Wstęp do Programowania potok funkcyjny

Marcin Kubica

2010/2011

Outline

1. Technika spamiętywania i programowanie dynamiczne
   - Technika spamiętywania
   - Programowanie dynamiczne

Technika spamiętywania

Technika spamiętywania

- Rekurencyjny zapis rozwiązania + Mapy.
- Unikamy wielokrotnych wywołań dla tych samych argumentów.
- Spamiętujemy wszystkie obliczone wcześniej wyniki wywołań.
- Skrzyżowanie rekurencji i programowania dynamicznego

Przykład spamiętywania

Example (Liczby Fibonacciego i spamiętywanie)

```
let tab = ref empty;;
let rec fib n =
  if dom !tab n then
    apply !tab n
  else
    let wynik =
      if n < 2 then n else fib (n-1) + fib (n-2)
    in begin
      tab := update !tab n wynik;
      wynik
    end;;
```

Schemat spamiętywania

Schemat spamiętywania jest dosyć uniwersalny:

- Implementacja rekurencyjna.
- Założenie: dla tych samych argumentów zawsze dostajemy takie same wyniki.
- Tabela do spamiętywania wyników.
- Przed wywołaniem rekurencyjnym sprawdzamy, czy już dla takiego wywołania nie obliczono wyniku.
- Jeśli tak, wykorzystujemy obliczony wcześniej wynik.
- Jeśli nie, wywołujemy rekurencyjnie procedurę i zapamiętujemy wynik.

Uniwersalny spamiętywacz

Uniwersalna „otoczka” spamiętująca:

```
let memoize tab f x =
  if dom !tab x then
    apply !tab x
  else
    let wynik = f x
    in begin
      tab := update !tab x wynik;
      wynik
    end;;
```

Przykład zastosowania spamiętywacza

Procedura obliczająca liczby Fibonacciego zapisana przy użyciu uniwersalnego spamiętywacza:

Example

```
let fib =
  let tab = ref empty
  in
    let rec f n =
      memoize tab (function n ->
        if n < 2 then n else f (n-1) + f (n-2)) n
    in f;;
```

Problem NWP

Example

Problem najdłuższego wspólnego podciągu:

- Dane: dwa ciągi $x = [x_1; x_2; \ldots; x_m]$ i $y = [y_1; y_2; \ldots; y_n]$.
- Szukamy najdłuższego ciągu, który jest podciągiem x i y. Oznaczenie: $NWP(x, y)$.

Idea rozwiązania problemu NWP

Example

Idea rozwiązania:

- Uogólnienie problemu na prefiksy:
  $nwp(i,j) = NWP([x_1;\ldots;x_i],[y_1;\ldots;y_j])$
- Własność podproblemu:
  - Jeśli $x_i = y_j$, to $nwp(i,j) = nwp(i-1,j-1)@[x_i]$.
  - Jeśli ostatni wyraz nwp jest różny od x_i, to $nwp(i,j) = nwp(i-1,j)$.
  - Jeśli ostatni wyraz nwp jest różny od y_j, to $nwp(i,j) = nwp(i,j-1)$.
  - Funkcja nwp jest monotoniczna względem obydwu współrzędnych.
  - Jeśli $x_i \neq y_j$, to $nwp(i,j)$ jest dłuższym z podciągów $nwp(i-1,j)$ i $nwp(i,j-1)$).
  - Warunki brzegowe: $nwp(i,0) = nwp(0,j) = []$.

Algorytm rozwiązania problemu NWP

Example

Algorytm: tabela rozwiązań dla podproblemów i jej wypełnianie.

		A	B	C	B	D	A	B
	0	0	0	0				
B	0	0	**1**	1	**1**			
D	0	0	1	1	1	**2**		
C	0	0	1	**2**	2	2		
A		**1**	1	2	2	2	**3**	
B					**3**	3		**4**
A							**4**	4

Najdłuższe wspólne podciągi: `BCBA`, `BCAB`, `BDAB`,
ostatni z nich występuje w pierwszym ciągu na dwa sposoby.

NWP — implementacja ze spamiętywaniem

Example

Wynikiem procedury pom jest para: (odwrócony) najdłuższy wspólny podciąg i jego długość.

```
let nwp ciag_x ciag_y =
  let x = Array.of_list ciag_x
  and y = Array.of_list ciag_y
  and tab = ref empty
  in
    let rec pom (i, j) = ...
    in
      rev (fst (pom (length ciag_x, length ciag_y)));;

nwp ['A'; 'B'; 'C'; 'B'; 'D'; 'A'; 'B']
    ['B'; 'D'; 'C'; 'A'; 'B'; 'A'];;
- : char list = ['B'; 'D'; 'A'; 'B']
```

NWP — implementacja ze spamiętywaniem

Example

```
let rec pom (i, j) =
  memoize tab (fun (i, j) ->
    if i = 0 || j = 0 then ([], 0)
    else if x.(i-1) = y.(j-1) then
      let (c, l) = pom (i-1, j-1)
      in (x.(i-1)::c, l+1)
    else
      let (c1, l1) = pom (i-1, j)
      and (c2, l2) = pom (i, j-1)
      in
        if l1 > l2 then (c1, l1) else (c2, l2)
  ) (i, j)
```

NWP – Implementacja z wykorzystaniem tablic

Example

- Dane: dwa ciągi $x = [x_1; x_2; \ldots; x_m]$ i $y = [y_1; y_2; \ldots; y_n]$.
- Szukamy najdłuższego ciągu, który jest podciągiem x i y. Oznaczenie: $NWP(x, y)$.
- Spamiętywanie możemy zaimplementować wprost, tworząc tablicę:

$$\mathrm{a}.(i).(j) = |NWP([x_1; \ldots; x_i], [y_1; \ldots; y_j])|$$

- Dwie fazy:
  - obliczenie tablicy a,
  - rekonstrukcja wyniku.

NWP – Implementacja z wykorzystaniem tablic

Example

```
let nwp ciag_x ciag_y =
  if (ciag_x = []) || (ciag_y = []) then [] else
  let n = length ciag_x
  and m = length ciag_y
  and x = Array.of_list ((hd ciag_x)::ciag_x)
  and y = Array.of_list ((hd ciag_y)::ciag_y)
  in let a = Array.make_matrix (n+1) (m+1) 0
     in let rec rekonstrukcja acc i j = ...
        in begin
          for i = 1 to n do
            for j = 1 to m do
              if x.(i) = y.(j) then
                a.(i).(j) <- a.(i-1).(j-1) + 1
              else
                a.(i).(j) <- max a.(i-1).(j) a.(i).(j-1)
            done
          done;
          rekonstrukcja [] n m
        end;;
```

NWP – Implementacja z wykorzystaniem tablic

Example

Rekonstrukcja podciągu na podstawie tablicy a:

```
let rec rekonstrukcja acc i j =
  if i = 0 || j = 0 then acc
  else if x.(i) = y.(j) then
    rekonstrukcja (x.(i)::acc) (i-1) (j-1)
  else if a.(i).(j) = a.(i-1).(j) then
    rekonstrukcja acc (i-1) j
  else
    rekonstrukcja acc i (j-1)
```

NWP – Implementacja z wykorzystaniem map

Example

- Zamiast tablicy możemy użyć mapy.
- Złożoność czasowa gorsza o czynnik $O(\log n)$.
- Główna procedura:

```
let nwp ciag_x ciag_y =
  let a = zbuduj_tablice1 ciag_x ciag_y
  in odtworz_podciag1 a ciag_x ciag_y;;
```

- Zależność między sąsiednimy elementami tablicy:

```
let dlugosc u v w x y =
  if x = y then v + 1
  else max u w;;
```

NWP – Implementacja z wykorzystaniem map

Example

```
let zbuduj_tablice ciag_x ciag_y =
  let dodaj_wiersz tablica y j = ...
  in
    let pierwszy = ...
    and buduj a yl = ...
    in buduj pierwszy ciag_y;;
```

NWP – Implementacja z wykorzystaniem map

Example

```
let pierwszy =
  let (a, _) =
    fold_left
      (fun (a, i) _ -> (update a (0, i) 0, i+1))
      (update empty (0, 0) 0, 1)
      ciag_x
  in a

and buduj a yl =
  let (w, _) =
    fold_left
      (fun (a, j) y -> (dodaj_wiersz a y j, j+1))
      (a, 1) yl
  in w
```

NWP – Implementacja z wykorzystaniem map

Example

```
let dodaj_wiersz tablica y j =
  let (t, _) =
    fold_left
      (fun (tab, i) x ->
        let u = apply tab (j, i-1)
        and v = apply tab (j-1, i-1)
        and w = apply tab (j-1, i)
        in
          (update tab (j, i) (dlugosc u v w x y), i+1))
      (update tablica (j, 0) 0, 1)
      ciag_x
  in t
```

NWP – Implementacja z wykorzystaniem map

Example

Rekonstrukcja podciągu:

```
let odtworz_podciag tablica ciag_x ciag_y =
  let rec odtworz akumulator tab ciag_x ciag_y i j = ...
  in
    odtworz [] tablica
            (rev ciag_x) (rev ciag_y)
            (length ciag_x) (length ciag_y);;
```

NWP – Implementacja z wykorzystaniem map

Example

```
let rec odtworz akumulator tab ciag_x ciag_y i j =
  match ciag_x with
    [] -> akumulator |
    (x::tx) ->
      match ciag_y with
        [] -> akumulator |
        (y::ty) ->
          if x = y then
            odtworz (x::akumulator) tab tx ty (i-1) (j-1)
          else
            let u = apply tab (j, i-1)
            and w = apply tab (j-1, i)
            in
              if u > w then
                odtworz akumulator tab tx ciag_y (i-1) j
              else
                odtworz akumulator tab ciag_x ty i (j-1)
```

Ogólny schemat programowania dynamicznego

- Sparametryzowanie problemu.
- Zależności między rozwiązaniami podproblemów.
- Rodzaj użytej struktury danych.
- Kolejność rozwiązywania podproblemów.
- Czy można lepiej sformułować podproblemy?
- Ewentualne wyznaczenie pomocniczych problemów.
- Ewentualna rekonstrukcja wyniku.
- Polepszenie złożoności pamięciowej.
- Czy lepiej zastosować spamiętywanie, czy programowanie dynamiczne?