12 Backtracking algoritmes

12.1 Techniek

Voor vele problemen kan een oplossing gevonden worden door een bepaalde methode stap voor stap (een algoritme) te volgen. Er bestaan echter nog veel meer problemen waarvoor zo'n algoritme niet voor handen is. Men heeft hier te maken met "general problem solving" en men kan zo'n probleem aanpakken met *trial-and-error*.
Gewoonlijk wordt een *trial-and-error* proces opgedeeld in partiële taken. Deze taken kunnen op een natuurlijke manier in recursieve termen uitgedrukt worden en bestaan uit het onderzoeken van een eindig aantal deeltaken. Het gehele proces kan gezien worden als een zoekproces dat stelselmatig opgebouwd wordt waarbij telkens een waaier van deeltaken moet verder onderzocht worden. Men spreekt soms van een *zoekboom*. Deze boom kan zeer snel groeien, meestal exponentieel snel.
Een karakteristiek element van de methode is dat er een poging ondernomen wordt in de richting van een volledige oplossing. Eens deze poging aanvaard is, doet men een volgende poging zodat men telkens dichter en dichter bij de volledige oplossing komt. Het is natuurlijk mogelijk dat men op een bepaald moment vaststelt dat men de volledige oplossing niet kan bereiken. Op zo'n moment kunnen aanvaarde pogingen echter ongedaan gemaakt worden: men keert dus terug op zijn stappen (*backtracking*) om dan met een alternatieve poging terug in de richting van de oplossing te werken.

```
probeer()
{
        initialiseer keuze van mogelijke kandidaten ;
        doe
        {
                genereer volgende kandidaat;
                indien aanvaadbaar
                {
                        voeg kandidaat aan oplossing toe;
                        indien oplossing onvolledig
                        {
                                probeer();
                                verwijder kandidaat van oplossing;
                        }
                        anders
                                er is een oplossing gevonden;
                }
        }
        totdat er geen kandidaten meer zijn;
}
```

12.2 Voorbeeld: het koninginnen-probleem

Plaats op een 8×8 schaakbord een koningin op elke rij zodat er geen enkele koningin op dezelfde kolom of dezelfde diagonaal staat.
Merk op dat deze oefening kan opgelost worden voor een $N \times$N bord met N koninginnen waarbij $N >= 4$.

```
1  /*   nqueen.c : N-koningingen probleem  */
   #include <stdio.h>
3  #include <stdlib.h>
   #include <unistd.h>
5  #include <setjmp.h>
```

```
7  #define LEN     21

9  int geldig( char a[][LEN], int  m, int i, int j );
   void probeer( char a[][LEN], int  m, int i );
11 void drukbord( char a[][LEN], int m );
   int verbose = 0;
13 int aantal_pogingen = 0;
   int aantal_oplossing = 0;
15 int maxtal_oplossing = 1;
   jmp_buf omgeving;          /* stack context/omgeving  */
17
   int main( int argc, char *argv[])
19 {
        char    b[LEN][LEN];
21      int             n = 4;
        char    c;
23
        while ( (c=getopt(argc,argv,"n:a:v")) != EOF )
25      {
                switch ( c )
27              {
                        case 'n': n = atoi(optarg);
29                                      break;
                        case 'a': maxtal_oplossing = atoi(optarg);
31                                      break;
                        case 'v': verbose++;
33                                      break;
                }
35      }
        memset(b, '\0', LEN*LEN);
37      if ( setjmp(omgeving) == 0 )
                probeer(b,n,1);
39      printf("Aantal pogingen : %d\n", aantal_pogingen);
   }
41
   void drukbord( char a[][LEN], int m )
43 {
        int i, j, l=2*m;
45
        for(i=1; i<=m; i++)
47      {
                for(j=0; j<=l; j++)
49                      printf("-");
                printf("\n");
51              for(j=1; j<=m; j++)
                        if ( a[i][j] != '\0' )
53                              printf("|*");
                        else
55                              printf("| ");
                printf("|\n");
57      }
        for(j=1; j<=l; j++)
59              printf("-");
```

```
        printf("\n");
}
int geldig( char b[][LEN], int  n, int rij, int kol )
{
        int i, k;

        for (i=1; i<rij; i++)
        {
                if ( b[i][kol] == 1 )
                        return 0;
                k = kol-i;
                if ( k >= 1 && b[rij-i][k] == 1 )
                        return 0;
                k = kol+i;
                if ( k <= n && b[rij-i][k] == 1 )
                        return 0;
        }
        return 1;
}
void probeer( char b[][LEN], int  n, int rij )
{
        int j;
        int dec;

        aantal_pogingen++ ;
        for (j=1; j<=n; j++)
        {
                if ( geldig(b, n, rij, j) == 1 )
                {
                        if ( verbose )
                                printf("rij %2d : kolom %2d \n", rij, j );
                        b[rij][j] = 1;
                        if ( rij == n )
                        {
                                aantal_oplossing++;
                                printf("Oplossing %4d/%4d na %d pogingen\n",
                                aantal_oplossing, maxtal_oplossing, aantal_pogingen)
                                drukbord( b, n);
                                if ( aantal_oplossing == maxtal_oplossing)
                                        longjmp(omgeving,1);
                        }
                        else
                                probeer(b, n, rij+1);
                        b[rij][j] = 0;
                }
        }
}
```

Een aantal oplossingen:

$N = 4$

```
Oplossing  1/2
---------
| |*| | |
---------
| | | |*|
---------
|*| | | |
---------
| | |*| |
--------
Aantal pogingen : 8

Oplossing    2/2
---------
| | |*| |
---------
|*| | | |
---------
| | | |*|
---------
| |*| | |
--------
Aantal pogingen : 15
```

$N = 8$

```
Oplossing   1/100
-----------------
|*| | | | | | | |
-----------------
| | | | |*| | | |
-----------------
| | | | | | | |*|
-----------------
| | | | | |*| | |
-----------------
| | |*| | | | | |
-----------------
| | | | | | |*| |
-----------------
| |*| | | | | | |
-----------------
| | | |*| | | | |
----------------
Aantal pogingen : 113
```

```
Oplossing   92/100
-----------------
| | | | | | | |*|
-----------------
| | | |*| | | | |
-----------------
|*| | | | | | | |
-----------------
| | |*| | | | | |
-----------------
| | | | | |*| | |
-----------------
| |*| | | | | | |
-----------------
| | | | | | |*| |
-----------------
| | | | |*| | | |
----------------
Aantal pogingen : 1965
```

Bij $N = 4$ zijn er 2 verschillende oplossingen: de eerste wordt gevonden na 8 pogingen, de tweede na 11 pogingen. Daarna zijn er nog 4 pogingen waaruit geconcludeerd wordt dat er geen bijkomende oplossingen meer zijn.

Bij $N = 8$ zijn er 92 verschillende oplossingen (waarvan een heleboel een spiegelbeeld van een andere oplossing zijn). In weze zijn er 12 echt verschillende oplossingen. Een uitbreiding in het programma zou kunnen zijn: het genereren van deze echt verschillende oplossingen.
Bij $N = 20$ zijn er 199635 pogingen nodig om een eerste oplossing te vinden. Een klassieke PC gebruikt hiervoor een drietal CPU seconden rekentijd.

12.3 Oefening

Gegeven een $n \times n$ bord met n^2 velden. Plaats in deze velden de getallen van 1 tot n^2 in oplopende volgorde waarbij een volgend getal slechts op bepaalde posities ten opzichte van het vorige getal geplaatst kan worden: naar links, rechts, onder of boven: twee velden tussen laten en in de diagonalen één veld tussen laten.

			7			
	6				8	
5			X			1
	4				2	
			3			

Een aantal oplossingen:

$n = 5$				
1	24	14	2	25
16	21	5	8	20
13	10	18	23	11
4	7	15	3	6
17	22	12	9	19

$n = 6$					
1	21	30	2	22	29
35	26	18	36	25	17
14	32	23	28	31	3
6	20	10	7	19	11
34	27	15	33	24	16
13	8	5	12	9	4