Programmazione

Esercizi svolti in aula

(26-05-2015)

il codice contenuto in questo .pdf è disponibile anche qui:
http://pastebin.com/H7u7f5VE

/* Esercizio 3 */

```
/*
data una lista(L) ed un array P[0..dimP-1] cercare un match completo (e non necessariamente contiguo) di P su L.
in caso di "successo", i nodi di lista(L) che matchano con elementi di P vanno staccati da L e concatenati in una lista(L2) da restituire.
(possibile variante: non utilizzare il parametro "succ" (hint: fermarsi quando dimP==1)).
*/
```

```
1.  //nel seguito viene utilizzata la seguente struttura nodo
2.  struct nodo
3.  {
4.    //membri - campi dati
5.    int info;
6.    nodo *next;
7.
8.    //costruttore
9.    nodo(int a=0, nodo* b=0)
10.   {
11.     info=a;
12.     next=b;
13.   }
14. };
```

```
1.  nodo *match_nc(nodo *&L, int *P, int dimP, bool &succ)
2.  {  //PRE(lista(L) corretta, P ha dimP>=0 elementi, succ è definito 0/1 e lista(L)=lista(vL)).
3.    if(!dimP)  //alla chiamata precedente ho esaurito il pattern da matchare, quindi ritorno stato positivo
4.    {
5.      succ=true;
6.      return 0;
7.    }
8.    if(!L)  //sono uscito dalla lista, non ho più nodi (e dato che se ho trovato il match mi son fermato prima) qui non ho trovato il match
9.      return 0;
10.   if(L->info==*P)  //potenzialmente questo nodo fa parte del match, ma lo staccherò solo se c'è il match completo
11.   {
12.     nodo *x=match_nc(L->next, P+1, dimP-1, succ);  //mi occupo della parte restante della lista
13.     if(succ)  //se il match completo c'è: x contiene i nodi del match del resto di L a partire da qua, L->next (per side-effect) punta ai restanti nodi
14.     {
15.       nodo *y=L;  //chiamo il nodo attuale "y" (backup)
16.       L=L->next;  //stacco il nodo attuale (y) dalla lista (lo by-passo)
17.       y->next=x;  //attacco in cima ad x il nodo attuale (che fa parte del match)
18.       return y;  //restituisco la lista-pattern aggiornata
19.     }
20.     else  //se il match completo non c'è, ritorno la lista vuota
21.       return 0;
22.   }
23.   else  //il nodo corrente non matcha in P, quindi non farà parte del match, è giusto che lo "salti" e lo lasci dov'è nella lista(L)
24.     return match_nc(L->next, P, dimP, succ);
25. }
```

```
//POST=(se esiste match completo (anche non contiguo) tra P[0..dimP-1] e lista(vL) allora:
//       - con return viene restituita una lista(L2) corretta contenente i nodi di lista(vL) che metchano con elementi di P[0..dimP-1],
//       - lista(L) corretta, conterrà i rimanenti nodi;
//      altrimenti:
//       - return restituisce 0,
//       - lista(L)==lista(vL).


/*
VARIANTE:
necessito di sapere se c'è il pattern completo oppure no.
non potendo usare il parametro "succ", potrei usare il puntatore alla lista-pattern ritornato come "test":
per poterlo fare, quando trovo il match completo mi devo fermare con dimP==1 e ritornare un puntatore ad un nodo (l'ultimo della lista pattern);
invece, quando NON trovo un pattern completo restituirò x=0;
in tal modo if(x) mi dirà se il pattern completo c'è o meno senza l'uso di "succ".
PS: bisogna prestare particolare attenzione a come cambia il caso base!
*/
```

/* Esercizio 4 */

```
/*
dato un albero(r) ed un array P[0..dimP-1] cercare un match (contiguo e completo) di P su albero(r).
in caso di "successo", va creata una lista(L) i cui nodi hanno come campo "info" un puntatore al nodo di albero(r) che matcha con l'elemento in P.
(per un esempio "visivo" vedi: Esercizio 1 del 25-05-2015 in cui si tratta una lista con suddette caratteristiche).
note: l'ordine di esplorazione di albero(r) è Depth-First Pre-Fissa; è sufficiente fermarsi al primo match incontrato (se c'è).
(possibile variante: cercare un match completo (e non necessariamente contiguo) di P su albero(r)).*/
```

```
1.  //nel seguito viene utilizzata la seguente struttura, per realizzare i nodi dell'albero
2.  struct nodo_albero
3.  {
4.    //membri - campi dati
5.    int info;
6.    nodo_albero *left, *right;
7.
8.    //costruttore
9.    nodo_albero(int a=0, nodo_albero* b=0, nodo_albero*c=0)
10.   { info=a;
11.     left=b;
12.     right=c;
13.   }
14. };
15.
16.
```

```
17. //e la seguente struttura, per realizzare i nodi della lista
18. struct nodo_lista
19. {
20.   //membri - campi dati
21.   nodo_albero *p;
22.   nodo_lista *next;
23.
24.   //costruttore
25.   nodo_lista(nodo_albero *a=0, nodo_lista* b=0)
26.   { p=a;
27.     next=b;
28.   }
29. };
```

```
1.  nodo_lista *match(nodo_albero *r, int P, int dimP)
2.  {  //PRE=().
3.    if(!r)  //caso base: albero vuoto
4.      return 0;
5.    else  //ho almeno un nodo
6.    {
7.      nodo_lista *x=check(r, P, dimP);  //cerchiamo il match a partire dal nodo attuale, cioè la lista non è vuota
8.      if(x)  //in caso affermativo
9.        return x;  //ritorniamo la lista
10.
11.     x=match(r->left, P, dimP);  //se ci è andata male, proviamo nel sotto-albero sx
12.     if(x)  //se l'ho trovato a sx
13.       return x;  //ritorno la lista
14.
15.     return match(r->right, P, dimP);  //altrimenti lo cerco nel sotto-albero dx, e restituirò quel che ottengo: se c'è la lista, altrimenti 0
16.   }
17. }  //POST=().
18.
19.
```

```
20. nodo_lista *check(nodo_albero *r, int *P, int dimP)
21. {  //PRE=(albero(r) corretto, P ha dimP>=0 elementi).
22.   if(!r)  //caso base: sono uscito dall'albero
23.     return 0;  //è andata male, ritorno la lista vuota
24.   if(r->info==*P)  //se il nodo corrente match
25.   {
26.     if(dimP==1)  //se il nodo corrente (che matcha) è l'ultimo da trovare
27.       return new nodo_lista(r,0);
28.     nodo_lista *x=check(r->left, P+1, dimP-1);  //altrimenti continuo la ricerca del resto del match a sx
29.     if(x)  //se a sinistra ho completato la parte di match che mancava
30.       return new nodo_lista(r,x);  //ci attacco in testa il nodo corrente e ritorno la nuova lista
31.     x=check(r->right, P+1, dimP-1);  //se non l'ho trovato a sx, cerco di completare il match a dx
32.     if(x)  //se a destra ho completato la parte di match che mancava
33.       return new nodo_lista(r,x);  //ci attacco in testa il nodo corrente e ritorno la nuova lista
34.     else  //se neanche a dx riesco a completare il match
35.       return 0;  //allora non c'è, ritorno la lista vuota
36.   }
37.   else  //il nodo attuale non matcha
38.     return 0;  //è andata male, ritorno la lista vuota
39. }  //POST=().
40.
41.
42. /*
43. VARIANTE:
44.
45. */
```