BreviarTeoretic - Recursivitate (II)

February 27, 2017

#
Structuri de date
##
Laborator 2 - Recursivitate (II)Breviar teoretic
###
Iulian-Gabriel Radu

0.1 1. Divide et Impera

0.1.1 1.1. Noiuni generale

Divide et impera se bazeaz pe principiul descompunerii problemei în dou sau mai multe subprobleme (mai uoare), care se rezolv, iar soluia pentru problema iniial se obine combinând soluiile subproblemelor.

De multe ori, subproblemele sunt de acelai tip i pentru fiecare din ele se poate aplica aceeai tactic a descompunerii în (alte) subprobleme, pân când (în urma descompunerilor repetate) se ajunge la probleme care admit rezolvare imediat.

0.1.2 1.2. Implementri

0.1.3 1.2.1. Cutare binar - Binary Search

Problema: Se da un sir de numere **ordonat crescator** cu **N** elemente i un numr **x**. S se verifice dac numrul **x** se gsete în sirul dat sau nu.

Algoritm:

```
/* Iniial: low = 0, high = N - 1 */
int binary_search(int A[0..N-1], int x, int low, int high)
{
    if (high < low)
        return -1

    int mid = (low + high) / 2
    if (x < A[mid])
        return binary_search(A, value, low, mid - 1)
    else if (x > A[mid])
        return binary_search(A, value, mid + 1, high)
    else
        return mid
}
```

0.1.4 1.2.2. Sortarea prin interclasare - Merge Sort

Problema: Se d un ir de numere cu **N** elemente. S se sorteze irul în mod **cresctor** folosind algoritmul de sortare prin interclasare.

Algoritm:

```
int* merge_sort(int A[0..N-1])
{
    if (N <= 1)
        return A
    else {
        int mid = N/2
        int left[0..N/2-1] = A[0] ... A[mid-1]
        int right[0..N/2-1] = A[mid] ... A[N-1]

        merge_sort(left)
        merge_sort(right)

        return merge(left, right)
    }
}

int* merge(int A[0..M-1], int B[0..N-1])
{
    int C[0..N-1]

    /* Add elements from either A or B */
    while (i < M and j < N) {
        if (A[i] <= B[j])
            add A[i++] to C
        else
            add B[j++] to C
    }

    /* Add remaining elements from A */
    while (i < M)
        add A[i++] to C

    /* Add remaining elements from B */
    while (j < N)
        add B[j++] to C

    return C
}
```

0.2 2. Backtracking

0.2.1 2.1. Noiuni generale

Backtracking este o metod de parcurgere sistematic a spaiului soluiilor posibile al unei probleme. Este o metod general de programare, i poate fi adapt pentru orice problem pentru care dorim s obinem toate soluiile posibile, sau s selectm o soluie optim, din mulimea soluiilor posibile. Backtracking este îns i cea mai costisitoare metod din punct de vedere al timpului de execuie.

0.2.2 2.2. Implementri

0.2.3 2.2.1. Permutri

Problema: S se scrie o funcie recursiv pentru a genera i afia pe ecran toate **permutrile** unui ir de caractere citit de la tastatur.

Algoritm:

```
void permute(char A[0..N-1], int l, int r)
{
   if (l == r)
     printf("%s\n", a)
   else
   {
       for (i = l; i <= r; i++)
       {
          swap(A+l, A+i)
          permute(A, l+1, r)
          swap(A+l, A+i) //backtrack
       }
   }
}
```