Informatica 3 – Proff. Ghezzi e Morzenti – Primo recupero – 22 luglio 2003

Cognome e Nome ..... Numero Matricola .....
(in stampatello)

Recupero della ☐ PRIMA ☐ SECONDA prova in itinere (barrare una o entrambe)

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.

NON CONSEGNARE ALTRI FOGLI. (spazio a uso dei docenti) | | | | | | |

RECUPERO PRIMA PROVA

Esercizio 1

Rispondere alle seguenti domande:

a) In che cosa consiste l'operazione di garbage collection che il supporto run-time di Java effettua? Perché è utile che l'operazione di garbage collection venga effettuata automaticamente dall'implementazione del linguaggio?

[R. Il garbage collector di Java si occupa di eliminare dallo Heap le istanze degli oggetti o di altre strutture dati che non sono piu' utili al programma in corso di esecuzione.
Se si affidano le operazioni di garbage collection all'utente ci sono i seguenti rischi:

- l'utente rischia di scordarsi di deallocare oggetti dallo heap, come conseguenza con lo scorrere del tempo lo heap si satura.
- L'utente rischia di deallocare oggetti ancora in uso, causando dei dangling reference.

]

b) La possibilità che una variabile (p) abbia come valore l'indirizzo di un'altra variabile (x) può generare un "dangling reference". Perchè?

[R. C'e' il rischio che attraverso p si possa cercare di accedere a x (per essere piu' precisi all'area di memoria che contiene l'r-value di x) quando x non esiste piu'. I casi in cui la variabile x puo' non esistere piu' sono i seguenti:

- x era stata originalmente allocata sullo heap e poi e' stata stata deallocata.
- x e' stata originalmente allocata sullo stack e uscendo dallo scope di x la porzione corrispondente dello stack e' stata deallocata

]

c) Nel caso precedente, sapresti individuare qualche controllo al tempo di compilazione che coinvolga lo scope delle variabili e che consenta di individuare potenziali sorgenti di dangling reference durante l'esecuzione?

[R. Si puo' controllare che lo scope di p sia strettamente incluso nello scope di x. In tal modo p viene deallocata prima di x. Tuttavia (in presenza di routines) questo controllo puo' essere compiuto integralmente solo a run-time.
Altri linguaggi inoltre ricorrono ai seguenti espedienti per cercare di minimizzare la presenza di dangling references (tutti controllabili a compile-time):

- Tipizzazione dei puntatori.
- Divieto di applicazione delle operazioni aritmetiche ai puntatori.

]

Esercizio 2

Si consideri un programma che opera sulle seguenti variabili:

```
int z, i; int vect [10]
```

e che esegue il codice seguente contenente due chiamate alla routine p:

```
z = 0;
p(z);
write(z);
for (i=0; i<10; i++) vect[i]=0;
i=1;
p(i);
write(i); write(vect[i]);
```

Ipotizziamo che la variabile z sia globale, visibile sia al programma che alla routine p. La routine p è così definita:

```
void p (int y) {
    write(z); write(y);
    z ++;
    y := y + z;
    write(z); write(y);
};
```

Si scrivano i risultati che vengono visualizzati dall'esecuzione del programma considerando le diverse possibilità di passaggio dei parametri (a) per valore, (b) per indirizzo, (c) per valore-risultato.

[R.

Passaggio per valore:	Passaggio per indirizzo:	Passaggio per valore risultato
1 z: 0, y: 0	1 z=0, y=0	1 z=0, y=0
2 z: 1, y: 1	2 z=2, y=2	2 z=1, y=1
3 z: 1	3 z=2	3 z=1
1 z: 1, y: 1	1 z=2, y=1	1 z=1, y=1
2 z: 2, y: 3	2 z=3, y=4	2 z=2, y=3
4 i: 1, vect[1]: 0	4 i=4, vect[4]=0	4 i=3, vect[3]=0

Esercizio 3

Si consideri in Java un oggetto condiviso, esemplare di una classe Pippo che contiene una variabile intera che deve essere sempre maggiore o uguale a zero. L'oggetto è accessibile in modo concorrente da diversi threads mediante le operazioni inc(x), che incrementa di x il valore della variabile condivisa, e dec(x) che la decrementa di x. Si tratteggi la classe Pippo nei seguenti casi:

1. il decremento può essere fatto solo se il valore della variabile condivisa prima del decremento è positiva, altrimenti il thread viene sospeso;
2. il decremento può essere fatto solo se il valore della variabile condivisa dopo il decremento non diventa negativa, altrimenti il thread viene sospeso;

Per quanto riguarda l'incremento, si considerino i seguenti casi:

3. l'incremento può essere sempre fatto;
4. l'incremento può essere fatto solo se il valore della variabile condivisa prima dell'incremento non supera una certa soglia (una costante definita per la classe dell'oggetto);
5. l'incremento può essere fatto solo se il valore della variabile condivisa dopo l'incremento non supera la soglia di cui sopra.

Come cambierebbe la soluzione se si utilizzasse Ada e il meccanismo di rendez vous tra task?

NB Non è necessario che le soluzioni che date siano rispettose della sintassi del linguaggio (in particolare, ciò vale per Ada. E' invece importante che la soluzione fornita sia rispettosa della semantica dei costrutti del linguaggio.

[R. 1)

JAVA

```
public class Pippo{
   private int value = …;
   public synchronized void dec(int x){
      while(value<=0)
         wait();
      value = value – x;
   }

   public synchronized void inc(int x){
      value = value + x;
      notifyAll();
   }
}
```

ADA

```
task Pippo is
   entry inc(x: in INTEGER);
   entry dec(x: in INTEGER);
end ContoCor;

task body Pippo is
   value: INTEGER := 0;
   loop
      select
         when value >= 0 accept dec(x: in INTEGER) do
              value := value – x;
         end dec;
         or
         accept inc(x: in INTEGER) do
              value := value + x;
         end deposita;
      end inc;
   end loop;
end Pippo;
```

2)

JAVA

```
public class Pippo{
   private int value = …;
   public synchronized void dec(int x){
      while(value<=x)
         wait();
      value = value – x;
   }

   public synchronized void inc(int x){
      value = value + x;
      notifyAll();
   }
}
```

ADA: non e' realizzabile. Il thread che gestisce l'accettazione del rendez vous non puo' conoscere il valore del parametro x prima di aver accettato la richiesta di rendez vous, tuttavia il valore di x serve per poter decidere se il rendez vous puo' essere accettato o meno.

3) Vedi esercizio 1, sia la parte java sia la parte ADA

4)

JAVA

```
public class Pippo{
    private int value = …;
    private int soglia;
    public Pippo (int soglia){
        this.soglia = soglia;
    }
    public synchronized boolean inc(int x){
    if (value>soglia)
        return false;
        value = value + x;
        notifyAll();
        return true;
    }

    public synchronized void dec(int x){
        while(value<x)
            wait();
        value = value – x;
    }
}
```

ADA

```
task Pippo is
   entry inc(x: in INTEGER);
   entry dec(x: in INTEGER);
end Pippo;

task body Pippo is
   value: INTEGER := 0;    soglia: INTEGER := 10000;
   loop
      select
        when value >= 0 accept dec(x in INTEGER) do
             value := value – importo;
          end dec;
        or
        when value < soglia accept inc(x: in INTEGER)
do
             value := value + importo;
          end inc;
     end select;
   end loop;
end Pippo;
```

5)

```
public class Pippo{
    private int value = …;
    private int soglia;
    public Pippo (int soglia){
        this.soglia = soglia;
    }
    public synchronized boolean inc(int x){
        if (value+x>soglia)
            return false;
        value = value + x;
        notifyAll();
        return true;
    }

    public synchronized void dec(int x){
        while(value<x)
            wait();
        value = value – x;
    }
}
```

ADA

Non e' possibile l'implementazione, per considerazioni analoghe a quelle del punto 2: per decidere se accettare o meno la richiesta di incremento, il thread che gestisce i rendez-vous dovrebbe conoscere x. Ma il valore di x puo' essere ottenuto solo dopo aver accettato il rendez-vous.

RECUPERO SECONDA PROVA

Esercizio 1

Si consideri un albero binario di ricerca in cui tutti i nodo hanno valore di chiave distinto. Si definisce *successore* di un nodo dato il nodo che in un attraversamento dell'albero in ordine simmetrico (inorder) verrebbe visitato immediatamente dopo di esso (equivalentemente, il successore è quel nodo che contiene il minimo tra i valori di chiave maggiori di quello del nodo dato). Simmetricamente, il *predecessore* di un nodo è il nodo che nell'attraversamento in ordine simmetrico viene visitato immediatamente prima.

Assumendo che l'interfaccia BinNode contenga, in aggiunta agli altri metodi, anche il seguente

```
public BinNode father(); // restituisce il nodo padre; NULL nel caso il padre non esista
```

implementare un ulteriore metodo della classe BinNode

```
public BinNode successor();
```

che per un nodo (che si suppone faccia parte di un albero binario di ricerca) restituisca il nodo successore (NULL se non c'è successore) *senza accedere al valore della chiave nei nodi dell'albero ma sfruttando solo la struttura dell'albero*. Valutare la complessità di calcolo asintotica di tale metodo per il caso pessimo nei due modi: a) come funzione del numero dei nodi *n* dell'albero, e b) come funzione della profondità *p* dell'albero.

[R.

```
public BinNode successor() {
    if (right != NULL) return left.minNode();
                    //t.minNode() dà il nodo minimo dell'albero di radice t
    BinNode s = this;
    BinNode f = s.father();
          // sali fino a incontrare un nodo che è figlio sinistro del proprio padre
          // o fino alla radice
    while (f != NULL && s == f.right()) {
        s = f;
        f = f.father();
    }
    return f;
}
```

La complessità nel caso pessimo è $\Theta(n)$ e $\Theta(p)$, p essendo $\Theta(n)$ nel caso pessimo di albero totalmente sbilanciato a destra.

]

Dimostrare che se un nodo di un albero binario di ricerca ha entrambi i figli allora il suo nodo successore non ha figlio sinistro, e il nodo predecessore non ha figlio destro.

[R. il successore è il nodo minimo del sottoalbero destro, il predecessore è il massimo del sottoalbero sinistro]

Esercizio 2

È dato un insieme di n giocatori di un certo sport (e.g., calcio). Fra alcuni di essi esiste una relazione di "rivalità personale", rappresentata con un insieme di r coppie di giocatori che sono rivali. (Si noti che tale rivalità è una relazione simmetrica, nel senso che se il giocatore *Tizio* è rivale del giocatore *Caio* allora senz'altro anche *Caio* è rivale di *Tizio*)

Si può facilmente rappresentare la relazione di rivalità mediante un grafo, in cui l'insieme V dei vertici corrisponde a quello dei giocatori. Indicare le caratteristiche di questo grafo (è orientato? è necessariamente ciclico o privo di cicli? è necessariamente connesso o non connesso?).

[R. Il grafo è non orientato, può essere connesso o no, può essere ciclico o no]

Si vogliono formare due squadre utilizzando tutti gli n giocatori in modo tale che tra qualsiasi coppia di giocatori della stessa squadra non esista rivalità; nel formare le squadre si è disposti a rinunciare a ottenere due squadre che abbiano lo stesso numero di giocatori.

Tratteggiare un algoritmo, di complessità $\Theta(n+r)$, che permetta di stabilire se è possibile dividere i giocatori in due squadre nel modo indicato e, nel caso in cui ciò sia possibile, produca l'elenco dei giocatori di ognuna delle due squadre. Si argomenti, informalmente ma in modo chiaro e sintetico, che l'algoritmo proposto è corretto.

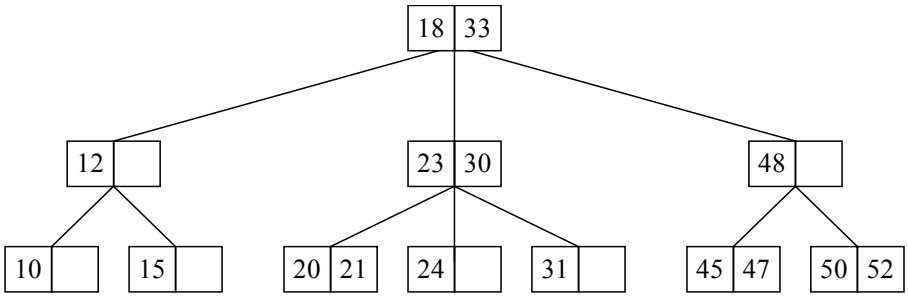
[R. l'algoritmo è una variazione di una visita breadth-first del grafo che tenta di etichettare i nodi del grafo con i "colori" delle due squadre in modo alternato]

Indicare in quali casi è possibile formare le squadre nel modo indicato e in quali casi non lo è, preferibilmente definendo una semplice proprietà dei cammini, presenti nel grafo corrispondente alla relazione di “rivalità personale”, che costituisca una condizione necessaria e sufficiente per poter formare le due squadre.

[R. non devono esistere cicli di lunghezza dispari]

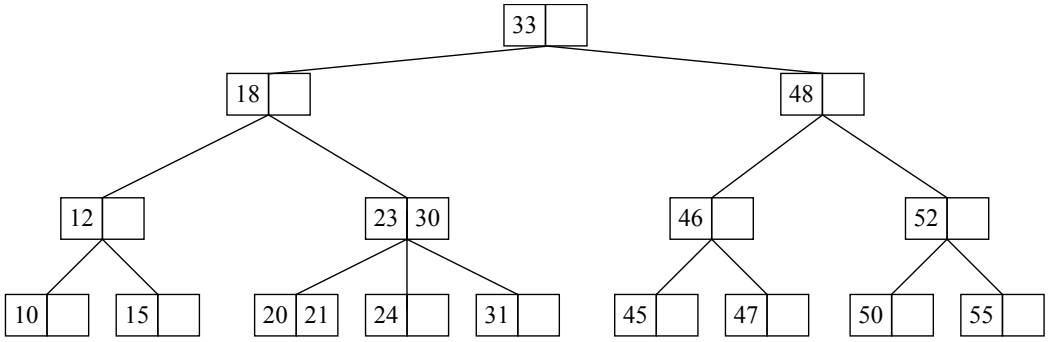
Esercizio 3

Si consideri il 2-3 albero in figura



1. Mostrare il risultato dell’inserimento dei valori 55 e 46.

[R:



]

2. Dimostrare che il numero di foglie di un 2-3 albero con k livelli è compreso tra 2^{k-1} e 3^{k-1}.

[R:

L’albero 2-3 che contiene il numero minimo di elementi, fissato il numero di livelli k, e` un albero binario completo bilanciato, dunque il livello k dell’albero ha 2^{k-1} elementi.

Analogamente l’albero 2-3 piu` grande e` ternario completo bilanciato, dunque il livello k dell’albero ha 3^{k-1} elementi.

]