Tentamen Algoritmiek
Vrijdag 5 juni, 13.15 – 16.15 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. [15 pt] De Fibonaccigetallen $F(n)$ worden gedefinieerd door

$$F(n) = \begin{cases} 0 & \text{als } n = 0 \\ 1 & \text{als } n = 1 \\ F(n-1) + F(n-2) & \text{als } n \geq 2 \end{cases}$$

   De eerste tien Fibonaccigetallen zijn:

n	0	1	2	3	4	5	6	7	8	9
$F(n)$	0	1	1	2	3	5	8	13	21	34

   We kunnen de Fibonaccigetallen met de volgende recursieve functie berekenen:

```
long long fib (int n)
{ if (n==0 || n==1)
    retun n;
  else
    return fib(n-1) + fib(n-2);
}
```

   Als basisoperatie van deze functie kunnen we de optelling in de tweede returninstructie kiezen. Laat $A(n)$ het aantal keer zijn dat deze basisoperatie wordt uitgevoerd bij de berekening van `fib`(n).

   (a) Geef een recurrente betrekking voor $A(n)$, met basisgeval(len) en recursieve stap.
   Als je dit antwoord niet weet, kun je het 'kopen' van de docent, zodat je de volgende onderdelen wellicht wel kunt maken.

   (b) Geef de waarden van $A(n)$ voor $n = 0, 1, 2, \ldots, 9$

   (c) Druk $A(n)$ voor $n \geq 0$ uit in de Fibonaccigetallen. Bewijs met inductie dat deze relatie tussen $A(n)$ en de Fibonacci getallen correct is.

---

2. [30 pt] Anna en Demi spelen een variant van het spel Nim. Hierbij liggen twee stapels met lucifers op de tafel, met (aan het begin) respectievelijk $m \geq 1$ en $n \geq 1$ lucifers. Om de beurt mogen de twee spelers een stapel kiezen en daar een aantal lucifers afhalen: minimaal 1 en maximaal de hele stapel. Als beide stapels leeg zijn, is het spel afgelopen. De speler die op dat moment aan de beurt zou zijn, verliest. De andere speler, die dus de laatste lucifers heeft weggehaald, wint dan.

   (a) Wat zijn voor dit spel de toestanden en de acties (voor algemene $m, n \geq 1$)?
   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 toestand-actie-boom van dit spel, voor het geval dat $m = 2$ en $n = 3$ en dat Anna (als oudste) begint. Geef bij *elke* toestand ook aan of die winnend is voor A (Anna) of voor D (Demi), te beginnen bij de onderste toestanden. Je mag hierbij gebruik maken van de volgende feiten:

- Als een van de twee stapels leeg is, en de ander niet, dan is de toestand winnend voor de speler die aan de beurt is. Zij kan immers de ene resterende stapel helemaal weghalen. Die toestand hoef je dan niet meer verder uit te werken.
- Als er meerdere toestanden zijn waarbij de ene stapel i lucifers en de andere stapel j lucifers bevat, dan zijn die feitelijk equivalent. Dan hoef je maar één van die toestanden uit te werken.

(c) 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:

```
bool 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, recursieve C++-functie `bool winnend (...)` voor de variant van Nim die Anna en Demi 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.

(d) Beredeneer dat een toestand waarbij beide stapels even veel lucifers bevatten verliezend is voor de speler die aan de beurt is. Je hoeft dit niet per se formeel met inductie te doen.

Hint: kijk of je een dergelijke toestand ook in je toestand-actie-boom van onderdeel (b) hebt.

---

3. [25 pt] We bekijken het landkaartkleuringsprobleem. We hebben een 2-dimensionale landkaart, waarbij elk land een samenhangend oppervlak heeft (geen enclaves in andere landen). We willen elk land een kleur geven, zodat buurlanden (met een gedeelde grens van

positieve lengte), verschillende kleuren krijgen. Dit noemen we een *geldige kleuring*. Volgens de vierkleurenstelling kan elk landkaart geldig gekleurd worden met maximaal vier verschillende kleuren.

De landkaart kun je zien als een ongerichte graaf G, waarbij de landen de knopen vormen, en de grenzen (van positieve lengte) tussen landen de takken. Laat n het aantal landen en m het aantal takken van G zijn. De namen van de landen zijn $0, 1, 2, \ldots, n-1$.

Laat G gerepresenteerd worden met behulp van een adjacency list in `Buur *graaf[n]`, waarbij de klasse Buur als volgt gedefinieerd is:

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

Als je de adjacency list representatie van grafen niet kent, mag je bij deze opgave ook uitgaan van een adjacency matrix `bool graaf[n][n]`*, waarbij* `graaf[i][j]` *true is, dan en slechts dan als er een tak tussen knoop i en knoop j in de graaf is. In dat geval kun je voor deze opgave niet 25, maar 20 punten verdienen.*

Je mag ervanuit gaan dat het array `graaf` bij beide representaties een globale variabele is. De vier kleuren die we willen gebruiken zijn 1,2,3,4.

(a) Schrijf een niet-recursieve C++-functie `bool kleurGretig (int n, int kleuring[])`, die op gretige wijze een geldige kleuring van de n landen met onze vier kleuren probeert te bepalen. De functie moet `true` retourneren als het lukt om (op gretige wijze) een geldige kleuring voor alle n landen te vinden. In dat geval moet `kleuring[i]` aan het eind de kleur van land i zijn. Als dat niet lukt (omdat er voor een bepaalde knoop i geen kleur meer gevonden kan worden, gegeven de al toegekende kleuren aan zijn buurknopen), moet de functie `false` retourneren.

(b) Met een gretig algoritme kun je niet voor alle mogelijke landkaarten een geldige kleuring vinden. Geef een voorbeeld van een landkaart, waarbij de landen ook $0, 1, \ldots, n-1$ heten, waarbij je functie uit onderdeel (a) geen geldige kleuring kan vinden. Laat ook zien welke landen (in welke volgorde) in de loop van de functie welke kleur krijgen, en bij welk land geen kleur meer kan worden gevonden.

(c) Schrijf nu een *recursieve* C++-functie `bool kleurBT (int n, int kleuring[], int i)`, die met behulp van backtracking een geldige kleuring van de n knopen met maximaal vier kleuren bepaalt. De parameters `kleuring` en i betekenen

- dat we tot nu toe kleuren `kleuring[0]`, ..., `kleuring[i-1]` een waarde gegeven hebben,
- en dat we in deze recursieve aanroep mogelijke waardes voor `kleuring[i]` gaan proberen.

De eerste aanroep zal van de vorm `kleurBT (n, kleuring, 0);` zijn.

---

4. [30 pt] Veel mensen denken dat grotere, zwaardere olifanten ook slimmer zijn. Deze gedachte willen we graag weerleggen.

   Zij gegeven een rij van n olifanten, genummerd $0, 1, 2, \ldots, n-1$. Olifant i heeft gewicht `gewicht[i]` en IQ `IQ[i]`. We hebben de olifanten al gesorteerd op oplopend gewicht: `gewicht[i]` $\leq$ `gewicht[i + 1]` voor $i = 0, 1, \ldots, n-2$. Gelijke gewichten zijn mogelijk. Een voorbeeldrij olifanten met hun gewicht en IQ is:

i	0	1	2	3	4	5	6	7	8	9
`gewicht[i]`	7	8	8	10	10	11	12	13	13	13
`IQ[i]`	3	4	5	6	5	5	9	9	2	8

   Bijvoorbeeld, olifanten 6 en 8 vormen een tegenvoorbeeld van de gedachte dat zwaardere olifanten slimmer zijn: olifant 8 is zwaarder, maar veel minder slim dan olifant 6. Een deelrij met olifanten waarbij elke volgende olifant in de deelrij (strict) zwaarder en (strict) minder slim is dan de huidige, noemen we een *tegenvoorbeeldrij*. We zoeken nu een langste tegenvoorbeeldrij, dat wil zeggen: een tegenvoorbeeldrij met zo veel mogelijk olifanten. In het algemeen kunnen er meerdere langste tegenvoorbeeldrijen zijn. Bij bovenstaand voorbeeld is er maar één, bestaande uit olifanten 3, 5 en 8.

   Voor algemene rijen met olifanten (gesorteerd op gewicht) gaan we dit oplossen met dynamisch programmeren. Daartoe definiëren we, voor $i = 0, 1, 2, \ldots, n-1$:

   > lengte(i) is de lengte van een langste tegenvoorbeeldrij die eindigt met olifant i (dus olifant i moet er zelf ook in zitten).

   Bij bovenstaand voorbeeld is bijvoorbeeld lengte(2) = 1 en lengte(8) = 3.

   (a) Geef voor bovenstaand voorbeeld de waarde lengte(i) voor elke i.

   (b) Beredeneer dat lengte(i) voldoet aan de volgende recurrente betrekking:

$$\text{lengte}(i) \;=\; \max(\; \{\text{lengte}(j)+1 \mid\; 0 \leq j < i \text{ en } \texttt{gewicht[j]<gewicht[i]} \text{ en } \texttt{IQ[j]>IQ[i]}\} \cup \{1\}\;)$$

   Als $i = 0$, is lengte(i) dus gelijk aan 1. Als $i > 0$, kijken we ook naar $j < i$.

   (c) Schrijf een niet-recursieve C++-functie `int langsteRij (int gewicht[], int IQ[], int n)`, die gebruikmakend van **bottom-up dynamisch programmeren** en de hierboven beschreven recurrente betrekking, de lengte bepaalt van een langste tegenvoorbeeldrij. Deze lengte wordt vervolgens geretourneerd.

   (d) Wat is de tijdcomplexiteit van je functie `langsteRij` uit onderdeel (c)? Motiveer je antwoord, door een basisoperatie aan te bewijzen en te bepalen hoe vaak die wordt uitgevoerd.

   (e) Stel dat je een array `lengte[0],lengte[1],...,lengte[n-1]` hebt met daarin de waardes lengte(i) uit ons probleem. Schrijf een niet-recursive C++-functie `vector<int> langsteRij (int lengte[], int n)` die een langste tegenvoorbeeldrij bepaalt, op basis van de opgeslagen waardes in array `lengte`, en arrays `gewicht` en `IQ`. Niet de *lengte* van een langste tegenvoorbeeldrij dus, maar zo'n rij zelf, dat wil zeggen: de nummers van de olifanten in de rij, in oplopende volgorde. Deze rij wordt in een vector geretourneerd.

   Probeer ervoor te zorgen dat de tijdcomplexiteit van je functie lineair is in n. Als dit niet lukt, kun je nog wel het grootste deel van de punten verdienen.