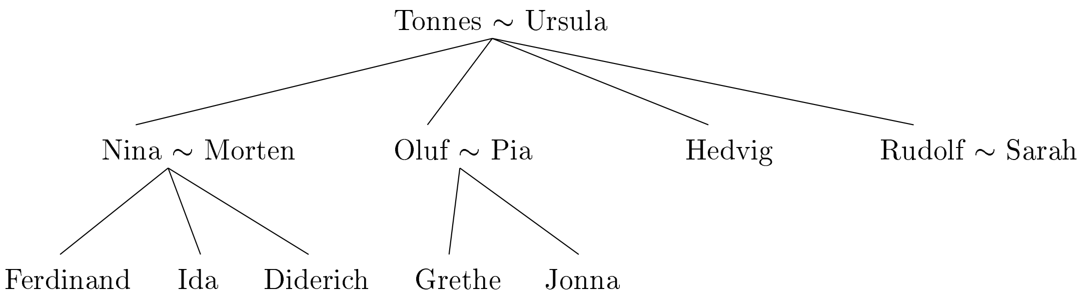
Opgave 1 (25%)

Følgende viser en simpel efterslægtstavle



De forskellige typer personer der optræder i tavlen kan repræsenteres ved en række Java klasser:

```
abstract class Person {
    String name;
}

class MarriedPerson extends Person {
    String spouse_name;
    Person[] children;
}

class Girl extends Person {}

class Boy extends Person {}
```

Med disse klassedefinitioner kan Tonnes og Ursulas efterslægt repræsenteres ved 4 `MarriedPerson`-objekter for Tonnes, Nina, Oluf og Rudolf, 2 `Boy`-objekter for Ferdinand og Diderich samt 4 `Girl`-objekter for Hedvig, Ida, Grethe og Jonna.

Efterslægtstavlen kan også udskrives som ren tekst med indrykning:

```
Tonnes
~Ursula
    Nina
    ~Morten
        Ferdinand
        Ida
        Diderich
    Oluf
    ~Pia
        Grethe
        Jonna
    Hedvig
    Rudolf
    ~Sarah
```

a) Tilføj en metode

`void printDescendants(String offset) { ... }`

til klassen `Person`, så et kald af `printDescendants("")` vil bevirke at den pågældende persons efterslægt udskrives som vist ovenfor i ren tekst. Bemærk, at det kan være nødvendigt at du overrider `printDescendants`-metoden i en eller flere af subklasserne `MarriedPerson`, `Girl` eller `Boy`. Der lægges vægt på at besvarelsen er letlæselig, detaljeret og korrekt.

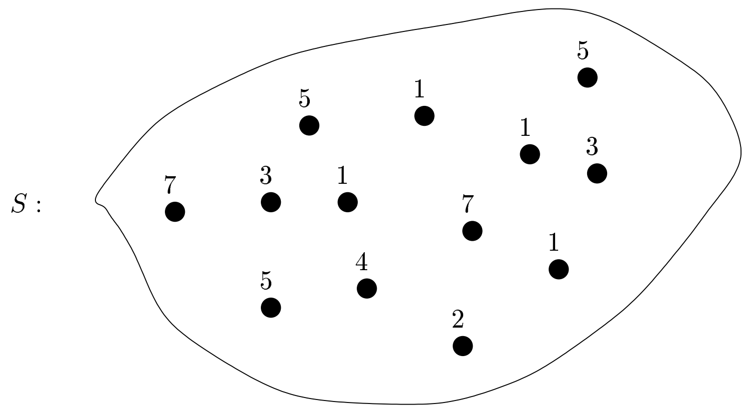
b) Tilføj en metode

`int numberOfBoyFamilies() { ... }`

til klassen `Person`, så et kald af `numberOfBoyFamilies()` returnerer antallet af familier, hvor der er flere (ugifte) drengebørn end (ugifte) pigebørn, inden for hele den pågældende persons efterslægt. F.eks. skal et kald på personen Tonnes i tavlen ovenover returnere 1, fordi inden for Tonnes efterslægt er det kun Ninas familie, der har flere ugifte drenge end piger. Bemærk, at det kan være nødvendigt at du overrider `numberOfBoyFamilies`-metoden i en eller flere af subklasserne `MarriedPerson`, `Girl` eller `Boy`. Der lægges vægt på at besvarelsen er letlæselig, detaljeret og korrekt.

Opgave 2 (20%)

En multimængde er som bekendt ligesom en mængde bortset fra, at den kan indeholde gentagelser af et element. Et eksempel er følgende multimængde af heltal



Man kan beskrive en multimængde ved at angive, hvor mange elementer der er af hver slags. Det giver en kort beskrivelse

$$S = [(1:4), (2:1), (3:2), (4:1), (5:3), (7:2)]$$

Vi vil beskæftige os med den abstrakte datatype **Multi**, der understøtter følgende operationer på en multimængde af heltal:

> `insert`(k): Tallet k indsættes i multimængden.
>
> `delete`(k): En forekomst af k slettes fra multimængden (såfremt k findes i multimængden).
>
> `lookup`(k): Antal forekomster af k i multimængden returneres.

a) Angiv en implementation af **Multi**, så alle tre operationer kan udføres i tid $O(\log n)$, hvor n er antal forskellige elementer i multimængden.

Vi ser nu på den abstrakte datatype **MultiInc**, der er ligesom **Multi**, pånær at den understøtter en ekstra operation

> `increment`(a, b): alle forekomster af et tal k, hvor $a \leq k \leq b$, ændres til $k + 1$.

Eksempel, hvis

$$S = [(1:4), (2:1), (3:2), (4:1), (5:3), (7:2)]$$

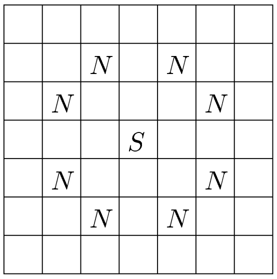
så vil udførsel af `increment`$(2, 4)$ bevirke en ændring til

$$S = [(1:4), (3:1), (4:2), (5:4), (7:2)]$$

b) Angiv en implementation af den udvidede datatype **MultiInc**, hvor alle de angivne operationer udføreres i tid $O(\log n)$, hvor n er antal forskellige elementer i multimængden.

Opgave 3 (30%)

En springer på et skakbræt kan i ét hop bevæge sig ét skridt i én retning og to skridt i en retning vinkelret herpå. En springer kan altså i et hop bevæge sig til et af 8 nabofelter

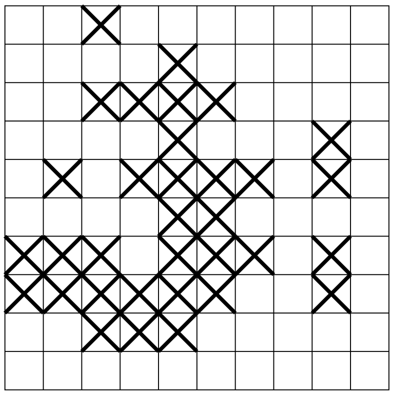


S : nuværende springer-position

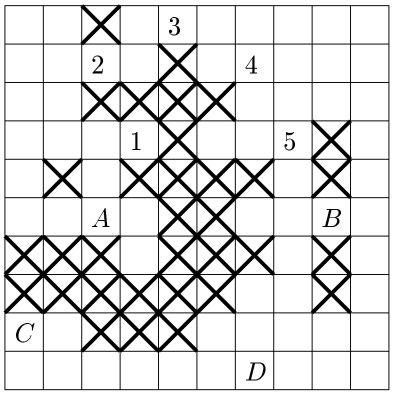
N : mulige nye positioner efter ét hop

.

Vi betragter et $n \times n$ spillebræt, hvor en række felter er forbudte (markeret med kryds på følgende figur)



På et sådant $n \times n$ bræt ser vi nu på følgende *springer-problem*: Givet to positioner på spillebrættet skal vi afgøre, om det er muligt for en springer at bevæge sig fra den ene position til den anden position uden at mellemlande på de forbudte felter.



Ovenstående tegning viser hvordan en springer kan komme fra A via 1, 2, 3, 4 og 5 til B, mens ingen springer kan komme fra C til D uden at mellemlande på forbudte felter.

a) Angiv en effektiv dynamisk programmering algoritme til at løse springer-problemet på et $n \times n$ spillebræt med forbudte felter. Hvad bliver udførelsestiden?

b) Som bekendt er en partition en opdeling af en grundmængde i disjunkte delmængder. Argumentér for at der findes en partition, hvis mængder indeholder lovlige felter på spillebrættet, så

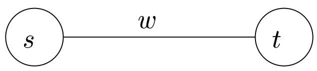
- To lovlige brætfelter ligger i samme mængde præcis hvis det ene felt kan nås fra det andet felt (eller omvendt) ved springerbevægelser, der undgår de forbudte områder.
- Ethvert lovligt spillefelt, ligger i en af mængderne.

c) Anvend nu b) til at finde en mere effektiv algoritme til løsning af springerproblemet. Hvad bliver den forbedrede tidskompleksitet?

Opgave 4 (25%)

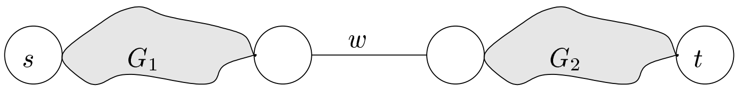
Denne opgave omhandler serie-parallelle grafer (*sp-grafer*), der er specielle ikke-orienterede vægtede grafer. De har altid to bestemte knuder s og t, og defineres i øvrigt induktivt som følger.

Den mindste sp-graf består blot af to knuder og en enkelt kant:



To sp-grafer:

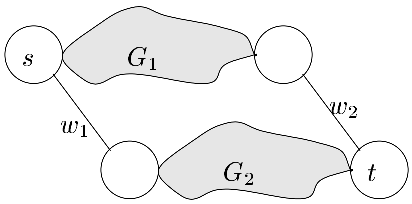
kan danne en ny ved at blive sat sammen i *serie*:



Her bliver t-knuden for G_1 og s-knuden for G_2 til almindelige knuder, og der tilføjes en ny kant.

To sp-grafer:

kan danne en ny ved at blive sat sammen i *parallel*

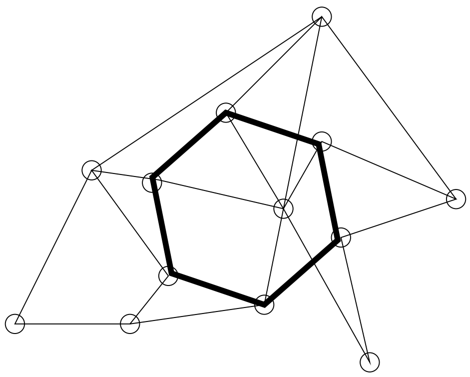


Her bliver t-knuden for G_1 og s-knuden for G_2 også til almindelige knuder, men der tilføjes to nye kanter.

a) Lad $k(n)$ være antallet af kanter i en sp-graf med n knuder. Argumentér for, at $k(n) \in \Theta(n)$.

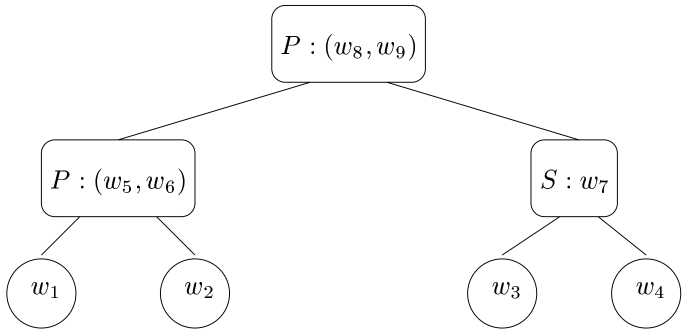
For en sp-graf er vi nu interesserede i at finde et minimalt udspændende træ

b) Hvor lang tid vil det med Kruskals algoritme tage at løse denne opgave for en sp-graf med n knuder?

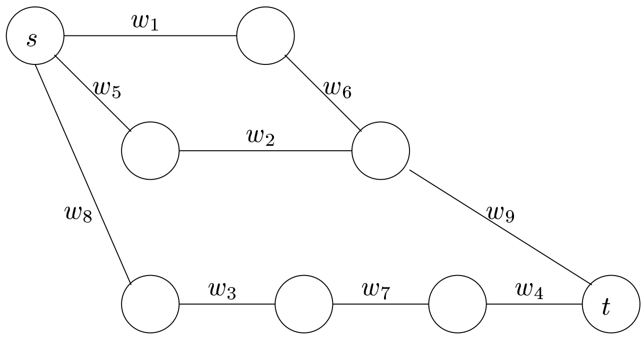


c) Figuren ovenfor viser en vægtet graf G, der indeholder en kreds markeret med fede kanter foruden en mængde andre kanter. Argumentér for, at hvis der findes en tungeste kant i kredsen, så kan den *ikke* indgå i et minimalt udspændende træ for G.

En sp-graf kan repræsenteres ved et binært strukturtræ. F.eks. repræsenterer træet



grafen



I strukturtræet repræsenterer en indre knude $S : w$ en seriel sammensætning af de to grafer, der er repræsenteret i undertræerne, med en ny kant af vægt w. En indre knude $P : (w_1, w_2)$ repræsenterer tilsvarende en parallel sammensætning af undertræerne, hvor de nye kanter har vægte w_1 og w_2. Endelig repræsenterer et blad w en graf der består af en enkelt kant med vægt w.

d) Hvis man repræsenterer en sp-graf som et binært strukturtræ, hvordan kan man så opnå en mere effektiv algoritme til beregning af et minimalt udspændende træ for sp-grafen?