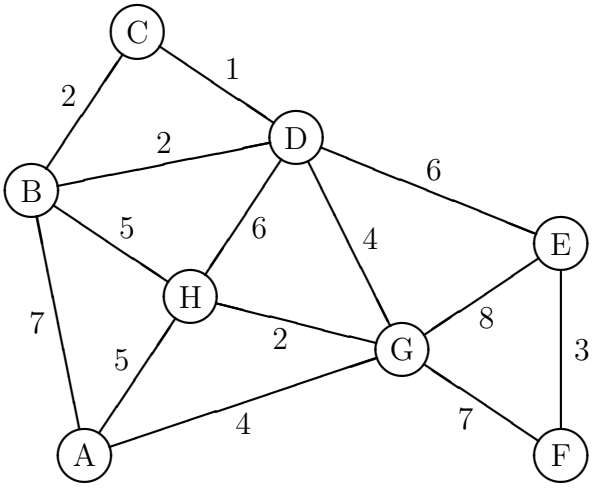
Tentamen Algoritmiek
Donderdag 4 juni 2020, 14.15 – 17.15 uur

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. [11 pt]

   (a) Laat G_1 de volgende graaf zijn:



   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.

2. [32 pt] Suzan en Freek spelen een variant van het spel Nim.

   Er liggen $n \geq 1$ lucifers op de tafel, en om de beurt mogen de twee vrienden er 1, 2 of 3 weghalen, net zo lang totdat alle lucifers weg zijn. Suzan begint. De speler die als laatste een of meer lucifers van de tafel heeft genomen, *verliest*.

   (a) Wat zijn voor dit spel de toestanden en de acties (voor algemene $n \geq 1$)? Wanneer is een toestand een eindtoestand?

   We noemen een toestand *winnend* voor een speler als die speler gaat winnen bij optimaal spel van beide spelers vanaf die toestand.

   (b) Teken de complete toestand-actie-boom van dit spel voor het geval dat $n = 4$. Teken toestanden die (feitelijk) equivalent zijn, toch iedere keer opnieuw.

   Geef bij *elke* toestand ook aan of die winnend is voor S (Suzan) of voor F (Freek), te beginnen bij de eindtoestanden.

   (c) Hoeveel knopen bevat de complete toestand-actie-boom voor $n = 1$, voor $n = 2$, voor $n = 3$ en voor $n = 4$?

   Beredeneer, zonder de boom te tekenen, hoeveel knopen de complete toestand-actie-boom voor $n = 5$ heeft.

(d) Laat $f(n)$ voor $n \geq 1$ het aantal knopen zijn in de complete toestand-actie-boom bij de variant van Nim van Suzan en Freek. Geef een recurrente betrekking waaraan $f(n)$ voldoet. Vergeet het basisgeval / de basisgevallen van de recursie niet. Leg ook uit waarom $f(n)$ aan deze betrekking voldoet.

(e) De algemene opzet voor een functie `winnend (...)`, die met behulp van brute force de toestand-actie-boom van een tweepersoonsspel afloopt om te bepalen of de huidige toestand (bij optimaal spel van beide spelers) winnend is voor de speler die nu aan de beurt is, ziet er als volgt uit:

```
winnend (stand)
{
  if eindstand (stand) then
    ...
  else
    for alle mogelijke zetten i do
      kopie := stand;
      doezet (kopie,i);
      if not winnend (kopie) then
        return true;
      fi
    od
    return false;
  fi
}
```

Maak van deze algemene opzet een concrete C++-functie `bool winnend (...)` voor de variant van Nim die Suzan en Freek spelen. De functie moet dus gebruik blijven maken van brute force, maar moet wel stoppen met het aflopen van mogelijke zetten in een toestand, als er een winnende zet is gevonden.

(f) Bij de standaard variant van Nim die we tijdens college behandeld hebben, heb je uiteindelijk geen brute force nodig om te bepalen of de huidige toestand winnend is of niet voor de speler die nu aan de beurt is (bij optimaal spel van beide spelers). De toestand is winnend voor de speler die aan de beurt is, als het aantal lucifers niet deelbaar is door 3.

Iets dergelijks geldt ook voor de variant die Suzan en Freek spelen. In welk geval / in welke gevallen is een toestand met $n \geq 1$ lucifers winnend voor de speler die nu aan de beurt is? Motiveer je antwoord.

3. [31 pt] De koning op het schaakbord kan in 1 zet naar een naburig vakje (horizontaal, verticaal of diagonaal) lopen. We hebben nu een schaakbord van $n + 1$ rijen en $n + 1$ kolommen, genummerd van 0 t/m n. Een positie (i, j) correspondeert met rij i en kolom j.

We zijn ge-interesseerd in het aantal kortste paden (zo min mogelijk stappen) voor een koning die op positie (n, k_1) staat voor zekere kolom k_1, en naar positie $(0, k_2)$ wil voor zekere kolom k_2. Het is in te zien dat zo'n pad altijd uit n stappen bestaat: in elke stap gaat de koning een rij naar beneden, waarbij hij eventueel ook een kolom naar links of rechts gaat.

Om dit aantal kortste paden te berekenen, maken we gebruik van de grootheid $P(i, j)$, die het aantal paden van lengte i voorstelt van positie (i, j) naar positie $(0, k_2)$.

(a) Beredeneer dat $P(i,j)$ voldoet aan de volgende recurrente betrekking:

$$P(i,j) = \begin{cases} 1 & \text{als } i = 0 \text{ en } j = k_2 \\ 0 & \text{als } i = 0 \text{ en } j \neq k_2 \\ P(i-1,0) + P(i-1,1) & \text{als } i > 0 \text{ en } j = 0 \\ P(i-1,n-1) + P(i-1,n) & \text{als } i > 0 \text{ en } j = n \\ P(i-1,j-1) + P(i-1,j) + P(i-1,j+1) & \text{als } i > 0 \text{ en } 0 < j < n \end{cases}$$

Hint: bedenk bij de twee basisgevallen, wat $P(0,j)$ ook al weer voorstelt.

(b) Geef een C++functie `int paden (int n, int k1, int k2)` die met behulp van bottom-up dynamisch programmeren, en met gebruikmaking van de bij (a) gegeven recurrente betrekking, het aantal kortste paden bepaalt (en retourneert) voor een koning op een $(n+1)\times(n+1)$ schaakbord van positie $(n, k1)$ naar positie $(0, k2)$. Je mag aannemen dat $n \geq 1$ en dat $k1$ en $k2$ geldige kolommen zijn. Maak gebruik van een passende tabel om resultaten van deelproblemen in op te slaan.

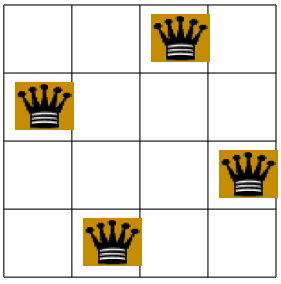
(c) Voer het algoritme van onderdeel (b) met de hand uit voor het geval dat $n = 4$, $k_1 = 2$ en $k_2 = 1$. Vul de complete tabel met resultaten van deelproblemen in, en maak duidelijk wat het eindantwoord is.

(d) Wat is de worst-case tijdcomplexiteit van je algoritme bij onderdeel (b)? Motiveer je antwoord, door een basisoperatie in het algoritme aan te wijzen en te bepalen hoe vaak deze operatie wordt uitgevoerd.

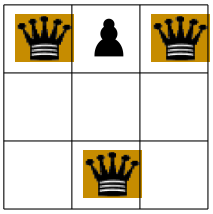
4. [26 pt] Tijdens college hebben we het n-koninginnen probleem behandeld: gegeven een $n \times n$ schaakbord, op hoeveel manieren kunnen we n dames (koninginnen) op het bord zetten, zodat geen enkele dame een andere dame kan aanvallen.

Aanvallen betekent dat een dame in 1 zet de positie van een andere dame kan bereiken. Omdat een dame in 1 zet een willekeurig aantal vakjes in dezelfde richting (horizontaal, verticaal, diagonaal, dus maximaal acht richtingen) kan bewegen, gaat het er dus om dat geen enkel tweetal dames in dezelfde rij, in dezelfde kolom, of in dezelfde diagonaal wordt gezet.

Een voorbeeld van een geldige oplossing voor $n = 4$ is



In deze opgave kijken we naar een variant van dit probleem. Gegeven een $n \times n$ schaakbord en een pion, op hoeveel manieren kunnen we n dames en een pion op het bord zetten, zodat geen enkele dame een andere dame kan aanvallen. Dames kunnen niet over de pion heenspringen, zodat er extra manieren komen om de dames neer te zetten. Bijvoorbeeld, het oorspronkelijke n-koninginnenprobleem had geen oplossing voor $n = 3$. Het n-koninginnenprobleem met een pion wel, bijvoorbeeld:



Merk op dat we niet langer eisen dat er per se maar één dame per rij of kolom gezet kan worden. Het feit dat een dame de pion kan aanvallen, negeren we.

(a) Schrijf een functie `bool aanval (int n, pair<int,int> dame1, pair<int,int> dame2, pair<int,int> pion)` die controleert of een dame op positie `dame1` een dame op positie `dame2` kan aanvallen, als de pion op positie `pion` staat. Zo ja, dan retourneert de functie `true`, anders `false`.

Neem aan dat een pair bestaat uit de rij en de kolom van de positie, dat alle rijen en kolommen genummerd zijn van 1 t/m n, en dat we met drie geldige, *verschillende* posities te maken hebben.

Merk op dat de slides van college 6 hier een beetje bij kunnen helpen. Je mag, desgewenst, gebruik maken van functies `min` en `max`, die het minimum en maximum van twee of meer getallen bepalen.

Voor wie pairs niet paraat heeft: je kunt eenvoudig de onderdelen van een pair benaderen met b.v. `dame1.first` en `dame1.second`. Je kunt een nieuw pair maken met b.v. een toekenning `pos = make_pair (rij, kolom);`. In het algemeen kun je ook eenvoudig pairs toekennen en vergelijken, met = respectievelijk ==.

De volgende functie kan gebruikt worden om te testen of een nieuw te plaatsen dame op positie `nieuwedame` op een $n \times n$ schaakbord een van de dames op de posities in vector `dames` kan aanvallen, gegeven de positie `pion` van de pion.

```
bool geeftaanval (int n, vector<pair<int,int>> dames, pair<int,int> pion,
                  pair<int,int> nieuwedame)
{
  for (size_t i=0;i<dames.size();i++)
  { if (aanval (n, dames[i], nieuwedame, pion))
      return true;
  }
  return false;
}
```

Merk op dat deze functie zonder controle vooraf de functie `aanval` aanroept. De functie `geeftaanval` gaat er dus vanuit dat positie `nieuwedame` verschilt van alle posities in vector `dames` en positie `pion`. Zorg daar ook voor als je deze nieuwe functie hieronder aanroept.

(b) Schrijf een recursieve functie `int plaatsdames (int n, vector<pair<int,int>> dames, pair<int,int> pion)` om het aantal manieren te bepalen waarop je n dames op een $n \times n$ schaakbord kunt zetten, zonder dat ze elkaar kunnen slaan, gegeven

de (vaste) positie `pion` van een pion. De functie moet de dames plaatsen met behulp van backtracking.

Bij de eerste aanroep zal de vector `dames` leeg zijn. Bij elk volgende niveau van de recursie bevat de vector een positie van een dame extra. De returnwaarde van een recursieve aanroep wordt dan het aantal mogelijkheden om de huidige plaatsing uit te breiden tot een volledige plaatsing (gebruik dus geen globale teller). Maak bij het plaatsen van een volgende dame gebruik van de functie `geeftaanval`.

Probeer te voorkomen dat je plaatsingen dubbel telt: als je twee dames van positie verwisselt, is het feitelijk dezelfde plaatsing. Die zou dan maar 1 keer geteld moeten worden.

Voor wie vectoren niet paraat heeft: je kunt met `dames.push_back` een nieuwe positie achteraan de vector toevoegen, en met `dames.pop_back` kun je het achterste element er weer uithalen.
