Hertentamen Algoritmiek
Maandag 29 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. [28 pt] Het 1-dimensionale *closest pair*-probleem wordt als volgt gedefinieerd:

   > Gegeven een array $A = A[0], A[1], \ldots, A[N-1]$ dat $N \geq 2$ integers bevat, wat is het kleinste verschil (in absolute waarde) tussen twee elementen van A?

   In deze opgave moet je drie verschillende functies in C++ schrijven die dit probleem oplossen: met brute force, met decrease-and-conquer en met divide-and-conquer. **We gaan ervanuit dat de getallen van klein naar groot gesorteerd zijn.**

   (a) Schrijf een eenvoudige, niet-recursieve C++-functie

   ```
   int closestPair1 (int A[ ], int N)
   ```

   die met **brute force** het kleinste verschil (in absolute waarde) tussen twee elementen van ons (gesorteerde) array A met $N \geq 2$ integers bepaalt, en dat retourneert.

   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.

   (b) Schrijf nu een recursieve C++-functie

   ```
   int closestPair2 (int A[ ], int i)
   ```

   die met behulp van **decrease-by-one-and-conquer** het probleem oplost en het gevraagde verschil retourneert voor het (deel)array $A[0], A[1], \ldots, A[i-1]$ ($i \geq 2$).

   De aanroep `closestPair2 (A, N)` geeft dan uiteraard het antwoord voor ons hele array A.

   (c) Schrijf nu een recursieve C++-functie

   ```
   int closestPair3 (int A[ ], int links, int rechts)
   ```

   die met behulp van **divide-and-conquer** het probleem oplost en het gevraagde verschil retourneert voor het deelarray $A[$`links`$], \ldots, A[$`rechts`$]$, met `links` $<$ `rechts`. Hierbij moet het deelarray steeds (tot de eindconditie van de recursie) in twee delen van (ongeveer) halve grootte worden verdeeld. Als het deelarray een oneven aantal getallen beslaat, zal het ene deel hierbij 1 groter worden dan het andere deel.

   De aanroep `closestPair3 (A, 0, N-1)` geeft dan uiteraard het antwoord voor ons hele array A.

   (d) De *Master Theorem* kan gebruikt worden om de tijdcomplexiteit te bepalen van een algoritme met divide-and-conquer. Stel dat zo'n algoritme een instantie van grootte n opsplitst in a instanties van elk grootte (ongeveer) n/b. Stel verder dat de rekentijd $C(n)$ voor een instantie van grootte n voldoet aan $C(n) = aC(n/b) + f(n)$, waarbij $f(n)$

de tijd is om de instantie op te splitsen in a instanties van grootte n/b, en om de (deel-)oplossingen van deze instanties te combineren tot een oplossing van het oorspronkelijke probleem. Als $f(n) \in \Theta(n^d)$ met $d \geq 0$, dan geldt:

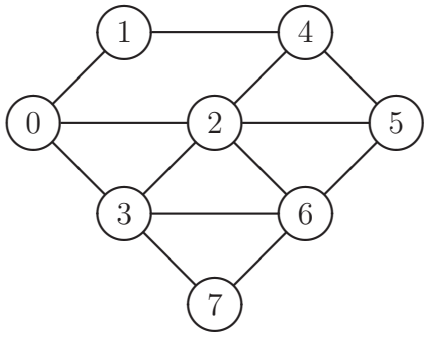
$$C(n) \in \begin{cases} \Theta(n^d) & \text{als } a < b^d \\ \Theta(n^d \log n) & \text{als } a = b^d \\ \Theta(n^{\log_b a}) & \text{als } a > b^d \end{cases}$$

Pas dit resultaat toe op het divide-and-conquer algoritme van onderdeel (c). Als je dat onderdeel niet hebt, ga er dan vanuit dat $f(n)$ constant is. Wat zijn a, b en d in dit geval, en wat is de resulterende tijdcomplexiteit volgens de Master Theorem? Motiveer de waarde voor d en de resulterende tijdcomplexiteit die je vindt.

---

2. [20 pt] We hebben een ongerichte graaf $G = (V, E)$, met knooppunten $V = \{0, 1, 2, \ldots, n-1\}$ en adjacency matrix `bool graaf[MaxN][MaxN]`, waarbij `graaf[i][j]` true is, dan en slechts dan als er een tak tussen knoop i en knoop j in de graaf is. Je mag ervanuit gaan dat het array `graaf` een globale variabele is.

   We noemen een deelverzameling C van V met $k \geq 1$ knooppunten een *k-kliek* als elk knooppunt i in C een tak heeft naar elk ander knooppunt j in C. In onderstaande graaf is bijvoorbeeld $\{0, 1, 2\}$ geen 3-kliek, maar $\{0, 2, 3\}$ is dat wel:



   (a) Schrijf een niet-recursieve C++-functie `bool isKliek (int k, int C[ ])`, die controleert of de elementen $C[0], C[1], \ldots, C[k-1]$ een k-kliek in onze graaf vormen. Je mag ervanuit gaan dat $C[i] \in \{0, 1, \ldots, n-1\}$ voor $i = 0, 1, \ldots, k-1$ (het zijn geldige knooppuntennummers). De functie moet `true` retourneren als de elementen inderdaad een k-kliek vormen, en `false` anders.

   (b) Schrijf nu een *recursieve* C++-functie `bool bevatKliek (int n, int k, int C[ ], int i)`, die met behulp van exhaustive search controleert of de graaf een k-kliek bevat. De functie moet alle deelverzamelingen C van k knooppunten genereren, en controleren of C een k-kliek is met behulp van functie `isKliek` uit onderdeel (a). De zoektocht mag stoppen wanneer er een k-kliek gevonden is. De parameters `C` en i betekenen
   • dat we tot nu toe waardes hebben bepaald voor de elementen `C[0]`, ..., `C[i-1]` van de deelverzameling,
   • en dat we in deze recursieve aanroep mogelijke waardes voor `C[i]` gaan proberen.

   De functie moet `true` retourneren als het lukt om de huidige deelverzameling uit te breiden tot een k-kliek, en `false` anders. De eerste aanroep zal van de vorm `bevatKliek (n, k, C, 0)` zijn.

   Probeer hierbij te voorkomen dat dezelfde deelverzameling meerdere keren geconstrueerd wordt. Bijvoorbeeld $\{0, 2, 3\}$ is hetzelfde als $\{3, 0, 2\}$. Als dit niet lukt, kun je nog wel het grootste deel van de punten verdienen.

(c) Wat is de worst case tijdcomplexiteit van een exhaustive search algoritme (zoals je antwoord bij (b)) om te controleren of een ongerichte graaf met n knopen en m takken een k-kliek bevat? Motiveer je antwoord door te kijken hoeveel deelverzamelingen van k knooppunten er worden gemaakt, en hoeveel werk er per deelverzameling kan worden gedaan, bij een adjacency matrix representatie.

Ook als je bij onderdeel (b) geen goed antwoord hebt, kun je hier misschien toch iets zinnigs zeggen.

---

3. [27 pt] Reguliere expressies kunnen gebruikt worden om strings te beschrijven. In deze opgave kijken we naar expressies die bestaan uit (gewone) letters uit het alfabet, en de symbolen `'*'` en `'?'` met de volgende betekenis:

   `'*'` een willekeurige string van 0 of meer letters uit het alfabet

   `'?'` een willekeurige letter uit het alfabet.

   Bijvoorbeeld:
   • De string `"Leiden"` voldoet aan de expressie `"L?iden"`.
   • De string `"Leiden"` voldoet niet aan de expressie `"L?den"`.
   • De string `"Algoritmiek"` voldoet aan de expressie `"Algo*"`.
   • De string `"tabel"` voldoet aan de expressie `"t*abel"`.
   • De string `"Universiteit"` voldoet niet aan de expressie `"Uni*y"`.

   We gaan dynamisch programmeren gebruiken om te controleren of een string aan een expressie voldoet. Laat s een string zijn van 0 of meer letters uit het alfabet, en laat p (van $p$atroon) een reguliere expressie zijn zoals hierboven beschreven. Laat $m \geq 0$ de lengte (het aantal symbolen) van s zijn en $n \geq 0$ de lengte van p.

   Dan definiëren we, voor $0 \leq i \leq m$ en $0 \leq j \leq n$, de boolean $match(i, j)$ als volgt:

   > $match(i, j)$ is true, dan en slechts dan als de eerste i letters van s voldoen aan de expressie bestaande uit de eerste j letters van p.

   Merk op dat in strings (en ook in expressies) de losse karakters vanaf positie 0 in de string staan. Dus bijvoorbeeld: $s[0]$ is de eerste letter van string s, en $s[i-1]$ is de i-de letter.

   (a) Bepaal voor de strings $s =$ "`aabab`", en $p =$ "`a*b?`" de waarde $match(i, j)$ voor elke combinatie van i en j. Presenteer deze waarden in een tabel van de volgende vorm:

	$j=0$	1	2	3	4
$i=0$					
1					
2					
3					
4					
5					

   (b) Er geldt:

$$match(i,j) = \begin{cases} \texttt{true} & \text{als } i=0,\ j=0 \\ \texttt{false} & \text{als } i=0,\ j>0 \text{ en } p[j-1] \neq '*' \\ match(i,j-1) & \text{als } i=0,\ j>0 \text{ en } p[j-1] = '*' \end{cases}$$

Maak de recurrente betrekking voor $match(i, j)$ compleet door ook de gevallen $(i > 0, j = 0)$ en $(i > 0, j > 0)$ met alle mogelijke karakters in s en p te beschrijven. Geef een korte motivatie van je antwoord.

Als je het antwoord op dit onderdeel niet weet, dan kun je het 'kopen' van de docent. Wellicht kun je dan wel onderdeel (c) maken.

(c) Schrijf een C++-functie

```
bool voldoetAan (string s, string p)
```

die gebruikmakend van **top-down dynamisch programmeren** en de recurrente betrekking van onderdeel (b), bepaalt of string s voldoet aan de expressie p. De functie moet `true` retourneren, dan en slechts dan als dat het geval is. Je mag ervanuit gaan dat de lengte van strings s en p maximaal `MaxLengte` is. Met behulp van bijvoorbeeld `s.length()` kun je de feitelijke lengte van s opvragen.

Je functie moet gebruik maken van een recursieve hulpfunctie met (o.a.) parameters i en j, die controleert of de eerste i letters van s voldoen aan de eerste j letters van p.

---

4. [25 pt]

(a) Twee 'ontwerptechnieken' die we tijdens de colleges hebben behandeld zijn *backtracking* en *best-first branch-and-bound*. Geef twee overeenkomsten en twee verschillen tussen deze twee ontwerptechnieken, zoals behandeld tijdens de colleges.

We gaan nu best-first branch-and-bound toepassen op het toewijzingsprobleem (*assignment problem*). Bij dit probleem hebben we n personen en n jobs. Elke persoon moet één job uitvoeren. Elke job moet door één persoon worden uitgevoerd. Wanneer persoon i job j uitvoert, kost dat $kosten[i][j]$. Zie als voorbeeld de volgende kostenmatrix met $n = 4$:

	job 1	job 2	job 3	job 4
Aké	8	7	2	4
Brobbey	2	8	4	5
Cruijff	2	6	3	4
Dumfries	3	4	4	7

Doel van het toewijzingsprobleem is om een zodanige toewijzing van jobs aan personen te krijgen, dat de totale kosten worden geminimaliseerd. Voor het toepassen van best-first branch-and-bound maken we gebruik van een ondergrens.

(b) Beschrijf een geschikte ondergrens voor het toewijzingsprobleem. Geef zo'n beschrijving zowel voor de begintoestand (als er nog helemaal geen jobs aan personen zijn toegewezen) als voor een algemene deeloplossing.

Ga ervanuit dat we per rij (dus per persoon) werken.

Als je het antwoord op dit onderdeel niet weet, dan kun je het 'kopen' van de docent. Wellicht kun je dan wel onderdeel (c) maken.

(c) Pas de methode best-first branch-and-bound toe op de voorbeeld kostenmatrix met $n = 4$ van hierboven. Teken de bijbehorende *state-space-tree*, met bij elke knoop (deeloplossing) de relevante informatie (waaronder ook de opbouw van de ondergrens). Geef ook aan in welke volgorde de knopen zijn aangemaakt, welke knopen gesnoeid worden en waarom.

Ga er ook bij het opbouwen van de state-space-tree vanuit dat we per rij werken.

Wat is dus de optimale oplossing?
