# DAKI Programmeeropdracht 6: Cykels verbreken

(Deze versie is van 28 mei 2020, 01:22)

Je moet deze programmeeropdracht **zelf** en **alleen** maken. Je mag zeker met anderen overleggen over je aanpak, maar code van anderen bekijken of overnemen of zelf code delen met anderen is uitdrukkelijk niet toegestaan. Je moet je programma schrijven in C#, en inleveren via DOMjudge.

## 1 Opdrachtbeschrijving

Nu je DAKIBOT van de nodige intelligentie hebt voorzien, vind je dat ze klaar is om jou persoonlijk te ondersteunen. Je hebt gemerkt dat je leven in veel opzichten bestaat uit het zoeken in grafen: je zoekt in je curriculum naar vakken die je kunt volgen, als je een vak volgt zoek je een geschikte weg door de stof, je zoekt in je sociale netwerk naar interessante vrienden, . . . , etc.

Het curriculum bestaat bijvoorbeeld uit vakken, maar daartussen bestaat een voorkennis-relatie: je gaat natuurlijk niet *Datastructuren en Algoritmen voor KI* volgen als je niet met vlag en wimpel voor *Modelleren en Programmeren* bent geslaagd! (Dat was een soort grapje, het is eventueel te doen.) En binnen DAKI bestaat er een relatie tussen de verschillende onderdelen van de stof: het is handig om eerst de array te leren kennen, en pas daarna te bestuderen hoe je die kunt gebruiken om een heap te implementeren. Gegeven die 'voorkennis' relatie $V \subseteq O \times O$ over $O$nderwerpen, moet je besluiten hoe je je studie gaat inrichten, en hoe je vakken gaat vullen—pardon, gaat volgen en halen. Hiervoor ga je natuurlijk DAKIBOT inzetten, die jou en al je makkers, vrinden, kompanen en bff's kan assisteren als je haar van een geschikt programma voorziet.

Je krijgt als input een gerichte graaf. Ik ga er in deze beschrijving vanuit dat die de voorkennis-relatie tussen studie-onderdelen beschrijft (maar op DOMjudge worden de knopen aangegeven met positieve gehele getallen). Dus de knopen zijn *onderwerpen*, en een kant $(u, v)$ zegt dat onderwerp $u$ vóór onderwerp $v$ moet worden bestudeerd. Je wil een volgorde weten waarin je alle onderwerpen kunt bestuderen, zodat die volgorde aan alle voorkennis-vereisten voldoet. Maar helaas zijn alle onderwerpen verdeeld over de vakken van allerlei verschillende docenten, en al die docenten specificeren redelijk autonoom de voorkennis-vereisten voor hun onderwerp(en).[1] Het kan daarom gebeuren dat er cykels in de graaf voorkomen, waardoor een onderwerp via de vakken van andere docenten z'n eigen voorkennis is!

> Tomas Klos: "Inductie leer je bij *Inleiding Logica*!"

[1] Maak je geen zorgen, dit voorbeeld is enigszins gekunsteld voor dramatisch effect. In het echt heeft een opleiding een directie en een opleidingscommissie die de consistentie en 'studeerbaarheid' van het curriculum in de gaten houden. Zoals je al hebt ondervonden, is die opleidingscommissie bevolkt door capabele en onderwijskwaliteitslievende studenten én docenten.

Johannes Korbmacher: "Inductie leer je bij *Wiskunde voor KI*!"

Ben Rin: "Inductie leer je bij *Datastructuren en Algoritmen voor KI*!"

Als je dit detecteert, kun je zo'n cykel doorbreken door op eigen houtje een onderwerp in een cykel te gaan bestuderen, zodat je in feite een kant uit de graaf verwijdert.

Stel bijvoorbeeld dat tussen de drie onderwerpen $a$, $b$ en $c$ de voorkennisrelatie $\{(a,b),(b,c),(c,a)\}$ bestaat. Je zou dan geen enkel onderwerp kunnen bestuderen, want je beschikt voor elk van de drie niet over de benodigde voorkennis. Maar je zou kunnen besluiten om in je eentje onderwerp $a$ te gaan bestuderen, zonder eerst over $c$ te hebben geleerd. Dit verwijdert in feite de kant $(c,a)$ uit de graaf. Daarna kun je $b$ doen, en daarna $c$, zodat je toch alles hebt geleerd! Hulde! Je hebt alleen in je drukke programma niet genoeg tijd om op deze manier méér dan 1 voorkennis-kant te verwijderen.

Je gaat dus voor de onderwerpen-graaf een topologische ordening maken. Als de graaf een cykel heeft kan dit niet, maar je vindt het in dat geval ook goed als je de graaf acyclisch kunt maken door met zelfstudie één kant te verwijderen. Dan rapporteer je zo'n kant, en natuurlijk alsnog ook een topologische ordening. Als het niet kan met het verwijderen van één kant, dan rapporteer je dat, en besteedt je de rest van je tijd aan protesteren bij de studentengeleding van de opleidingscommissie, die dit manco onverwijld aankaart in de eerstvolgende vergadering van de opleidingscommissie van KI, die onder andere "de kwaliteit van het onderwijs [bewaakt], eventuele knelpunten aan de orde [stelt] en adviseert over de ontwikkeling en uitvoering van het onderwijsbeleid."

# 2 Invoer en Uitvoer

Op de eerste regel van de invoer staat een positief geheel getal, het aantal grafen in de testcase. (Je doet dit natuurlijk ook voor de curricula van al je vrienden en vriendinnen, en recursief voor die van hun BFF's, met BFS gevonden in je vriendengraaf.) Daarna volgt per graaf:

- Eén regel met een positief getal $n$, het aantal knopen in de graaf. De knopen hebben de namen $0$ t/m $n-1$.
- Eén regel met een positief getal $m$, het aantal gerichte kanten in de graaf.
- $m$ regels met op elk daarvan twee getallen $a$ en $b \in \{0,1,2,\ldots,n-1\}$, steeds van elkaar gescheiden door een spatie. Er is dan een kant *van* knoop $a$ *naar* knoop $b$.

Je programma geeft per testcase de volgende uitvoer (zie sectie 3 voor een voorbeeld). Per graaf in de testcase schrijf je 1 of 2 regels.

**Als de graaf geen cykels bevat,** schrijf je één regel met de tekst '`acyclic`', gevolgd door één regel met daarop een topologische ordening van de knopen, gegeven als de namen van de knopen, gescheiden door spaties.

**Als de graaf $\geq 1$ cykel bevat, maar reparabel is door één kant $(a,b)$ te verwijderen,** schrijf je één regel met de tekst '`fix `' (inclusief spatie dus) gevolgd door de namen—gescheiden door een spatie—van twee knopen $a$ en $b$ die samen de kant $(a,b)$ vormen, gevolgd door één regel met daarop een topologische ordening van de knopen van de graaf $G = (V, E - \{(a,b)\})$, gescheiden door spaties.

**Als de graaf $\geq 1$ cykel bevat en niet reparabel is,** schrijf je één regel met de tekst '`no fix`'.

# 3 Voorbeeld

Bij de volgende input:

```
3
6
7
2 0
2 1
0 1
0 5
0 4
1 3
5 3
6
7
2 0
2 1
1 0
0 5
0 4
3 1
5 3
6
8
0 2
0 3
2 1
3 1
1 4
1 5
4 0
5 0
```

hoort *bijvoorbeeld* de volgende output:

```
acyclic
2 0 1 4 5 3
fix 0 5
2 5 3 1 0 4
no fix
```

De cykel in de tweede graaf $\langle 0, 5, 3, 1, 0 \rangle$ kan worden doorbroken door één van deze vier kanten te verwijderen. Je programma moet een kant en een bijbehorende topologische ordening geven. De topologische ordening van een dag is niet uniek, dus op DOMjudge staat geen expliciete beschrijving van de output die bij een input hoort. Er draait op DOMjudge een script dat de output van je programma neemt, en verwerkt om te checken of het een correct antwoord geeft. De derde graaf heeft vier cykels: $\langle 0, 2, 1, 4, 0 \rangle$, $\langle 0, 2, 1, 5, 0 \rangle$, $\langle 0, 3, 1, 4, 0 \rangle$, en $\langle 0, 3, 1, 5, 0 \rangle$. Deze graaf kan niet acyclisch worden gemaakt door één kant te verwijderen.

# 4 Algoritmische eisen

Je moet een algoritme voor topologische ordenen aanpassen, om het specifieke probleem in deze opdracht op te lossen.

# 5 Hints, Tips

Je kunt het op DFS gebaseerde algoritme in CLRS sectie 22.4 gebruiken. Het is handig om je algoritme, net als in het boek, in tweeën te splitsen: in het 'bovenste' deel doe je $\mathrm{DFS}(G)$ voor de gehele graaf, waarbij je achtereenvolgens op elke knoop $u$, als die nog wit is, $\mathrm{DFS\text{-}Visit}(G, u)$ aanroept. De graaf kan namelijk uit verschillende delen bestaan, en niet elke knoop in de graaf hoeft bereikbaar te zijn vanuit de eerste knoop $u$ waarop je $\mathrm{DFS\text{-}Visit}(G, u)$ aanroept. Zo wordt in het eerste voorbeeld hierboven, knoop $2$ niet bereikt met de aanroep $\mathrm{DFS\text{-}Visit}(G, 0)$, mocht dat je eerste aanroep zijn.

Je mag ook het algoritme uit exercise 22.4-5 aanpassen (maar dat raad ik niet aan!). Dat laatste algoritme wordt op de wikipedia pagina over topologisch ordenen beschreven als "Kahn's algorithm". Het is behoorlijk amazing: "A PERT network of $30\,000$ activities can be ordered in less than one hour of machine time"! OK..., dat schreef Arthur Kahn in 1962. Anyway, **check vooral** ook even de vraag aan het eind van exercise 22.4-5. Wat zijn de consequenties van het antwoord op deze vraag voor de manier waarop je dit algoritme kunt aanpassen voor het probleem in deze programmeeropdracht?