De eerste programmeeropgave — Bynum game
Algoritmiek voorjaar 2009, Universiteit Leiden

Toestand-actie-ruimte en strategie

Het tweepersoonsspel Bynum game wordt gespeeld met $m * n$ speelkaarten (of eenhapscakejes, of kiezelsteentjes, of ...), die in een m bij n rechthoek (m rijen en n kolommen) worden gelegd. Er zijn twee spelers, V (Verticaal) en H (Horizontaal), die om de beurt een zet doen. We spreken af dat V begint.
Als speler V een zet doet neemt deze steeds een hele kolom weg, speler H een hele rij. Door het wegnemen van een rij of kolom uit een rechthoek zal in het algemeen die rechthoek in twee kleinere rechthoeken worden verdeeld. Indien van de rand wordt weggehaald, resteeert één kleinere rechthoek. Bij aanvang van het spel is er slechts één rechthoek, maar tijdens het spel worden dit er meer. Als een speler aan de beurt is neemt deze een kolom (V) of rij (H) weg uit precies één van die rechthoeken. Een zet bestaat dus uit het aanwijzen van een kolom of rij in een van de rechthoeken, en die daar vervolgens weghalen. De speler die de laatste kaart(en) weghaalt heeft gewonnen.

Een voorbeeld van een mogelijk spelverloop, met $m = 3$ en $n = 4$:

```
X X X X     V      X   X X     H      X   X X     V      X   X       H     X   X
X X X X    --->    X   X X    --->    X           --->    X           --->  X
X X X X            X   X X            X   X X            X   X X         X
  (*)                                                                    (**)
```

In het spelverloop hierboven geven de `X`-en de speelkaarten aan. In toestand `(**)` is V aan de beurt. Als deze in de volgende zet de meest linker kolom weghaalt verliest hij. Veronderstel dus dat hij de derde kolom (één speelkaart) weghaalt. Dan blijft er een toestand over die winnend is voor H. (Zij neemt namelijk de middelste rij (één speelkaart) en haalt dan in haar volgende zet de laatste kaart weg.) Stand `(**)` is derhalve verliezend voor V (ofwel winnend voor H), want zijn twee mogelijke zetten leiden allebei tot verlies.

Bij deze programmeeropdracht moet een kort **verslag** gemaakt worden, waarin de volgende vragen/opdrachten beantwoord/uitgewerkt moeten worden. Het verslag is bij voorkeur getypt (het liefst in LaTeX), maar mag ook handgeschreven zijn; als het maar duidelijk is, zowel wat betreft inhoud als leesbaarheid. Zie www.liacs.nl/mwitsenb/Algoritmiek/ voor enkele richtlijnen bij het maken van het verslag.

Bij het tekenen van toestand-actie-ruimtes, zoals hieronder gevraagd, mag je rekening houden met de symmetrie van het probleem. Bijvoorbeeld: vanuit de beginstand `(*)` in bovenstaand voorbeeld, zijn 4 zetten van V mogelijk. Echter zowel het weghalen van de tweede kolom als de derde kolom laat een 3 bij 1 en een 3 bij 2 rechthoek over. Je mag daarom in je toestand-actieplaatje een van beide zetten weglaten. Iets soortgelijks geldt voor het wegnemen van de eerste of de vierde kolom: beide zetten laten een 3 bij 3 rechthoek over. Verder gaan we ervan uit dat V altijd begint.

1. Teken de toestand-actie-ruimte voor de gevallen $m = 2, n = 4$ en $m = n = 3$. Geef bij elke toestand aan of deze winnend is voor V of voor H. Bepaal zo de winnende zet en de winnende strategie.
2. Teken de toestand-actie-ruimte voor de gevallen $m = 4, n = 2$; $m = 5, n = 2$; $m = 6, n = 2$ en $m = 7, n = 2$. Geef bij elke toestand aan of deze winnend is voor V of voor H. Bepaal zo de winnende zet en de winnende strategie. Je hoeft hier een toestand waarvan je al gezien hebt of die winnend is voor V dan wel voor H, niet meer helemaal uit te werken.
3. We bekijken nu het algemene geval waarin begonnen wordt met een m bij 2 rechthoek. Bewijs de volgende beweringen en geef in beide gevallen aan hoe men moet spelen om te winnen:

1. als m even is, dan is het spel winnend voor V.
2. als m oneven is, dan is het spel winnend voor H.

Hint: gebruik inductie.

Programma

Er moet een (brute force) programma worden geschreven dat voor Bynum game met willekeurige m en n de/een winnende zet oplevert voor de beginnende speler. In ons geval is de beginnende speler altijd degene die kolommen weghaalt (V dus); zijn tegenstander (H) neemt rijen weg. De resultaten moeten voor zo veel mogelijk waarden van m en n in een tabelletje in het verslag worden opgenomen. Verder moet het spel gespeeld kunnen worden tegen de computer.

Het programma vraagt de gebruiker om een waarde voor m en n in te voeren. Vervolgens berekent het programma wat een/de winnende zet is door voor alle mogelijke directe vervolgstanden een *recursieve* boolese functie `winst` aan te roepen, die bepaalt of de betreffende stand winnend is voor de speler die aan de beurt is of niet. Nog beter is het om een recursieve functie te schrijven die meteen de winnende zet bepaalt (zie de opmerkingen hieronder). Merk op dat een stand winnend is voor degene die aan de beurt is dan en slechts dan als een van zijn directe vervolgstanden niet winnend is voor de tegenstander. Druk de/een winnende zet af op het beeldscherm.
Vervolgens kan het spel —indien de gebruiker dat wil— gespeeld worden tussen de gebruiker (die begint) en de computer. Zetten van de gebruiker worden ingevoerd door de coördinaten (i, j) te geven van het onderste vakje uit de kolom die hij weghaalt. Daarmee ligt zijn zet helemaal vast. De zetten van V in het voorbeeld worden zo aangegeven met respectievelijk $(2, 1)$ en $(0, 3)$. Druk bij elke tussenstand waarbij de gebruiker aan de beurt is een mogelijke winnende zet af op het beeldscherm. Als een stand niet winnend is voor de gebruiker, moet dit ook gemeld worden.
De computer moet optimaal spelen, dat wil zeggen: als er een winnende zet voor hem bestaat doet hij deze. Voor het vinden van zo'n winnende zet kan de functie `winnendezet` weer worden aangeroepen. Als er in een zekere toestand geen winnende zet bestaat, mag de computer een willekeurige zet doen: bedenk maar wat. Zijn er meer winnende zetten mogelijk, kies er dan een.

Opmerkingen:

- Kies een verstandige klasse-structuur (genaamd `bynum`), met als member-functies in elk geval de recursieve functie `winst`, een functie `winnendezet` die de/een winnende zet

bepaalt en `drukaf` (voor het afdrukken van een stand). Ook degene die aan de beurt is moet als membervariabele worden opgenomen. Denk verder aan de constructor en de destructor (indien nodig).

- Laat de memberfunctie `winnendezet` bijvoorbeeld $(-1, -1)$ opleveren als er geen winnende zet bestaat. Overigens kun je natuurlijk ook de functie `winnendezet` zelf recursief maken en dus recursief een winnende zet laten bepalen; dan heb je geen aparte functie `winst` nodig. Dit verdient de voorkeur.

- Implementeer het Bynum-rooster bijvoorbeeld met behulp van een `MaxM` bij `MaxN` array, waarbij `MaxM` en `MaxN` niet te groot zijn (denk aan een orde van grootte van het totaal aantal vakjes 40). Er zijn overigens wel andere en misschien efficiëntere implementaties van het rooster te bedenken.

- Bij het invoeren van m en n en de zetten van de gebruiker mag gewoon `cin` worden gebruikt (je mag aannemen dat er een geheel getal is ingevoerd). Er moet wel worden gecontroleerd dat de gebruiker niet te grote m en n kiest en dat hij een geldige zet invoert, dus bijvoorbeeld dat $m \leq$ `MaxM`, dat $0 \leq i \leq$ `m-1` en dat het vakje (i, j) nog niet leeg is.

- Het werkende programma mag er op het scherm eenvoudig uitzien (gebruik bij voorkeur alleen `iostream.h`), maar moet natuurlijk wel duidelijk zijn.

- Probeer een programma te schrijven dat voor zo groot mogelijke rechthoeken binnen redelijke tijd kan bepalen of die winnend of verliezend zijn voor V (met winnende zet). Op de website zal een ranglijst worden bijgehouden met de grootte van de rechthoeken die de ingeleverde programma's aankunnen, en de bijbehorende tijden.

- Leg in je verslag duidelijk (maar niet te uitgebreid) uit hoe je recursieve functie werkt en wat je eventueel extra hebt gedaan om het probleem voor grotere rechthoeken te kunnen oplossen.

- 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).

- Voor eventuele aanvullingen of tips bij de programmeeropdracht, zie:
  `http://www.liacs.nl/home/mwitsenb/Algoritmiek/`. Hier komt t.z.t. ook de bovengenoemde ranglijst te staan, alsmede de behaalde cijfers.
  Voor meer informatie over het vak, laatste wijzigingen, etcetera, raadplege men:
  `http://www.liacs.nl/home/graaf/ALGO/`

Uiterste inleverdatum: vrijdag 20 maart 2009. Voor elke week te laat inleveren gaat er een punt van het cijfer af.
Het programma per e-mail sturen aan de hoofdnakijker: `mwitsenb@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 3; commentaar en layout 1,5; modulaire opbouw en OO: 1,5; werking 4