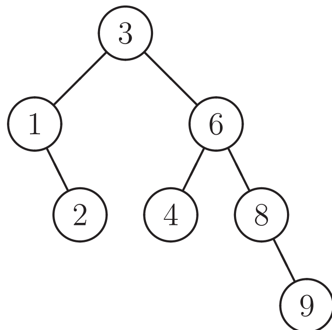
Hertentamen Algoritmiek
Maandag 8 juli 2024, 13.00 – 16.00 uur

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. [26 pt] Een binaire zoekboom is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom. Een eenvoudig voorbeeld van een binaire zoekboom is de boom hieronder.




```
class Knoop
{ public:
    Knoop () {}  // constructor
    Knoop* links;
    Knoop* rechts;
    int info;
}; // Knoop
```

In deze opgave gaan we er dus vanuit dat alle waarden in een binaire zoekboom verschillend zijn en moeten zijn. Bij de implementatie van een binaire zoekboom maken we gebruik van knopen die objecten zijn uit de klasse `Knoop` hierboven.

(a) Geef een *niet-recursieve* C++-functie `Knoop* zoek (Knoop* w, int getal)` die in een binaire zoekboom met wortel w controleert of parameter `getal` voorkomt als infowaarde. Zo ja, dan wordt een pointer naar de knoop met deze infowaarde geretourneerd. Zo nee, dan wordt nullptr geretourneerd. De boom verandert niet.

(b) Wat is de worst-case tijdcomplexiteit van je functie `zoek` uit het vorige onderdeel. Druk de tijdcomplexiteit uit als functie van n, het aantal knopen in de boom. Motiveer je antwoord door een worst case (een zo slecht mogelijke binaire zoekboom met n knopen, en een daarin te zoeken getal) te geven.

(c) De voorbeeldboom hierboven heeft hoogte 3. Er bestaat een binaire zoekboom met dezelfde getallen en hoogte 2. Geef deze binaire zoekboom.

(d) Laat A een oplopend gesorteerd array met $n \geq 1$ verschillende getallen zijn. Geef een efficiënte *recursieve* C++-functie `Knoop* maakBZBoom (int A[], int links, int rechts)` die de getallen `A[links],A[links+1],...,A[rechts]` in een nieuwe binaire zoekboom **met minimale hoogte** zet, en een pointer naar de wortel van die boom retourneert. De eerste aanroep zal van de vorm `maakBZBoom (A, 0, n-1)` zijn.

Hint: Maak gebruik van divide-and-conquer en verdeel het subarray met toe te voegen getallen steeds (ongeveer) in twee even grote delen.

---

2. [20 pt] Een *Latijns vierkant* van orde n is een n bij n vierkant (matrix) dat in elke rij en elke kolom de getallen 1 t/m n bevat. Er geldt dus dat in elke rij en in elke kolom elk van die getallen precies één keer voorkomt. De bedoeling is nu om alle Latijnse vierkanten van een bepaalde orde te genereren en te tellen. Zonder beperking der algemeenheid mogen we aannemen dat in de eerste rij (rij 0) en de eerste kolom (kolom 0) altijd de getallen 1 t/m n **in die volgorde** komen te staan. Onder deze aanname is het enige Latijnse vierkant voor orde $n = 3$ gelijk aan het volgende:

$$\begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix}$$

Een voor de hand liggende manier om een Latijns vierkant stap voor stap op te bouwen is om het vierkant rij voor rij, en per rij van links naar rechts te vullen. In elke stap wordt dan gecontroleerd of aan de restricties is voldaan.

(a) Geef alle Latijnse vierkanten voor orde $n = 4$ (opnieuw onder de aanname over de eerste rij en de eerste kolom). Dit zijn er vier.

(b) Geef nu een *recursieve* C++-functie `int latijnseVierkanten(int A[MaxN][MaxN], int n, int i, int j)` die alle Latijnse vierkanten van orde n met behulp van backtracking genereert en telt. De parameters i en j geven aan dat je nu het vakje met coördinaten (i, j) probeert te vullen, waarbij alle eerdere vakjes (in eerdere rijen en links in dezelfde rij) reeds goed gevuld zijn.

Een aanroep van de functie moet het aantal Latijnse vierkanten retourneren dat vanaf het huidige, deels ingevulde array A te maken is. Je moet alleen de Latijnse vierkanten genereren en tellen die aan de aanname over de eerste rij en de eerste kolom voldoen. De eerste aanroep van de functie zal van de vorm `latijnseVierkanten (A, n, 0, 0)` zijn. Op dat moment is het array A nog helemaal leeg. Desgewenst mag je van `latijnseVierkanten` een wrapperfunctie met een recursieve hulpfunctie maken.

Het is niet de bedoeling dat je functie gebruik maakt van een globale teller, of dat je een teller als parameter meegeeft aan een recursieve hulpfunctie. Doe je dat wel, dan verlies je 3 van de punten die je voor dit onderdeel kunt verdienen.

---

3. [24 pt] Bij het handelsreizigersprobleem is het de bedoeling om bij een graaf met minstens drie knopen en 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. We gaan er net als tijdens de colleges vanuit dat de graaf ongericht en compleet (alle knopen met alle andere knopen verbonden) is.

   We kunnen een Hamiltonkring in een graaf met n knopen weergeven met een rijtje van $n + 1$ elementen: de achtereenvolgende knopen in de kring, waarbij het laatste element gelijk is aan het eerste.

   We nemen in het vervolg van deze opgave aan dat onze graaf n knopen heeft, genummerd $0, 1, 2, \ldots, n-1$ en dat de graaf gerepresenteerd wordt met behulp van een adjacency list `Buur *graaf[n]`, waarbij de klasse Buur als volgt gedefinieerd is:

```
class Buur
{ public:
    int knoopnummer;
    int gewicht;
    Buur* volgende;
};  // Buur
```

   Een gretig algoritme voor het handelsreizigersprobleem werkt als volgt:

   - Begin in knoop 0.
   - Voeg vanuit de huidige knoop steeds de dichtstbijzijnde, nog-niet-bezochte knoop (via tak met laagst mogelijke gewicht dus) toe aan de route.
   - Als alle knopen in de route zitten, sluit dan de kring.

   (a) Geef een *niet-recursieve* C++-functie `void gretigTSP (int n, Buur *graaf[], int route[])` die dit gretige algoritme implementeert, en de gevonden route in parameter `route` teruggeeft, als achtereenvolgens `route[0],route[1],...,route[n]`. Merk op dat de functie niet expliciet de kosten van deze route teruggeeft.

   Probeer ervoor te zorgen dat je in constante tijd kunt controleren of een toe te voegen knoop nog niet bezocht is. Lukt dat niet, dan verlies je 2 van de punten die je voor dit onderdeel kunt verdienen.

   (b) Wat is de tijdcomplexiteit, als functie van n, van je functie `gretigTSP` uit het vorige onderdeel? Motiveer je antwoord door een basisoperatie aan te wijzen, en te bepalen hoe vaak die operatie wordt uitgevoerd.

   (c) Ons gretige algoritme levert niet voor elke complete, ongerichte graaf een optimale route. Geef een voorbeeld van een volledige, ongerichte graaf met maximaal vijf knopen en met gewichten op de takken waarbij het algoritme een niet-optimale route oplevert.

   Wat is voor deze graaf een optimale route? Welke route levert het gretige algoritme op?

---

4. [30 pt]

   (a) Leg uit hoe best-first branch-and-bound werkt voor maximalisatieproblemen in het algemeen. Geef daarbij o.a. aan hoe (deel)oplossingen gegenereerd worden, wat met branch bedoeld wordt en wat met bound, wat best-first betekent, wanneer gesnoeid wordt, enz.

Bij het knapzakprobleem hebben we een verzameling objecten i ($1 \leq i \leq n$), elk met een gewicht w_i en een waarde v_i. Verder hebben we een knapzak met een capaciteit W. In deze knapzak kunnen we objecten uit onze verzameling stoppen, zolang het totaalgewicht niet groter wordt dan W. Doel is om een deelverzameling van (verschillende) objecten in de knapzak te stoppen met een zo groot mogelijke totale waarde. Een voorbeeld van een knapzakprobleem met vier objecten is:

object i	w_i	v_i	v_i/w_i
1	6	54	9
2	4	32	8
3	8	56	7
4	5	20	4

W = 15

Tijdens het college hebben we een branch-and-bound algoritme voor het knapzakprobleem behandeld. Hierbij wordt gebruikt dat de objecten gesorteerd zijn op de gemiddelde-waarde-per-gewicht v_i/w_i, van groot naar klein. Iedere stap van het algoritme wordt een object wel of juist niet aan de knapzak toegevoegd, te beginnen bij object 1.

   (b) Wat is de hierbij gebruikte bovengrens bij een deeloplossing? Geef zowel de bovengrens bij aanvang (als we nog geen object behandeld hebben), als de bovengrens na stap i (als we net object i wel of niet gekozen hebben). In dit laatste geval nemen we aan dat we nog niet alle objecten behandeld hebben, ofwel $i \leq n - 1$.

   Geef antwoorden voor een algemeen knapzakprobleem met $n \geq 1$ objecten, dus niet specifiek voor het voorbeeld van de vier objecten hierboven. Je hoeft niet toe te lichten waarom je antwoorden daadwerkelijk bovengrenzen zijn.

   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.

   (c) Pas het branch-and-bound algoritme met de bovengrens uit het vorige onderdeel toe op het voorbeeld en teken de bijbehorende state-space-tree, met bij elke knoop (deeloplossing) de relevante informatie (waaronder ook de opbouw van de bovengrens). Geef ook aan in welke volgorde de knopen zijn aangemaakt, welke knopen gesnoeid worden en waarom.

   Wat is dus de optimale oplossing?