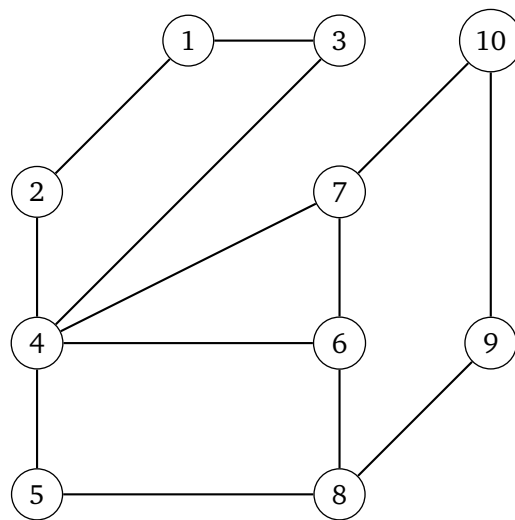


10

Grafen - Verdieping


Oefening 10.1

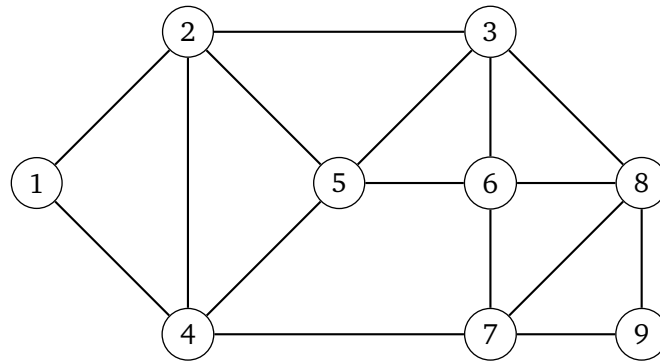
 Zoek in figuur 10.1 het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.



Figuur 10.1 Zoek het pad van knoop 1 naar 10


Oefening 10.2

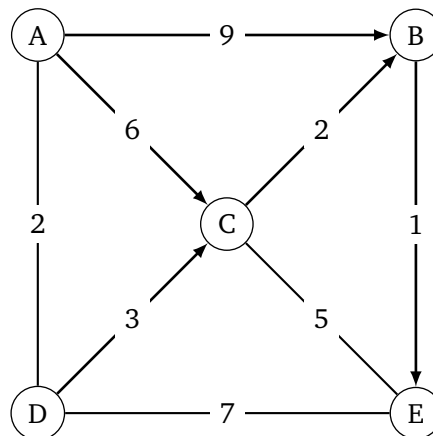
 Zoek in figuur 10.2 het pad van het eerste naar het laatste knooppunt dat zo weinig mogelijk knooppunten telt door gebruik te maken van het breadth-first algoritme.



Figuur 10.2 Zoek het pad van knoop 1 naar 9

Oefening 10.3

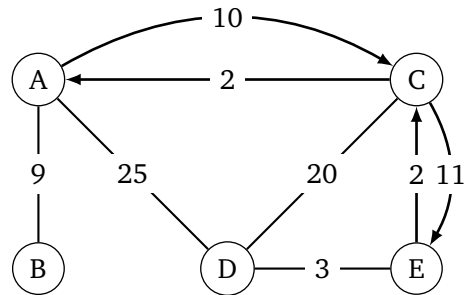
 Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 10.3 te berekenen.



Figuur 10.3 Netwerk bij oefening 10.3

Oefening 10.4

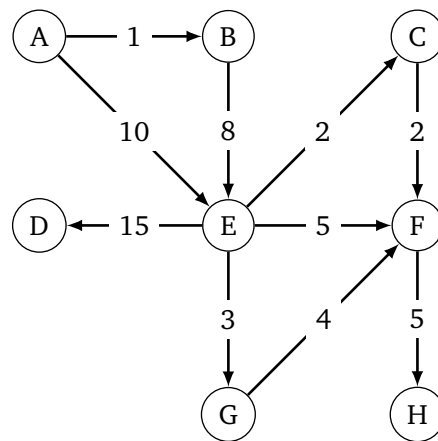
 Gebruik de methode van Floyd om het kortste pad tussen de verschillende knooppunten van het netwerk van figuur 10.4 te berekenen.



Figuur 10.4 Netwerk bij oefening 10.4

Oefening 10.5

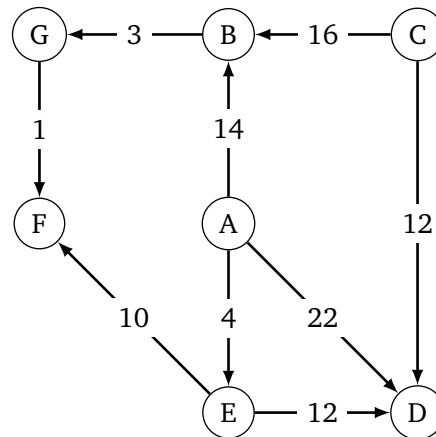
Gebruik de methode van Dijkstra om het kortste pad te bepalen van knooppunt A naar de andere knooppunten van het netwerk in figuur 10.5. Geef ook aan over welke knooppunten dat pad loopt.



Figuur 10.5 Netwerk bij oefening 10.5

Oefening 10.6

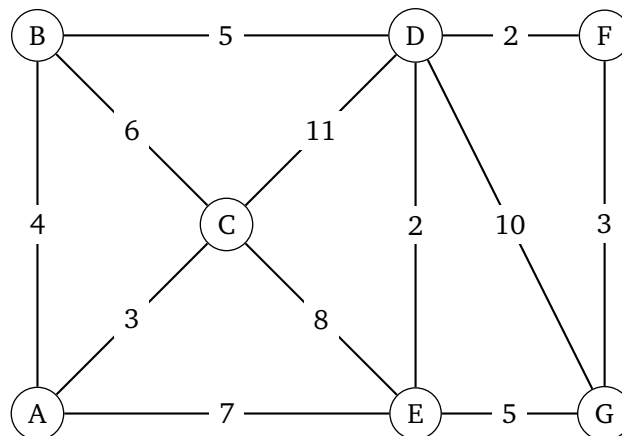
 Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 10.6 te berekenen.



Figuur 10.6 Netwerk bij oefening 10.6

Oefening 10.7

 Gebruik de methode van Dijkstra om een lijst te maken met de kortste afstanden van knooppunt A naar alle andere van het netwerk van figuur 10.7 te berekenen.



Figuur 10.7 Netwerk bij oefening 10.7

Oefening 10.8



(Examen augustus 2018) Een *universele vergeetput* in een gerichte of gemengde graaf met n knooppunten is een knooppunt dat het doel is van $n - 1$ pijlen en de bron van geen enkele pijl. Soms wordt dit ook een *beroemdheid* genoemd: iedereen in de ruimte herkent de beroemdheid, maar de beroemdheid herkent niemand.

Schrijf een methode die het nummer van het knooppunt dat een vergeetput is, teruggeeft. De graaf wordt voorgesteld door een verbindingsmatrix. Als de graaf geen vergeetput heeft, dan geeft de methode `null` terug.

Tip: teken een netwerk dat voldoet aan bovenstaande beschrijving. Stel de verbindingsmatrix hiervoor op. Ga op zoek naar de specifieke eigenschappen waaraan de verbindingsmatrix moet voldoen. Test je code met een driverklasse in `ui`.

Oefening 10.9



(Examen juni 2019) Je beschikt over een *ongewogen, gerichte* graaf met n knooppunten. Elk knooppunt bevat een getal van 1 tot en met n dat met een persoon overeen komt. Een verbinding van knoop i naar knoop j betekent dat persoon i persoon j als één van zijn beste vrienden beschouwt (niet noodzakelijk omgekeerd).

Persoon i ($i : 1 \dots n$) wint de lotto en verdient hiermee een bepaald bedrag. Hij houdt de helft zelf en verdeelt de andere helft gelijkmatig over zijn beste vrienden (behalve als die al iets van dit bedrag hebben verkregen van een andere persoon). Elke beste vriend zet dit proces verder totdat iedereen zijn deel gekregen heeft.

Schrijf een methode `verdeel(i: integer, bedrag: double)` die een array (met n elementen van het type `double`) teruggeeft. De elementen van deze array zijn het eindbedrag dat elke persoon uiteindelijk bezit als persoon i het gegeven bedrag heeft uitgedeeld en zijn vrienden het proces verder gezet hebben. Test je code via een `main` methode in `ui`.

Voor deze oefening zijn er vele juiste oplossingen mogelijk. Het hangt af van je implementatie en o.a. van de volgorde waarin je de mensen behandelt. Je mag altijd commentaar aan je code toevoegen als je wilt verduidelijken hoe je dit probleem aangepakt hebt.

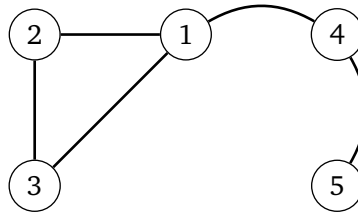
Tip: Maak voor jezelf eerst een kleine graaf die aan de beschrijving voldoet. Voer dit verdelen zelf eens uit op papier zodat je goed snapt hoe alles werkt. Pas dan probeer je een algoritme te bedenken.

Oefening 10.10



(Examen augustus 2019) Een *brug* in een niet-gerichte graaf is een verbinding tussen twee punten die, als ze wegvalt als gevolg heeft dat de graaf niet meer verbonden is. Figuur 10.8 toont een eenvoudig voorbeeld van een niet-gerichte graaf met twee bruggen (de gebogen

lijnen). Verbinding 1–4 is een brug, want als je die weglaat zijn er twee subgrafen (1, 2, 3) en (4, 5) waar je onmogelijk van het ene stuk in het andere kan geraken. Analooog is de verbinding 4–5 ook een brug.



Figuur 10.8 Een niet-gerichte graaf

Deze programmeervraag valt uiteen in twee delen die elk op punten staan. Wie weet kan je wel deel (b) programmeren, maar niet deel (a)? Schrijf dan die code (ook al kan je niet testen, want je hebt natuurlijk wel de code van deel (a) nodig). Nog een belangrijke tip: je kan veel code recyclen van BFS. Maak zeker ook gebruik van het feit dat de graaf niet-gericht is, en dus dat de verbindingsmatrix ... is). Werk eerst bovenstaand voorbeeld even op papier uit vooraleer je begint te programmeren!

1. Schrijf een methode `isBrug(van: int, naar: int): boolean` die op een ongerichte graaf nakijkt of een verbinding tussen twee punten van en naar een brug vormt.
2. Schrijf een methode `aantalBruggen(): int` die het aantal bruggen telt in een gegeven ongerichte graaf, gebruik makend van de methode `isBrug`.

Oplossingen

Oplossing 10.1 $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 10$ of $1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 10$

Oplossing 10.2 $1 \rightarrow 4 \rightarrow 7 \rightarrow 9$

Oplossing 10.3 Enkele paden: $A \rightarrow D \rightarrow C \rightarrow B \rightarrow E$; $C \rightarrow B \rightarrow E \rightarrow D \rightarrow A$; $D \rightarrow C \rightarrow B \rightarrow E$

Oplossing 10.4

$$D^{(5)} = \begin{bmatrix} 0 & 9 & 10 & 24 & 21 \\ 9 & 0 & 19 & 33 & 30 \\ 2 & 11 & 0 & 14 & 11 \\ 7 & 16 & 5 & 0 & 3 \\ 4 & 13 & 2 & 3 & 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 1 & 5 & 3 \\ 0 & 1 & 0 & 5 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Enkele paden: $4 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 2$ met lengte 16; $3 \rightarrow 5 \rightarrow 4$ met lengte 14

Oplossing 10.8 Om deze oefening op te lossen bekijken we de `verbindingsMatrix`. Als er in de verbindingsmatrix een rij bestaat met allen maar nullen (`false`) en als overeenkomstige kolom alleen maar eentjes (`true`) bevat uitgezonderd waar op de plaats waar de rij-index gelijk is aan de kolom-index, dan hebben we te maken met een vergeetput.

Listing 2 `isVergeetput(verbindingsMatrix)` methode

```
public Integer isVergeetput(boolean[][] verbindingsMatrix){
    for (int i = 0; i < verbindingsMatrix.length; i++){
        boolean[] fromRow = verbindingsMatrix[i];
        boolean vergeetputFound = true;
        for(int j = 0; j < verbindingsMatrix.length; j++){
            if(fromRow[j] || (!verbindingsMatrix[j][i] && i!=j)){
                vergeetputFound = false;
                break;
            }
        }
        if(vergeetputFound){
            return i;
        }
    }
    return null;
}
```


Oplossing 10.9 Deze oefening is opgelost in 2 delen.

Het eerste deel maakt een vertaling van ons probleem waarbij 1 persoon en 1 bedrag gegeven zijn naar het zelfde probleem waarbij er n personen zijn en waar dat iedere persoon een ander bedrag kan hebben.

Vervolgens roepen we een helper functie op die dat de verdeling zal maken. Deze verdeling gebeurt van links naar rechts en per diepte. Deze methode zal zichzelf recursief oproepen waarbij we steeds verder kijken naar vrienden die verder weg zijn van de oorspronkelijke winnaar.

Listing 3 verdeel(i , bedrag) methode

```
public ArrayList<Double> verdeel(int i, double bedrag){
    ArrayList<Double> verdeelt = new ArrayList<>();
    ArrayList<Integer> delers = new ArrayList<>();
    for(int j =0; j<this.getAantalKnopen(); j++){
        if(i == j){
            verdeelt.add(bedrag);
            delers.add(i);
        }
        else{
            verdeelt.add(0.0);
        }
    }
    return verdeelHelper(verdeelt,delers);
}

private ArrayList<Double> verdeelHelper(ArrayList<Double> verdeelt,
    ArrayList<Integer> delers){
    if(delers.size() == 0){
        return verdeelt;
    }
    else{
        ArrayList<Integer> nieuweDelers = new ArrayList<>();
        for(int i = 0; i < delers.size(); i++){
            int deler = delers.get(i);
            boolean[] kinderen = this.verbindingsMatrix[deler];
            ArrayList<Integer> deelMetKinderen = new ArrayList<>();
            for(int j = 0; j<kinderen.length; j++){
                if(kinderen[j] && verdeelt.get(j) == 0.0){
                    deelMetKinderen.add(j);
                }
            }
            if(deelMetKinderen.size() != 0){
                double verdeelValue = verdeelt.get(deler)/2;
                verdeelt.set(deler,verdeelValue);
                verdeelValue /= deelMetKinderen.size();
                for(int j = 0; j<deelMetKinderen.size();j++){
                    int kind = deelMetKinderen.get(j);
                    verdeelt.set(kind,verdeelValue);
                    if (!nieuweDelers.contains(kind)) nieuweDelers.add(kind);
                }
            }
        }
        return verdeelHelper(verdeelt,nieuweDelers);
    }
}
```

```
}  
}
```

Oplossing 10.10