Algoritmiek Programmeeropdracht 2

Johanna Dekker (s1771337)

6 april 2022

1 Introductie

Klaverjassen is een kaartspel dat met vier personen wordt gespeeld, waarbij twee spelers tegen de andere twee spelen. Deze vier personen zitten aan een tafel, waarbij de teamgenoten tegenover elkaar zitten. Bij een klaverjastoernooi is het dan de bedoeling om een schema te maken van tafels per ronde. Iedere deelnemer wilt één keer met iedere andere deelnemer in het team zitten, en twee keer tegen iedere deelnemer spelen. Dit is alleen mogelijk wanneer het aantal deelnemers gelijk 0 of 1 modulo 4 is. e programmeeropdracht van het vak Algoritmiek, voorjaar 2022.

2 Backtracking

Hoe ik backtracking heb geimplementeerd is zonder recursie. In plaats daarvan gebruik ik de functie *addSpeler*. Deze houdt een teller bij, putHere. Deze variabele wordt geïnitialiseerd bij het inlezen of creeëren van een nieuw schema. Iedere keer als er een speler wordt toegevoegd, wordt er 1 bij putHere opgeteld zowel als bij speler (waar de modulus wordt gebruikt zodat er nooit een ongeldige speler wordt ingevoerd). In de functie *bepaalSchemaBT* wordt er gecontroleerd of het schema met de nieuwe speler nog geldig is. Zo niet, wordt er 1 afgetrokken bij putHere zodat de volgende keer dat *addSpeler* wordt aangeroepen de speler op dezelfde plek wordt gezet. Indien het schema geldig is, returnt de functie true. De functie *addSpeler* zorgt er ook voor dat mijn matrices maatGeweest, tegenstanderGeweest, en vrijGeweest weer worden geüpdate. Deze matrices is hoe ik bepaal of het schema geldig is.

Deze functies gaan de 1D array af, en roepen vervolgens *getSpelerAt* aan. Het probleem van de 1D array was dat het zeer lastig te onderscheiden was waar een tafel begon en eindigde ten opzichte van de index van de array. Met de functie die ik heb geschreven kan er per tafel worden gekeken naar welke spelers samen en tegen elkaar spelen. Dit doe ik door middels van een pointer die het beginpunt van een gegeven tafel aangeeft. Dit is ook hoe ik rekening houd met symmetrie.

Mijn grote probleem is dat ik binnen mijn functie bepaalSchemaBT geen gebruik kan maken van *isOngeldigSchema*. Ik kreeg telkens de melding 'zsh: segmentation fault ./schema.'. Oftewel, ergens gebruik ik een illegal memory location. Wat er gebeurt is als ik bepaalSchemaBT oproep, dat de invoer verdwijnt. Hiermee bedoel ik dat bij $ds8_1 nadefunctietehebbenaangeroepenschema[0]totschema[15](eerstetweeregels)verdwij$

3 Symmetrie

In mijn programma maak ik 3 matrices aan: maat, tegen, en vrij. Deze worden aangemaakt door naar de hele array te kijken. In plaats van dit direct via de index te doen, loop ik over iedere tafel en alle posities daarin. Het aantal tafels is geïnitialiseerd bij het iunlezen/aanmaken van een schema. De functie *getSpelerAt* houdt rekening met een tafel door het gebruik van 0 mod 4. Op deze manier kan je je partner vinden door je eigen positie + 2 te doen. Zonder iets in deze geest te doen zal de speler op tafel 0 op positie 3 iemand op de volgende tafel aanzien als partner. Hierdoor kan het programma altijd correct zien wie de tegenstanders zijn en wie de partner is.

De matrices werken als volgt. Uit schema wordt de juiste waarde voor de speler gehaald, en wordt opgeslagen in 'ik'. Een partner is iemand die +2 plekken van 'ik' verwijderd is. Deze twee waardes worden gebruikt om in de matrix de desbetreffende positie op te hogen. Bijvoorbeeld, Indien spelers 2 en 3 elkaars partner zijn in het schema, dan zal er in de matrix *maat* op posities [3][2] en [2][3] een 1 staan. Dat was ook de hoofdreden waarom ik heb gekozen om met de voorgenoemde drie matrices te werken. Op deze manier herkent hij de symmetrie. Alle matrices worden aangemaakt met een loop door de hele array. In feite controleert hij dus per tafel al alle symmetrische mogelijkheden.

4 Zoekruimte

Met een aantal spelers n, is het aantal rondes gelijk aan (n-1). Het aantal tafels is gelijk aan (n/4)(n-1), het aantal spelers gedeeld door 4 maal het aantal rondes. Per tafel zijn er 3 schemas die feitelijk gelijk zijn, en per ronde is het aantal schemas die feitelijk gelijk zijn evenveel als het aantal tafels.

In totaal, lijdt dat tot het volgende: (3(n/4)(n-1)+(n/4). 3 maal het aantal tafels + het aantal tafels maal het aantal rondes.