TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

**Lehrstuhl für Sprachen und Beschreibungsstrukturen**
**Einführung in die Informatik 2**
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz

WS 2008/09
**Übungsblatt 7**
25.11.2008

Abgabe: 02.12.2008 (vor der Vorlesung)

## Aufgabe 7.1 (H) Matrixmultiplikation

In dieser Aufgabe soll ein Webseiten-Ranking berechnet werden. Wir betrachten folgende Web-Seiten:

1) www.nichtsowichtig.de
2) www.tum.de
3) wwwseidl.in.tum.de
4) portal.mytum.de

Untersuchungen haben ergeben, dass ein Benutzer der www.nichtsowichtig.de besucht, mit einer Wahrscheinlichkeit von $40$ % danach die Seite www.tum.de besuchen. Mit einer Wahrscheinlichkeit von $60$ % wird danach wwwseidl.in.tum.de besucht. Insgesamt lassen sich die Ergebnisse der Untersuchung in folgender Matrix zusammenfassen:

$$P = \begin{pmatrix} 0 & 0 & 0.2 & 0.3 \\ 0.4 & 0 & 0.8 & 0 \\ 0.6 & 0 & 0 & 0.7 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Dabei bezeichnet der Eintrag in der $i$-ten Zeile und der $j$-ten Spalte die Wahrscheinlichkeit, dass nach Webseite $j$ die Webseite $i$ besucht wird. Ein Webseiten-Ranking sortiert die Seiten danach, mit welcher erwarteten Wahrscheinlichkeit sie besucht werden. Auf Grundlage solcher Daten ergibt sich das Webseiten-Ranking dann durch

$$\lim_{i \to \infty} P^i \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Ihre Aufgabe besteht darin, eine Approximation dieses Wertes zu bestimmen.

Zur Lösung der gestellten Aufgabe gehen wir wie folgt vor. Einen Vektor $x = (x_1, \ldots, x_n)$ repräsentieren wir durch eine streng monoton aufsteigend sortierte Liste von Tupeln $(i, x_i)$. Dabei werden nur Paare $(i, x_i)$ mit $x_i \neq 0$ aufgeführt. Z.B. wird der Vektor $(0, 0, 0.1, 0, 0.5, 0.4, 0, 0)^\top$ durch `[(3,0.1); (5,0.5); (6,0.4)]` repräsentiert.

Eine Matrix wird durch eine Liste von Paaren `(j,v)` repräsentiert, wobei `j` ein Zeilenindex und `v` ein Vektor ist, welcher die `j`-te Zeile repräsentiert. Z.B. wird die Matrix $P$ durch die Liste

$$[(1, [(3, 0.2); (4, 0.3)]); (2, [(1, 0.4); (3, 0.8)]); (3, [(1, 0.6); (4, 0.7)]); (4, [(2, 1)])]$$

repräsentiert.

a) Schreiben Sie eine Funktion `v_mult`, die das Skalarprodukt zweier Vektoren berechnet. Zur Erinnerung: $(a_1, \ldots, a_m) \cdot (b_1, \ldots, b_m)^\top = \sum_{i=1}^{m} a_i \cdot b_i$

b) Schreiben Sie eine Funktion `m_v_mult`, die für eine Matrix $\mathtt{a} = (\mathtt{a_1}, \ldots, \mathtt{a_m})^\top$ und einen Vektor `v` die Multiplikation $a \cdot v$ durchführt.
Zur Erinnerung: $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} v = \begin{pmatrix} a_1 \cdot v \\ \vdots \\ a_n \cdot v \end{pmatrix}$

c) Schreiben Sie eine Funktion `iter`. Falls `p` die Matrix $P$ und `x` den Vektor $x$ repräsentieren, soll der Aufruf `iter p x i` den Vektor $x_i := P^i x$ berechnen. Hinweis: Für alle $i \in \mathbb{N}_0$ gilt:

$$x_i = \begin{cases} Px_{i-1} & \text{falls } i > 0 \\ x & \text{falls } i = 0. \end{cases}$$

d) (**freiwillige Zusatzaufgabe**) Schreiben Sie eine Funktion `m_m_mult`, die zwei Matrizen miteinander multipliziert.

**Lösungsvorschlag 7.1**

```
(* Beispiele *)
let ex_v = [ (3,0.1); (5,0.5);  (6,0.2) ]

let ex_a = [(2,[(3,23.);(5,25.)]);(5,[(2,52.)]);(6,[(2,62.)])]

(* Vektormultiplikation *)
let rec v_mult p x y =
  match (x,y) with
    ((px,vx)::x',(py,vy)::y') ->
      if px = py then
        v_mult (p +. vx *. vy) x' y'
      else if px < py then
        v_mult p x' y
      else
        v_mult p x y'
  | _ -> p

let v_mult x y = v_mult 0. x y

(* Matrix * Vektor *)
let m_v_mult a v =
  let l = List.map (fun (j,aj) -> (j, v_mult aj v)) a in
    (* Multipliziert Zeilenvektor aj mit dem Vektor v*)
    List.filter (fun (j,aj) -> (aj <> 0.)) l (* Nullen entfernen *)

(* Mergesort wird fuer die Zusatzaufgabe benoetigt,
       Loesung von Aufgabe 6.4 *)

let init l = List.map (fun x -> [x]) l

let rec merge l1 l2 =
```

```
  match (l1,l2) with
    ([],l) | (l,[]) -> l
  | (x::xs,y::ys) ->
      if x <= y then
        x::(merge xs l2)
      else
        y::(merge l1 ys)

let rec merge_list = function
    [] -> []
  | [l] -> [l]
  | l1::l2::ls -> (merge l1 l2)::(merge_list ls)

let rec mergesort l =
  match l with
    [] -> []
  | [x] -> x
  | _ -> mergesort (merge_list l)

let mergesort l = mergesort (init l)


(* Transponieren oder
     Zeilenrepraesentation zu Spaltenrepraesentation *)

let rec v_to_list i = function
  (j,x)::v -> (i,j,x)::v_to_list i v
| _ -> []

let rec m_to_list = function
  (i,v)::a -> v_to_list i v @ m_to_list a
| _ -> []

let m_from_list l =
  let l = mergesort l in
  List.fold_right
    (
    fun (i,j,x) ->
      function ((i',v)::vs) ->
        if i = i' then
          ((i,(j,x)::v)::vs)
        else
          (i,[(j,x)])::((i',v)::vs)
      | _ -> [(i,[(j,x)])]
    )
    l
    []

let transpose a =
  let l = m_to_list a in
  let l = List.map (fun (i,j,x) -> (j,i,x)) l in
  m_from_list l
```

```
(* Matrixmultiplikation *)
let m_m_mult a b =
  let b = transpose b in
  let handle_row_a (i,ai) = (i, m_v_mult b ai) in
  let r = List.map handle_row_a a in
  List.filter (fun (_,ai) -> ai <> []) r

let p =
  [
    (1, [(3,0.2);(4,0.3)]);
    (2, [(1,0.4);(3,0.8)]);
    (3, [(1,0.6);(4,0.7)]);
    (4, [(2,1.)])
  ]

let x = [(1,1.)]
let y = [(2,1.)]

let rec iter p x i =
  if i = 0 then
    x
  else
    iter p (m_v_mult p x) (i-1)

let rec iterm p i =
  if i = 0 then
    p
  else
    iterm (m_m_mult p p) (i-1)
```

**Aufgabe 7.2 (P) Listenfunktionale, Funktionen höherer Ordnung**

Definieren Sie die Funktion `fold_right` mithilfe der Funktion `fold_left`. Konstruieren Sie eine aus 1.000.000 Einsen bestehende Liste `list`. Testen Sie ihre Funktion anhand des Aufrufs `fold_right (+) list 0`. Testen Sie den gleichen Aufruf anhand der in der Vorlesung angegebenen Implementierung. Was fällt Ihnen auf?

**Lösungsvorschlag 7.2**

```
open List

(* Erzeugt eine Liste l mit 1000000 Eintraegen. *)

let rec b n l = if n = 0 then l else b (n-1) (1::l)

let l = b 1000000 []

(* Version 1 *)

let rev l = fold_left (fun r x -> x::r) [] l

let fold_right f l s =
  let l = rev l in
  fold_left (fun x y -> f y x) s l

let _ = print_int (fold_left (+) 0 l)

(* -- Version 1 *)
(* Version 2 *)

let (++) f g x = f (g x)

let fold_right f l =
  fold_left (fun f' x -> f' ++ (f x)) (fun x -> x) l

let _ = print_int (fold_left (+) 0 l)

(* -- Version 2 *)
```

**Aufgabe 7.3 (P) MiniJava-Interpreter**

Ziel dieser Aufgabe und der Hausaufgabe auf dem nächsten Blatt ist es, einen Interpreter für eine Java-ähnliche Sprache in OCaml zu implementieren. Die arithmetischen Ausdrücke `e` dieser Sprache sind durch folgende Grammatik spezifiziert:

$$\texttt{e} ::= \texttt{Const}(<\texttt{int}>) \mid \texttt{Var}(<\texttt{string}>) \mid \texttt{Add}(\texttt{e},\texttt{e}) \mid \texttt{Sub}(\texttt{e},\texttt{e}) \mid \texttt{Mul}(\texttt{e},\texttt{e}) \mid \texttt{Div}(\texttt{e},\texttt{e})$$

Beispielsweise ist

$$\texttt{Add}(\texttt{Var}(\texttt{x}), \texttt{Mul}(\texttt{Const}(2), \texttt{Const}(3)))$$

ein arithmetischer Ausdruck. Gehen Sie wie folgt vor:

a) Definieren Sie zunächst einen Typ `expr` zur Repräsentation arithmetischer Ausdrücke.

b) Nutzen Sie den auf der Übungsseite bereitgestellten Parser, um Ausdrücke aus einer Datei zu lesen und diese anschließend in einen Wert vom Typ `expr` zu konvertieren.

c) Definieren Sie eine Funktion `eval : expr -> int`, die arithmetische Ausdrücke auswertet. Gehen Sie dabei zunächst davon aus, dass alle Variablen den Wert $0$ haben.

d) In dieser Aufgabe werden Variablen als `string`-Werte repräsentiert. Eine Funktion

```
rho : string -> int
```

ist eine Variablenbelegung. Sie ordnet jeder Variablen einen `int`-Wert zu. Schreiben Sie eine Funktion

```
update : (string->int)->string->int->string->int
```

zum Aktualisieren von Variablenbelegungen. Der Aufruf `update rho x v` soll die Variablenbelegung `rho`$'$ zurückgeben, die `rho` entspricht, bis darauf, dass `rho`$'$ `x` $=$ `v` gilt.

e) Ändern Sie die Funktion `eval` entsprechend. Ihre Funktion `eval` benötigt jetzt zusätzlich zu dem auszuwertenden Ausdruck `e` eine Variablenbelegung `rho` als Parameter.

f) **(Anspruchsvoll)** Ergänzen Sie Ihre Implementierung, so dass der Benutzer nach Start des Programms dazu aufgefordert wird Werte für alle im Ausdruck vorkommenden Variablen einzugeben. Die eingegebenen Werte sollen dann zur Auswertung des Ausdrucks verwendet werden.

**Hinweis:** Verwenden Sie die OCaml-Funktionen `string_of_int`, int_of_string, `print_string` und `read_int`.

**Lösungsvorschlag 7.3**

```
open List
open Mo
exception ParseError of string

(* Teil a) *)
type var = string
type expr =
  Const of int
| Var   of var
| Add   of expr * expr
| Sub   of expr * expr
| Mul   of expr * expr
| Div   of expr * expr

(* Teil b) *)
let rec get_expr t =
  match t with
    Node("Const", [Node(x,[])]) -> Const(int_of_string x)
  | Node("Var", [Node(x,[])]) -> Var(x)
  | Node("Add", [a1;a2]) -> Add(get_expr a1, get_expr a2)
  | Node("Sub", [a1;a2]) -> Sub(get_expr a1, get_expr a2)
  | Node("Mul", [a1;a2]) -> Mul(get_expr a1, get_expr a2)
  | Node("Div", [a1;a2]) -> Div(get_expr a1, get_expr a2)
  | _ -> raise (ParseError (string_from_term t))

(* Teil c) *)
let rec eval = function
  Add (e1,e2) -> (eval e1) + (eval e2)
| Sub (e1,e2) -> (eval e1) - (eval e2)
| Mul (e1,e2) -> (eval e1) * (eval e2)
| Div (e1,e2) -> (eval e1) / (eval e2)
| Const i     -> i
| Var v       -> 0

(* Teil d) *)
let update rho x v y = if x = y then v else rho y

(* Teil e) *)
let rec eval rho = function
  Add (e1,e2) -> (eval rho e1) + (eval rho e2)
| Sub (e1,e2) -> (eval rho e1) - (eval rho e2)
| Mul (e1,e2) -> (eval rho e1) * (eval rho e2)
| Div (e1,e2) -> (eval rho e1) / (eval rho e2)
| Const i     -> i
| Var v       -> rho v

(* Teil f) *)
(* Variante 1 *)
let union a b =
  fold_left (fun m x -> if mem x m then m else x::m) a b
```

```
let rec vars = function
  Add(e1,e2) | Sub(e1,e2) | Mul(e1,e2) | Div(e1,e2) ->
    union (vars e1) (vars e2)
| Var(x) -> [x]
| _ -> []
(*-- Variante 1 *)
(* Variante 2 *)
let rec vars vs = function
  Add(e1,e2) | Sub(e1,e2) | Mul(e1,e2) | Div(e1,e2) ->
    let vs = vars vs e1 in
    vars vs e2
| Var(v) -> if mem v vs then vs else v::vs
| _ -> vs

let vars = vars []
(*-- Variante 2 *)

let read_rho vars =
  fold_left
    (fun rho x -> let v = print_string (x ^ " = "); 
      read_int () in update rho x v)
    (fun _ -> 0)
    vars

let term = term_from_file Sys.argv.(1)
let ausdruck = get_expr term
let _ = print_string ("Eingelesener Ausdruck : "
                ^ (string_from_term term) ^ "\n")
let rho = read_rho (vars ausdruck)
let _ = print_string "Der Wert des Ausdrucks ist "
let _ = print_int (eval rho ausdruck)
let _ = print_string ".\n"
```