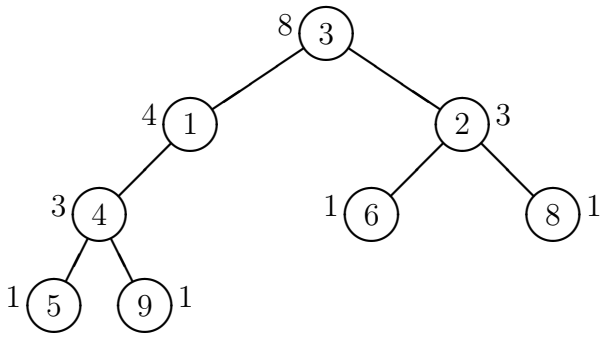
Tentamen Algoritmiek
Dinsdag 13 juli 2021, 13.00 – 16.00 uur

Waneer 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.

---

1. [25 pt] Deze opgave gaat over binaire bomen, bestaande uit knopen uit de klasse `knoop` die er als volgt uitziet:

```
class knoop
{ public:
    knoop* links;
    knoop* rechts;
    int info;
    int gewicht;
}; // knoop
```



In de boom hierboven zijn de getallen *in* de knopen de velden `info`. De getallen naast de knopen zijn de velden `gewicht`. Dit laatste veld is bedoeld voor het gewicht van een knoop: het aantal knopen in de subboom met die knoop als wortel. Zo heeft elk blad gewicht 1, want de subboom met een blad als wortel bevat alleen dat blad. Interne knopen hebben een hoger gewicht.

Bij aanvang hebben de velden `gewicht` in de boom overigens **nog geen zinvolle waarde**. Ze gaan gevuld worden bij onderdeel (b).

(a) Schrijf een *recursieve* C++-functie `void wandel (knoop *w)` die de velden `info` in de binaire boom (of subboom) met wortel `w` in preorde volgorde op het scherm afdrukt. Voor de boom hierboven betekent dat de volgorde 3, 1, 4, 5, 9, 2, 6, 8.

(b) Schrijf een *recursieve* C++-functie `void vulGewicht (knoop *w)` die in elke knoop in de (sub)boom met wortel `w` het veld `gewicht` vult met de juiste waarde.

(c) Bij dit onderdeel mag je ervan uitgaan dat de binaire boom niet leeg is, en dat van alle knopen in de boom de velden `gewicht` de juiste waardes hebben. We willen deze velden gebruiken om te bepalen waar we een nieuwe knoop gaan toevoegen: als we een knoop toevoegen aan de boom, dan wordt hij als blad toegevoegd aan de subboom met het kleinste aantal knopen. Binnen die subboom wordt hij toegevoegd aan de subsubboom met het kleinste aantal knopen, enzovoort.

Als beide subbomen van een knoop evenveel knopen bevatten, kiezen we voor de linker subboom om de nieuwe knoop aan toe te voegen. In het voorbeeld hierboven zal een nieuwe knoop worden toegevoegd als linker kind van de knoop met infoveld 6.

Schrijf een *niet-recursieve* C++-functie `void voegToe (knoop *w, int getal)`, die een nieuwe knoop met infoveld `getal` toevoegt aan een niet-lege binaire boom met wortel `w`, op de hierboven beschreven positie.

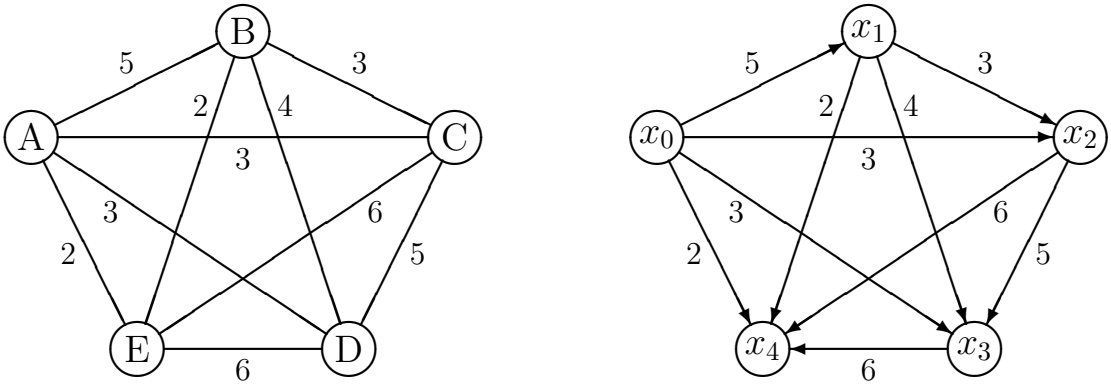
Na afloop van de functie moeten de velden `gewicht` in de boom kloppen met de nieuwe situatie.

---

2. [35 pt] Deze opgave gaat over grafen met gewichten op de takken, waarbij alle gewichten **positief en geheel** zijn. Bij het algoritme van Dijkstra zoeken we kortste paden in een graaf. In deze opgave gaan we op zoek naar het **langste**, enkelvoudige pad.[1]

   (a) Stel dat onze graaf G ongericht is, n knopen bevat, en compleet is (alle takken zijn aanwezig), zoals de linker graaf hieronder. Beschrijf in woorden (enkele zinnen is voldoende) een exhaustive search algoritme om het langste enkelvoudige pad in G te bepalen.

   (b) Wat is de tijdcomplexiteit van dit exhaustive search algoritme? Motiveer je antwoord.



In de rest van deze opgave gaan we ervanuit dat de graaf gericht is en acyclisch. Het is dus een *directed acyclic graph* (DAG).

We kijken bij de onderdelen (c) en (d) eerst naar een speciaal geval: een DAG G, met $n \geq 1$ knopen $x_0, x_1, \ldots, x_{n-1}$, zódat er van **elke** knoop x_i naar **elke** knoop x_j met $i < j$ een tak is. Omdat alle takken er zijn, noemen we dit een complete DAG. De rechter graaf hierboven is hiervan een voorbeeld. We gebruiken $d[i][j]$ om het gewicht op de tak van x_i naar x_j aan te duiden.

Laat lengte(j) de lengte van het langste pad van knoop x_0 naar knoop x_j zijn.

   (c) i. Beredeneer dat lengte(j) voldoet aan de volgende recurrente betrekking:

$$\begin{aligned} \text{lengte}(0) &= 0 \\ \text{lengte}(j) &= \max_{0 \leq i < j} \{\ \text{lengte}(i) + d[i][j]\ \} \qquad \text{als } 0 < j < n \end{aligned}$$

   ii. Beredeneer ook dat de lengte van het langste pad in de graaf (gemaximaliseerd over alle mogelijke beginknopen en eindknopen) gelijk is aan lengte($n-1$), dat wil zeggen aan de lengte van het langste pad van knoop x_0 naar knoop x_{n-1}.

   (d) Bij dynamisch programmeren slaan we alle waardes lengte(i) op in een tabel (array) `lengte`. We noemen het dan `lengte[i]` (met blokhaken).

   Geef een niet-recursieve C++-functie `int langstePad (int n)` die met behulp van bottom-up dynamisch programmeren, gebruikmakend van de recurrente betrekking hierboven, de lengte van het langste pad berekent (en retourneert). Je mag er voor het gemak vanuitgaan dat het array d met de gewichten een globale variabele is.

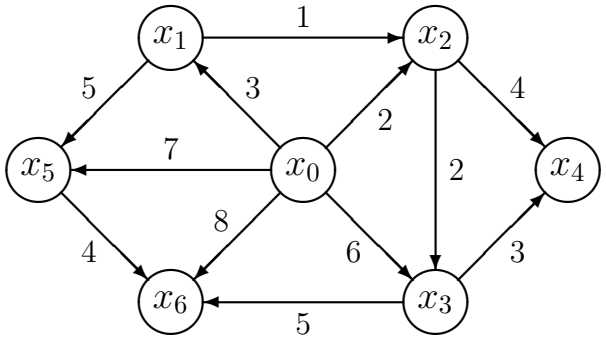
[1] Een enkelvoudig pad is een pad waarin elke knoop maximaal één keer voorkomt.

We laten nu de aanname los dat de DAG G compleet is. Het is nog wel zo dat er alleen (gerichte) takken zijn van knopen x_i naar knopen x_j met $i < j$, maar het is niet per se meer zo dat er een tak is voor **elke** x_i en **elke** x_j met $i < j$.

(e) Leg uit (in woorden of code, maar in ieder geval duidelijk en volledig) hoe je functie `langstePad` uit onderdeel (d) aangepast moet worden om ook voor deze algemenere DAG G de lengte van het langste pad in G te berekenen.

Als je geen (goed) antwoord bij onderdeel (d) hebt, mag je ook uitleggen hoe de definitie van lengte(j), de recurrente betrekking uit onderdeel (c) en het doel (wat retourneer je uiteindelijk) aangepast moeten worden.

(f) Pas het aangepaste algoritme uit onderdeel (e) toe op onderstaande DAG G:

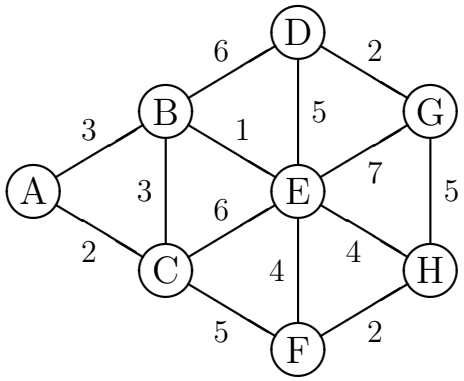


Laat voor iedere knoop x_i (in de juiste volgorde) zien hoe je de waarde lengte(i) berekent. Wat is de resulterende lengte van het langste pad in G?

N.B.: Ook als je geen antwoord hebt voor onderdeel (e), kun je dit onderdeel misschien toch maken.

---

3. [15 pt] Laat G_1 de volgende graaf zijn:



(a) Pas het algoritme van Kruskal toe om een minimale opspannende boom van G_1 te bepalen. Leg uit hoe je te werk gaat. Geef ook duidelijk aan, welke takken je in welke volgorde bekijkt en waarom je een tak wel of niet aan je oplossing toevoegt. Teken ook de resulterende boom.

Z.O.Z.

(b) Bij een implementatie van het algoritme van Kruskal kunnen we gebruik maken van *union-find*. Hiermee worden disjuncte deelverzamelingen van knopen bijgehouden. Wat zijn bij het voorbeeld hierboven de deelverzamelingen van knopen, nadat bij toepassing van het algoritme vijf takken aan de oplossing zijn toegevoegd? (Je hoeft niet de deelverzamelingen na nul, één, twee, drie en vier takken te geven.)

Representeer de deelverzamelingen als gerichte bomen, zoals we ook tijdens het college hebben gedaan. Als je niet weet wat er bedoeld wordt met deze specifieke representatie van de deelverzamelingen, dan kun je nog wel een deel van de punten verdienen met een gewone opsomming van de deelverzamelingen.

---

4. [25 pt] Bij het toewijzingsprobleem (*assignment problem*) hebben we n personen en n jobs. Elke persoon moet één job uitvoeren. 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.

   (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
Anna	9	6	7	2
Demi	3	9	7	9
Lorena	1	2	8	3
Marianne	8	4	8	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 is dan bruikbaarder voor het branch-and-bound algoritme bij dit voorbeeld? Motiveer je antwoord.
