008AA – ALGORITMICA E LABORATORIO

Appello del 5 settembre 2011: Soluzioni

Esercizio 1.
Si implementi in C una rudimentale gestione di array dinamico. In particolare:

1. si definisca una struttura `ArrayDinamico` per la gestione dell'array dinamico. La struttura deve contenere campi per memorizzare gli elementi dell'array, la dimensione corrente dell'array, e il numero di elementi effettivi memorizzati;

2. si definisca una procedura `stampaArray()` che, preso come argomento un puntatore a una struttura, stampi tutti gli elementi dell'array;

3. si definisca un funzione `Crea()` che, preso come argomento un intero `size`, restituisca il puntatore a una struttura contenente un array di dimensione `size`;

4. si definisca una funzione `addElem()` che, presi come argomenti un puntatore a una struttura e un intero `elem`, aggiunga `elem` in fondo all'array, e restituisca il puntatore alla struttura così modificata. Si gestisca il caso in cui l'array sia pieno con i metodi visti a lezione, in modo tale che il costo ammortizzato dell'operazione risulti costante.

NOTA: si utilizzi opportunamente la funzione di libreria `free(ptr)` per liberare la memoria puntata dal puntatore `ptr` passato come argomento.

```
#include <stdio.h>
#include <stdlib.h>

struct ArrayDinamico {
    int *punt;
    int dim;
    int numElem;
};

void stampaArray (struct ArrayDinamico *a){
    int i;
    //struct ArrayDinamico b = *a;
    int dimArray = a -> dim;
    for (i=0; i<dimArray; i++){
        printf ("%d ", a-> punt[i]);
    }
    printf ("%d, %d \n", a -> dim, a -> numElem);
}

struct ArrayDinamico * Crea (int size){
    struct ArrayDinamico *a = malloc (sizeof(struct ArrayDinamico));
    (*a).punt = malloc (size * sizeof(int));
    (*a).dim = size;
    (*a).numElem = 0;
    return a;
```

```
}

struct ArrayDinamico* addElem (struct ArrayDinamico *a, int elem){
    struct ArrayDinamico b = *a;

    if (a->numElem == a->dim) {
        struct ArrayDinamico *a2 = Crea((a->dim)*2);
        //struct ArrayDinamico b2 = *a2;
        int i;
        for (i=0; i<(a->dim); i++) {
            a2->punt[i] = a->punt[i];
        }
        a2 -> punt[i] = elem;
        a2 -> numElem = a -> numElem + 1;
       free(a);
        a = a2;
    } else{
        a -> punt[a -> numElem]=elem;
        a -> numElem++;
    }
    return a;
}

main()
{
    struct ArrayDinamico *a = Crea(4);
    a = addElem(a, 10);
    stampaArray (a);
    a = addElem(a, 11);
    stampaArray (a);
    a = addElem(a, 12);
    stampaArray (a);
    a = addElem(a, 13);
    stampaArray (a);
    a = addElem(a, 14);
    stampaArray (a);
}
```

Soluzione.

Esercizio 2.
Sia dato il problema del calcolo del valore massimo in un array.

1. Descrivere a parole e formalizzare in pseudocodice due procedure ricorsive che risolvano il problema suddetto e le cui complessità in tempo soddisfino rispettivamente le seguenti relazioni di ricorrenza:

- $T(n) = T(n-1) + O(1)$
- $T(n) = 2T(n/2) + O(1)$

2. Si risolvano le relazioni precedenti.

3. Si dimostri che i due algoritmi proposti sono ottimi.

Soluzione.

1.
```
Max(a, n) // n = dim dell'array
{
   if (n == 1) return a[0];
   max = Max(a, n-1);
   if (max > a[n-1]) return max;
   else return a[n-1];
   }
```

```
Max(a, i, j)
{
   if (i == j) return a[i];
   c = (i+j)/2;
   max1 = Max(a, i, c);
   max2 = Max(a, c+1, j);
   if (max1 >= max2) return max1;
   else return max2;
```

2. Sia c il termine costanmte della ricorrenza $T(n) = T(n-1) + O(1)$.

   $T(n) = T(n-1)+c = T(n-2)+2c = ... = T(n-i)+ic = ... = T(1)+(n-1)\,c = O(n)\,.$

   La soluzione della ricorrenza $T(n) = 2T(n/2) + O(1)$ si ottiene applicando il teorema fondamentale (primo caso): $T(n) = O(n)$.

3. I due algoritmi sono ottimi in quanto non e' possibile individuare il massimo in un array non ordinato con meno di $n-1$ confronti.

Esercizio 3.
Data la sequenza di chiavi intere

$$10, 5, 20, 15, 18, 25, 29,$$

costruire un albero AVL per inserzioni successive, disegnando l'albero dopo l'inserzione di ogni chiave e indicando, prima di ogni rotazione, il nodo critico e il tipo di rotazione eseguita.

Soluzione. Sono necessarie tre rotazioni: DS(20), DD(10) e DD(20).

Esercizio 4.
Dato un insieme di n chiavi, si vogliono realizzare le seguenti operazioni: inserzione di una nuova chiave, estrazione della chiave minima ed estrazione della chiave massima. Discutere la complessità in tempo di queste operazioni nel caso in cui l'insieme venga realizzato mediante

1. array,
2. tabella hash con liste concatenate,
3. albero binario di ricerca,
4. albero di ricerca bilanciato (es: AVL).

Soluzione.

1. array (non ordinato)
   - inserzione di una nuova chiave: $O(1)$
   - estrazione della chiave minima: $O(n)$
   - estrazione della chiave massima: $O(n)$
2. tabella hash con liste concatenate
   - inserzione di una nuova chiave: $O(1)$
   - estrazione della chiave minima: $O(n)$
   - estrazione della chiave massima: $O(n)$
3. albero binario di ricerca
   - inserzione di una nuova chiave: $O(h) = O(n)$
   - estrazione della chiave minima: $O(h) = O(n)$
   - estrazione della chiave massima: $O(h) = O(n)$
4. albero di ricerca bilanciato (es: AVL)
   - inserzione di una nuova chiave: $O(h) = O(\log n)$
   - estrazione della chiave minima: $O(h) = O(\log n)$
   - estrazione della chiave massima: $O(h) = O(\log n)$