De eerste programmeeropgave — Schuiven
Algoritmiek voorjaar 2008, Universiteit Leiden

Toestand-actie-ruimte

Deze programmeeropdracht gaat over (gegeneraliseerde) eenpersoons schuifpuzzels.
Bekend is de zogenaamde 15-puzzel: in een 4 bij 4 rooster staan de getallen 1 tot en met 15 in een willekeurige volgorde. Verder is er één lege plek. Je mag een getal verschuiven naar de lege plek, mits dat getal er horizontaal of verticaal direct aan grenst. Het doel is om (zo snel mogelijk) de volgende stand te bereiken:

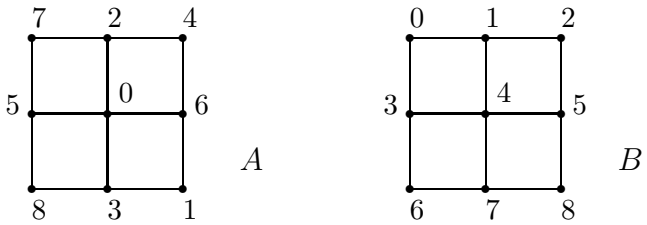
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Merk op dat er $16! = 20922789888000$ verschillende toestanden zijn. Hiervan blijkt altijd precies de helft bereikbaar vanuit een gegeven begintoestand.

Geheel analoog kunnen we de 3-puzzel, de 8-puzzel, de 24-puzzel, ... definiëren. We kunnen het probleem generaliseren tot rechthoekige schuifpuzzels, of zelfs tot grafen.

Gegeven een samenhangende ongerichte graaf met n knopen. In alle knopen zit een uniek getal (0 tot en met $n-1$) opgeslagen, in een of andere volgorde. Dat getal noemen we het *label* van de knoop. Het label 0 is speciaal: je mag een label naar een buurknoop schuiven als die buurknoop label 0 heeft. De vraag is dan of je, door herhaald te schuiven, een gegeven andere verdeling van de getallen over de knopen kunt bereiken, dat wil zeggen een andere *labelling* van de knopen. En zo ja, wat is dan de snelste manier? Voor een gegeven graaf met n knopen zijn er $n!$ labellingen mogelijk, corresponderend met alle mogelijke verdelingen van de labels 0 tot en met $n-1$ over de knopen.

Voorbeeld. Gegeven de volgende twee labellingen van de graaf $\mathcal{G}$ die een 3 bij 3 rooster modelleert:

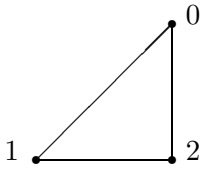


Kun je vanuit toestand (= labelling van $\mathcal{G}$) A toestand B bereiken door volgens de spelregels de getallen rond te schuiven? Merk op dat deze graaf met labellingen correspondeert met standen van de 8-puzzel.

Bij deze programmeeropdracht moet een kort **verslag** gemaakt worden, waarin de volgende vragen/opdrachten beantwoord/uitgewerkt moeten worden. Het verslag is bij voorkeur getypt (bijv. in LaTeX), maar mag ook handgeschreven zijn; als het maar duidelijk is, zowel wat betreft inhoud als leesbaarheid.

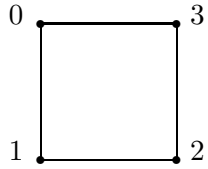
1. Teken voor onderstaande drie grafen met aangegeven labelling van de knopen (het deel van) de toestand-actie-ruimte vanuit die beginsituatie. Naast elke graaf staat de betreffende toestand in een verkorte notatie. Hint bij het overzichtelijk tekenen: teken de toestanden nivo voor nivo en nummer de verschillende toestanden. Dezelfde toestand hoef je maar één keer uit te werken. Zijn alle mogelijke toestanden bereikbaar vanuit de begintoestand?

a.



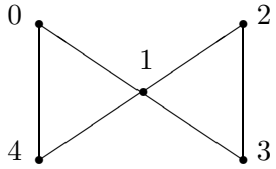
0
1 2

b.



0 3
1 2

c.



0 2
1
4 3

2. Geef de serie van achtereenvolgende verschuivingen die je moet doen om zo snel mogelijk vanuit stand I in stand II (zie hieronder) te komen. De onderliggende graaf is die uit **b**.

I: 0 1 / 2 3 II: 3 2 / 0 1

3. Beredeneer hoe je zo snel mogelijk vanuit stand III in stand IV (zie hieronder) kan komen. Gebruik hierbij alleen het deel van de toestand-actie-ruimte dat je in **1.** hebt getekend. De onderliggende graaf is weer die uit **b**.

III: 0 2 / 1 3 IV: 3 0 / 2 1

4. De volledige toestand-actie-ruimte die alle labellingen van de graaf uit **b.** bevat valt uiteen in twee samenhangscomponenten. Beredeneer hoeveel samenhangscomponenten de volledige toestand-actie-ruimte behorend bij de graaf uit **c.** heeft.

Programma

Er moet een (brute force) programma worden geschreven dat voor een samenhangende graaf met bijbehorende beginlabelling een lijst oplevert met alle toestanden (= labellingen van de graaf) die vanuit die beginsituatie te bereiken zijn. Verder moet berekend worden hoeveel verschuivingen de kortste "route" tussen twee labellingen gebruikt. De gebruiker moet het spel voor rechthoekige schuifpuzzels kunnen spelen.

De gebruiker moet in het begin kunnen kiezen of een graaf wordt ingelezen uit een invoerfile (optie 1), of dat hij/zij een rechthoekige m bij n schuifpuzzel invoert (via toetsenbord of

invierfile) waarop het spel gespeeld kan worden (optie 2).

1. Het programma leest eerst een graaf in uit een invoerfile. Op de eerste regel van de invoerfile staat het aantal knopen n en het aantal takken van de graaf. Daarna volgen de takken van de graaf: op elke regel één, gegeven als een paar i j met i en j knoopnummers. De knopen zijn genummerd met 0 t/m $n-1$. Vervolgens wordt de beginlabelling ingelezen (nieuwe invoerregel). Deze bestaat uit een permutatie van de getallen, 0 tot en met $n-1$, waarbij het eerste getal het label van knoop 0 voorstelt, het tweede het label van knoop 1, etcetera. Voorbeeld: 3 4 0 2 1 betekent dat knoop 0 label 3 heeft, knoop 1 label 4, knoop 2 label 0 en knoop 3 label 2.

   Op een of andere manier moeten (bijvoorbeeld in een enkelverbonden lijst, maar het kan ook anders en efficiënter) alle labellingen worden opgeslagen die vanuit de beginlabelling bereikbaar zijn. Schrijf daartoe een (al dan niet recursieve) functie `tebereiken`, die dit realiseert. Let op: voordat je een toestand opbergt moet je steeds eerst in je lijst kijken of je die toestand al gehad hebt. Sla de toestand altijd op voordat je de vervolgtoestanden onderzoekt. Het aantal bereikbare toestanden moet afgedrukt worden.

   Verder moet herhaald voor twee door de gebruiker in te voeren labellingen ℓ_1 en ℓ_2, *mits* deze beide bereikbaar zijn vanuit de oorspronkelijke beginlabelling, kunnen worden bepaald wat het minimale aantal verschuivingen is dat van de ene naar de andere labelling leidt. Dit kun je doen door ℓ_1 op te zoeken in de opgeslagen bereikbare standen, deze met afstand 0 te nummeren, en vervolgens alle standen die in één zet bereikbaar zijn op te zoeken en afstand 1 te geven, de direct bereikbare toestanden van daaruit afstand 2, etcetera, tot je ℓ_2 bereikt hebt. Het kan hierbij van belang zijn dat je de toestanden op een handige manier hebt opgeslagen, zodat het zoeken niet te lang duurt. Er moet overigens eerst worden gecontroleerd dat de ingevoerde labellingen inderdaad bereikbaar zijn vanuit de beginlabelling. Zo niet, dan moet dit op het scherm medegedeeld worden.

2. Het rechthoekige bord wordt gegeven door m en n en de inhoud van de vakjes van het m bij n- rooster. Het rooster wordt op het scherm getoond en de gebruiker kan een schuifzet doen, in te lezen van het toetsenbord, waarna steeds de vervolgstand wordt getoond, Voordat het zetten begint moet worden gemeld of de (gebruikelijke) eindstand bereikbaar is en hoeveel verschuivingen het minimaal kost om deze te bereiken. Voor dit alles moet de rechthoek dus als graaf worden geïnterpreteerd en daarin worden omgezet, zodat de bij optie 1. te schrijven functies gebruikt kunnen worden.

Opmerkingen:

- Kies een verstandige klasse-structuur. Gebruik een klasse `graaf`, om de graaf(structuur) in op te slaan, en een geschikte klasse `opslag` waarin alle bereikbare toestanden worden opgeslagen (eventueel twee verschillende opslagstructuren indien dat beter uitkomt). Een toestand bevat in elk geval de betreffende labelling van de graaf, en de positie van de 0 daarin (dus het knoopnummer van de knoop waarin de 0 zit). Denk ook aan de constructor en de destructor (indien nodig).

- Gebruik een constante `MAX` om het maximale aantal knopen mee aan te geven. Deze moet niet te groot zijn. Afhankelijk van het aantal takken in de graaf en de gekozen implementatie van de opslag kan 10 al te groot blijken te zijn.

- Het werkende programma mag er op het scherm eenvoudig uitzien, maar moet natuurlijk wel duidelijk zijn. Gebruik bij voorkeur alleen `iostream` en `fstream` (voor het uit file lezen). Inlezen van getallen mag gewoon met `cin` en `invoer >>`. Je mag ervan uitgaan dat de in te lezen getallen in het juiste bereik zitten, en dat een rijtje getallen ook inderdaad en goede labelling voortstelt.

- Leg in je verslag kort maar duidelijk uit hoe je programma werkt. Het gaat hier met name om het bepalen van de bereikbare labellingen, hoe die zijn opgeslagen en waarom (!), en het bepalen van het kleinste aantal verschuivingen dat je kan doen om van de ene naar de andere labelling te komen.

- Boven elke functie moet een commmentaarblokje komen met daarin een (zeer) korte beschrijving van wat de functie doet. Noem daarin ook de gebruikte parameters en geef aan hoe ze eventueel veranderd worden door de functie. Let ook op de layout (consequent inspringen) en op het overige commentaar bij de programmacode (alleen zinvol en kort commentaar).

- Op de website zullen enkele voorbeeldgrafen met bijbehorende uitvoer komen te staan. Verder zal daar een ranglijst worden bijgehouden met de (executietijden van de) snelste programma's.

- *Voor de liefhebbers*: bepaal het minimale aantal verschuivingen nodig om van de ene toestand naar de andere te komen voor *willekeurige* toestanden. Hierbij mag alleen gebruik gemaakt worden van de opgeslagen toestanden die vanuit de ingelezen labelling bereikbaar zijn.

- Informatie over de programmeeropdracht is te vinden op:
  `http://www.liacs.nl/home/jlaros/edu/alg.shtml`.
  Voor meer informatie over het vak, laatste wijzigingen, etcetera, raadplege men:
  `http://www.liacs.nl/home/graaf/ALGO/algo2008.html`

Uiterste inleverdatum: vrijdag 28 maart 2008. Voor elke week te laat inleveren gaat er een punt van het cijfer af.
Het programma per e-mail sturen aan de hoofdnakijker: `jlaros@liacs.nl`. Listing en verslag moeten op papier worden ingeleverd in de daartoe bestemde doos met opschrift Algoritmiek in de postkamer van Informatica, kamer 156. Vermeld overal duidelijk de namen van de makers.

Normering: verslag 2; commentaar en layout 1,5; modulaire opbouw en OO: 1,5; werking 5