Vorlesung 4 (Dienstag 20.2.2018)

4.4 Dynamisches Programmieren

Fibonacci Zahlen:

$$\text{fib}(n) = \begin{cases} 1 & (n = 1) \\ 1 & (n = 2) \\ \text{fib}(n-1) + \text{fib}(n-2) & (n > 2) \end{cases} \tag{5}$$

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

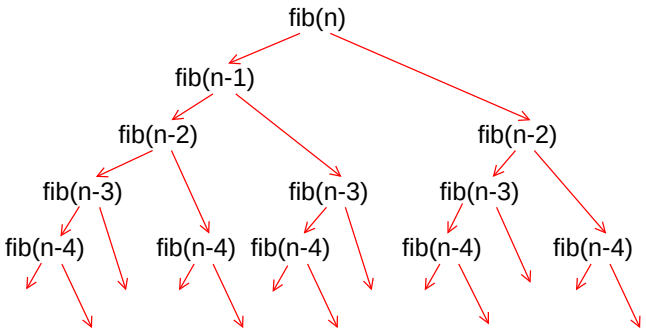

Figure 3: Hierarchie der Aufrufe für $\text{fib}(n)$.

[Selbsttest]

Wie man man einen schnelleren Algorithmus finden?

ACHTUNG: Lesen Sie den Rest des Abschnitts NICHT, bevor Sie sich etwas überlegt haben

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);
}
```

[Selbsttest]

Wie ist die Laufzeit des Verfahrens?
Wettbewerb: was ist das größte n, das fib(n) noch eine endliche Ausgabe erzeugt: jede(r) gibt eine Schätzung ab.

Noch schneller: Formel

$$\text{fib}(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right) \tag{6}$$

4.5 Backtracking

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

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

Example: 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. □

[Selbsttest]

Überlegen Sie sich erst für 4 Minuten einen Algorithmus (Grundidee), der das N Damen Problem löst.
Dann diskutieren Sie 3 Minuten mit ihrem Nachbarn ihre Lösungen.

ACHTUNG: Lesen Sie den Rest des Abschnitts NICHT, bevor Sie sich etwas überlegt haben

---

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

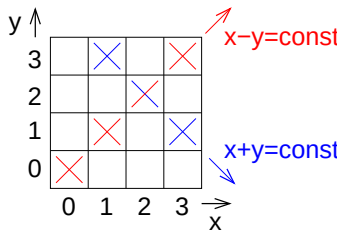

Figure 4: Testvariablen, ob in den jeweiligen Diagonalen Damen stehen.

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 c, int N, int *pos, int *row,
            int *diag_up, int *diag_down)
{
  int r, c2;                                    /* loop counters */
  if(c == -1)                                 /* solution found ? */
  {
      /* omitted here */                       /* print solution */
  }
  for(r=N-1; r>=0; r--)   /* place queen in all rows of column c */
  {
    if(!row[r]&&!diag_up[c-r+(N-1)]&&!diag_down[c+r]) /*  place ?*/
    {
      row[r] = 1; diag_up[c-r+(N-1)] = 1; diag_down[c+r] = 1;
      pos[c] = r;
      queens(c-1, N, pos, row, diag_up, diag_down);
      row[r] = 0; diag_up[c-r+(N-1)] = 0; diag_down[c+r] = 0;
    }
  }
  pos[c] = 0;
}
```

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

[Selbsttest]

Lösen Sie das 4×4 Problem mit Hilfe des ausgeteilten Musters. Wieviele verschiedene Lösungen gibt es?

Wenn man für kleine N die Zahl der Lösungen zählt → wächst exponentiell mit N.

5 Fortgeschrittene Datentypen

Zum Durchführen von Elementaren Operationen (Speichern, Suchen, Auslesen und Löschen von Daten).

Mittels guten Datenstrukturen: fast immer größere Systeme oder schneller Simulationen möglich.

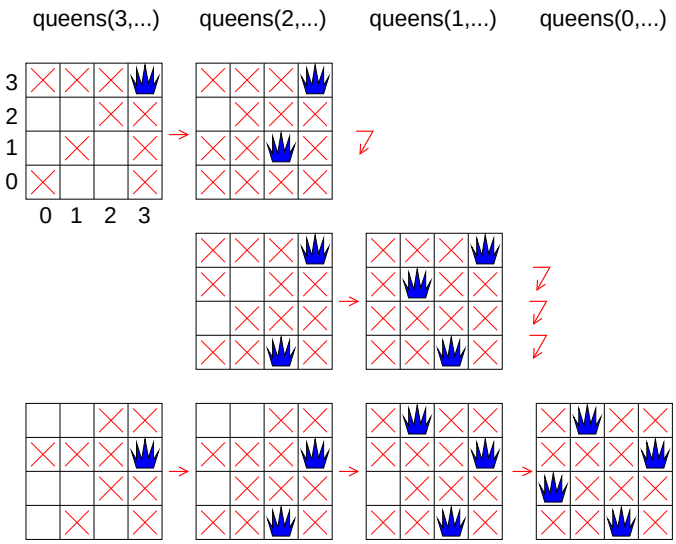

Figure 5: Wie der Algorithmus das 4-Damen Problem löst.

5.1 Listen

Listen = Verallgemeinerungen von Arrays: auch lineare Ordnung aber flexibler. Bsp.: Löschen eines Array Elements $O(N)$, (linked) Liste: $O(1)$.
Hier: einfach verbundene Liste. Datenstruktur:

```
/* data structures for list elements */
struct elem_struct
{
    int                 info;               /* holds ‘‘information’’ */
    struct elem_struct *next;  /* points to successor (NULL if last) */
};

typedef struct node_struct elem_t;       /* define new type for nodes */
```

Doppelt verbundene Listen: auch Eintrag `struct elem_struc *prev`
Erzeugen und Löschen von Elementen:

```
/****************** create_element() ****************/
/** Creates an list element an initialized info      **/
/** PARAMETERS: (*)= return-paramter                 **/
/**        value: of info                            **/
/** RETURNS:                                         **/
/**     pointer to new element                       **/
/*******************************************************/
elem_t *create_element(int value)
{
  elem_t *elem;

  elem = (elem_t *) malloc (sizeof(elem_t));
  elem->info = value;
  elem->next = NULL;
  return(elem);
}

/****************** delete_element() ****************/
/** Deletes a single list element (i.e. only if it   **/
/** is not linked to another element)                **/
/** PARAMETERS: (*)= return-paramter                 **/
/**         elem: pointer to element                 **/
/** RETURNS:                                         **/
/**    0: OK, 1: error                               **/
/*******************************************************/
int delete_element(elem_t *elem)
{
  if(elem == NULL)
  {
    fprintf(stderr, "attempt to delete ‘nothing‘\n");
    return(1);
  }
  else if(elem->next != NULL)
  {
    fprintf(stderr, "attempt to delete linked element!\n");
    return(1);
  }
  free(elem);
  return(0);
}
```

Liste = Pointer auf erstes Element:

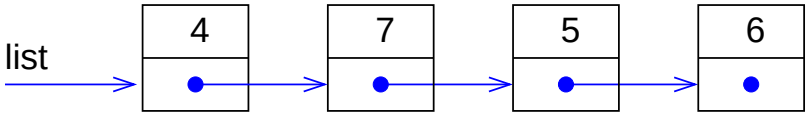

Figure 6: A single-linked list.

Erzeugung von Listen: Einfügen von Elementen a) am Anfang b) nach einem anderen Element:

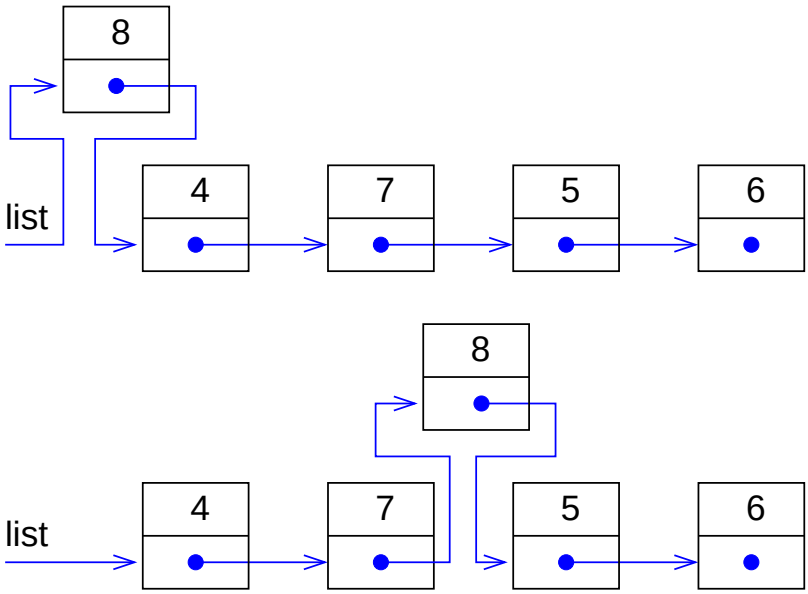

Figure 7: Einsetzen eines Elements in Liste.

```
/****************** insert_element() ****************/
/** Inserts the element ‘elem‘ in the ‘list          **/
/** BEHIND the ‘where‘. If ’where’ is equal to NULL  **/
/** then the element is inserted at the beginning of **/
/** the list.                                        **/
/** PARAMETERS: (*)= return-paramter                 **/
/**         list: first element of list              **/
/**         elem: pointer to element to be inserted  **/
/**        where: position of new element            **/
/** RETURNS:                                         **/
/**  (new) pointer to the beginning of the list      **/
/****************************************************/
elem_t *insert_element(elem_t *list, elem_t *elem, elem_t *where)
{
  if(where==NULL)                    /* insert at beginning ? */
  {
    elem->next = list;
    list = elem;
  }
  else                                    /* insert elsewhere */
  {
    elem->next = where->next;
    where->next = elem;
  }
  return(list);
}
```

Ausgeben einer Liste: Gehe durch alle Elemente:

```
/****************** print_list() ****************/
/** Prints all elements of a list               **/
/** PARAMETERS: (*)= return-paramter            **/
/**           list: first element of list       **/
/** RETURNS:                                    **/
/**          nothing                            **/
/************************************************/
void print_list(elem_t *list)
{
  while(list != NULL)              /* run through list */
  {
    printf("%d ", list->info);
    list = list->next;
```

```
  }
  printf("\n");
}
```

Löschen von Elementen: a) erstes Element b) andere Elemente:

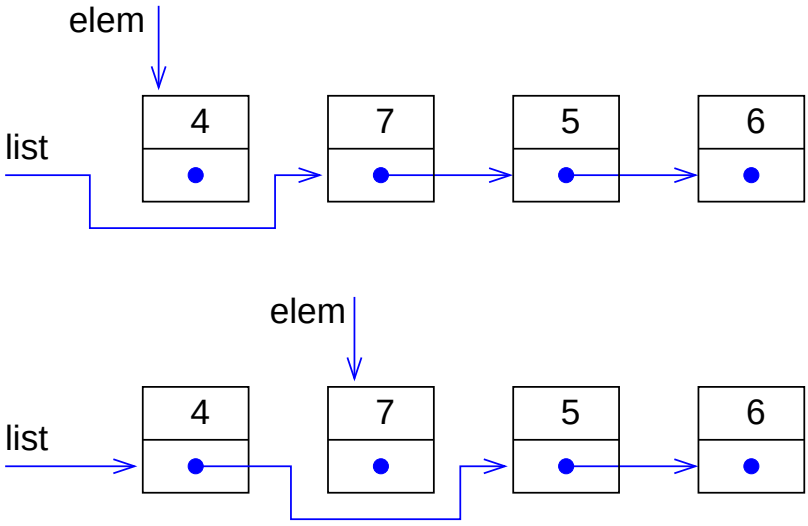

Figure 8: Löschen eines Listenelements

Man muss das Element vor dem zu löschenden Element finden. Einfacher: doppelt verkettete Listen.