Backtracking

dr. Walter Kosters, Universiteit Leiden

Capelle — vrijdag 17 april 2009

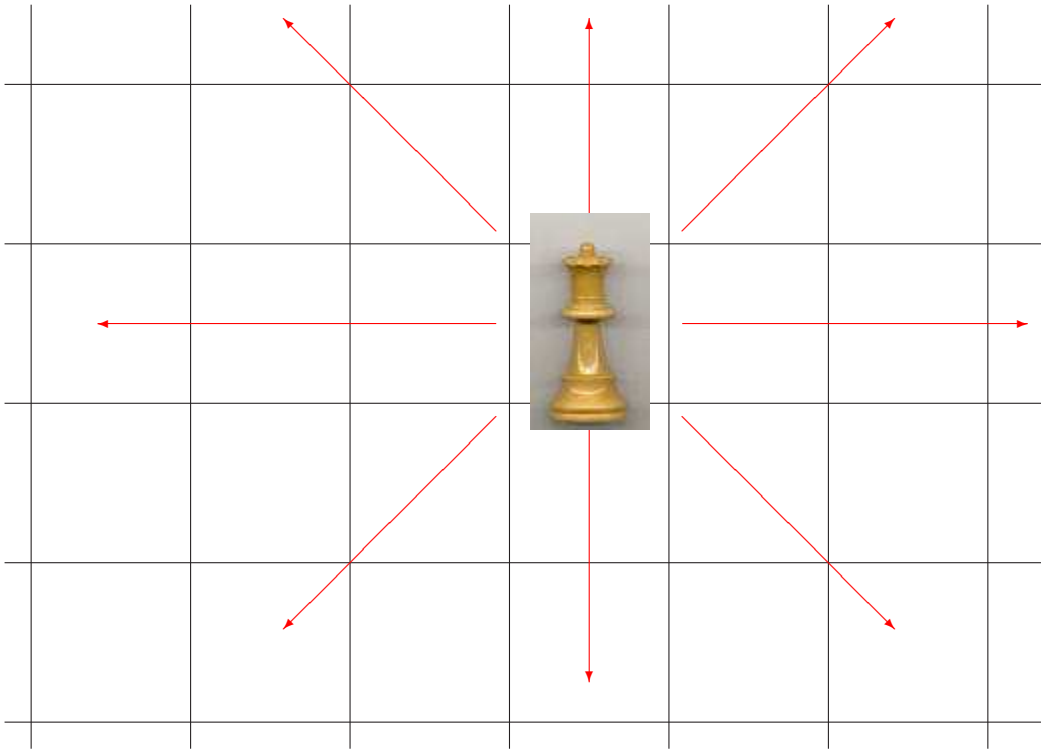
www.liacs.nl/home/kosters/gastlessen/

Backtracking is een techniek die je helpt om uit doolhoven te komen — en soortgelijke problemen op te lossen.

Er zijn veel algoritmen (rekenvoorschriften, stappenplannen) die met backtracking werken, onder meer in de robotica, en ook bij het programmeren van spellen.
Het basisidee is: als je bij het zoeken vastloopt, ga je terug op je pad om het laatste "open" alternatief te proberen.

Probleem

We willen nu het volgende probleem oplossen: zet 8 dames op een 8×8 schaakbord, zodanig dat geen dame een andere in één keer kan slaan.

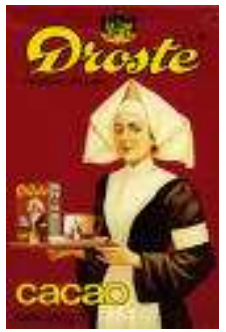


Het probleem kan systematisch met behulp van het volgende idee aangepakt worden:

als er nog een lege rij is

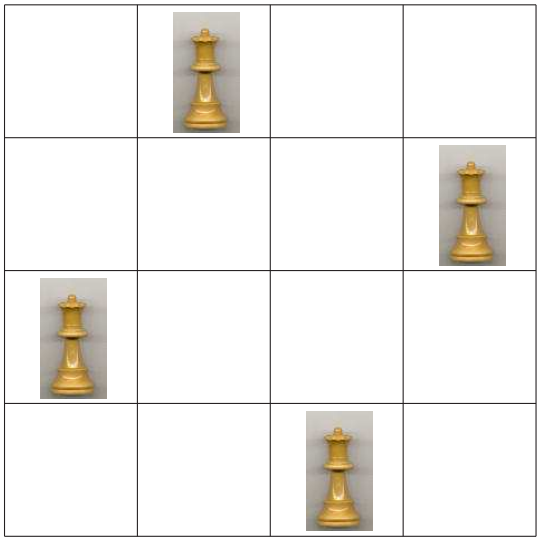
probeer in elke kolom een dame te zetten

als dat mag: ga door . . .



Samengevat: je zoekt steeds verder zolang het nog kan ("depth first search") en gaat — als het mis loopt — terug ("backtracking") naar de laatste plek waar je nog een keuze open hebt staan.
Een heuristiek kan helpen bij de keuze van de volgende rij.

Op deze manier worden alleen “redelijke” kandidaten bekeken, allemaal één keer, en komt alles aan de beurt.



`www.liacs.nl/home/kosters/gastlessen/dame.html`

Vaak wordt backtracking uitgeprogrammeerd met behulp van recursie. Het is namelijk zo dat je soms het oorspronkelijke probleem als een deelprobleem herkent.

Een eenvoudig voorbeeld van recursie vind je bij het sorteren van een rijtje getallen. Zoek het kleinste getal op, zet dit vooraan, en (recursie!) sorteer de rest. Bijvoorbeeld: $7\ 5\ 9\ 2\ 6\ 4\ 4 \rightarrow 2 \cdot 7\ 5\ 9\ 6\ 4\ 4 \rightarrow 2\ 4 \cdot 7\ 5\ 9\ 6\ 4 \rightarrow \ldots$

Let er wel op dat de recursie een keer moet afbreken: in een eenvoudig basisgeval. Bij sorteren is dat wanneer je nog maar één getal over hebt. En bij de Russische poppetjes bij het kleinste poppetje.

De wiskunde maakt ook gebruik van recursieve definities. Zo wordt $n!$ (n faculteit) vaak netjes gedefinieerd als:

$$n! = \begin{cases} 1 & \text{als } n = 0 \\ n \times (n-1)! & \text{als } n > 0 \end{cases}$$

in plaats van

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

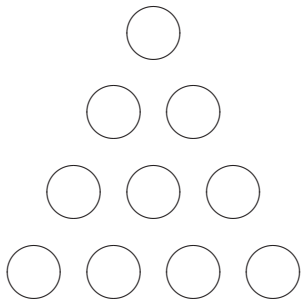
Er geldt namelijk zeker niet

$$4! = 4 \times 3 \times 2 \times \cdots \times 3 \times 2 \times 1$$

Hoe programmeer je recursie? Laten we een voorbeeld bekijken in de programmeertaal C++.

De som van de getallen 1 tot en met 100 is gelijk aan 100 plus de som van de getallen 1 tot en met 99, en die laatste som is weer 99 plus de som van de getallen 1 tot en met 98, . . . enzovoort. In C++:

```
int som (int n) {
   if ( n == 1 ) return 1;  // basisgeval
   else return ( n + som (n-1) );
}  // som
```



C++ gebruikt functies, bijvoorbeeld een functie `geenaanval` die controleert of de dame op de zoveelste rij geen problemen heeft met dames op eerdere rijen.

De belangrijkste functie is `zetdames`. De recursieve functie probeert een dame in de `rij`-de rij te zetten. Het idee is:

- Als het de 9-de rij is (van een 8×8 bord) hebben we blijkbaar een nieuwe oplossing gevonden: succes! We drukken deze stand af — en turven hem.

- Anders: probeer één voor één alle kolommen. Als we ergens een dame neer mogen zetten (gebruik `geenaanval`) doen we dat, en gaan op dezelfde manier door: recursie!

De functie zetdames in C++

```
void zetdames (int grootte, int rij, int S[ ], int & teller) {
  int kolom;
  if ( rij == grootte + 1 ) {
    drukaf (grootte, S);
    teller++;
  }  // if
  else
    for ( kolom = 1; kolom <= grootte; kolom++ ) {
      S[rij] = kolom;
      if ( geenaanval (rij, S) )
        zetdames (grootte, rij+1, S, teller);
    }  // for
}  // zetdames
```

Het gehele C++-programma

```
#include <iostream>
using namespace std;
const int MAX = 20;
bool geenaanval (int rij, int S[ ]) {
  bool veilig = true; int hulprij = 1;
  while ( veilig && ( hulprij < rij ) ) {
    veilig = ( ( S[rij] != S[hulprij] ) && ( S[rij] - S[hulprij] != rij - hulprij )
             && ( S[rij] - S[hulprij] != hulprij - rij ) );
    hulprij++; }//while
  return veilig; }//geenaanval
void drukaf (int grootte, int S[ ]) {
  for ( int i = 1; i <= grootte; i++ ) cout << S[i] << " ";
  cout << endl; }//drukaf
void zetdames (int grootte, int rij, int S[ ], int & teller) {
  int kolom;
  if ( rij == grootte + 1 ) { drukaf (grootte, S); teller++; }//if
  else
    for ( kolom = 1; kolom <= grootte; kolom++ ) { S[rij] = kolom;
      if ( geenaanval (rij, S) ) zetdames (grootte, rij+1, S, teller); }//for
}//zetdames
int main ( ) {
  int S[MAX]; int grootte; int teller = 0;
  do {
    cout << "Grootte schaakbord ( < " << MAX << " ) .. "; cin >> grootte;
  } while ( grootte < 1 || grootte >= MAX );
  zetdames (grootte, 1, S, teller);
  cout << endl << "Aantal: " << teller << endl << endl; return 0;
}//main
```

Op het 8×8 schaakbord vind je zo 92 oplossingen. In essentie zijn er 12 verschillende, waaruit je door draaien en spiegelen (8 mogelijkheden) ze allemaal kunt maken. Er is één wat meer symmetrische oplossing.