Programmeeropgave: de kabinetsformatie

Daniel von Asmuth

17 april 2021

Dit is een eenvoudige programmeeropgave: gegeven de zetelverdeling in de Tweede Kamer na de verkiezingen van maart 2021, schrijf een programma dat de mogelijke coalities afdrukt. Om de lijst niet onnodig lang te maken dienen coalities te worden beperkt tot maximaal 6 partijen en als een combinatie samen een meerderheid heeft, dienen er geen partijen aan te worden toegevoegd. Uiteraard zoeken we een efficiënte oplossing. Hier is C de voorkeurstaal, maar de voorgestelde oplossing kan in vrijwel alle talen worden omgezet.

Recursie versus iteratie

De nieuwe Tweede Kamer telt 17 partijen. Een simpele, maar weinig efficiënte oplossing is om alle deelverzamelingen te genereren en elk ervan te testen. Dit zijn er 2^{17} = 131.072. Het aantal mogelijke deelverzamelingen met 4, 5 of 6 partijen bedraagt dan

$$\binom{17}{6}+\binom{17}{5}+\binom{17}{4}=20.994$$

Er zijn verschillende mogelijkheden om alle deelverzamelingen van een verzameling op te sommen: de eerste vorm is met een genestelde reeks lussen

```
for( indx[1] = from[1]; indx[1] <= to[1]; indx[1]++)
  for( indx[2] = from[2]; indx[2] <= to[2]; indx[2]++)
    for( indx[3] = from[3]; indx[3] <= to[3] ; indx[3]++)
      .....
         print();
```

Dit werkt in de meeste talen wel voor een constante nesteldiepte, maar geen programmeertaal kent dergelijke 'variadische' loops met een dynamische diepte. De bijpassende datastructuur zou het variadische array zijn met dynamische dimensie. We kunnen variadische lussen realiseren met een dubbel recursieve functie.

```
void            rec( int pos, int val, int n)
{
  if( pos < 1 || val < 1)
    return;
  indx[pos-1] = n + 1 - val;
  if( pos == 1)
    print();
  else
    rec( pos - 1, n, n);
  rec( pos, val - 1, n);
}
```

Theo van Haandel schreef een iteratief alternatief.

```
void            var_loop( int n)
{
  int           i               = 1;

  indx[i] = from[i] - 1;
  while( i > 0)
  { if( indx[i]++ < to[i])
      if( i < n)
      { i++;
        indx[i] = from[i] - 1;
      }
      else
        print( n);
    else
      i--;
```

```
  }
}
```

Erik von Asmuth liet zien dat een gewone for-loop volstaat van de vorm

```
  for (unsigned int coalitie = 1 << 15; coalitie < (1 << 17); coalitie++)
```

Daarin worden de bits van een integer gebruikt als een verzameling; om de elementen te lezen en schrijven gebruik je dan functies als:

```
static inline unsigned int
get_bit( const unsigned int x, const unsigned int i)
{
  return (x & (1 << i)) == 0 ? 0 : 1;
}

static inline unsigned int
set_bit( const unsigned int x, const unsigned int i, const BOOL val)
{
  return val == 1 ? x | (1 << i) : x & ~(1 << i);
}
```

Hier volgt eerst een recursieve oplossing voor de opgave die met bovenstaande ideeën tot een iteratieve oplossing is omgewerkt.

Een dubbel recursieve oplossing in C

```
#include        <stdio.h>

#define         PARTIES         17              /* parties elected in parliament */
#define         MAX_COALITION    6              /* max. practical coalition size */
#define         SEATS           150             /* total seats in parliament */
#define         MAJORITY        76              /* seats needed in parliament */

typedef         struct party
{ char          name[ 8];
  int           seats;
} PARTY;                                        /* Election result */

PARTY           chamber[PARTIES]        = { { "Bij1", 1},
                                            { "BBB", 1},
                                            { "50Plus", 1},
                                            { "DENK", 3},
                                            { "Volt", 3},
                                            { "SGP", 3},
                                            { "JA21", 3},
                                            { "CU", 5},
                                            { "PvdD", 6},
                                            { "GL", 8},
                                            { "FvD", 8},
                                            { "PvdA", 9},
                                            { "SP", 9},
                                            { "CDA", 15},
                                            { "PVV", 17},
                                            { "D66", 24},
                                            { "VVD", 34}
                                          };    /* Sorted by party size */

int             coalition[MAX_COALITION];       /* (Partial) next solution */
/* 'coalition' holds indexes into 'chamber' of parties in a coalition */

/* Print a possible coalition, smallest party first */
void            print( int m, int seats)
{
  int           i;

  printf( "%02d: ", seats);
  for( i = m; i < MAX_COALITION; i++)
    printf( "%s, ", chamber[coalition[i]].name);
  printf( "\n");
}
```

```c
/* Search next result
 * 'member is index in 'coalition' array
 * 'party' is index in 'chamber' array
 * 'coalition' contains indexes for 'chamber' array
 * 'needed' is min. nr. of seats short of majority
 * 'remaining' is nr. of seats in parties not counted yet
 * 'seats' is nr. of seats of current coalition
 */
void            search( int member, int party, int needed, int remaining, int seats)
{
  int           s               = chamber[party].seats;

  if( party < 0 || member < 0 || needed > remaining)
    return;
  remaining -= s;

  /* examine coalitions that exclude 'party' */
  search( member, party - 1, needed, remaining, seats);

  /* examine coalitions that include 'party' */
  coalition[member] = party, seats += s, needed -= s;
  if( needed < 1)
    print( member, seats);
  else
    search( member - 1, party - 1, needed, remaining, seats);
}

int             main( void)
{
  search( MAX_COALITION - 1, PARTIES -1, MAJORITY, SEATS, 0);
  return 0;
}
```

Een iteratieve oplossing in C

```c
#include        <stdio.h>

#define         PARTIES          17             /* parties elected in parliament */
#define         MAX_COALITION     6             /* max. practical coalition size */
#define         SEATS           150             /* total seats in parliament */
#define         MAJORITY         76             /* seats needed in parliament */

typedef         struct party
{ char          name[ 8];
  int           seats;
  int           selected;                       /* 1 if party belongs to coalition */
} PARTY;                                        /* election result */

PARTY           chamber[PARTIES]        = { { "Bij1", 1, 0},
                                            { "BBB", 1, 0},
                                            { "50Plus", 1, 0},
                                            { "DENK", 3, 0},
                                            { "Volt", 3, 0},
                                            { "SGP", 3, 0},
                                            { "JA21", 3, 0},
                                            { "CU", 5, 0},
                                            { "PvdD", 6, 0},
                                            { "GL", 8, 0},
                                            { "FvD", 8, 0},
                                            { "PvdA", 9, 0},
                                            { "SP", 9, 0},
                                            { "CDA", 15, 0},
                                            { "PVV", 17, 0},
                                            { "D66", 24, 0},
                                            { "VVD", 34, 0}
                                          };    /* Sorted by party size */

/* Print current solution (possible coalition) */
void            print( int seats)
{
  int           i;

  printf( "%02d: ", seats);
```

```
  for( i = PARTIES - 1; i >= 0; i--)
    if( chamber[i].selected == 1)
      printf( "%s, ", chamber[i].name);
  printf( "\n");
}
/*
 * Search results (iterative)
 */
void            search( void)
{ int           needed          = MAJORITY;
  int           remaining       = SEATS;
  int           seats           = 0;
  int           party           = PARTIES -1;
  int           members         = 0;

  chamber[party].selected = -1;
  while( party < PARTIES)
  { if( chamber[party].selected++ > 0)
    { /* restore number of seats and proceed to next party */
      seats -= chamber[party].seats; needed += chamber[party].seats;
      remaining += chamber[party].seats;
      members--; party++;
    }
    else
    { /* adjust number of seats */
      if( chamber[party].selected == 1)
      { seats += chamber[party].seats; needed -= chamber[party].seats;
        members++;
      }
      else
        remaining -= chamber[party].seats;

      /* check if we found a solution */
      if( needed < 1 )
      { print( seats);
        chamber[party].selected = 2;
      }
      else
        /* check for absence of solutions */
        if( chamber[party].selected==1 && (needed>remaining || members>=MAX_COALITION))
          chamber[party].selected = 2;
        else
          if( party > 0)
            chamber[--party].selected = -1;
    }
  }
}

int             main( void)
{ search();
  return 0;
}
```

Toelichting

Het recursieve algoritme begint met het array *chamber* te vullen met het resultaat van de verkiezingen; het array *coalition* stelt de verzameling van partijen voor die een coalitie vormen en de *print()* functie drukt een coalitie af. De *search()* functie doorzoekt de mogelijke combinaties vanaf de grootste partij naar de kleinste.

De *member* parameter telt de grootte van de huidige coalitie en *party* de volgende partij; beide worden verlaagd. Verder telt *seats* het aantal zetels in de huidige coalitie, *needed* is het aantal zetels dat nog nodig is voor een meerderheid en *remaining* telt het aantal zetels dat nog niet getest; *needed* en *remaining* worden gebruikt om partijen over te slaan die geen meerderheid opleveren en *seats* wordt enkel afgedrukt.

De code begint met de parameters te testen of de recursie beëindigd moet worden. De werking wordt begrijpelijker als je ziet dat beide recursies het array van partijen afwerken van de grootste tot de kleinste; dit zou ook iteratief kunnen. De eerste aanroep slaat de geteste partij gewoon over; daarna wordt die partij toegevoegd aan de coalitie om coalities met die partij te testen. Als de coalitie een meerderheid heeft, worden ze afgedrukt en niet verder uitgebreid. Merk op dat in beide gevallen de zetels van de huidige partij worden afgetrokken van het restant.

Het iteratieve algoritme zoekt de mogelijke coalities af met 17 geneste for-loops die van 0 t/m 1 lopen; het veld *selected* geeft dan aan of een partij deel uitmaakt van de geteste coalitie. Het test de zelfde variabelen om combinaties over te slaan die geen oplossing kunnen vormen.

Resultaten

Bovenstaand programma vindt 380 mogelijke coalities met maximaal 6 partijen. Er zijn 38 mogelijkheden zonder de VVD, maar die zijn alle met CDA, PVV en D66, wat geen goede samenwerking belooft. Omdat er maar weinig partijen met de VVD of PVV willen regeren, kun je denken aan een alternatief met 7 partijen: zo hebben D66, CDA, SP, PvdA, GL, PvdD en CU samen ook 76 zetels.

Het recursieve programma onderzoekt 1210 mogelijke coalities in 0.002 s op oude hardware, wat een goed resultaat is, waar de Java versie op 0.2 s komt. De iteratieve variant test 8679 combinaties en is vrijwel even snel.

Conclusie

Recursieve algoritmes kunnen worden omgezet naar iteratieve uitvoeringen die bijna even snel zijn. Voor beginners is recursie moeilijk te doorgronden, maar dat geldt hier ook voor de iteratie.

Literatuur

'**Do** considered **od**', E.C.R. Hehner, Acta Informatica **11**, 287 – 304 (1979),
http://www.cs.toronto.edu/~hehner/dco-acta.pdf