Tentamen Algoritmiek
Donderdag 9 juni 2022, 14.15 – 17.15 uur

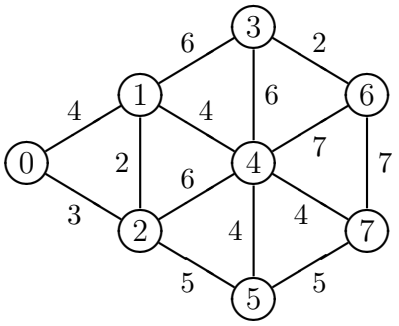
Wanneer er in een opgave gevraagd wordt om uitleg, toelichting of motivatie van je antwoord, is het belangrijk om die ook te geven.
De aantallen punten die bij het begin van elke opgave vermeld worden, zijn indicatief. Ze kunnen dus nog iets wijzigen.
Veel succes!

---

1. [32 pt] Bij het handelsreizigersprobleem is de bedoeling om bij een graaf met gewichten op de takken een Hamiltonkring met minimaal totaal gewicht te vinden. Een Hamiltonkring is een pad in de graaf dat begint en eindigt in dezelfde knoop, en alle andere knopen precies één keer bevat.

   Tijdens de colleges gingen we er bij het handelsreizigersprobleem altijd vanuit dat de graaf ongericht en compleet (alle knopen met alle andere knopen verbonden) was. Die aanname laten we nu los.

   We kunnen een Hamiltonkring in een graaf met n knopen weergeven met een rijtje van $n + 1$ elementen: de achtereenvolgende knopen in de kring. Beschouw bijvoorbeeld de volgende (ongerichte) graaf:



   De rijtjes $[0, 1, 3, 6, 7, 5, 4, 2, 0]$ en $[3, 4, 6, 7, 5, 2, 0, 1, 3]$ corresponderen bijvoorbeeld met een Hamiltonkring. De rijtjes $[0, 1, 2, 3, 4, 5, 6, 7, 0]$, $[0, 1, 3, 4, 7, 5, 4, 2, 0]$ en $[0, 1, 2, 5, 4, 3, 6, 7, 5]$ corresponderen om verschillende redenen niet met een Hamiltonkring.

   We nemen in deze opgave aan: dat graaf G n knopen heeft, genummerd $0, 1, 2, \ldots, n-1$, dat alle gewichten van de takken positief (> 0) zijn, en dat de graaf gerepresenteerd wordt met een globale adjacency matrix `adj`, waarbij `adj`$[i][j]$ het gewicht van de tak van i naar j voorstelt. **Een niet-bestaande tak is herkenbaar aan** `adj`$[i][j] = 0$**.**

   (a) Geef een C++-functie `int kostenHamiltonkring (int n, int route[])` die controleert of het rijtje `route[0], route[1], ..., route[n]` een Hamiltonkring voorstelt voor graaf G. Zo ja, dan wordt het totale gewicht van de Hamiltonkring berekend en geretourneerd. Zo nee, dan wordt -1 geretourneerd. Je mag ervan uitgaan dat array `route` inderdaad $n + 1$ getallen uit $\{0, 1, 2, \ldots, n-1\}$ bevat.

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

   Je functie mag gebruikmaken van losse variabelen (integers, booleans, ...), of arrays met zulke waarden. Om eenvoudig de tijdcomplexiteit te kunnen bepalen, zijn 'ingewikkelder' datastructuren, zoals `vector` en `set` niet toegestaan.

(b) Geef een recursieve C++-functie `void bepaalRoute (int n, int route[], int pos, int besteRoute[], int &best)` die met behulp van *exhaustive search* **alle** rijtjes van $n+1$ getallen uit $\{0, 1, 2, \ldots, n-1\}$ in array `route` construeert, waarbij we alleen eisen dat `route[0]` en `route[n]` allebei gelijk zijn aan 0. Als zo'n rijtje compleet is, wordt met behulp van functie `kostenHamiltonkring` gecontroleerd of het rijtje een Hamiltonkring voorstelt voor graaf G, en zo ja wat het totale gewicht daarvan is. Voor de goede orde: **alle** rijtjes betekent dat bijvoorbeeld ook het rijtje met $n+1$ keer een 0 wordt geconstrueerd.

De parameter `pos` betekent dat we al een deelroute hebben geconstrueerd in `route[0]`, ..., `route[pos-1]`, en dat we in deze recursieve aanroep waardes voor `route[pos]` gaan proberen. In parameter `besteRoute` wordt de beste complete route die we tot nu toe zijn tegengekomen, bijgehouden. De parameter `best` bevat het totale gewicht van deze route.

De eerste aanroep zal van de vorm `bepaalRoute (n, route, 0, besteRoute, best);` zijn, waarbij `best` is geïnitialiseerd op `INT_MAX`. Deze aanroep zal dus een oplossing van het handelsreizigersprobleem voor G opleveren (als die bestaat).

(c) Wat is de tijdcomplexiteit van je functie `bepaalRoute` uit onderdeel (b)? Motiveer je antwoord.

(d) Beschrijf, (in woorden of in pseudo-code, maar in ieder geval duidelijk en volledig) een *gretig* algoritme om een Hamiltonkring met minimaal totaal gewicht te bepalen.

Pas je algoritme toe op de voorbeeldgraaf aan het begin van deze opgave. Levert het algoritme voor deze graaf een geldige Hamiltonkring?

---

2. [23 pt] Gegeven een array $A[0], A[1], \ldots, A[n-1]$ dat $n \geq 1$ integers bevat. In deze opgave gaan we drie keer in A op zoek naar een specifiek getal x. Als x voorkomt in (een deelarray van) A, moet een positie i in het (deel-)array geretourneerd worden waarvoor $A[i]$ gelijk is aan x. Als x niet voorkomt in het (deel-)array, moet -1 geretourneerd worden.

(a) Schrijf een eenvoudige, niet-recursieve C++-functie `int vindPositie1 (int A[], int n, int x)`, die met behulp van *brute force* het getal x in array A zoekt.

(b) Neem aan dat n een 2-macht is. Schrijf een recursieve C++-functie `int vindPositie2 (int A[], int links, int rechts, int x)`, die met behulp van *divide-and-conquer* het getal x zoekt in het deelarray A[`links`], ..., A[`rechts`], ter lengte een 2-macht. Bij het 'verdelen' van *divide-and-conquer* moet het deelarray steeds in twee even grote delen worden verdeeld.

De eerste aanroep zal dus van de vorm `vindPositie2 (A, 0, n-1, x);` zijn.

(c) Neem nu aan de getallen in array A gesorteerd zijn van klein naar groot. De dimensie n is niet per se meer een 2-macht. Schrijf een niet-recursieve C++-functie `int vindPositie3 (int A[], int n, int x)`, die met behulp van binair zoeken het getal x in array A zoekt. Bij binair zoeken wordt x herhaaldelijk met het 'middelste' getal in het huidige deelarray vergeleken, waarna, zo nodig, verder gezocht wordt met de getallen links of rechts van dit 'middelste' getal.

(d) Is binair zoeken een voorbeeld van *divide-and-conquer*, van *decrease-by-a-constant-and-conquer*, van *decrease-by-a-constant-factor-and-conquer* of van *variable-size-decrease-and-conquer*? Motiveer je antwoord.

---

3. [28 pt] In deze opgave hebben we twee strings A en B, van respectievelijk lengte m en n, en we zoeken een langste gemeenschappelijke substring van A en B. De letters van de substring hoeven niet opeenvolgend in A en in B voor te komen, als de volgorde van

de letters maar overeenkomt. Als bijvoorbeeld A gelijk is aan "algoritmisch" en B gelijk is aan "logistiek". is een langste gemeenschappelijke substring "loiti", van lengte 5. Een andere langste gemeenschappelijke substring bij dit voorbeeld is "lgiti".

Merk op dat een (voorkomen van een) letter van A maar gematched kan worden met één (voorkomen van een) letter van B en vice versa. De langste gemeenschappelijke substring van "zes" en "zee" is dus gewoon "ze", van lengte 2.

We gaan de lengte van een langste gemeenschappelijke substring van A en B bepalen met behulp van dynamisch programmeren. Daartoe definiëren we $L(i, j)$ als de lengte van een langste gemeenschappelijke substring van de eerste i letters van A (dus $A[0 \ldots i-1]$) en de eerste j letters van B ($B[0 \ldots j-1]$). Uiteindelijk willen we natuurlijk $L(m, n)$ weten. Bij de strings "algoritmisch" en "logistiek" is deze waarde dus 5.

(a) Beredeneer dat $L(i, j)$ voldoet aan de volgende recurrente betrekking:

$$L(i,j) = \begin{cases} 0 & \text{als } i = 0 \text{ en/of } j = 0 \\ 1 + L(i-1, j-1) & \text{als } 1 \le i \le m \text{ en } 1 \le j \le n \text{ en } A[i-1] = B[j-1] \\ \max\{L(i-1,j), L(i,j-1)\} & \text{als } 1 \le i \le m \text{ en } 1 \le j \le n \text{ en } A[i-1] \neq B[j-1] \end{cases}$$

(b) Bepaal, met behulp van de recurrente betrekking hierboven, voor strings $A =$"lift" en $B =$"fiets" alle waardes $L(i, j)$. Geef je antwoord in een tabel van de volgende vorm:

$L(i,j)$			f	i	e	t	s
		$j=0$	1	2	3	4	5
	$i=0$	·	·	·	·	·	·
l	1	·	·	·	·	·	·
i	2	·	·	·	·	·	·
f	3	·	·	·	·	·	·
t	4	·	·	·	·	·	·

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

Geef een niet-recursieve C++-functie `int overlap (string A, int m, string B, int n)` die met behulp van bottom-up dynamisch programmeren, gebruikmakend van de recurrente betrekking hierboven, de waarde $L[m][n]$ berekent (en retourneert).

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

---

4. [17 pt] Het algoritme van Prim bepaalt voor een samenhangende, ongerichte, graaf G met gewichten op de takken een minimale opspannende boom.

(a) Wat verstaan we onder een minimale opspannende boom? Leg in het bijzonder uit wat we *minimaliseren* en wat een *opspannende boom* van G is.

Als je het antwoord op dit onderdeel niet weet, dan kun je het 'kopen' van de docent. Wellicht heb je er iets aan voor onderdeel (b).

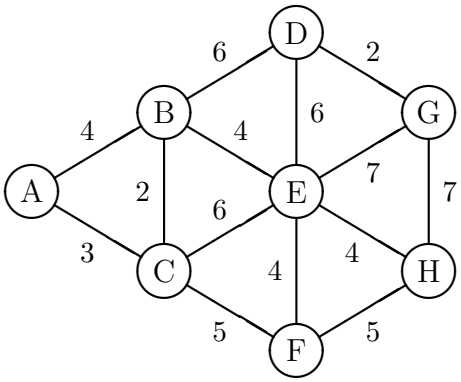
(b) Laat nu V de verzameling knopen van G zijn. Het algoritme van Prim begint vanuit een gegeven knoop s, en wordt beschreven door de volgende pseudo-code:

for $v \in V$ **do**
    tak$[v] := \infty$;
od

```
tak[s] := 0;
U := ∅;
E_T := ∅;
while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met tak[v*] minimaal;
        U := U ∪ {v*};
        voeg bijbehorende tak naar v* toe aan E_T (als v* ≠ s)
        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
```

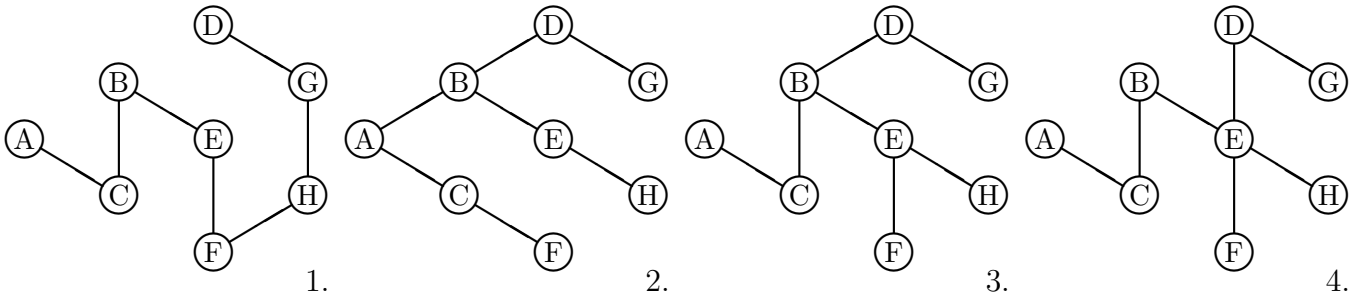
Hierbij is gewicht(v^*, v) (jawel) het gewicht van de tak tussen knopen v^* en v. E_T bevat aan het eind de takken van de boom.

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



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

ii. 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.