Informatica Generale II - Prova teorica

A.A. 2005/2006

Esame: 23 giugno 2006
Codice: GMZK

1. Si consideri l'algoritmo di ordinamento per fusione (mergesort):
   (a) si avvale dell'algoritmo di fusione tra due array ordinati (merge) che ha complessità lineare. Tale algoritmo di fusione viene attivato un numero di volte pari a $\mathcal{O}(log\, n)$ e da ciò si ha che la complessità dell'algoritmo mergesort è $\mathcal{O}(n * log\, n)$;
   (b) si avvale dell'algoritmo di fusione tra due array ordinati (merge) che ha complessità $\mathcal{O}(log\, n)$. Tale algoritmo di fusione viene attivato un numero di volte pari a $\mathcal{O}(n)$ e da ciò si ha che la complessità dell'algoritmo mergesort è $\mathcal{O}(n * log\, n)$;
   (c) si avvale dell'algoritmo di fusione tra due array ordinati (merge) che ha complessità $\mathcal{O}(1)$. Tale algoritmo di fusione viene attivato un numero di volte pari a $\mathcal{O}(log\, n)$ e da ciò si ha che la complessità dell'algoritmo mergesort è $\mathcal{O}(log\, n)$;
   (d) si avvale dell'algoritmo di fusione tra due array ordinati (merge) che ha complessità $\mathcal{O}(n^2)$. Tale algoritmo di fusione viene attivato un numero di volte pari a $\mathcal{O}(log\, n)$ e da ciò si ha che la complessità dell'algoritmo mergesort è $\mathcal{O}(n^2 * log\, n)$;
   (e) non rispondo

2. Si consideri il seguente frammento di codice:

```
struct Tipo1 {
 int a;

 Tipo1(int _a) { a = _a; }
};

struct Tipo2 {
 Tipo1 a;

 Tipo2(int _a) {
   a = Tipo1(_a);
 }
};
```

   (a) è errato perché Tipo1 e Tipo2 contengono due campi con lo stesso nome.
   (b) è errato, ma sarebbe corretto se esistesse anche il costruttore `Tipo1() {a = 0;}`.
   (c) è errato perché Tipo2 non ha un costruttore di default.
   (d) è corretto e compilabile.
   (e) non rispondo

3. Si supponga di avere una funzione `visit` di visita di un albero che inserisce il valore del nodo visitato in fondo ad una lista. Per ottenere una lista ordinata in senso decrescente dall' attraversamento di un BST utilizzando la funzione di visita `visit`:
   (a) si può attraversare il BST in postorder e poi invertire la lista;
   (b) si può attraversare il BST inorder;
   (c) si può attraversare il BST in preorder;
   (d) si può attraversare il BST inorder e poi invertire la lista;
   (e) non rispondo

4. Quale delle seguenti affermazioni è *falsa*

(a) Diciamo che un nodo n1 è sotto n2 (n2 è sopra n1) se n1 è nel cammino tra n2 e la radice.
(b) I nodi senza figli vengono detti foglie.
(c) Un albero ordinato è un albero con radice e è specificato l'ordine dei figli di ciascun nodo.
(d) Ogni nodo di una albero tranne la radice ha un solo nodo sopra di se, detto nodo padre.
(e) non rispondo

5. Si consideri la rappresentazione di un grafo mediante *matrice delle adiacenze* (sia n il numero di nodi). Quale tra le seguenti affermazioni è *falsa*?

(a) tale rappresentazione richiede un'occupazione di memoria proporzionale al numero di nodi del grafo;
(b) l'elemento nella riga i e nella colonna j della matrice è pari a 1 se nel grafo rappresentato c'è un arco dal nodo i al nodo j, è pari a 0 nel caso contrario;
(c) tale rappresentazione richiede un'occupazione di memoria proporzionale al numero massimo di archi del grafo;
(d) l'operazione di accesso ai successori di un nodo richiede l'accesso ad n elementi della matrice;
(e) non rispondo

6. In quale ordine partendo dal nodo C vengono visitati i nodi del grafo in figura 1 da un algoritmo di visita in profondità?

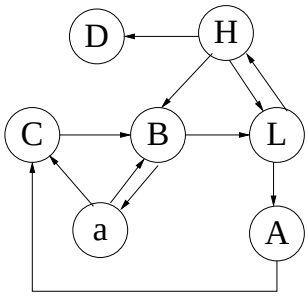

Figura 1:

(a) Nessuna risposta è accettabile;
(b) CBLaHAD
(c) CBLHDAa
(d) CBLaADH
(e) non rispondo

7. Si supponga di avere la sequenza di numeri *1,6,4,2,3*. Dopo *3* iterazioni soltanto di Bubblesort su tale sequenza, il risultato sarà:

(a) 1,6,2,3,4
(b) 1,2,6,3,4
(c) 1,2,3,6,4
(d) 1,2,6,4,3
(e) non rispondo

8. Si consideri la seguente funzione:

```
int* funz(int dim) {
  int* punt;
  punt=new int[dim];
  for (int i=0; i < dim; ++i) punt[i]=i;
  return punt;
}
```

La memoria allocata all'interno della funzione `funz` mediante l'operatore `new`:

(a) viene deallocata quando il controllo passa all'esterno della funzione `funz`
(b) non può più venire deallocata poiché l'operatore `delete` non compare all'interno della funzione stessa
(c) rimane allocata fin quando il programma non termina o finché non venga esplicitamente deallocata tramite una istruzione `delete []`
(d) rimane allocata fin quando il programma non termina o finché non viene esplicitamente deallocata tramite l'istruzione `delete`
(e) non rispondo

9. Quale tipo di attraversamento d'albero implementa il seguente codice?

```
void foo-order(Node* l, void visit(Node*))
  StackPtr s = new Stack();

  Push(s, i);
  while(! StackIsEmpty(s)) {
    Node * h = Pop(s);

    if ((h->left == NULL) && (h->right == NULL)) visit(h);
    else Push(s, new Node(h->data, true));
    if (h->right != NULL) Push(s, h->right);
    if (h->left != NULL) Push(s, h->left);
    if (h->flag == true) delete h;
  }

  delete s;
}
```

(a) level-order
(b) postorder
(c) inorder
(d) preorder
(e) non rispondo

10. Le operazioni di base di stack e code sono:

(a) push e pop per code, put e get per stack
(b) put e get
(c) push e pop per stack, put e get per code
(d) push e pop
(e) non rispondo