Tentamen Algoritmiek
Woensdag 21 mei 2025, 13.15 – 16.15 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. [33 pt] Een *doorloper* is een puzzel waarbij je woorden moet zien te zoeken in een $m \times n$ rechthoek met letters. Een woord mag in principe in elk vakje van de rechthoek beginnen, en vanuit die positie in een van de acht mogelijke richtingen (horizontaal, verticaal of diagonaal) te lezen zijn. In de figuur hieronder links zijn bijvoorbeeld de woorden "SINT" en "PIET" te vinden (vet gedrukt).

	0	1	2	3	4	5
0	R	**T**	S	E	E	I
1	**P**	O	**N**	Y	F	Z
2	**I**	O	B	**I**	R	U
3	**E**	B	M	O	**S**	I
4	**T**	T	U	O	T	S

	0	1	2	3	4	5
0	R	**T**	S	E	E	I
1	P	**O**	N	Y	F	Z
2	I	**O**	B	I	R	U
3	E	**B**	**M**	**O**	**S**	I
4	T	T	U	**O**	**T**	S

In deze opgave gaan we kijken naar een variant van deze puzzel, waarin de woorden een slang vormen: opeenvolgende letters van het woord moeten in horizontaal of verticaal **(niet diagonaal)** aangrenzende vakjes liggen. **Ze mogen daarbij geen vakjes dubbel gebruiken.** In de figuur hierboven rechts is bijvoorbeeld een slang voor het woord "STOOMBOOT" aangegeven (vet gedrukt, en met lijntjes tussen de opeenvolgende letters).

Formeel is een slang voor een woord een reeks van verschillende vakjes (i, j), die steeds horizontaal of verticaal bij elkaar aansluiten, zodat de letters in de vakjes (in de gegeven volgorde) het woord vormen. Bij een vakje (i, j) is i dan de rij en j de kolom. Rijen zijn van boven naar beneden genummerd als $0, 1, 2, \ldots, m-1$, kolommen van links naar rechts als $0, 1, 2, \ldots, n-1$. De slang voor het woord "STOOMBOOT" in het voorbeeld hierboven is dus $(3, 4), (4, 4), (4, 3), (3, 3), (3, 2), (3, 1), (2, 1), (1, 1), (0, 1)$.

Een woord kan verschillende slangen in dezelfde puzzel hebben. Twee slangen zijn verschillend, als ten minste één vakje in de ene slang afwijkt van het corresponderende vakje in de andere slang. In het bijzonder dus ook, als ze dezelfde vakjes bevatten, maar in een andere volgorde.

In deze opgave gaan we twee functies schrijven om alle slangen op te bouwen en te tellen. We maken gebruik van een `pair` in C++ om een vakje (i, j) op te slaan. Voor wie deze datastructuur niet paraat heeft: je kunt eenvoudig de onderdelen van een pair `vakje` benaderen met `vakje.first` en `vakje.second`. Je kunt een nieuw pair maken met b.v. een toekenning `vakje = make_pair(i,j);`. In het algemeen kun je ook eenvoudig pairs toekennen en vergelijken, met respectievelijk `=` en `==`.

(a) Hoeveel verschillende slangen voor het woord "STOOMBOOT" kent bovenstaande puzzel? Het is voldoende om het aantal te geven.

(b) Schrijf een niet-recursieve C++-functie `bool checkSlang (char A[MaxN][MaxN], int m, int n, pair<int,int> sl[], string str)`, die controleert of het array met vakjes `sl` daadwerkelijk een slang is voor het woord `str`, in de puzzel die is opgeslagen in 2-dimensionaal array `A`, waarbij parameters `m` en `n` de daadwerkelijke aantallen rijen en kolommen zijn. Zo ja, dan wordt `true` geretourneerd, en anders `false`.

Je mag ervanuit gaan het aantal vakjes in `sl` gelijk is aan `str.length()`, dat het geldige vakjes in de puzzel zijn, en dat de vakjes steeds horizontaal of verticaal bij elkaar aansluiten.

Je mag er ook vanuit gaan dat `str.length()` hoogstens `m·n` is. Probeer ervoor te zorgen dat de worst case tijdcomplexiteit van je functie in $O(m \cdot n)$ is. Als dit niet lukt, kun je nog wel het grootste deel van de punten verdienen.

(c) Schrijf een recursieve C++-functie `int aantalSlangen (char A[MaxN][MaxN], int m, int n, pair<int,int> sl[], string str, int pos)` die met behulp van **exhaustive search** het aantal slangen voor het woord `str` in de puzzel `A` bepaalt. De functie moet alle reeksen van `str.length()` aansluitende vakjes in de puzzel genereren, en dan van elke reeks bepalen of het een slang is met behulp van functie `checkSlang`.

De slang wordt stap voor stap opgebouwd in array `sl`. Je mag ervanuit gaan dat dit array groot genoeg is voor `str.length()` vakjes, en dat dit aantal vakjes minstens 1 is (het woord is niet leeg). Bij binnenkomst in een aanroep van de functie, zijn de posities 0,1,...,`pos` − 1 van array `sl` al gevuld. In deze aanroep moet `sl[pos]` zelf dus alle mogelijke waardes/vakjes krijgen. Het aantal slangen dat vanuit deze deeloplossing gegenereerd kan worden, moet geretourneerd worden. De eerste aanroep zal van de vorm `aantalSlangen (A, m, n, sl, str, 0)` zijn. Bedenk dat de slang op elk vakje van de puzzel mag beginnen.

(d) Leg uit in woorden (maar wel precies) hoe je functie `aantalSlangen` uit het vorige onderdeel zou veranderen, als ze geen gebruik zou maken van exhaustive search, maar van **backtracking**.

---

2. [27 pt] Mergesort is een sorteeralgoritme gebaseerd op het principe van verdeel-en-heers (**divide-and-conquer**). Neem aan dat we een array $A[0], A[1], \ldots, A[N-1]$ met $N \geq 1$ getallen van klein naar groot willen sorteren.

   Het idee bij mergesort is dat we eerst recursief de voorste $N/2$ getallen sorteren, dan recursief de achterste $N/2$ getallen sorteren, en deze twee gesorteerde deelrijen dan in elkaar schuiven met de functie `merge`. Als N oneven is, dan wordt $N/2$ uiteraard omhoog of omlaag afgerond.

   (a) Pas mergesort toe op de rij getallen $3, 6, 5, 1, 2, 4, 8, 7$, met $N = 8$.

   Geef voor elke deelrij die onstaat aan, welke deelrijtjes daar weer uit ontstaan. En voor elke twee gesorteerde deelrijtjes wat de gecombineerde deelrij wordt.

   (b) Schrijf een niet-recursieve C++-functie `void merge (int B[], int p, int C[], int q, int A[])` die de twee arrays B (met elementen $B[0], B[1], \ldots, B[p-1]$) en C (met elementen $C[0], C[1], \ldots, C[q-1]$) samenvoegt tot een gesorteerd array A (als $A[0], A[1], \ldots, A[p+q-1]$). Je mag ervanuit gaan dat de arrays B en C al gesorteerd zijn.

   Probeer ervoor te zorgen dat de tijdcomplexiteit van je functie lineair is in het aantal elementen $p + q$. Als dit niet lukt, kun je nog wel het grootste deel van de punten verdienen.

   (c) Schrijf een recursieve C++-functie `void sort (int A[], int N)` die het array $A[0]$, $A[1], \ldots, A[N-1]$ sorteert met behulp van mergesort, gebruikmakend van de functie `merge` uit onderdeel (b).

   (d) Wat is de tijdcomplexiteit van mergesort?

   Motiveer je antwoord door uit te leggen hoe vaak alle getallen in totaal bekeken worden door de hulpfunctie `merge`. Het is niet nodig om een recurrente betrekking te gebruiken, zoals we wel op college gedaan hebben bij dit algoritme.

---

3. [24 pt] We hebben op tafel een rij van n speelkaarten genummerd $i = 0, 1, 2, \ldots, n-1$. Elke kaart heeft zowel aan de voorkant als aan de achterkant een getal staan. Het getal aan de voorkant van kaart i noemen we $a[i][0]$, het getal aan de achterkant van die kaart noemen we $a[i][1]$. Een voorbeeld van een rij van $n = 10$ kaarten is:

	$i=$ 0	1	2	3	4	5	6	7	8	9
voorkant: $j = 0$	3	4	5	6	5	5	9	9	2	8
achterkant: $j = 1$	1	1	9	2	6	8	7	3	3	4

We willen nu de n kaarten zó draaien (elk met de voorkant of de achterkant boven), dat een zo lang mogelijke aaneengesloten deelrij kaarten ontstaat, waarvan de getallen, van links naar rechts lezend, oplopen of gelijk blijven. In het voorbeeld hierboven kunnen we met kaarten 0 t/m 7 de rij getallen 3,4,5,6,6,8,9,9 krijgen door kaarten 4 en 5 om te draaien. Dit is de langst mogelijke deelrij in ons voorbeeld. Toevallig begint deze deelrij met kaart 0, maar dat is in het algemeen niet per se zo. Een andere oplopende (althans niet dalende) deelrij is 2,5,5,7, die we kunnen vormen met kaarten 3 t/m 6.

Om de langst mogelijke aaneengesloten deelrij kaarten te vinden, definiëren we de grootheid reeks(i, j) als volgt:

> reeks(i, j) is de lengte van de langst mogelijke aaneengesloten deelrij kaarten, eindigend op kaart i, waarvan de getallen van links naar rechts lezend, oplopen of gelijk blijven. Van kaart i eisen we dat getal $a[i][j]$ zichtbaar is (dus als $j = 0$ ligt de voorkant boven, anders de achtekant). De kaarten vóór kaart i mogen we zo gunstig mogelijk draaien.

In ons voorbeeld is reeks$(7, 0) = 8$ (we eindigen de reeks dus met de voorkant van kaart 7) en reeks$(6, 1) = 4$ (we eindigen de reeks met de achterkant van kaart 6).

(a) Geef de complete tabel met 10×2 waarden reeks(i, j) bij ons voorbeeld.

(b) Beredeneer dat reeks(i, j) voldoet aan de volgende recurrente betrekking:

$$\text{reeks}(i,j) = \begin{cases} 1 & \text{als } i = 0 \\ 1 & \text{als } i \geq 1 \text{ en } a[i-1][0] > a[i][j] \text{ en } a[i-1][1] > a[i][j] \\ 1 + \text{reeks}(i-1, 0) & \text{als } i \geq 1 \text{ en } a[i-1][0] \leq a[i][j] \text{ en } a[i-1][1] > a[i][j] \\ 1 + \text{reeks}(i-1, 1) & \text{als } i \geq 1 \text{ en } a[i-1][0] > a[i][j] \text{ en } a[i-1][1] \leq a[i][j] \\ 1 + \max\{\text{reeks}(i-1, 0), \text{reeks}(i-1, 1)\} & \\ & \text{als } i \geq 1 \text{ en } a[i-1][0] \leq a[i][j] \text{ en } a[i-1][1] \leq a[i][j] \end{cases}$$

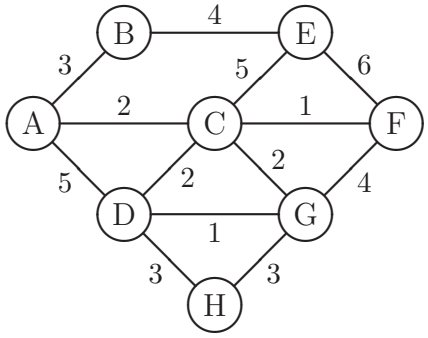
(c) Schrijf een niet-recursieve functie `int langsteRij (int a[MaxN][2], int n)`, die gebruikmakend van **bottom-up dynamisch programmeren** en de hierboven beschreven recurrente betrekking, de lengte bepaalt van de langste aaneengesloten deelrij kaarten, waarvan de getallen van links naar rechts lezend oplopen of gelijk blijven, bij een zo gunstige mogelijke draaiïng van de kaarten. Deze lengte wordt vervolgens geretourneerd.

---

4. [16 pt] Het algoritme van Dijkstra bepaalt voor samenhangende, gewogen grafen G met knooppuntenverzameling V de (lengtes van) kortste paden vanuit een gegeven knoop s naar alle andere knopen. Het algoritme wordt beschreven door de volgende pseudo-code:

```
for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;
while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
              if pad[v*]+ gewicht(v*, v) < pad[v] then
                    pad[v] := pad[v*]+ gewicht(v*, v);
                    nieuwe kandidaattak voor v: (v*, v)
              fi
        od
od
```

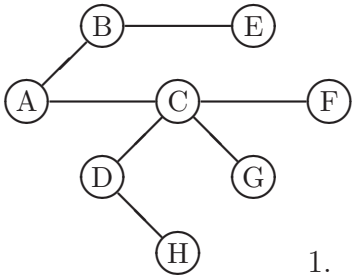
Pas (op kladpapier, dus niet inleveren) het algoritme van Dijkstra toe op onderstaande graaf, beginnend in knoop A.



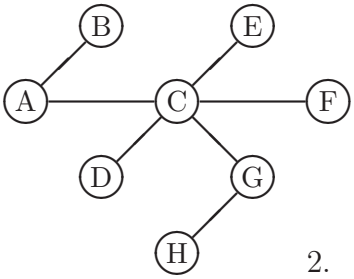
(a) In welke volgorde kunnen de knopen in de loop van het algoritme van Dijkstra (bij bovenstaande pseudocode) aan de boom van kortste paden worden toegevoegd? Nul of meer van de volgende volgordes kunnen goed zijn. Geef de nummers van alle goede volgordes.

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

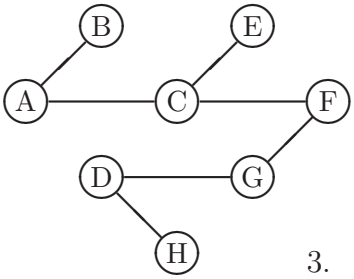
(b) Hoe kan de resulterende boom van kortste paden (bij bovenstaande pseudocode) eruit zien? Nul of meer van de volgende bomen kunnen goed zijn. Geef de nummers van alle goede bomen.



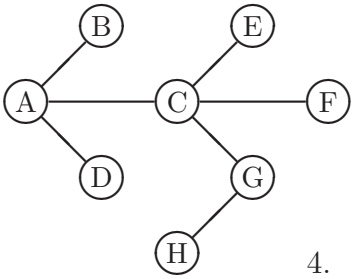
1.



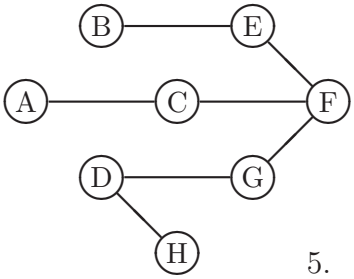
2.



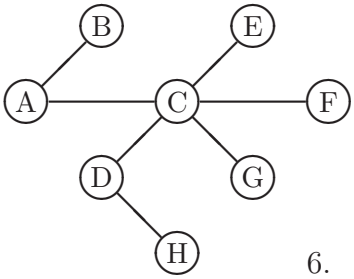
3.



4.



5.



6.