Nome e cognome _____ N. di matricola (10 cifre num.) _____

UNIVERSITA' DEGLI STUDI DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
CORSO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2006/2007
TEST DI VERIFICA DELLE CONOSCENZE PREGRESSE - 28 settembre 2006

Esercizio 0: Scrivere correttamente nome, cognome e numero di matricola prima di svolgere ogni altro esercizio seguente.

MATEMATICA:

Esercizio 1.1:
Sia (v_i) una successione di valori. La media esponenziale m_n dei primi n valori e' definita per ricorrenza come segue:
$m_0=v_0$; $m_{i+1}=am_i + (1-a)v_{i+1}$ dove il parametro a della combinazione lineare e' scelto nell'intervallo [0,1].

- Calcolare il valore di m_n in funzione di n per la successione $v_i = k$ per ogni i diverso da r, $v_r = h$.
- Studiare la funzione ottenuta del punto precedente al variare del parametro a.

Esercizio 1.2:
Si considerino i vettori v_1=(5x 3x 12), v_2=(9 4y 30), v_3=(-8 10 -4), v_4=(-15 -20 -12). Per quali valori dei coefficienti x e y e' possibile disporre totalmente i vettori in maniera strettamente crescente secondo l'ordine $(\ldots x_i \ldots) \leq (\ldots y_i \ldots)$ sse $\forall i. x_i \leq y_i$?

LOGICA:

Esercizio 2.1:
Costruire la tabella di verita' per l'espressione f(A,B) = $(A \vee B) \Rightarrow (\neg B \; xor \; A)$
f(A,B) e' una tautologia? g(C)=f(C,C) e' una tautologia?

Esercizio 2.2:
Considerare le proposizioni su interi non negativi
P_1= $\forall n \exists m, n < m$; P_2= $\exists m \forall n, n < m$; P_3= $\exists n \forall m, n \leq m$; P_4= $\forall m \exists n, n < m$
Per ogni proposizione dimostrare che e' vera oppure che e' falsa.

Esercizio 2.3:
Considerare la seguente definizione di algoritmo a stack:
"Un algoritmo e' a stack quando indipendentemente dalla stringa di riferimenti s accade che l'insieme P(s,n,t) delle pagine mantenute in memoria all'istante t utilizzando n frame e' un sottoinsieme di P(s,n+1,t)"
Riscrivere formalmente la definizione, facendo attenzione a quantificare nel modo corretto tutte le variabili.

PROGRAMMAZIONE:

Esercizio 3.1:
Scrivere il codice imperativo di una funzione che inverta sul posto un vettore (o array) utilizzando al piu' una sola variabile d'appoggio.

Esercizio 3.2:
Scrivere un programma che legga quattro numeri e stampi la coppia di elementi consecutivi la cui differenza in valore assoluto risulti la massima.

Esercizio 3.3:
Dato un albero binario di interi, scrivere in un linguaggio imperativo una funzione ricorsiva che calcoli la somma degli elementi nell'albero.

ARCHITETTURA:

Esercizio 4.1:
Descrivere il ciclo di Von Neumann senza tralasciare la gestione degli interrupt.

Esercizio 4.2:
Descrivere a cosa serve e come funziona il Direct Memory Access (DMA).

ALGORITMI:

Esercizio 5.1:
Scrivere l'algoritmo verifichi se un grafo diretto ha cicli o e' aciclico. Che complessita' computazionale ha?

UNIVERSITA' DEGLI STUDI DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
CORSO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2006/2007
SOLUZIONI TEST DI VERIFICA DELLE CONOSCENZE PREGRESSE - 28 settembre 2006

MATEMATICA:

Esercizio 1.1:
Sia (v_i) una successione di valori. La media esponenziale m_n dei primi n valori e' definita per ricorrenza come segue:
$m_0=v_0$; $m_{i+1}=am_i + (1-a)v_{i+1}$ dove il parametro a della combinazione lineare e' scelto nell'intervallo [0,1].
- Calcolare il valore di m_n in funzione di n per la successione $v_i = k$ per ogni i diverso da r (r > 0), $v_r = h$.
- Studiare la funzione ottenuta del punto precedente al variare del parametro a.

Soluzione:
$v_m = k$ per m < r; $v_m =$ $a^{m-r}(1-a)(h-k)+a^m k+(1-a)\sum_{i=0}^{m-1} a^i k=a^{m-h}(1-a)(h-k)+k$ altrimenti
Per a = 0 si ha $m_i = k$ per ogni i diverso da r, $m_r=h$.
Per a = 1 si ha $m_i = v_0$ per ogni i, quindi se r=0 allora $m_i = h$ altrimenti $m_i = k$
Per 0 < a < 1 la funzione m_i ha un massimo (o minimo) assoluto per i=r pari a ak+(1-a)h, un asintoto orrizzontale di valore k e decresce (o cresce) esponenzialmente con ragione a a partire dal valore estremante.

Esercizio 1.2:
Si considerino i vettori v_1=(5x 3x 12), v_2=(9 4y 30), v_3=(-8 10 -4), v_4=(-15 -20 -12). Per quali valori dei coefficienti x e y e' possibile disporre totalmente i vettori in maniera strettamente crescente secondo l'ordine $(\dots x_i \dots)\leq(\dots y_i \dots)$ sse $\forall i.x_i \leq y_i$?
Soluzione:
$v_4 < v_3 < v_1 < v_2$ impossibile (vincoli x > 10/3 e x < 9/5)
tutti gli altri ordinamenti impossibili perche' violano i vincoli sulla terza componente dei vettori
Pertanto non e' possibile soddisfare la richiesta

LOGICA:

Esercizio 2.1:
Costruire la tabella di verita' per l'espressione f(A,B) = $(A \vee B)\Rightarrow(\neg B \; xor \; A)$
f(A,B) e' una tautologia? g(C)=f(C,C) e' una tautologia?
Soluzione:

A	B	$A \vee B$	$\neg B \; xor \; A$	$(A \vee B)\Rightarrow(\neg B \; xor \; A)$
0	0	0	1	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

f(A,B) non e' una tautologia perche' f(0,1) non fa 1
g(C) e' una tautologia perche' g(0)=g(1)=1

Esercizio 2.2:
Considerare le proposizioni su interi non negativi
$P_1=$ $\forall n \exists m, n<m$; $P_2=$ $\exists m \forall n, n<m$; $P_3=$ $\exists n \forall m, n\leq m$; $P_4=$ $\forall m \exists n, n<m$
Per ogni proposizione dimostrare che e' vera oppure che e' falsa.
Soluzione:
P_1 e' vera: fissiamo un n generico (per provare un quantificatore universale);
scegliamo m = n+1 (per provare un quantificatore esistenziale basta scegliere un esempio);
n < n+1 Qed.
P_2 e' falsa: fissiamo un m generico (per refutare un quantificatore esistenziale);
scegliamo n = m + 1 (per refutare un quantificatore universale basta scegliere un esempio);
m + 1 < m e' falso Qed.
P_3 e' vera: scegliamo n = 0 (per provare un quantificatore esistenziale basta scegliere un esempio);
fissiamo un m generico (per provare un quantificatore universale);
$0\leq m$ Qed.
P_4 e' falsa: scegliamo m = 0 (per refutare un quantificatore universale basta scegliere un esempio);
fissiamo un n generico (per refutare un quantificatore esistenziale);
n < 0 e' falso Qed.

Esercizio 2.3:
Considerare la seguente definizione di algoritmo a stack:
"Un algoritmo e' a stack quando indipendentemente dalla stringa di riferimenti s accade che l'insieme P(s,n,t) delle pagine mantenute in memoria all'istante t utilizzando n frame e' un sottoinsieme di P(s,n+1,t)"
Riscrivere formalmente la definizione, facendo attenzione a quantificare nel modo corretto tutte le variabili.
Soluzione:
$\forall s \forall n \forall t, P(s,n,t)\subseteq P(s,n+1,t)$

PROGRAMMAZIONE:

Esercizio 3.1:
Scrivere il codice imperativo di una funzione che inverta sul posto un vettore (o array) utilizzando al piu' una sola variabile d'appoggio.

```
void flipv(int *v, int len)
{
    int i,j;
    for (i=0,j=len-1; i<j; i++,j--) {
      int tmp;
      tmp=v[i];
      v[i]=v[j];
      v[j]=tmp;
    }
}
```

Esercizio 3.2:
Scrivere un programma che legga quattro numeri e stampi la coppia di elementi consecutivi la cui differenza in valore assoluto risulti la massima.

```
main()
{
    int i, maxi;
    int maxdiff;
    int v[4];
    for (i=0;i<4;i++)
       sscanf("%d", v[i]);
    for (i=maxi=madiff=0;i<3;i++) {
       int diff;
       if (v[i+1] > v[i])
            diff=v[i+1]-v[i];
       else
            diff=v[i]-v[i+1];
       if (diff > maxdiff) {
            maxdiff = diff;
            maxi=i;
       }
    }
    printf("max gap: v[%d]=%d   v[%d]=%d\n",i,v[i],i+1,v[i+1]);
}
```

Esercizio 3.3:
Dato un albero binario di interi, scrivere in un linguaggio imperativo una funzione ricorsiva che calcoli la somma degli elementi nell'albero.

Soluzione:
se il nodo dell'albero ha il seguente formato:

```
struct nodo {
    int val;
    struct nodo *left, *right;
}
```

La funzione sara':

```
int somma(struct nodo *n)
{
    if (n == NULL)
        return 0;
    else
        return n->val + somma(n->left) + somma(n->right);
}
```

ARCHITETTURA:

Esercizio 4.1:
Descrivere il ciclo di Von Neumann senza tralasciare la gestione degli interrupt.

Fase 1: fetch dell'istruzione. Viene caricato il contenuto della memoria all'indirizzo specificato dal program counter (o instruction counter) e viene posto nell'instruction register.

Fase 2: fase di decodifica. l'istruzione viene decodificata. Durante la fase di decodifica il program counter viene incrementato.

Mentre le fasi precedenti sono indipendenti dalla istruzione, quelle seguenti dipendono dal codice operativo dell'istruzione da eseguire
Fase 3: fetch degli operandi (se richiesto dalla istruzione): se l'istruzione prevede una lettura in memoria si provvede all'operazione

Fase 4: execute. Il processore esegue l'istruzione

Fase 5: store del risultato (se richiesto dall'istruzione).

Le fasi dalla 1 alla 5 sono non interrompibili. Prima di ritornare alla fase 1 si verifica se vi sono state richieste di interrupt non mascherate. Se ve ne sono state si prosegue con la fase 6 altrimenti si torna alla fase 1.

Fase 6: salvataggio del program couter corrente e caricamento del program couter previsto per la gestione dell'interrupt.

La spiegazione fa riferimento ad una ipotetica didattica macchina RISC, il ciclo di macchina per i processori CISC si riferisce all'esecuzione di una microistruzione, l'istruzione viene infatti eseguita tramite un vero e proprio interprete scritto in microcodice. In questo caso la fase 6 viene eseguita solo quando il microcodice lo consente (alla conclusione dell'istruzione e non della singola microistruzione).

Esercizio 4.2:
Descrivere a cosa serve e come funziona il Direct Memory Access (DMA).

Il DMA serve per migliorare le prestazioni dei sistemi multitasking. In questi sistemi infatti le operazioni di I-O di un processo avvengono contemporaneamente alla esecuzione di altri processi. Le periferiche prive di DMA richiedono che l'I-O avvenga sulla base di dati presenti sull'interfaccia della periferica. E' quindi necessario che il sistema operativo provveda a scrivere il dato nella memoria dell'interfaccia prima di attivare una operazione **write** e dopo una operazione **read** deve copiare i dati letti nella memoria centrale. Con DMA l'interfaccia puo' direttamente scambiare i dati con la memoria centrale evitando il costo delle copie al processore. DMA e' possibile perche' il processore non usa sempre il bus di interscambio con la memoria e le periferiche. Le interfacce possono quindi usare i cicli non occupati dal processore centrale (cycle stealing) per provvedere al trasferimento dei propri dati. Perche' DMA possa funzionare occorre che nel sistema sia presente un **bus arbiter** capace di dirimere le contese di accesso al bus. Infatti se nei sistemi senza DMA il processore e' l'unica componente attiva sul bus, con DMA il processore e tutte le interfacce diventano componenti attive che contendono l'uso del bus.

ALGORITMI:

Esercizio 5.1:
Scrivere l'algoritmo verifichi se un grafo diretto ha cicli o e' aciclico. Che complessita' computazionale ha?

```
DAGverify(int N, Arcset R,)  //,R matrice NxN
{
      R1=R;      //copia l'insieme degli archi
      for h=1,...,log2N
        for i=1,..,N          //chiusura transitiva
            for j=1,...,N
               for k=1,...,N
                  if (R1[i,j] and R1[j,k]) then R1[i,k] = 1;   // oppure =R1[i,j]+R1[j,k] se si vuole calcolare la distanza
       result = 0;
       for i=1,...N
            result = result or R1[i,i];    // se nella chiusura R1[i,i] e' 1 per qualche i allora e' ciclico.
       return (not result);   //se non ha cicli e' un DAG (grafo diretto aciclico).
}
```

L'algoritmo e' in $O(N^3 \log_2 N)$ dove N e' il numero di nodi.