RO: Metode de programare recursive si nerecursive. Metoda Backtracking. Metoda Divide et Impera. Tehnici de cautare.

EN: Recursive and non-recursive programming methods. The Backtracking method. The Divide et Impera method. Searching techniques.

Obiective:

- Înțelegerea tipurilor de probleme la care se pretează folosirea metodelor „backtracking” si „divide et impera”.
- Înţelegerea mecanismului de implementare a unor aplicatii folosind metode de tip „backtracking” si „divide et impera”.
- Înţelegerea tehnicilor de căutare.

Objectives:

- Understanding the problem types that can be solved using the „backtracking” and “divide et impera” techniques.
- Understanding the programming mechanism for the applications that use „backtracking” and „divide et impera”.
- Understanding the searching techniques.

Rezumat:

A. Metoda *backtracking*

Soluţia are mai multe componente x[k], k=0,n-1 şi fiecare componenta poate lua valori x[k]=1,…,h.

1. VARIANTA NERECURSIVĂ

```
//initializari
k=0;
x[k]=0;
/*repeta pana cand ne-am intors
inapoi mai mult decat trebuia*/
do{
        //atata timp cat mai sunt valori de ales
      while   ( x[k]<h )

                              {
      //trec la urmatoarea valoare
      x[k]=x[k]+1;

      if  (posibil(k))    //daca este solutie partiala corecta

        if     ( k==n-1)     //daca am ajuns la o solutie completa
                              afiseaza_solutia(X);
                      else  //trec la urmatoarea componenta
                                    {
                                                k=k+1;
                                                x[k]=0;
                                    }
          }//while
      k=k-1; /*fac pasul inapoi*/
}while(! (k<0));// (k>=0)
```

2. VARIANTA RECURSIVĂ

```
void Backtracking(int k)
{
//pentru toate valorile pe care le poate lua X[k]
for(int i=1;   i <=   h   ;  i++)
{
  x[k]=i;
        if  ( posibil(k)) //sol. partiala posibila

        if  (k==n-1)  //solutie completa

                afiseaza_solutia(X);
        else
                Backtracking(k+1);
                //apel recursiv
   }//for
}//Backtracking
```

Apel in main():

```
//initializari
Backtracking(0);
```

- Solutia este reprezentata ca un *vector* (implementat printr-un tablou unidimensional) ***X***:
  $\boldsymbol{X}(x_1, x_2, \ldots, x_n) \in \boldsymbol{S}=S_1xS_2x\ldots xS_n$
- unde:
  - *S* e spatial solutiilor posibile care este finit (produs cartezian a S_i)
  - x_i, *i=1,...n,* sunt componentele solutiei S, cu valori in S_i

Funcţia ***posibil(k)*** returnează**:**
-TRUE dacă e soluţie parţială corectă, adică posibilă şi se poate continua sau

-FALSE în caz contrar.
Se poate să nu fie o funcţie pentru cazurile mai simple sau funcţia poate avea mai mulţi parametrii în cazul în care condiţia de continuitate este mai complexă sau ea oferă valori în etapa verificării dacă avem o soluţie completă sau nu (ce se poate implementa dacă e mai complicată cu o funcţie ***gata_solutie()***).
Funcţia ***afiseaza_solutia(X),*** are rolul de a afişa o soluţie completă obţinută la un moment dat. Funcţie de problema care e rezolvată poate să aibă diferite forme de implementare.

La ori ce problema standard de backtracking se stabilesc:
1. *k = 0,...n-1*, adica numarul de componente ale unei solutii
2. *x[k]=1,...h*, adica domeniul valorilor posibile ale unei componente
3. *posibil(k)*, adica conditia de continuitate ce e o conditie interna care stabileste ce relatii trebuie respectate intre componentele unei solutii.
Atât în varianta recursivă cât şi în cea nerecursivă iniţial se determină (citesc) date referitoare la:
- numărul componentelor, *n*
- domeniul valorilor, *h*
- alte date necesare pentru elaborarea algoritmului

De asemenea sunt unele aplicaţii în care se impun unele restricţii de pornire sau de altă natură ele fiind tratate în consecinţă. Există variante de backtracking în care soluţiile nu au aceiaşi lungime (nr. variabil de componente) sau se cere obţinerea unor soluţii optimale, etc.

B. Metoda *divide et impera*

Principiul propus de metodă este:
- Descompune problema în subprobleme în mod recursiv, până când ajungem la o subproblemă pe care o putem rezolva.
- Soluţiile subproblemelor se vor combina obţinând soluţia finală a problemei

Tehnici de căutare

În general se pune problema căutării unor obiecte pe baza valorii unui câmp (cheie) asociat fiecărui obiect. Vom considera că obiectele sunt grupate în tablouri unidimensionale şi că pentru cheie este definită o relaţie de ordine.

În cazul în care obiectele nu sunt ordonate, nu este posibilă decât o căutare directă, adică parcurgerea liniară a tabloului şi compararea valorii cheii cu valoarea de căutat. Dacă însă, obiectele sunt ordonate găsirea unui obiect se poate face mai rapid. Un algoritm în acest sens este cel de căutare binară atât în variantă recursivă, cât și nerecursivă, ce folosește înjumătățirea intervalului, în fond un algoritm "Divide et impera".

În cazul unor tablouri neordonate, se pot folosi următoarele funcţii de bibliotecă (exista si alte variante ale acestor functii in noile variante ale mediului VC++ care pot fi folosite):

*void *lfind(const void *key, const void *base, size_t *num, size_t width, int (*fcmp)(const void *, const void*));*
*void *lsearch(const void *key, const void *base, size_t *num, size_t width, int (*fcmp)(const void *, const void *));*

În cazul tablourilor ordonate există funcţia de bibliotecă *bsearch( )* pentru căutarea binară (înjumătăţirea intervalului):

*void *bsearch(const void *key, const void *base, size_t nelem, size_t width, int (*fcmp)(const void*, const void*));*

Exemple:

1. Colorarea unui drapel cu n (<=5) benzi, fiecare bandă putând avea h culori
a)Varianta nerecursivă

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define DIM 5

int posibil(int);
void afis_sol(void);

int x[DIM], n;

int main( )
        {
        int k,h;
        printf("\nIntrodu numarul de culori ale unui drapel (benzi <=DIM=5): ");
        scanf("%d",&n);
        printf("\nIntrodu numarul maxim de culori posibile ale unei benzi: ");
        scanf("%d",&h);
        k=0;
```

```
	x[k]=0;
	do  {
		while(x[k] < h) //mai sunt valori posibile pentru componenta k
			{
			x[k]++; //trec la urmatoarea valoare
			if(posibil(k))
				if(k==(n-1))
					afis_sol();  //e gata solutia
				else
				{
					k++;
					x[k]=0;
				} //trec la urmatoarea componenta cu valori de la zero
			}//while
		k--; //nu mai sunt valori pentru componenta k; revin la componenta k-1
	    }while(!(k<0));//m-am intors mai mult decat se putea
	}//main

int posibil(int k)
	{
	if(k==0) return 1;			//intial este posibil tot
	if(x[k-1] == x[k]) return 0;		//am doua culori alaturate ce nu e posibil
	return 1;				//nu sunt culori alaturate
	}//posibil

void afis_sol(void)
	{
	//afisez solutia curenta pentru componentele drapelului
	for(int i=0;i<n;i++)
		printf("%d ",x[i]);
	printf("\n");
	}//afis_sol
```

b)Varianta recursivă

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define DIM 5

int posibil(int);
void afis_sol(void);
void dr_rec(int k);
int x[DIM],n;
int h;

int main( )
	{
	printf("\nIntrodu numarul de culori ale unui drapel(benzi <=DIM=5): ");
	scanf("%d",&n);
	printf("\nIntrodu numarul maxim de culori posibile ale unei benzi: ");
	scanf("%d",&h);
	dr_rec(0);
	}//main

void dr_rec(int k)
	{
	for(int i=1;i<=h;i++)
		{
		 x[k]=i;
		if(posibil(k))
			if(k==(n-1))
				afis_sol();  //e gata solutia
			else
				dr_rec(k+1); //trecla urmatoarea componenta
```

```
            }
        }//dr_rec
```

Functiile:

```
int posibil(int k);
void afis_sol(void);
```

sunt identice cu cele de la varianta nerecursiva

2. Descompunerea unui număr n, în suma de numere naturale (backtracking cu soluţii având un număr variabil de componente)

a) Varianta recursivă :

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define DIM 20

void back_SubSet(int);
int posibil(int, int&);
void afis_sol(int);

int x[DIM],n;

int main( )
{
        printf("\nIntroduceti numarul care urmeaza sa fie descompus( <=DIM=20) : ");
        scanf("%d",&n);
        printf("\n\tSolutii posibile :\n");
        back_SubSet(0);
}//end main

void back_SubSet(int k)
{
int sum;
        x[k] = 0;
        while(x[k] < n)
        {
                x[k]++;
                if(Posibil(k, sum))
                {
                        if(sum==n)
                                afis_sol(k);
                        else
                                back_SubSet(k+1);
                }//end else
        }//end while
}//end back_SubSet

int posibil(int k, int &s)
{
        s=0;
        if(k==0)return 1;//initial e posibil orice
        if(x[k] >= x[k-1])
        {
                for( int i=0;i<=k;i++)
                        s += x[i];
                if(s <= n) return 1;
        }//end if
        return 0;
}//end posibil

void afis_sol(int k)
{
        printf("\n\t");
```

```
	for(int i=0;i<=k;i++)
		printf("%d  ",x[i]);
}//end afis_sol
```

b) Varianta iterativă :

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define DIM 20

int posibil(int, int&);
void afis_sol(int);

int x[DIM], n;

int main( )
{
int k,sum;
	printf("\nIntroduceti numarul care urmeaza sa fie descompus( <=DIM=20) : ");
	scanf("%d",&n);
	printf("\n\tSolutii posibile :\n");

	k=0;
	x[k]=0;
	do  {
		while(x[k] < n)
		{
			x[k]++;
			if(posibil(k, sum))
			{
				if(sum==n)
					afis_sol(k);
				else
				{
					k++;
					x[k]=0;
				}//end else
			}//end if
		}//end while
		k--;
	}while(!(k<0)); //end do…while
}//main
```

Functiile:

```
	int posibil(int k, int &s);
	void afis_sol(int k);
```

sunt identice ca si la varianta recursiva

3. Divide et impera. Turnurile din Hanoi

```
//*************************************************************************
//Problema turnurilor din Hanoi (cu solutie in mod text)
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

double mutari; // numar de mutari
void hanoi(int n, char a, char b, char c);

int main( )
{
int n;
	printf("Numarul de discuri este = ");
	scanf("%d",&n);
	mutari=0;
```

```
	hanoi(n, 'A', 'B', 'C');
	printf("Numarul de mutari este = %lf\n", mutari);
}  //main

void hanoi(int n, char a, char b, char c)
 //n-nr. discuri, a-sursa, b-destinatia, c-manevra
{
	if (n==1)// n=1
		{
		printf("Se muta discul 1 de pe tija %c pe tija %c\n",a,b);
		 mutari++;
		 return;
		 }

	// n>1, Etapa 1, Problema pt.n-1 discuri a-sursa, c-destinatia, b-manevra
		 hanoi(n-1,a,c,b);

	// Etapa 2, Discul n de pe tija a se muta pe b
		 printf("Discul %d de pe tija %c se muta pe tija %c\n",n,a,b);
		 mutari++;

	//Etapa 3, Problema pt. n-1 discuri c-sursa, b-destinatia, a-manevra
		 hanoi(n-1,c,b,a);
} // hanoi
```

4. Exemple complete pentru utilizarea funcțiilor de bibiotecă ce realizează căutări

//Programul foloseste functia de biblioteca *lsearch()* pentru a gasi numele unei luni
//daca nu o gaseste o adauga la finalul sirului in care se face cautarea.
// Modificati aplicatia astfel incat luna se va introduce de la tastatura

```
#include <stdio.h>
#include <string.h>
#include <search.h>

#define DIM 12

int cmp(const char *arg1, const char *arg2);
int addelem(const char **key, const char **tab, int nelem);

int main( ) {
	const char *luni[DIM] = { "ian", "feb", "mar", "apr", "mai", "iun" };
	int nluni = 6;
	int i;
	const char *key = "iul";
	if (addelem(&key, luni, nluni))
		printf("Luna %s este deja in tablou.\n", key);
	else {
		nluni++;
		printf("Luna \"%s\" a fost adaugata in tablou: ", key);
		for (i = 0; i < nluni; i++)
			printf("%s, ", luni[i]);
	}
}

int addelem(const char **key, const char **tab, int nelem)
{
	int oldn = nelem;
	_lsearch(key, tab, (size_t *)&nelem, sizeof(char *),
		(int(*)(const void *, const void *))cmp);
	return(nelem == oldn);
}
```

```
int cmp(const char *arg1, const char *arg2)
{
        return(strcmp(arg1, arg2));
}
//*****************************************************************************
// utilizarea functiei de biblioteca lfind( )
#include <stdlib.h>
#include <stdio.h>
#include <search.h>
#define DIM 10

int compare_int(int *a, int *b);

int main( )
{
int int_values[DIM] = {1, 3, 2, 4, 5};
int *int_ptr, key_value, num = 5; //key_value stocheaza valoarea cautata
printf("Introduceti valoarea pe care o cautati in tablou: ");
scanf_s("%d",&key_value);
        //apelul functiei lfind()
        int_ptr = (int *)_lfind(&key_value, int_values, (size_t *)&num, sizeof(int),
                (int (*) (const void *, const void *)) compare_int);
        if(int_ptr!=0)
printf("Numarul %d se gaseste in tablou pe pozitia : %d si la adresa %p\n",key_value,int_ptr-int_values+1,  int_ptr);
        else
           printf("Numarul %d nu se gaseste in tablou\n",key_value);
}//end main( )

int compare_int(int *a, int *b)
{
        return(*a - *b);
}//end compare_int( )

//*****************************************************************************
// utilizarea functiei de biblioteca bsearch( )

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

int compare_int(int *a, int *b);
int compare_float(float *a, float *b);

int main( )
{
int int_values[ ] = {1, 2 , 3, 4, 5};
float float_values[ ] = {1.1f, 2.2f, 3.3f, 4.4f, 5.5f};

int *int_ptr, int_value = 2, num;
float *float_ptr, float_value = 33.3f;
        num = sizeof(int_values)/sizeof(int);
        //apel la functia de bibioteca bsearch() pentru sirul de numere intregi
        int_ptr = (int *)bsearch(&int_value, int_values, num, sizeof(int),
                (int (*) (const void *, const void *)) compare_int);
        if (int_ptr)
          printf("Valoarea %d a fost gasita!\n", int_value);
        else
          printf("Valoarea %d nu a fost gasita!\n", int_value);

        num = sizeof(float_values)/sizeof(float);
        //apel la functia de bibioteca bsearch() pentru sirul de numere reale
        float_ptr = (float *)bsearch(&float_value, float_values, num, sizeof(float),
          (int (*) (const void *, const void *)) compare_float);
```

```
        if (float_ptr)
          printf("Valoarea %3.1f a fost gasita!\n", float_value);//end if
        else
          printf("Valoarea %3.1f nu a fost gasita!\n", float_value);//end else
}//end main( )

int compare_int(int *a, int *b)
{
        return(*a - *b);
}//end compare_int( )

int compare_float(float *a, float *b)
{
        if(*a < *b) return -1;
        if(*a > *b) return 1;
        return 0;
}//end compare_float( )
```

Teme:

1. Testați aplicațiile privind metodele de programare oferite în cadrul laboratorului. Folosiți aceste aplicații pentru a implementa problemele 2 și 3 propuse, considerând în acest sens aplicațiile adiționale din laborator *note.cpp* și *bakopti.cpp*.
2. Fie un sistem de calcul care urmăreşte controlul unui proces de transmisie a datelor pe o linie principală având un debit maxim de 40MB/s. Fluxul de date de pe această linie este împărţit de către maxim 10 utilizatori, traficul pe liniile oferite lor putându-se efectua cu debite între 2 şi 40 MB/s (valori intregi). Impărţirea debitelor pe cele maxim 10 linii se face în mod dinamic de către sistemul de control considerând pentru fiecare linie i, o pondere subunitară p_i, asociată la configurarea sistemului funcţie de utilizator. Introduceţi iniţial cele maxim 10 ponderi p_i astfel încât suma lor să fie egală cu 1. Dacă această condiţie este verificată generaţi toate soluţiile posibile considerând că pe linia principală vom avea cel puţin un debit de 2MB/s, deci de 2...40MB/s determinând în aceste cazuri debitele posibile pe cele maxim 10 linii de intrare ale utilizatorilor (ajustate la intregi MB/s). Se pune pe 0, dacă nu se poate asigura minimul de 2MB/s. Afişaţi aceste soluţii. Implementarea poate fi recursivă sau nerecursivă.
3. În cadrul unei companii de software se pune problema difuzării unor pachete de date la mai multe filiale, maxim 7, valoarea fiind introdusă de la intrarea standard. Filialele sunt dispuse în mai multe puncte din lume, costurile de transmisie fiind dependente de poziţia lor geografică. Pachetele trebuie să ajungă la fiecare filială chiar dacă nu există o legătură directă între toate filialele ci prin intermediul unei alte filiale caz în care pachetul va urma o rută ocolitoare. Costurile necesare transmiterii datelor în reţea se introduc iniţial la generarea sistemului. Să se determine şi afişeze ruta pe care trebuie transmise pachetele de date astfel încât pornind de la filiala i, citită de la intrarea standard pachetul să fie transmis cu cost minim (eventual cu confirmare adică pachetul poate să se întoarcă înapoi). Implementarea poate fi recursivă sau nerecursivă.
4. Problema investirii optime de capital: pentru un investitor cu un capital C şi n oferte la care trebuie avansate fondurile f_i şi care aduc beneficiile b_i, se cere selectarea acelor oferte care îi aduc beneficiul maxim.
5. Creaţi un fişier în care stocaţi un şir de numere întregi, generate in mod aleator. Preluaţi prin program aceste valori. Folosiţi metoda *divide et impera* pentru a determina minimul şi maximul din şir şi afisaţi rezultatele pe ecran. Actualizaţi fişierul iniţial adăugând aceste două valori.
6. Scrieţi o aplicaţie care determină c.m.m.d.c. dintr-un tablou unidimensional de numere întregi (max. 2000 de valori), utilizând metoda divide et impera. Elementele tabloului se pot citi de la tastatură sau dintr-un fisier.
7. Să se calculeze $\int_a^b 1/(1+x^2)dx$, cu ajutorul metodei trapezelor, astfel încât inălţimea fiecărui trapez a cărui arie se însumează să fie mai mică decât $\varepsilon=0.0001$. Aria trapezului cu vârfurile în punctele (a,0), (b,0), (a,f(a)) şi (b,f(b)) este (b-a)*(f(a)+f(b))/2, iar $f(x)=1/(1+x^2)$. Se citesc de la tastatură numerele reale a şi b, a≤b. Utilizaţi metoda divide et impera.
8. Să se scrie o aplicație C/C++ care să genereze aleator maxim 10 valori întregi, ce vor fi memorate într-un tablou unidimensional. Să se verifice dacă o altă valoare generată aleator aparține acestui tablou, utilizând funcția *_lsearch()*.
9. Scrieţi o aplicaţie C/C++ care să găsească imaginile cele mai apropiate folosind pe rând o cheie de căutare din antetul imaginii. Antetul este reprezentat printr-o structură ce conţine un nume de fişier (sir de caractere), o cale (sir de caractere), o rezoluţie de intensitate (întreg), o dimensiune în octeţi (întreg). Pentru fiecare cheie folosiţi o metodă diferită de căutare. Antetul imaginilor se află stocat într-un fişier sau se citeşte într-un tablou de structuri de la tastatură.
10. Citiți de la intrarea standard un tablou unidimensional de maxim 20 de valori întregi. Folosind mecanismul de căutare binară, determinați dacă o nouă valoare, *a*, introdusă de la tastatură există în șir. Dacă da, determinați toți factorii primi ai acestei valori pe care îi veți afișa pe ecran.

11. Folosind un fișier care conține numere reale ordonate, căutați o valoare reală *a* introdusă de la tastatură în cadrul șirului, folosind algoritmul de căutare binară iterativ, recursiv și funcția de bibliotecă *bsearch()*. Afișați șirul citit, valoarea *a* și poziția folosind cei trei algoritmi specificați.

Activitate suplimentară:
Analizaţi alte probleme practice în care pot fi utilizate metodele de programare utilizând în acest caz soluţii didactice sau nu. Realizaţi implementări în limbajul C/C++.

Homework:

1. Test the applications related to the programming methods described in this document (and in the additional file). Use these applications for implementing problem 2 and problem 3 (*note.cpp* and *backopti.cpp*)
2. Consider a computing system that monitors the data transfer on a main communication channel that has a maximum flow capacity of 40 MBps. The data stream can be shared among max 10 users. The channel's data flow is divided automatically by the monitoring program using max 10 (0...9) sub-unitary weights, each of them being associated with a user. The weights are entered from the keyboard when the program starts, and their sum must be 1. If the weights are entered correctly, generate all the possible solutions for each one of the max 10 individual channels, considering that the main channel has the debit between 2MBps and 40 MBps. Calculate the solutions with a precision of MBps. The 0 MBps will be assigned for those channels that cannot have the minimum of 2MBps. Display the solutions. The implementation can be both recursive and non-recursive.
3. In a software company, some data packets must be broadcasted to N subsidiaries, N being entered from the keyboard. The subsidiaries are located at different distances and the transmission costs depend on those distances. The data packets must reach their destination, irrespective of the existence or of a direct connection between the main company and a certain subsidiary (use other intermediate subsidiaries if necessary). The costs associated to each direct link are entered from the keyboard. Any subsidiary can be considered as the transmitting company. Determine and display the route that the data packets have to follow in order to minimize the total transmission cost. The implementation can be recursive or non-recursive.
4. The optimum capital investment: an investor has a capital C and n offers; he has to invest it into f_i investment funds each of them being associated with a corresponding benefit b_i. Generate all the possible investing solutions. Indicate the solution that brings the maximum profit.
5. Using the "divide et impera" programming method, read the integer values generated in a random mode previously stored into file. Determine the minimum and maximum values from the array. Update the original file by adding the determined values.
6. Write a program that uses the „divide et impera” method for determining the greatest common divider of a series of maximum 2000 values stored in an array. All the numbers are entered from the keyboard or read from a previously created file.
7. Calculate $\int_a^b 1/(1+x^2)dx$, using the trapezes method. The height of each considered trapeze must be smaller than $\varepsilon=0.0001$. The area of a trapezes defined by the points (a,0), (b,0), (a,f(a)) and (b,f(b)) is (b-a)*(f(a)+f(b))/2, where $f(x)=1/(1+x^2)$. The float values of a and b are read from the keyboard (a must be smaller or equal to b). Use the „divide et impera” method.
8. Write a C/C++ application that generates max 10 integer random values and stores them in an array. Check if another randomly generated value belongs to this array, using *_lsearch()* function.
9. Write a C/C++ application that finds and displays the images that represent the closest match to a certain searching key. The searching key is stored in each image's header and it is represented using a structure with the following fields: a filename (string of characters), a path (string of characters), an intensity resolution (integer value) and a dimension in bytes (integer value). Use a different searching technique for each key type. The headers are either stored into a file or the correspondent data is read from the keyboard.
10. Read from the keyboard a one-dimensional array of maximum 20 integer values. Using the binary search mechanism, determine if another value *a* (also read from the standard input) is part of the array. If so, determine and display all its prime factors.
11. Using a file that contains real ordered numbers, look for a value read from the keyboard. The searching method will rely on an iterative binary search, the recursive algorithm and the *bsearch()* library function. Display the values read from the file, the value to be searched and the position it was found (if any).