Seminar 8 - Greedy

1. Problema rucsacului, varianta continuă. Să se scrie subprogramul pentru rezolvarea problemei rucsacului. Se consideră un mijloc de transport cu o capacitate dată (C). Cu acesta trebuie transportate obiecte dintr-o mulțime $A = \{a_i | \ i = 1, \ n\}$. Fiecare obiect ocupă o *capacitate specifică* c_i și aduce un *câștig specific* v_i – fiecare obiect luat separat încape în mijlocul de transport. Se cere să se determine o modalitate de încărcare care să maximizeze cîștigul obținut la un transport. Obiectele transportate pot fi fractionate. În acest caz se va utiliza întotdeauna întreaga capacitate de transport.

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

struct Obiect {
	float valoare; float cost;
};

//Problema rucsacului in varianta continua
//=>Se ocupa intreaga capacitate a rucsacului
//obiectele adaugate in rucsac pot fi fractionate
//Pentru un algoritm eficient obiectele sunt deja sortate descrescator
//dupa castigul adus de fiecare obiect.
//q - capacitatea totala a rucsacului
//n - nr. obiecte
//ob - lista de obiecte
//sol - vector care marcheaza obiectele introduse in rucsac

void rucsacContinuu(float q, int n, Obiect* ob, float* sol) {
	int i;

	//parcurgem pana trecem prin toate obiectele din lista sau pana
	//cand capacitatea rucsacului este 0
	for (i = 0; i < n && q > 0; i++) {
		if (q >= ob[i].cost) {
			sol[i] = 1; //marcam obiectul ca introdus in rucsac
			q -= ob[i].cost;
		}
		else {
			//daca capacitatea este mai mica decat costul obiectului,
			//acesta se fractioneaza (doar pentru varianta continua)
			sol[i] = q / ob[i].cost;
			q = 0;
		}
	}

	//pentru eventualele obiecte neprelucrate in for se da valoarea 0
	//in vectorul solutie, deoarece acestea nu au fost puse in rucsac
	for (int j = i; j < n; j++)
		sol[j] = 0;
}

#define n 10

void main() {
	float q;
	Obiect* obiecte;
```

```
	float* obiecteAcceptate;

	obiecte = (Obiect*)malloc(n * sizeof(Obiect));
	obiecte[0].cost = 1; obiecte[0].valoare = 3;
	obiecte[1].cost = 2; obiecte[1].valoare = 2;
	obiecte[2].cost = 3; obiecte[2].valoare = 1;
	obiecte[3].cost = 4; obiecte[3].valoare = 4;
	obiecte[4].cost = 5; obiecte[4].valoare = 5;
	obiecte[5].cost = 6; obiecte[5].valoare = 3;
	obiecte[6].cost = 7; obiecte[6].valoare = 2;
	obiecte[7].cost = 8; obiecte[7].valoare = 7;
	obiecte[8].cost = 9; obiecte[8].valoare = 1;
	obiecte[9].cost = 10; obiecte[9].valoare = 8;

	//prelucrare preliminara
	//sortarea obiectelor in mod descrescator
	float aux;
	for (int i = 0; i < n - 1; i++)
		for (int j = i + 1; j < n; j++)
			if (obiecte[i].valoare / obiecte[i].cost < obiecte[j].valoare /
obiecte[j].cost)     //castig unitar
			{
				aux = obiecte[i].valoare; obiecte[i].valoare =
obiecte[j].valoare; obiecte[j].valoare = aux;

				aux = obiecte[i].cost; obiecte[i].cost = obiecte[j].cost;
obiecte[j].cost = aux;
			}

	printf("Capacitatea rucsacului: "); scanf("%f", &q);
	obiecteAcceptate = (float*)malloc(n * sizeof(float));

	rucsacContinuu(q, n, obiecte, obiecteAcceptate);

	double castig = 0;
	for (int i = 0; i < n; i++) {
		castig += obiecte[i].valoare * obiecteAcceptate[i];

		printf("%d: c = %.2f   v = %.2f  bucati de obiecte adaugate = %.2f;  castig =
%.2f\n", i + 1,
			obiecte[i].cost, obiecte[i].valoare, obiecteAcceptate[i], castig);
	}
 }
```

2. Se dă o mulțime de elemente reale $A = \{a_1, a_2, \ldots, a_n, \}$. Se cere să se scrie funcția care determină o submulțime $S \subseteq A$ astfel încât suma elementelor submulțimii S să fie cea mai mare posibilă (problema sumei maxime).

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

//Se obtine submultimea de elemente din vector
//care dau suma maxima.
//Se iau doar elementele pozitive.
//v - multimea primita de elemente
//n - nr elemente multime primita
//sol - submultimea elementelor care dau suma maxima
//nrSol - nr elemente din submultimea rezultat
int sumaMaxima(int * v, int n, int * sol, int &nrSol) {
	nrSol = 0;
	int max = 0;

	//parcurgem toate elementele din multimea primita
	//si selectam doar elementele pozitive care dau suma maxima
	//la final
	for (int i = 0; i < n; i++)
		if (v[i] >= 0) { //se ia si 0 pentru ca nu afecteaza suma
			sol[nrSol++] = v[i];
			max += v[i];
		}

	return max;
}

void main() {
	int n, nrSol;
	int * multimeInitiala;
	int * submultimeRezultat;

	printf("Numarul de elemente din multime: ");
	scanf("%d", &n);
	multimeInitiala = (int*)malloc(n * sizeof(int));
	submultimeRezultat = (int*)malloc(n * sizeof(int));

	printf("Elementele din multime sunt: ");
	for (int i = 0; i < n; i++)
		scanf("%d", &multimeInitiala[i]);

	int sMax = sumaMaxima(multimeInitiala, n, submultimeRezultat, nrSol);

	printf("\nSuma maxima este: %d", sMax);
	printf("\nSubmultimea de elemente care formeaza suma maxima: ");
	for (int i = 0; i < nrSol; i++)
		printf("%d ", submultimeRezultat[i]);

	free(submultimeRezultat);
	free(multimeInitiala);
}
```

3. Să se scrie subprogramul C care permite plata unei sume $S \in \mathbb{N}$ folosind cît mai puține bancnote din tipurile (valorile) a_i, $i = 1, n$, știind că printre acestea se află și bancnota cu valoare unitate. Sunt disponibile cantități nelimitate din fiecare tip de bancnotă.

```c
#include<stdio.h>

//Se face plata unei sume cu bancnotele furnizate intr-o lista.
//Printre acestea se afla si bancnota unitate.
//Se doreste sa se faca plata cu un numar cat mai mic de bancnote.
//Lista de bancnote contine valorile acestora in ordine descrescatoare.
//suma - suma de platit
//tipuriBancnote - bancnotele disponibile pentru plata
//n - nr de tipuri de bancnote
//sol - vector care retine cate bancnote s-au folosit din fiecare tip
void plataSumei(int suma, int * tipuriBancnote, int n, int * sol) {
	//parcurgem bancnotele de la cea mai mare valoare la cea mai mica
	for (int i = 0; i < n; i++) {
		sol[i] = suma / tipuriBancnote[i];
		suma = suma % tipuriBancnote[i]; //suma ramasa reprezinta restul
	}
}

void main() {
	int n = 7;
	int valoriBancnote[] = { 500, 200, 100, 50, 10, 5, 1 };
	int bancnoteUtilizate[] = { 0, 0, 0, 0, 0, 0, 0 };

	int sumaDePlata;
	printf("Suma de plata: ");
	scanf("%d", &sumaDePlata);

	plataSumei(sumaDePlata, valoriBancnote, n, bancnoteUtilizate);

	printf("\nBanconte utilizate:\n");
	for (int i = 0; i < n; i++)
		printf("%d - %d Subtotal: %d\n", valoriBancnote[i], bancnoteUtilizate[i],
			valoriBancnote[i] * bancnoteUtilizate[i]);
}
```

Probleme propuse

- Problema rucsacului, varianta întreagă. Să se scrie un subprogram care să implementeze problema rucsacului de la exercițiul 1 și în varianta întreagă. Problema rucsacului în varianta întreagă specifică că obiectele transportate nu pot fi fracționate, ele trebuie transportate întregi, ceea ce înseamnă că este posibil ca spațiul rucsacului să nu fie mereu folosit în întregime.

- Fiind dat numărul natural k>1 să se scrie programul care determină cel mai mic număr natural n având exact k divizori naturali proprii (diferiți de 1 și de n). De exemplu pentru numărul 6 avem 2 divizori naturali proprii, 2 și 3.