Programmeeropdracht 1 — Champions League
Algoritmiek, voorjaar 2026

Inleiding

De Europese voetbalbond UEFA organiseert elk jaar de Champions League, een competitie waarin de beste clubs van Europa met elkaar strijden om de 'cup met de grote oren'. Na de voorrondes van de Champions League vindt de competitiefase plaats. Hierin spelen de N deelnemende clubs elk $r \leq N-1$ wedstrijden in evenzovele rondes. Als r gelijk is aan $N-1$, speelt elke club tegen elke andere club. In het algemeen is r echter (veel) kleiner dan $N-1$. Doel is om een evenwichtig schema samen te stellen voor deze competitiefase. Hierin speelt elke club (ongeveer) even vaak thuis als uit, en tegen een eerlijke selectie van de andere clubs: deels sterke, en deels minder sterke tegenstanders. Er zijn ook nog andere eisen aan het schema. Zie verderop.

Representatie van schema

Een schema kan gerepresenteerd worden als een vector van clubnamen. Om dit duidelijk te maken, vervangen we voor het gemak de clubnamen door nummers, b.v. $0, 1, \ldots, N-1$. Een gedeeltelijk schema bij een invoer van $N = 8$ clubs kan er dan als volgt uit zien:

```
Ronde 1:   Ronde 2:   Ronde 3:
0-1        5-0        0-3
2-3        1-4        ...
4-5        3-6
6-7        7-2
```

In ronde 1 speelt dan club 0 thuis tegen club 1, club 2 thuis tegen club 3, enzovoort. Het complete schema kun je dan intern opslaan in een vector of array van integers:

0 1 2 3 4 5 6 7 5 0 1 4 3 6 7 2 0 3 ...

Dus eerst de wedstrijden uit ronde 1, dan de wedstrijden uit ronde 2, enzovoort. De corresponderende strings/clubnamen vormen dan het echte schema.

Eisen aan een geldig schema

In een geldig competitieschema

- Speelt elke club in elke ronde precies 1 wedstrijd.
- Speelt elke club hoogstens 1 keer tegen elk van de andere clubs.
- Speelt elke ronde ongeveer de helft van alle clubs uit hetzelfde land thuis, en de andere helft van de clubs uit dat land uit. Er mag maar 1 verschil zitten tussen het aantal thuis- en het aantal uitspelende clubs zijn. Als er bijvoorbeeld vijf clubs uit Spanje meedoen, dan spelen elke ronde twee of drie daarvan thuis, en de andere drie of twee clubs spelen uit.

- Speelt elke club (ongeveer) even vaak thuis als uit. Om precies te zijn: na elke twee rondes heeft elke club precies even vaak thuis als uit gespeeld.
- Speelt elke club (ongeveer) even vaak tegen een club uit de bovenste helft van de invoer (de sterkste helft) als tegen een club uit de onderste helft van de invoer (de minst sterke helft). Om precies te zijn: na elke twee rondes heeft elke club precies even vaak tegen een club uit de bovenste helft als tegen een club uit de onderste helft van de invoer gespeeld.
- Speelt geen enkele club ooit tegen een andere club uit zijn eigen land.
- Speelt geen enkele club te vaak tegen clubs uit hetzelfde land. De gebruiker kan de variabele `keerTegenLand` invoeren. Als deze waarde bijvoorbeeld 3 is, mag bijvoorbeeld elke club van buiten Spanje tegen hoogstens drie Spaanse clubs spelen.

Als de gebruiker kiest voor de optie `omEnOm`, dan zijn twee van de eisen hierboven nog iets strenger:

- Elke club speelt om-en-om een thuiswedstrijd en een uitwedstrijd.
- Elke club speelt om-en-om tegen een club uit de bovenste helft van de invoer en tegen een club uit de onderste helft van de invoer.

Tegenstanders eerlijk verdeeld

Naast bovenstaande 'harde' eisen is er ook een zachter doel: de tegenstanders van elke club moeten een beetje eerlijk verdeeld zijn qua sterkte. Deels wordt dit al afgedwongen doordat elke club ongeveer even vaak tegen een club uit de bovenste helft als tegen een club uit de onderste helft van de invoer moet spelen. Maar binnen die helften zou je dan nog heel scheve keuzes kunnen maken. Dat proberen we te voorkomen.

Stel dat we vier rondes spelen, en dat club 7 achtereenvolgens tegen club 0, club 5, club 2 en club 4 speelt. Dan is de som van zijn tegenstanders $0 + 5 + 2 + 4 = 11$. Dit was een gemakkelijk geval, waarin de tegenstanders allemaal sterker zijn dan de club zelf. Stel nu dat club 2 achtereenvolgens tegen club 1, club 6, club 7 en club 3 speelt. Dan berekenen we niet de som $1+6+7+3$, maar de som $1+5+6+2 = 14$. Club 2 kan namelijk niet tegen zichzelf spelen en clubs 6, 7 en 3 zijn allemaal minder sterk dan club 2. We schuiven ze dan als het ware één plekje omhoog qua sterkte.

In een ideaal schema, zou de som van tegenstanders van elke club gelijk zijn. Bij dit voorbeeld met acht clubs en vier rondes zou het dan 12 zijn (ga na!). Bij een willekeurig (niet per se ideaal) geldig schema berekenen we de som van de kwadratische afwijkingen van dit gemiddelde. Club 7 levert dan een bijdrage $(11 - 12)^2$, en club 2 een bijdrage $(14 - 12)^2$. De bijdragen van alle N clubs bij elkaar opgeteld noemen we de 'waarde' van het schema. In een minimaal schema is deze waarde zo klein mogelijk.

Opdracht

Je moet een C++-programma schrijven dat **met behulp van backtracking**

- simpelweg *een* geldig schema voor een opgegeven aantal ronden construeert bij een instantie (een lijst ingelezen clubs), met opgegeven waarden voor `keerTegenLand` en `omEmOm`,

- een *minimaal* schema voor een opgegeven aantal ronden construeert voor een dergelijke instantie,

- een experiment uitvoert (zie verderop).

Voor een minimaal schema moet je in principe alle geldige schema's aflopen. Voor simpelweg *een* schema is het voldoende om één geldig schema te construeren.

Backtracking betekent dat we de vector met getallen die het huidige deelschema representeert van links naar rechts positie voor positie gaan invullen. Bij elk deelschema controleren we of het tot nu toe nog aan alle eisen van een geldig schema voldoet, dwz: of het nog kan uitgroeien tot een geldig compleet schema van het gevraagde aantal rondes. Zo nee, breid het deelschema dan niet verder uit.

Voor elke positie in de vector lopen we in principe alle clubs af, om te zien of ze daar geplaatst mogen worden. De volgorde waarin we de clubs hierbij aflopen kan bij het construeren van *een* schema invloed hebben op de kwaliteit/waarde van het gevonden schema. Je moet hierbij drie varianten implementeren:

standaard: We lopen de clubs in een vaste volgorde af, b.v. $0, 1, 2, \ldots, N-1$.

random: We lopen de clubs in een random volgorde af: voor elke positie weer een nieuwe random volgorde.

slim: We lopen de clubs in een 'slimme' volgorde af, waarbij we al rekening houden met het doel om een schema met een lage waarde te krijgen. Concreet: als we de positie van een thuisspelende club moeten vullen, kiezen we een vaste volgorde, b.v. $0, 1, 2, \ldots, N-1$. Als we de positie voor een uitspelende club moeten vullen, waarbij dus de vorige positie de thuisspelende club i bevat, dan proberen we eerst de uitspelende club(s) j die 'zo goed mogelijk' bij club i passen. Dat wil zeggen: als club i tot nu toe relatief sterke tegenstanders heeft gehad, dan mag club j wel wat zwakker zijn. En omgekeerd. Maar ook: als club i relatief sterk is, dan mag club j wel een club zijn die tot nu toe relatief zwakke tegenstanders heeft gehad. Hoe je dit precies implementeert, bepaal je zelf.

Het kan natuurlijk zijn dat we met de 'ideale' tegenstander j uiteindelijk geen compleet schema kunnen vormen. Dan moet je alsnog de andere clubs aflopen, in een slimme volgorde.

Symmetrie

De toestand-actie-boom bij het construeren van een schema kan heel groot worden, te groot om helemaal door te lopen. We kunnen echter tijdens het zoeken werk uitsparen door gebruik te maken van symmetrie. Verschillende schema's kunnen namelijk symmetrisch/equivalent zijn, en dan hoef je ze niet allemaal opnieuw te gaan maken.

Je moet tijdens het opbouwen van het schema rekening houden met de volgende twee vormen van symmetrie:

- Een ronde met wedstrijden 0-1, 2-3, 4-5, 6-7 is equivalent aan b.v. een ronde met wedstrijden 2-3, 6-7, 4-5, 0-1: dezelfde wedstrijden, alleen in een andere volgorde genoteerd. Je kunt ervoor zorgen dat je altijd maar één van zulke rondes construeert, door te eisen dat de thuisspelende clubs in een ronde, in oplopende volgorde staan. Dus wel: $0, 2, 4, 6$, maar niet $2, 6, 4, 0$.

- Als je rondes 1 en 2 verwisselt met rondes 3 en 4, blijft een geldig schema gewoon geldig, en elke club speelt nog steeds tegen dezelfde andere clubs, alleen in een andere volgorde. In het algemeen kun je zo willekeurige tweetallen rondes verwisselen met andere tweetallen, zonder het schema echt te veranderen. Als tenminste de eerste ronde van zo'n tweetal maar een oneven nummer heeft (1 en 3 in het voorbeeld hierboven).

Zowel je functie `construeerSchema` als je functie `construeerMinSchema` moet rekening houden met deze twee vormen van symmetrie. Misschien bedenk je nog een andere vorm van symmetrie waarmee je rekening kunt houden. Daar kun je maximaal 0.5 punt bonus mee verdienen.

Voor u beschikbaar

Op de website van het vak

`https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

is een pakketje beschikbaar met een skeletprogramma. In het skeletprogramma wordt een klasse `Competitie` gedefinieerd, die door het hoofdprogramma gebruikt wordt. Het programma wordt onder Linux gecompileerd met het commando `make`. Vervolgens kun je het met het commando `./Competitie` runnen. Je wordt dan gevraagd om de naam van een bestand met clubs, dat ingelezen moet worden. Hierna krijg je een menu voorgeschoteld, met de keuze om een schema te construeren, een minimaal schema te construeren of een experiment te doen voor de ingelezen clubs. Bedoeling is om de `TODO`'s in `competitie.cc`, `competitie.h` en `main.cc` uit te voeren.

In de bestanden `standaard.h` en `standaard.cc` zijn enkele standaardfuncties uitgewerkt: `integerInBereik`, `randomGetal`, `genereerRandomPermutatie` en `intKwadraat`. Je kunt die gebruiken om te testen of een integer tussen bepaalde grenzen ligt (al dan niet met een foutmelding), om een random getal tussen bepaalde grenzen te genereren, om een random permutatie te genereren of om het kwadraat van een integer te berekenen.

De meeste functies die je moet implementeren worden toegelicht in het skeletprogramma, speciaal in `competitie.h`. Goed om te weten: als er in het commentaar boven een functie `Pre:` staat, dan volgt de **preconditie** voor die functie: een aantal aannames waar je vanuit mag gaan als je in die functie binnenkomt. Je hoeft dat dus in de functie niet meer te controleren. De gebruiker van de functie moet daar van tevoren voor zorgen. Net zo staat `Post:` voor de **postconditie** van de functie: zaken die na afloop van de functie moeten gelden. Daar moet je in de functie dus voor zorgen.
Van twee functies geven we nu een nadere toelichting:

- `Competitie::construeerSchema (...)`

  Dit is de functie die dus, met behulp van backtracking, *een* geldig schema probeert te construeren voor de ingelezen clubs en het opgegeven aantal rondes. Zonder deze functie zal je oplossing voor de opdracht **nooit** voldoende worden, hoe goed de rest ook is.

  Drie opmerkingen over de parameters van deze functie:

  - `methode`
    De corresponderende actuele parameter in `main.cc` heet `volgorde`.
  - `aantalDeelschemas`
    Deze parameter moet aan het eind het aantal deelschema's bevatten dat we in de hele constructie hebben opgebouwd. **Je mag er niet vanuit gaan** dat deze parameter bij de aanroep gelijk is aan 0. De functie moet dus zelf voor deze initialisatie zorgen.

  - `schema`
    Deze parameter moet aan het eind het gevonden schema bevatten, als er een schema gevonden is. Dat is dus de lijst met clubnamen van alle wedstrijden in alle rondes, overeenkomend met de lijst met getallen die we in deze specificatie een keer als voorbeeld gebruikt hebben.

- `doeExperiment (...)`

  Deze functie moet de drie volgordes waarmee we clubs kunnen aflopen bij het vullen van een positie in het schema met `construeerSchema` vergelijken. Roep daartoe deze functie een keer aan voor de standaard volgorde, tien keer voor de random volgorde, en een keer voor de slimme volgorde. In het verslag beschrijf je dan de uitkomsten voor drie invoerbestanden. Probeer het hele experiment met zo min mogelijk input van de gebruiker uit te voeren.

Invoer

Je programma leest de deelnemende clubs, en de landen waar ze uit komen in uit een tekstbestand. Dit tekstbestand heeft het volgende formaat

- Een regel met daarop alleen een geheel getal: het aantal deelnemende clubs N.
- N regels, elk bestaande uit twee strings gescheiden door een spatie: de naam van de club, en de naam van het land waaruit de club afkomstig is. Zowel de clubnaam als de landnaam is een niet-lege string bestaande uit alleen letters en cijfers (dus ook geen spaties in de namen). De clubs zijn geordend naar sterkte, van sterkst tot minst sterk, op basis van hun resultaten in de afgelopen vijf jaar. Er kunnen meerdere clubs uit hetzelfde land meedoen, bijvoorbeeld vijf clubs uit Spanje.

De invoer is geldig als:

- N even is. We willen immers elke ronde elke club een wedstijd laten spelen.
- N minstens 2 en hoogstens 36 is.
- Alle clubnamen verschillend zijn.
- Hoogstens de helft van de clubs afkomstig is uit hetzelfde land. We willen immers nooit een club tegen een (andere) club uit hetzelfde land laten spelen.

De eerste en de laatste van deze vier eisen zijn ook te herkennen als onderdeel van de eisen aan een geldig schema. Zie eerder. Bij het skeletprogramma bevindt zich een bestand `clubs1.txt` met een geldige invoer met $N = 8$. Bij het inlezen van een tekstbestand moet je programma controleren dat aan de vier eisen is voldaan.

Enkele tips bij het programmeren

- Kijk in het skeletprogramma waar allemaal `TODO` staat, voor met name de functies die geïmplementeerd moeten worden.
- Het is verstandig om goed na te denken over datastructuren waarmee je informatie bij kunt houden, die je kunt gebruiken om efficiënt alle eisen aan een geldig schema te controleren.

- Implementeer eerst de betrekkelijk eenvoudige memberfuncties `leesInClubs` en `drukAfClubs`.
- Wellicht ten overvloede: als je in C++ `ifstream fin` gebruikt voor je invoer-tekstbestand, dan kun je heel eenvoudig een getal of een string inlezen, b.v. met

```
fin >> getal;
fin >> naam;
```

- Bij diverse memberfuncties van klasse `Competitie` moet je eerst controleren of er al een geldige lijst met clubs is. Het is handig om daarvoor een boolean membervariabele te gebruiken die in `leesIn` een waarde krijgt.

Let op: het is niet toegestaan om de headers van de gegeven public memberfuncties in `competitie.h` te veranderen. Dan zou onze automatische test (met een ander main programma) bij de beoordeling namelijk niet goed kunnen werken. Je mag wel functies toevoegen. Verder kan het inleveren van lelijke, niet-elegante, of erg inefficiënte code 0.5 punt aftrek opleveren.

Algemene opmerkingen

- Maak / behoud een verstandige klassenstructuur.
- Je klassen mogen alleen `public` membervariabelen en -functies hebben die buiten de klasse bekend moeten zijn. Andere membervariabelen en -functies moeten `private` zijn.
- Als er in je klassen dynamisch geheugen wordt gealloceerd, denk dan ook aan een destructor.
- Functies mogen niet te lang zijn (maximaal 35 regels).
- Gebruik constantes waar dat zinvol is. Kijk bijvoorbeeld in bestand `constantes.h` in het skeletprogramma.
- Het werkende programma mag er op het scherm eenvoudig uitzien, maar moet wel duidelijk zijn. De enige te gebruiken headerfiles zijn in principe `iostream`, `sstream`, `iomanip`, `fstream`, `cstring`, `string`, `vector`, `set`, `unordered_set`, `cstdlib`, `ctime` en `climits`. Wil je een andere headerfile gebruiken, vraag de docent. De headerfile `algorithm` mag in ieder geval niet gebruikt worden.
- Boven elke functie in `competitie.h` moet een commentaarblokje komen met daarin een korte beschrijving van wat de functie doet. Noem daarin tevens de gebruikte parameters: geef hun betekenis en geef aan hoe ze eventueel veranderd worden door de functie. Geef ook aan wat de memberfuncties met de membervariabelen van het object doen. Let verder op de layout (consequent inspringen) en op het overige commentaar bij de programmacode (zinvol en kort).
- Als je bij de vereiste controles in de memberfuncties een fout ontdekt, geef dan ook een passende foutmelding aan de gebruiker.
- Het programma moet onder Linux bij LIACS getest zijn en werken. Dat kan vanuit huis bijvoorbeeld op de huisuil, zie de instructie op de website.

Verslag

Het verslag moet getypt zijn in LaTeX, en moet bevatten:

- Een korte introductie, met uitleg over het probleem en de opdracht.
- Een paragraaf waarin je uitlegt
  - hoe je, voor de functie `construeerSchema`, schema's voor de competitie stap voor stap opbouwt,
  - wanneer je bij dat opbouwen welke eisen van een geldig schema controleert,
  - welke datastructuren je gebruikt om dat controleren efficiënt uit te kunnen voeren.
- Een paragraaf waarin je uitlegt hoe je programma rekening houdt met symmetrie tussen schema's. Beredeneer ook hoeveel werk het scheelt dat je rekening houdt met symmetrie, dat wil zeggen: hoeveel meer schema's (als functie van het aantal clubs N en het aantal rondes r) je zou construeren als je geen rekening hield met symmetrie.
- Een paragraaf waarin je beredeneert waarom er voor $N = 6$ geen geldig schema bestaat met twee of meer rondes, zelfs niet als alle clubs uit verschillende landen komen.

  Hint: onderscheid verschillende gevallen, afhankelijk van hoeveel clubs uit de bovenste helft van de lijst met clubs in de eerste ronde tegen clubs uit de onderste helft spelen.

  Generaliseer deze redenering vervolgens naar algemene N met $N \bmod 4 = 2$.
- Een paragraaf met resultaten van je programma voor bestand `clubs1.txt`. Zoek voor elke combinatie van parameters `keerTegenLand` (1 of 2) en `omEnOm` (true of false) uit wat het maximale aantal rondes is dat je kunt vullen, en geef voor dat aantal rondes in een overzichtelijke tabel weer:
  - hoeveel geldige schema's er te construeren zijn
  - wat de waarde van een minimaal schema is
  - hoeveel deelschema's er bekeken worden bij het construeren van een minimaal schema
  - hoeveel tijd dit kost
- Een paragraaf waarin je voor drie verschillende bestanden met respectievelijk 8, 12 en 16 clubs de uitkomst van je experiment weergeeft. Je mag hierbij zelf kiezen welke combinatie van de parameters `keerTegenLand` en `omEnOm` je gebruikt. Vermeld in een overzichtelijke tabel voor elk bestand:
  - hoeveel clubs erin staan,
  - hoeveel rondes je hebt geconstrueerd (minstens drie),
  - welke parameters `keerTegenLand` en `omEnOm` je hebt gebruikt,
  - wat de waarde is van het schema dat je met standaard backtracking hebt gevonden,
  - wat de minimale, gemiddelde en maximale waardes zijn van de schema's die je hebt gevonden bij tien runs met random backtracking,
  - wat de waarde is van het schema dat je met slim backtracking hebt gevonden.

  Lever de drie gebruikte bestanden ook in bij je inzending.

  Welke conclusie trek je uit het experiment?

Aanvullingen / tips / vragen

Eventuele verdere aanvullingen of tips bij de programmeeropdracht komen op de website van het vak

`https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

Daar is ook een subpagina met onder andere een template voor het LaTeX-verslag. De behaalde cijfers komen te zijner tijd in Brightspace te staan.

Heb je vragen over de opdracht, dan kun je die uiteraard stellen tijdens de practicumbijeenkomsten van het vak. Je kunt ook emailen naar **algoritmiek@liacs.leidenuniv.nl**

In te leveren

Via Brightspace:

- je programma (alle .cc/.h bestanden en Makefile voor Linux bij LIACS) met de drie invoerbestanden van het experiment
- en een PDF van je verslag

samen in één .zip, .tgz of .tar.gz bestand. Vermeld overal duidelijk de namen van de makers. Om via Brightspace in te leveren, moet je eerst **een groepje vormen voor deze opdracht** (ook als je geen programmeerpartner hebt). Vervolgens kun je namens dat groepje je oplossing inleveren.
Uiterste (!) inleverdatum:
Voor maximale punten: maandag 20 april 2026, 23.59 uur.
Met aftrek van 1 punt: dinsdag 28 april 2026, 23.59 uur.
Met aftrek van 2 punten: dinsdag 5 mei 2026, 23.59 uur.

Normering:

Onderdeel:	standaard	bonus voor extra symmetrie
werking	5 punten	0.25 punt
commentaar en layout	1 punt	
modulaire opbouw en OOP	1 punt	
verslag	3 punten	0.25 punt

Het is niet toegestaan om je opdracht (code en/of verslag) te maken met behulp van een large language model als ChatGPT of DeepSeek. Bij opvallende inzendingen kunnen de makers worden uitgenodigd om hun oplossing toe te lichten.