Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Prof. Ghezzi, Lanzi e Morzenti
Prova di recupero
15 Giugno 2004

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

RECUPERO I PARTE ☐ Es. 1,2,3

RECUPERO II PARTE ☐ ES. 4,5,6

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Barrare le caselle relative alle parti recuperate. Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1. Parte I.
Si consideri il seguente frammento di programma C:

```
int x;
int *p;
int fun1(int *q) {
  p= &x;
  ...
  return x+1;
};
int fun2(int *t) {
  int z:
  z= *(t++);
  p= &z;
  ...
  return z;
}
main {
  int x;
  ...
  x = fun1(p);    (***)
  ...
  x =  fun2(p);
  ...
}
```

a) In quale delle due chiamate di funzione dall'interno di main si possono generare "dangling reference"?
b) Si schizzi lo stato della macchina astratta supponendo che la chiamata di fun1 nel punto (***) provochi la chiamata di fun2 e che questa chiami ricorsivamente fun1;
c) Si generino le istruzioni SIMPLESEM che traducono l'istruzione z=*(t++); all'interno di fun2;
d) Supponendo che in fun1 appaia l'istruzione *p = *q++; quali sono gli attributi distanza, offset per p e q?

Esercizio 1. Parte I. (continua)

Soluzione

a) Nella chiamata di fun2() dopo il termine della funzione p punta a una cella non piu' allocata (la vecchia variabile locale z di fun2) ho quindi una dangling reference.

b)

	0	Current	20
	1	Free	24
global	2	x	
	3	p	
main()	4	Return Pointer	
	5	Dynamic Link	
	6	Static Link	2
	7	x	
	8	Return Value	
fun1()	9	Return Pointer	
	10	Dynamic Link	4
	11	Static Link	2
	12	q	
	13	Return Value	
fun2()	14	Return Pointer	
	15	Dynamic Link	9
	16	Static Link	2
	17	t	
	18	z	
	19	Return Value	
fun1()	20	Return Pointer	
	21	Dynamic Link	14
	22	Static Link	2
	23	q	

c)
set D[0]+4, D[D[D[0]+3]]
set D[0]+3, D[D[0]+3] + 1

d)
p = <1,1>
q = <0,3>

Esercizio 2. Parte I. Si consideri un linguaggio che consente di annidare staticamente i sottoprogrammi. La macchina astratta SIMPLESEM illustrata a lezione che consente di modellare a runtime l'accesso ai dati non locali si basa sull'uso dei link statici e sulla rappresentazione delle variabili come coppia <distanza, offset>.
Quale delle seguenti affermazioni rappresenta in maniera piu' accurata la complessita' dell'accesso a un dato non locale rappresentato dalla coppia <dist, off>?

a) la complessita' e' proporzionale a x, dove x e' la profondita' corrente della pila
b) la complessita' e' proporzionale a dist
c) la complessita' e' proporzionale al logaritmo di dist
d) la complessita' e' proporzionale a logaritmo di x, dove x e' la profondita' corrente della pila

MOTIVARE con una SINTETICA spiegazione la risposta.

Soluzione

Dato che devo risalire la catena statica per un numero di link uguale alla distanza, la complessita' e' proporzionale alla distanza. L'accesso all'offset corretto e' a tempo costante.

Esercizio 3. Parte I. Si consideri un linguaggio che consente di annidare staticamente i sottoprogrammi. La macchina astratta SIMPLESEM illustrata a lezione che consente di modellare a runtime l'accesso ai dati non locali si basa sull'uso dei link statici e sulla rappresentazione delle variabili come coppia <distanza, offfset> e sulla percorrenza a runtime della catena statica. In alternativa, si puo' pensare a una implementazione che memorizza nel record di attivazione di un sottoprogramma un DISPLAY, e cioe' un array in cui in posizione i viene memorizzato il riferimento alla base del record di attivazione che si trova a distanza i lungo la catena statica.

1. Supponendo che il DISPLAY sia memorizzato a partire da un certo offset fisso K del record di attivazione, si scrivano le azioni della macchina SIMPLESEM che consentono di accedere alla variabile descritta dalla coppia di valori <D, OFF>.
2. Descrivere a parole le azioni necessarie per l'esecuzione a runtime delle operazioni che memorizzano nel record di attivazione di un sottoprogramma, all'atto della chiamata, il relativo DISPLAY;
3. Confrontare l'efficienza tra la realizzazione della macchina astratta che usa la catena statica e quella che usa il DISPLAY in termini di complessita' delle operazioni di chiamata di sottoprogramma e di accesso ai dati non locali.

Soluzione

1. Secondo la definizione dell'array DISPLAY nella cella ad offset K sarà memorizzata l'indirizzo della base del record di attivazione corrente (ovvero il record di attivazione a distanza zero) mentre nella cella successiva (offset K + 1) sarà memorizzato l'indirizzo della base del record di attivazione a distanza 1, quindi nella cella con offset K+2 l'indirizzo della base del record di attivazione a distanza 2 e così via. Di conseguenza l'indirizzo della base del record di attivazione a distanza D si troverà nella cella ad offset K + D. La variabile desiderata avrà quindi come indirizzo sullo stack I + OFF dove I rappresenta il valore contenuto nella cella di offset K + D del record di attivazione corrente.
2. All'atto dell'invocazione di una generica funzione f(), la macchina SIMPLESEM accede al record di attivazione della funzione che include staticamente f() (l'identità di questa funzione è già disponibile a compile time tramite lo *static nesting tree).*
   Si accede quindi all'array DISPLAY di questa funzione e si ricopiano i valori ivi contenuti a partire dalla prima cella nell'array DISPLAY di f() secondo la seguente formula : DISPLAY'[i+1] = DISPLAY[i] (DISPLAY' rappresenta il DISPLAY di f()). In DISPLAY'[0] verrà infine copiato l'indirizzo della base del record di attivazione di f().
3. Nella versione standard la creazione del link statico ha una complessità constante poichè richiede esclusivamente di collegare il link medesimo al record di attivazione corrispondente. Tuttavia l'accesso ad una variabile ha invece un complessità che dipende dal numero di salti richiesti sulla catena statica. Viceversa nella versione in esame, al momento della creazione del record di attivazione è necessario scorrere tutto l'array DISPLAY del padre per copiare i valori ma l'accesso ad una variabile richiede invece complessità costante in quanto è sufficiente accedere alla relativa cella nel proprio array DISPLAY. Occorre quindi valutare il rapporto tra numero di invocazioni e numeri di accessi con relative distanze per poter stabilire quale sia la soluzione più efficiente.

Esercizio 4. Parte II. Un albero completamente equilibrato è un albero pieno in cui tutte le foglie hanno la stessa distanza dalla radice. Dimostrare per induzione che per ogni a>=1 un albero completamente equilibrato di altezza a ha 2^a-1 nodi e 2^{a-1} foglie.

Soluzione

Base: per a=1 l'albero è composto dalla sola radice che è anche foglia, ha quindi 2^1-1=1 nodi e $2^{1-1}=2^0=1$ foglie.

Ipotesti induttiva: si assuma che un albero t di altezza a abbia 2^a-1 nodi e 2^{a-1} foglie.

Passo induttivo. Un albero t' di altezza $a'=a+1$ può essere pensato come ottenuto da un albero t di altezza a, attaccando due foglie a ogni foglia di t (foglia che diventa quindi un nodo interno di t'). Quindi il numero delle foglie di t` è il doppio di quello di t, cioè $2\cdot 2^{a-1}= 2^a=2^{(a+1)-1}$; inoltre il numero complessivo di nodi di t' è uguale a quello di t incrementato delle 2^a "nuove foglie", cioè $(2^a-1)+2^a=2^{(a+1)}-1$, cvd.

Esercizio 5. Parte II. Si ipotizzi che esista un algoritmo SPLITk che può dividere una lista L di *n* elementi in *k* sottoliste, ognuna delle quali contiene uno o più elementi, tale che tutti gli elementi della sottolista *i* sono minori di tutti gli elementi della sottolista *j*, per *i*<*j*<=*k*. Se *n*<*k*, allora *k*-n delle sottoliste prodotte da SPLITk sono vuote. Si assuma che SPLITk abbia complessità $\Theta(n)$, *n* essendo la lunghezza della lista che viene divisa in *k* sottoliste. Si assuma inoltre che le *k* liste possano essere concatenate in un tempo costante. Si consideri ora il seguente algoritmo.

```
List SORTk (List L) {
  List sub[k];      // contiene le sottoliste
  if (L.length()>1) {
    SPLITk(L, sub);  // SPLITk pone le sottoliste in sub
    for (int i = 0; i <k; i++)
      sub[i] = SORTk(sub[i]);  // ordina ricorsivamente ogni sottolista
    L = …concatenazione delle k sottoliste contenute in sub…
  }
  return L;
}
```

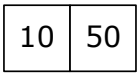
1. Qual è la complessità asintotica di SORTk nel caso ottimo?
   (giustificare opportunamente la risposta)

Risposta. Il caso ottimo è quello in cui le sottoliste hanno tutte la stessa lunghezza, quindi SORTk può essere analizzato in modo simile a mergesort (in mergesort lo split è a costo costante e il merge lineare, mentre in SORTk lo split è lineare e il merge costante), quindi la complessità è $\square(n\log n)$.

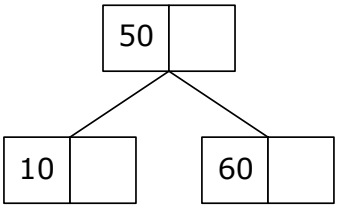
2. Qual è la complessità asintotica di SORTk nel caso pessimo?
   (giustificare opportunamente la risposta)

Risposta. Il caso pessimo è quello in cui la lunghezza delle sottoliste è totalmente sbilanciata (e.g., una lista lunga n-4, e quattro liste lunghe 1), nel qual caso SORTk si comporta come un quicksort nel caso pessimo, con complessità quadratica.

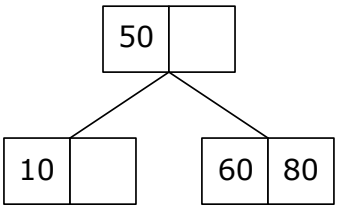
Esercizio 6. Parte II. Si consideri un "*2-3 tree*" e si inseriscano nell'ordine i seguenti elementi: 10,50,60,80,70, 69, 65, 15, 90, 95, disegnando gli alberi intermedi che si ottengono.



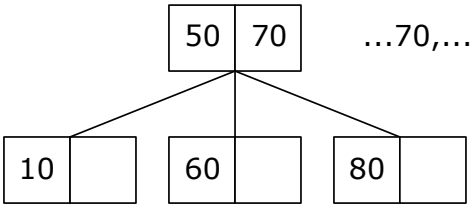
10,50,...



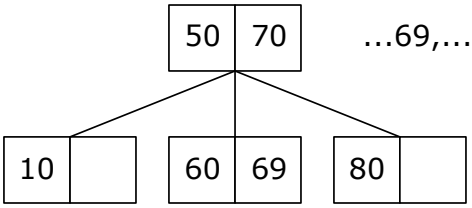
...60,...



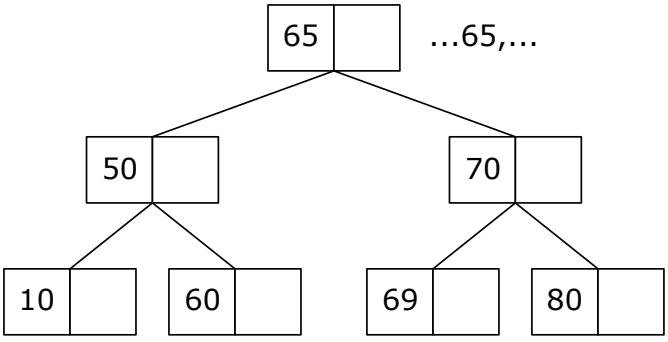
...80,...



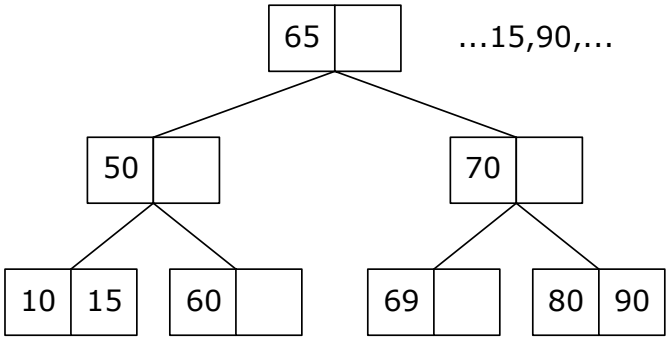
...70,...



...69,...



...65,...



...15,90,...

...95.

Infine...