Computergestütztes wissenschaftliches Rechnen SoSe 2004

Alexander K. Hartmann, Universität Göttingen

24. Mai 2004

4.4 Dynamisches Programmieren

Fibonacci Zahlen:

$$\mathrm{fib}(n) = \begin{cases} 1 & (n=1) \\ 1 & (n=2) \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & (n>2) \end{cases} \tag{1}$$

Z.B. $\mathrm{fib}(4) = \mathrm{fib}(3) + \mathrm{fib}(2) = (\mathrm{fib}(2) + \mathrm{fib}(1)) + \mathrm{fib}(2) = 3$,
$\mathrm{fib}(5) = \mathrm{fib}(4) + \mathrm{fib}(3) = 3 + 2 = 5$.
Wächst sehr schnell: $\mathrm{fib}(10) = 55$, $\mathrm{fib}(20) = 6765$, $\mathrm{fib}(30) = 83204$, $\mathrm{fib}(40) > 10^8$
Anzal der Aufrufe bei rekursiver Implentierung wächst auch exponentiell mit n, schneller als die Funktion!

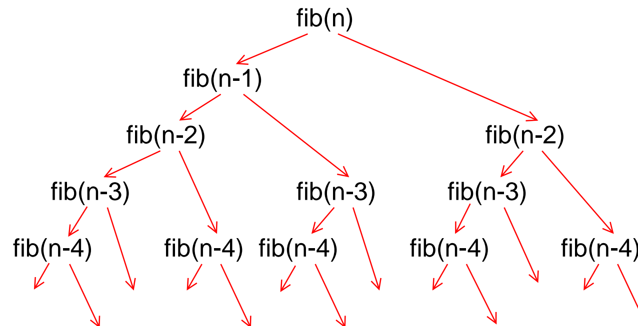

Abbildung 1: Hierarchie der Aufrufe für $\mathrm{fib}(n)$.

Besser: dynamisches Programmieren. Prinzip: Berechne Lösung für kleinere Probleme und benutze sie (nicht rekursiv) um größere Probleme iterativ zu lösen.

```
/** calculates Fibonacci number of 'n' dynamically **/
double fib_dynamic(int n)
{
  double *fib, result;
  int t;
  if(n<=2)                              /* simple case ? */
    return(1);                    /* return result directly */
  fib = (double *) malloc(n*sizeof(double));
  fib[1] = 1.0;                  /* initialise known results */
  fib[2] = 1.0;

  for(t=3; t<n; t++)       /* calculate intermediate results */
    fib[t]=fib[t-1]+fib[t-2];

  result = fib[n-1]+fib[n-2];
  free(fib);
  return(result);
}
```

Algorithmus enthält nur eine Schleife → Laufzeit $O(n)$.
Noch schneller: Formel

$$\mathrm{fib}(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) \tag{2}$$

4.5 Backtracking

Grundidee: Wenn man Lösung nicht direkt berechnen kann: versuche verschiedene Möglichkeiten.

Backtracking: Zuvor gemachte Entscheidungen verhindern die Lösung → nehme Entscheidungen in systematischer Weise zurück und versuche anderen Weg.

Beispiel: N Damen Problem

N Damen sind auf einem $N \times N$ Schachbrett so zu platzieren, dass keine Dame eine andere bedroht.

Das bedeutet, dass in jeder Reihe, jeder Spalte und jeder Diagonale maximal eine Dame steht. □

Grundidee des Verfahrens: stelle in jede Spalte (`column[i]`, $i = 0, \ldots, N-1$) in systematischer Weise eine Dame auf. Wenn es keine Lösung gibt, *backtracke*. Zusätzlich Variable für die Diagonalen → Test wird schneller.

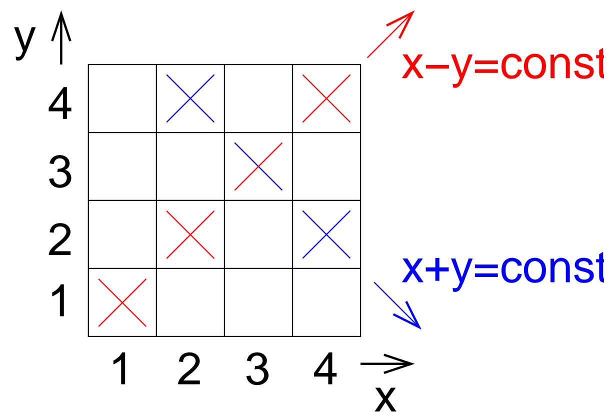

Abbildung 2: Testvariablen, ob in den jeweiligen Diagonalen Damen stehen.

Da $x, y = 0, \ldots, N-1 \rightarrow$ Abwärtsdiagonalen: $x + y \in 0, \ldots, 2N-2$, Aufwärtsdiagonalen: $x - y \in -N+1, \ldots, N-1$ (für C array: immer $N-1$ addieren)

```
void queens(int n, int N, int *column, int *row,
            int *diag_up, int *diag_down)
{
  int t, t2;
  if(n == -1)                                           /* solution found ? */
  {
    for(t=0; t<N; t++)                                 /* print solution */
    {
      for(t2=0; t2<column[t]; t2++)
        printf("*");
      printf("X");
      while(++t2<N)
        printf("*");
      printf("\n");
    }
    printf("----------------------\n");
    return;
  }
  for(t=N-1; t>=0; t--)        /* place queen in all rows of column n */
  {
    if(!row[t]&&!diag_up[n-t+(N-1)]&&!diag_down[n+t])  /* can place ? */
    {
      row[t] = 1; diag_up[n-t+(N-1)] = 1; diag_down[n+t] = 1;
      column[n] = t;
      queens(n-1, N, column, row, diag_up, diag_down);
      row[t] = 0; diag_up[n-t+(N-1)] = 0; diag_down[n+t] = 0;
    }
  }
  column[t] = 0;
}
```

Anfänglich: `column[i]=row[i]=diag_down[i]=diag_up[i]=0` für alle `i` und rufe auf:
`queens(N-1,N,column,row,diag_up,diag_down)`.

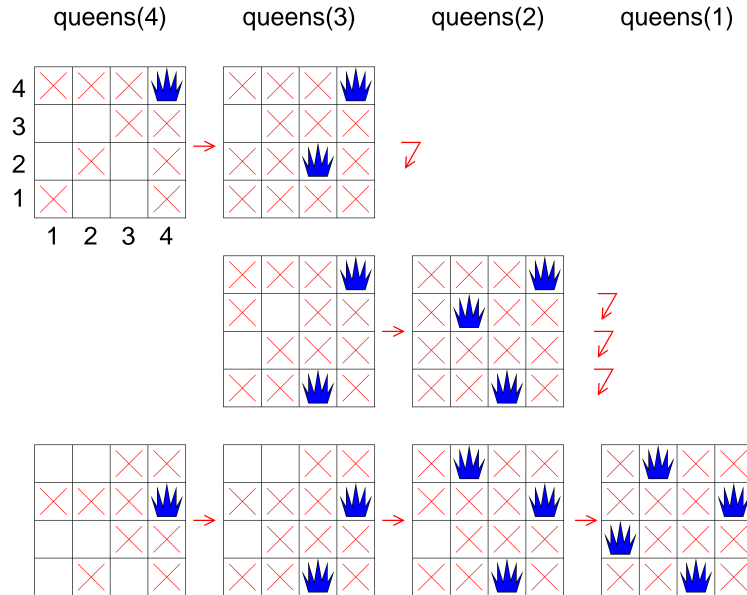

Abbildung 3: Wie der Algorithmus das 4-Damen Problem löst.

5 Speicherfehler

Speicherfehler, insbesondere by dynamisch alloziertem Speicher:

- Lesen von uninitialisiertem Speicher
- Lesen/Schreiben außerhalb der Array Grenzen
- Zugriff auf bereits freigegebenen Speicher
- Benutzung falscher Pointer
- Nicht-Freigabe von nicht mehr benutzem Speicher (Programm wird immer größer)

Speicherfehler sind schwer zu finden. Meistens werden sie (falls überhaupt) in völlig anderen Programmteilen sichtbar

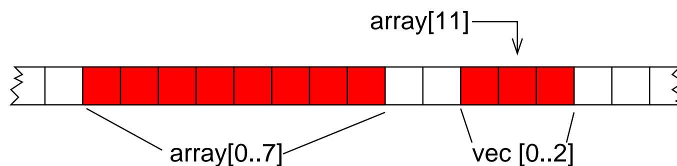

Abbildung 4: Zwei dynamisch allozierte Arrays. Wenn man über die Grenze des ersten hinaus schreibt, z.B. in `array[11]`, zerstört man den Inhalt des zweiten Arrays.

Beispiel `memory.c`

```
#include <stdio.h>
#include <stdlib.h>

int main()
```

```
{
    int *array;

    array = (int *) malloc (100*sizeof(int));
    array[50] = 1;
    array[200] = 1;              /* writes past boundaries */
    free(array);
    array[50] = 1;              /* writes in freed memory */
    return(0);
}
```

Compile `gcc -o memory memory.c -g`
Laufen lassen: Kein Problem!!
Daher: Benutzung des Programms `valgrind`. Es kontrolliert alle Speicherzugriffe → läuft viel langsamer (beim Testen). Man schreibt es einfach vor das aufgerufene Programm, d.h. keine Änderung am Compilieren, geht auch für fertige Programm, z.B. `valgrind ls`.
Run: `valgrind memory` gibt:

```
==26019== valgrind-1.0.4, a memory error detector for x86 GNU/Linux.
==26019== Copyright (C) 2000-2002, and GNU GPL'd, by Julian Seward.
==26019== Estimated CPU clock rate is 2700 MHz
==26019== For more details, rerun with: -v
==26019==
==26019== Invalid write of size 4
==26019==    at 0x80483AC: main (memory.c:16)
==26019==    by 0x402537F7: __libc_start_main (in /lib/i686/libc-2.3.1.so)
==26019==    by 0x80482D1: free@@GLIBC_2.0 (in /net/theorie/auto/home6/hartmann/
texte/lehre/comp_phys2004/programs/memory)
==26019==    Address 0x42B06344 is not stack'd, malloc'd or free'd
==26019==
==26019== Invalid write of size 4
==26019==    at 0x80483C8: main (memory.c:18)
==26019==    by 0x402537F7: __libc_start_main (in /lib/i686/libc-2.3.1.so)
==26019==    by 0x80482D1: free@@GLIBC_2.0 (in /net/theorie/auto/home6/hartmann/
texte/lehre/comp_phys2004/programs/memory)
==26019==    Address 0x42B060EC is 200 bytes inside a block of size 400 free'd
==26019==    at 0x4003D129: free (vg_clientfuncs.c:180)
==26019==    by 0x80483BD: main (memory.c:17)
==26019==    by 0x402537F7: __libc_start_main (in /lib/i686/libc-2.3.1.so)
==26019==    by 0x80482D1: free@@GLIBC_2.0 (in /net/theorie/auto/home6/hartmann/
texte/lehre/comp_phys2004/programs/memory)
==26019==
==26019== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==26019== malloc/free: in use at exit: 0 bytes in 0 blocks.
==26019== malloc/free: 1 allocs, 1 frees, 400 bytes allocated.
==26019== For a detailed leak analysis,  rerun with: --leak-check=yes
==26019== For counts of detected errors, rerun with: -v
```

Man sollte **ALLE** Programm einmal mit valgrind testen. Achtung: Einige Bibliotheksfunktionen haben Fehler.