ZAAWANSOWANE PROGRAMOWANIE KOMPUTEROWE WNE (2023)

KRZYSZTOF ZIEMIAŃSKI

12. Gry

12.1. **Gra w patyczki.** Gra w patyczki to gra dla dwóch osób. W grze używa się patyczków, które leżą na kilku stosach. Gracze na przemian wykonują ruchy; każdy ruch polega na zdjęciu pewnej liczby patyczków (przynajmniej jednego) z jednego ze stosów. Gracz, który zostanie zmuszony do zdjęcia ostatniego patyczka przegrywa. Naszym celem będzie napisanie programu, który gra (dobrze) w patyczki.

Aktualny stan gry można zapisać jako ciąg liczb, np. `7531` oznacza, że w danym momencie mamy cztery stosy: na jednym jest 7, na drugim 5, a trzecim 3, a na ostatnim 1 patyczek. Ruchy będziemy zapisywać za pomocą strzałek z zaznaczonym graczem wykonującym ruch. Przykładowy przebieg partii:

$$\texttt{7531} \xrightarrow{A} \texttt{6531} \xrightarrow{B} \texttt{6131} \xrightarrow{A} \texttt{611} \xrightarrow{B} \texttt{111} \xrightarrow{A} \texttt{11} \xrightarrow{B} \texttt{1} \xrightarrow{A} \texttt{0}$$

W pierwszym ruchu gracz A zabiera jeden patyczek z pierwszego stosu, w drugim ruchu B zabiera 4 patyczki ze stosu 2. Następnie gracz A zabiera wszystkie patyczki ze stosu 3, a gracz B — 5 patyczków ze stosu 1. Następne ruchy są wymuszone: każdy z graczy musi zabierać po jednym patyczku. Przegrywa gracz A, który jest zmuszony do zabrania ostatniego patyczka.

12.2. **Jak odróżnić dobry ruch od złego?** Jaki ruch wybrać? Na pewno dobre są ruchy, które pozostawiają na planszy tylko jeden patyczek: przeciwnik będzie zmuszony go zabrać i przegra. Inaczej mówiąc, stan `1` jest **przegrywający**: jeśli się w nim znajdziemy to przegramy. Stany `11`, `21`, `31`, `2`, `3` są natomiast **wygrywające**, bo możemy wykonać ruch, który prowadzi do stanu przegrywającego. Oczywiście znalezienie się w stanie wygrywającym nie gwarantuje zwycięstwa — możemy popełnić błąd i wybrać ruch prowadzący do stanu wygrywającego i wtedy przeciwnik dostanie swoją szansę. Na przykład ze stanu `31` możemy wykonać ruchy do stanów `3`, `21`, `11` i `1`, ale tylko ten ostatni ruch jest dobry. Widzimy więc, że kluczowa jest wiedza, które stany są wygrywające.

Przyjmujemy, że stan `0` jest wygrywający (jeśli się w nim znaleźliśmy, to już wygraliśmy). Możemy więc rodzaj stanu wyliczyć przy pomocy następującego algorytmu rekurencyjnego:

```
bool wygrywa(Stan s) {
    if(s==0)
        return true;
    for(t: istnieje ruch z s do t)
        if(!wygrywa(t))  // t jest przegrywający
            return true;
    return false; // nie znaleźliśmy dobrego ruchu
}
```

Oczywiście warunek "istnieje ruch z s do t" należy odpowiednio zaimplementować.

Kiedy mamy już wiedzę o tym, które stany wygrywają, możemy napisać program, który dobrze gra. Jeśli jesteśmy w stanie wygrywającym, przechodzimy do stanu przegrywającego (dla przeciwnika), jeśli nie, wykonujemy dowolny ruch i liczymy na błąd (pewnie w praktyce dobrze jest zdjąc jak najmniej patyczków).

12.3. **Ograniczenia.** Powyższy algorytm można zastosować do każdej dwuosobowej gry z pełną informacją. Przykładami takich gier są szachy, warcaby oraz Go. Używając funkcji `wygrywa` opisanej powyżej można napisać program, który będzie grał optymalnie w każdą w tych gier. Jedynym (ale za to poważnym) problemem jest to, że raczej nie doczekamy się na zakończenie działania tej funkcji, bo liczba stanów do przeanalizowania jest zbyt duża. W przypadku gry w patyczki (o ile patyczków jest nie więcej niż kilkanaście) funkcja działa zadowalająco szybko. Można jej działanie nieco przyspieszyć:

- Łatwo zauważyć, że rodzaj wielu stanów jest wyliczany wielokrotnie. Można temu zapobiec zapisując raz wyliczone stany w tablicy (tak jak w lekcji o rekurencji).
- Zamiana kolejności stosów nie wpływa na to, czy stan jest wygrywający czy przegrywający (np. oba stany `31` i `13` są wygrywające). Wystarczy więc zapisywać tylko "posortowane" stany.

Wprowadzenie tych zmian oczywiście wiąże się ze znaczną komplikacją programu.

12.4. **Zadania.**

(1) Zmodyfikować program tak, aby umożliwiał grę w patyczki z komputerem.

(2) Zaimplementować jakąś inną grę dla dwóch graczy.