[])` die de maximale gezamenlijke waarde retourneert van een deelverzameling objecten die in de knapzak past en die 'voortbouwt op' de deelverzameling die is vastgelegd in `inDeelverz[0]`, ..., `inDeelverz[i-1]`. Voor objecten $0, 1, \ldots, i-1$ ligt dus al vast of ze wel of niet in de deelverzameling zitten. Voor objecten $i, i+1, \ldots, n-1$ moet dat nog, object voor object, worden bepaald. Als er geen deelverzameling meer te vormen is die in de knapzak past, dient een passende defaultwaarde geretourneerd te worden.

   De parameters `totWaarde` en `totGewicht` bevatten de gezamenlijke waarde en het gezamenlijke gewicht van de objecten uit $0, 1, \ldots, i-1$ die al in de deelverzameling zitten. De arrays `waarde` en `gewicht` bevatten de waardes en de gewichten van de objecten, dus bijvoorbeeld `waarde`$[i] = v_i$.

   De functie moet gebruik maken van *exhaustive search* (niet van backtracking) en dus alle deelverzamelingen van de resterende objecten opbouwen. De functie moet gebruik maken van de functie `voorwaardenOK` uit onderdeel (a) om te controleren of een opgebouwde deelverzameling aan de voorwaarden voldoet.

   Merk op dat de aanroep `maxWaarde (n, 0, W, inDeelverz, 0, 0, waarde, gewicht)` de oplossing van het oorspronkelijke probleem geeft, althans de maximale gezamenlijke waarde. De corresponderende deelverzameling wordt niet geretourneerd.

   (c) Hoe groot is het aantal deelverzamelingen (als functie van n) dat door de functie `maxWaarde` uit onderdeel (b) wordt opgebouwd? Wat wordt daarmee de worst case tijdcomplexiteit van het complete exhaustive search algoritme om ons probleem op te lossen? Motiveer je antwoorden.

(d) Geef duidelijk aan hoe je de functie `maxWaarde` uit onderdeel (b) aan zou moeten passen, zodat het niet meer exhaustive search, maar backtracking implementeert. Je mag ook een compleet nieuwe functie geven. Ook als je geen antwoord op onderdeel (b) hebt, kun je nog wel in woorden vertellen hoe het exhaustive search algoritme dat object voor object alle deelverzamelingen opbouwt en controleert, moet veranderen.

---

2. [33 pt]

(a) We kennen twee vormen van dynamisch programmeren: top-down en bottom-up. Beide methodes maken gebruik van een tabel om oplossingen van deelproblemen in op te slaan, zodat die maar één keer hoeven te worden berekend. De volgende drie subonderdelen gaan over drie verschillen tussen de twee vormen. Je hoeft je antwoorden hierbij niet toe te lichten.

i. Welke van de twee vormen (top-down of bottom-up) dynamisch programmeren is recursief en welke is iteratief?

ii. Bij welke van de twee vormen (top-down of bottom-up) dynamisch programmeren worden (in principe) altijd alle deelproblemen opgelost, en bij welke worden alleen de deelproblemen opgelost die nodig zijn voor de oplossing van het oorspronkelijke, complete probleem?

iii. Bij welke van de twee vormen (top-down of bottom-up) dynamisch programmeren kan vaak geheugenruimte voor de tabel met deeloplossingen worden uitgespaard, en bij welke kan dat niet?

We gaan nu kijken naar het probleem Subset-Sum:

> Gegeven $n \geq 1$ verschillende objecten, genummerd $1, 2, \ldots, n$, met waardes (respectievelijk) $v_1, v_2, \ldots, v_n$, en gegeven een getal S: is er een deelverzameling van de objecten waarvan de gezamenlijke waarde precies S is? Alle waardes en het getal S zijn positieve integers.

(b) Stel dat we de volgende vijf objecten hebben, met bijbehorende waardes, en dat $S = 9$:

i	1	2	3	4	5
v_i	5	6	1	8	2

Voor dit voorbeeld is het antwoord op het probleem 'ja'. Geef alle deelverzamelingen van objecten $\{1, 2, 3, 4, 5\}$, zódat de som van de bijbehorende waardes v_i gelijk is aan $S = 9$.

We gaan Subset-Sum oplossen met behulp van dynamisch programmeren. Voor de deelproblemen kijken we naar de eerste i objecten (met $0 \leq i \leq n$) en de som j (met $0 \leq j \leq S$). Laat $F(i, j) = 1$ als er een deelverzameling is van objecten $\{1, 2, \ldots, i\}$, zódat de gezamenlijke waarde van de objecten in de deelverzameling gelijk is aan j. Anders is $F(i, j) = 0$. Het antwoord op het oorspronkelijke, complete probleem is dus 'ja', dan en slechts dan als $F(n, S) = 1$.

(c) Geef een tabel met alle waardes $F(i, j)$ voor het volgende, kleinere voorbeeld, met $n = 3$, $S = 6$ en

i	1	2	3
v_i	3	1	5

(d) Geef een recurrente betrekking voor $F(i, j)$. Denk zowel aan het basisgeval / de basisgevallen, als aan de recursieve stap. Leg ook uit waarom de recurrente betrekking juist is.

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

(e) Bij dynamisch programmeren slaan we alle waardes $F(i, j)$ op in een tabel (array) `F`. We noemen het dan `F[i][j]` (met blokhaken).

Geef een niet-recursieve C++-functie `int losOpSubsetSum (int n, int waarde[], int S)` die met behulp van bottom-up dynamisch programmeren, gebruikmakend van de recurrente betrekking uit onderdeel (d), de waarde `F[n][S]]` (1 of 0 dus) berekent (en retourneert). Het array `waarde` bevat (op posities $1, 2, \ldots, n$) de waardes $v_1, v_2, \ldots, v_n$. Er is geen globaal array `F` gedeclareerd.

---

3. [24 pt] Bij het toewijzingsprobleem (*assignment problem*) hebben we n personen en n jobs. Elke persoon moet één job uitvoeren. Elke job moet door één persoon worden uitgevoerd. Wanneer persoon i job j uitvoert, kost dat $kosten[i][j]$. Doel van het toewijzingsprobleem is om een zodanige toewijzing van jobs aan personen te krijgen, dat de totale kosten worden geminimaliseerd. We willen het toewijzingsprobleem oplossen met behulp van *best-first branch-and-bound*. Daarbij maken we gebruik van een ondergrens.

(a) Beschrijf een geschikte ondergrens voor het toewijzingsprobleem. Geef zo'n beschrijving zowel voor de begintoestand (als er nog helemaal geen jobs aan personen zijn toegewezen) als voor een algemene deeloplossing.

Ga ervanuit dat we per rij werken.

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) Pas de methode best-first branch-and-bound toe op de volgende kostenmatrix met $n = 4$:

	job 1	job 2	job 3	job 4
Aké	5	2	8	6
Bruyne	6	3	5	1
Courtois	8	3	7	4
Dumfries	4	8	2	3

Teken de bijbehorende *state-space-tree*, met bij elke knoop (deeloplossing) de relevante informatie (waaronder ook de opbouw van de ondergrens). Geef ook aan in welke volgorde de knopen zijn aangemaakt, welke knopen gesnoeid worden en waarom.

Ga er ook bij het opbouwen van de state-space-tree vanuit dat we per rij werken.

Wat is dus de optimale oplossing?

(c) Wat zou bij bovenstaand voorbeeld de ondergrens zijn voor de begintoestand, als we bij de berekening *per kolom* zouden werken in plaats van per rij?

Als je deze waarde vergelijkt met de waarde voor de begintoestand in onderdeel (b), waar we per rij werkten, welke waarde lijkt dan bruikbaarder voor het branch-and-bound algoritme? Motiveer je antwoord.

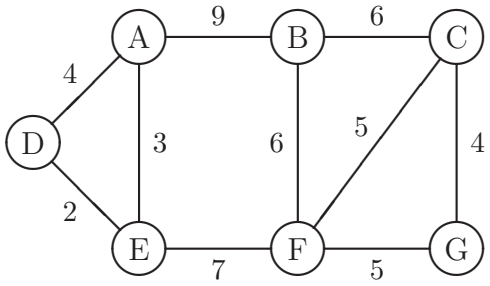
---

4. [12 pt] Je hoeft je antwoorden bij deze opgave **niet** toe te lichten.

Het algoritme van Prim bepaalt voor een samenhangende, ongerichte graaf G met gewichten op de takken een minimale opspannende boom, uitgaande van een startknoop s. Het algoritme wordt beschreven door de volgende pseudo-code, waarbij V de verzameling knopen van G is:

for $v \in V$ **do**
    tak$[v] := \infty$;
od
tak$[s] := 0$;
$U := \emptyset$;
while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met tak$[v^*]$ minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if gewicht$(v^*, v) <$ tak$[v]$ **then**
            tak$[v] :=$ gewicht(v^*, v);
            nieuwe kandidaattak voor v: (v^*, v)
        fi
    od
od

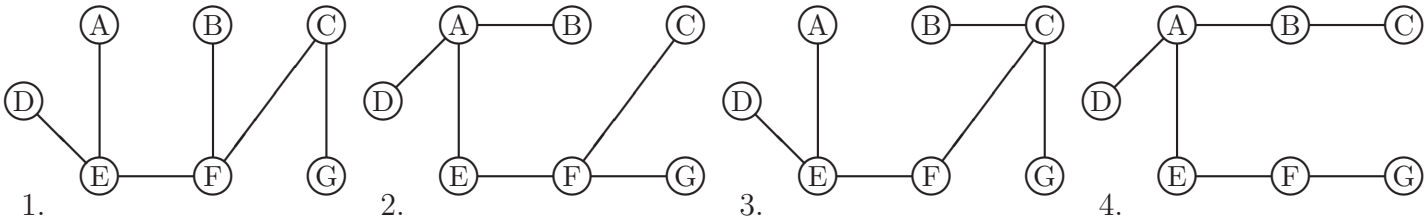
Pas (op kladpapier, dus niet inleveren) het algoritme van Prim toe op onderstaande graaf, beginnend in knoop A.



(a) In welke volgorde kunnen de knopen in de loop van het algoritme van Prim (bij bovenstaande pseudocode) aan de minimale opspannende boom worden toegevoegd? Nul of meer van de volgende volgordes kunnen goed zijn. Geef de nummers van alle goede volgordes.

1. A, E, D, B, F, G, C
2. A, E, D, B, F, C, G
3. A, E, D, F, G, C, B
4. A, E, D, F, C, G, B

(b) Hoe kan de resulterende minimale opspannende boom (bij bovenstaande pseudocode) eruit zien? Nul of meer van de volgende bomen kunnen goed zijn. Geef de nummers van alle goede bomen.



1. 2. 3. 4.