## Programare dinamica si strategia greedy

## Problema rucsacului

```
// numarul de obiecte
#define  N 5
// greutatea admisa
#define  W 11

int v[N+1];   // valoarea
int w[N+1];   // greutatea


void init(void)
{
    int i;
    for (i = 0; i <= W; i++)
        OPT[0][i] = 0;

    v[0] = 0; w[0] = 0;
    v[1] = 1; w[1] = 1;  // primul obiect
    v[2] = 6; w[2] = 2;  // al 2-lea obiect
    v[3] = 18; w[3] = 5; // al 3-lea obiect
    v[4] = 22; w[4] = 6; // al 4-lea obiect
    v[5] = 28; w[5] = 7; // al 5-lea obiect

    for (i = 0; i <= N; i++)
    {
        sel[i] = 0;
        sel_g[i] = 0;
    }
}


void rucsac_programare_dinamica(void)
{
    int n, g;
    for (n = 1; n <= N; n++ )
        for (g = 1; g <= W; g++)
    {
        if (w[n] > g)
            OPT[n][g] = OPT[n - 1][g];
        else
        {
            if (OPT[n - 1][g] >= v[n] + OPT[n - 1][g - w[n]])
            {
                OPT[n][g] = OPT[n - 1][g];
            }
            else
            {
                OPT[n][g] = v[n] + OPT[n - 1][g - w[n]];
            }
        }
    }

}
```

**Programare dinamica**

Functia de optim

```
0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1
0	1	6	7	7	7	7	7	7	7	7	7
0	1	6	7	7	18	19	24	25	25	25	25
0	1	6	7	7	18	22	24	28	29	29	40
0	1	6	7	7	18	22	28	29	34	35	40
```

Valoarea maxima = 40
Greutatea maxima obtinuta = 11
Numarul de obiecte din care se alege = 4
Obiectele alese =4, 3,

**Greedy**

Valoarea maxima = 35
Greutatea maxima obtinuta = 10
Obiectele alese =5, 2, 1,

```
// rucsac.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

// numarul de obiecte
#define  N 5
// greutatea admisa
#define  W 11

int v[N+1];   // valoarea
int w[N+1];   // greutatea

int vmax; // valoarea maxima Programare dinamica
int gmax; // greutatea maxima Programare dinamica
int ob; // nr. de obiecte din care se alege Programare dinamica

int ob_alese[N + 1]; // Programare dinamica
int nob;

int vmax_g; // valoarea maxima Greedy
int gmax_g; // greutatea maxima Greedy


int ob_alese_g[N + 1]; // Greedy
int nob_g;

int sel[N + 1];
int sel_g[N + 1];

int OPT[N+1][W+1]; // solutia optima OPT[nr de obiecte][greutate]

void init(void);
void rucsac_programare_dinamica(void);
void print_OPT(void);
void maxime(void);
void alegere_obiecte(void);

void alegere_greedy(void);


int _tmain(int argc, _TCHAR* argv[])
{
       init();

       rucsac_programare_dinamica();

       maxime();

       alegere_obiecte();

       alegere_greedy();

       print_OPT();

       return 0;
```

```
}

void init(void)
{
	int i;
	for (i = 0; i <= W; i++)
		OPT[0][i] = 0;

	v[0] = 0; w[0] = 0;
	v[1] = 1; w[1] = 1;  // primul obiect
	v[2] = 6; w[2] = 2;  // al 2-lea obiect
	v[3] = 18; w[3] = 5; // al 3-lea obiect
	v[4] = 22; w[4] = 6; // al 4-lea obiect
	v[5] = 28; w[5] = 7; // al 5-lea obiect

	for (i = 0; i <= N; i++)
	{
		sel[i] = 0;
		sel_g[i] = 0;
	}
}

void rucsac_programare_dinamica(void)
{
	int n, g;
	for (n = 1; n <= N; n++ )
		for (g = 1; g <= W; g++)
	{
		if (w[n] > g)
			OPT[n][g] = OPT[n - 1][g];
		else
		{
			if (OPT[n - 1][g] >= v[n] + OPT[n - 1][g - w[n]])
			{
				OPT[n][g] = OPT[n - 1][g];
			}
			else
			{
				OPT[n][g] = v[n] + OPT[n - 1][g - w[n]];
			}
		}
	}

}

void print_OPT(void)
{
	int n, g;

	printf("\nProgramare dinamica\n");
	printf("Functia de optim\n");
	for (n = 0; n <= N; n++)
	{
		for (g = 0; g <= W; g++)
		{
			printf("%d\t", OPT[n][g]);
		}
		printf("\n");
```

```c
       }
       printf("\n");
       printf("Valoarea maxima = %d\n", vmax);
       printf("Greutatea maxima obtinuta = %d\n", gmax);
       printf("Numarul de obiecte din care se alege = %d\n", ob);
       printf("Obiectele alese =");
       for (n = 0; n <= nob; n++)
              printf("%d, ", ob_alese[n]);
       printf("\n");

       printf("\nGreedy\n");
       printf("Valoarea maxima = %d\n", vmax_g);
       printf("Greutatea maxima obtinuta = %d\n", gmax_g);
       printf("Obiectele alese =");
       for (n = 0; n <= nob_g; n++)
              printf("%d, ", ob_alese_g[n]);
       printf("\n");
}

void maxime(void) // determina din cite obiecte se va alege ca sa se obtina val
maxima
{
       int n,g;
       int max;
       int indn,indg;

       max = 0;

       // determina maximul pe linia OPT[n][g]

       for (n = 0; n <= N; n++)
       {
              for (g = 0; g <= W; g++)
              if (OPT[n][g] > max)
              {
                     max = OPT[n][g];
                     indn = n;
                     indg = g;
              }
       }

       ob = indn;           // numarul de obiecte
       gmax = indg;  // greutatea maxima
       vmax = max;          // valoarea maxima

}

void alegere_obiecte(void)
{
       int i,n;
       int vcurent, vm;
       int obcurent=0;

       vcurent = 0;
       i = 0;

       // alege obiecte incepind cu obiectul de valoare maxima
       while (vcurent <= vmax)
```

```
        {
        // alege obiectul de valoare maxima
        vm = 0;
        obcurent = 0;
        for (n = 1; n <= ob; n++)
        {
                if (sel[n]==1) continue;
                if (v[n] > vm)
                {
                        vm = v[n];
                        obcurent = n;
                }
        }

                vcurent = vcurent + vm;
                ob_alese[i] = obcurent;
                sel[obcurent] = 1;
                nob = i;
                i++;

                if (vcurent == vmax) break;
        }

}


void alegere_greedy(void)
{
        int i, n;
        int gcurent, vcurent, vm;
        int obcurent = 0;

        int sum_sel;


        gcurent = 0;
        vcurent = 0;
        i = 0;
        sum_sel = 0;

        // alege obiecte incepind cu obiectul de valoare maxima
        while ((gcurent <= gmax) && sum_sel <N)
        {

        // alege obiectul de valoare maxima
                vm = 0;
                obcurent = 0;
                for (n = 1; n <= N; n++)
                {
                        if (sel_g[n] == 1) continue;
                        if (v[n] > vm)
                        {
                                vm = v[n];
                                obcurent = n;
                        }
                }

                if (vm > 0)
        {
            sel_g[obcurent] = 1;
            sum_sel = sum_sel + sel_g[obcurent];
        }

        if (gcurent + w[obcurent] <= gmax  && sel_g[obcurent] == 1)
        {
            gcurent = gcurent + w[obcurent];
            vcurent = vcurent + vm;
            ob_alese_g[i] = obcurent;
            nob_g = i;
            i++;
        }
    }
        vmax_g = vcurent;
        gmax_g = gcurent;
}
```

### Problema alegerii activitatilor

```
// initializeaza q[j]
       for (j = 0; j < N+1; j++)
       {
              q[j] = 0;
       }
       // initializeaza OPT[j]
for (j = 0; j <= N; j++)
       {
              OPT[j] = 0;
       }

// calculeaza q[j] = cel mai mare index i cu proprietatea i < j si activitatea i
compatibila cu activitatea j (finish i <= start j )

void calc_q(void)
{
       int i, j;

       for (j = 1; j <= N; j++)
       {
              // calculeaza q[j]
              for (i = j-1; i >= 0; i--)
              {
                     if (ACT[i].finish <= ACT[j].start)
                     {
                            q[j] = i;
                            break;
                     }
              }
       }
}

int rec_OPT(int j)
{
       if (j == 0) return 0;

if (ACT[j].weight + rec_OPT(q[j]) >= rec_OPT(j - 1))

       return (ACT[j].weight + rec_OPT(q[j]));
else
       return rec_OPT(j - 1);

}

void calc_OPT(void)
{
       int j;
       for (j = 0; j <= N; j++)
              OPT[j] = rec_OPT(j);
}
```

**Cazul 1**

Activitati
ACT[1]= start:1, finish:4, weight:1
ACT[2]= start:3, finish:5, weight:1
ACT[3]= start:0, finish:6, weight:1
ACT[4]= start:4, finish:7, weight:1
ACT[5]= start:3, finish:8, weight:1
ACT[6]= start:5, finish:9, weight:1
ACT[7]= start:6, finish:10, weight:1
ACT[8]= start:8, finish:11, weight:1

Programare dinamica

Valorile q
q[1]= 0
q[2]= 0
q[3]= 0
q[4]= 1
q[5]= 0
q[6]= 2
q[7]= 3
q[8]= 5
Valoare OPT
OPT[1]= 1
OPT[2]= 1
OPT[3]= 1
OPT[4]= 2
OPT[5]= 2
OPT[6]= 2
OPT[7]= 2
OPT[8]= 3
**Activitatile alese:1, 4, 8,**
**Valoarea activitatilor alese = 3**

Greedy

**Activitatile alese :1, 4, 8**,
**Valoarea activitatilor alese = 3**

**Cazul 2**

Activitati
ACT[1]= start:1, finish:4, weight:1
**ACT[2]= start:3, finish:5, weight:2**
ACT[3]= start:0, finish:6, weight:1

ACT[4]= start:4, finish:7, weight:1
ACT[5]= start:3, finish:8, weight:1
ACT[6]= start:5, finish:9, weight:1
ACT[7]= start:6, finish:10, weight:1
ACT[8]= start:8, finish:11, weight:1

Programare dinamica

Valorile q
q[1]= 0
q[2]= 0
q[3]= 0
q[4]= 1
q[5]= 0
q[6]= 2
q[7]= 3
q[8]= 5
Valoare OPT
OPT[1]= 1
OPT[2]= 2
OPT[3]= 2
OPT[4]= 2
OPT[5]= 2
OPT[6]= 3
OPT[7]= 3
OPT[8]= 3
**Activitatile alese:2, 6,**
**Valoarea activitatilor alese = 3**

Greedy

**Activitatile alese :1, 4, 8,**
**Valoarea activitatilor alese =** 3

Cazul 3

Activitati
ACT[1]= start:1, finish:4, weight:1
**ACT[2]= start:3, finish:5, weight:2**
ACT[3]= start:0, finish:6, weight:1
ACT[4]= start:4, finish:7, weight:1
ACT[5]= start:3, finish:8, weight:1
**ACT[6]= start:5, finish:9, weight:2**
ACT[7]= start:6, finish:10, weight:1
ACT[8]= start:8, finish:11, weight:1

Programare dinamica

Valorile q
q[1]= 0
q[2]= 0
q[3]= 0
q[4]= 1
q[5]= 0
q[6]= 2
q[7]= 3
q[8]= 5
Valoare OPT
OPT[1]= 1
OPT[2]= 2
OPT[3]= 2
OPT[4]= 2
OPT[5]= 2
OPT[6]= 4
OPT[7]= 4
OPT[8]= 4
**Activitatile alese:2, 6,**
**Valoarea activitatilor alese = 4**

Greedy

**Activitatile alese :1, 4, 8,**
**Valoarea activitatilor alese = 3**

**Cazul 4**

Activitati
ACT[1]= start:1, finish:4, weight:1
**ACT[2]= start:3, finish:5, weight:2**
ACT[3]= start:0, finish:6, weight:1
ACT[4]= start:4, finish:7, weight:1
ACT[5]= start:3, finish:8, weight:1
**ACT[6]= start:5, finish:9, weight:2**
**ACT[7]= start:6, finish:10, weight:5**
ACT[8]= start:8, finish:11, weight:1

Programare dinamica

Valorile q
q[1]= 0
q[2]= 0
q[3]= 0

```
q[4]= 1
q[5]= 0
q[6]= 2
q[7]= 3
q[8]= 5
Valoare OPT
OPT[1]= 1
OPT[2]= 2
OPT[3]= 2
OPT[4]= 2
OPT[5]= 2
OPT[6]= 4
OPT[7]= 7
OPT[8]= 7
```

**Activitatile alese:7, 2,**
**Valoarea activitatilor alese = 7**

Greedy

**Activitatile alese :1, 4, 8,**
**Valoarea activitatilor alese = 3**

```
// Alegere_activ.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

// numar de activitati
#define N 8

struct activitate {
       int start;
       int finish;
       int weight;
};

void init(void);   // initializari, activitatile sint ordonate crescator dupa
momentul de finish
void calc_q(void); // calculeaza q[j] = cel mai mare index i cu proprietatea i < j
si activitatea i compatibila cu activitatea j (finish i <= start j )
void print_info(void);
int rec_OPT(int j); // calculul optimului
void calc_OPT(void);

void alegere_act(int n); // alege din n activitati activitatile care conduc la
valoarea optima
void alegere_act_greedy(int n); // alege (greedy) din n activitati

activitate ACT[N+1]; // activitati sortate dupa finish (ordine crescatoare),
ACT[0] nu e folosita
int q[N + 1];
int OPT[N + 1]; // valoarea optima, in functie de numarul de activitati alese
int sel_act[N+1]; // activitate selectata

int act_alese[N]; // activitatile alese
int nr_act_alese; // numarul de activitati alese
int VAL;              // valoarea activitatilor alese


int sel_act_g[N + 1]; // activitate selectata Greedy
int act_alese_g[N]; // activitatile alese Greedy
int nr_act_alese_g; // numarul de activitati alese Greedy
int VAL_G;            // valoarea activitatilor alese Greedy


int _tmain(int argc, _TCHAR* argv[])
{
       init();
       calc_q();

       calc_OPT();

       alegere_act(8);

       alegere_act_greedy(8);

       print_info();

       return 0;
}
```

```
void init(void)
{
        int j;
        ACT[0].start = 0; ACT[0].finish = 0;  ACT[0].weight = 0; // nu e folosita

        ACT[1].start = 1; ACT[1].finish = 4;  ACT[1].weight = 1;
        ACT[2].start = 3; ACT[2].finish = 5;  ACT[2].weight = 2;
        ACT[3].start = 0; ACT[3].finish = 6;  ACT[3].weight = 1;
        ACT[4].start = 4; ACT[4].finish = 7;  ACT[4].weight = 1;
        ACT[5].start = 3; ACT[5].finish = 8;  ACT[5].weight = 1;
        ACT[6].start = 5; ACT[6].finish = 9;  ACT[6].weight = 2;
        ACT[7].start = 6; ACT[7].finish = 10; ACT[7].weight = 5;
        ACT[8].start = 8; ACT[8].finish = 11; ACT[8].weight = 1;

        // initializeaza q[j]
        for (j = 0; j < N+1; j++)
        {
                q[j] = 0;
        }
        // initializeaza OPT[j] si sel_act
        for (j = 0; j <= N; j++)
        {
                OPT[j] = 0;
                sel_act[j] = 0;
        }
        VAL = 0;
        nr_act_alese = 0;
        VAL_G = 0;
        nr_act_alese_g = 0;
}

void calc_q(void)
{
        int i, j;

        for (j = 1; j <= N; j++)
        {
                // calculeaza q[j]
                for (i = j-1; i >= 0; i--)
                {
                        if (ACT[i].finish <= ACT[j].start)
                        {
                                q[j] = i;
                                break;
                        }
                }
        }
}

int rec_OPT(int j)
{
        if (j == 0) return 0;
        if (ACT[j].weight + rec_OPT(q[j]) >= rec_OPT(j - 1)) return (ACT[j].weight
+ rec_OPT(q[j]));
        else return rec_OPT(j - 1);

}
```

```
void calc_OPT(void)
{
       int j;
       for (j = 0; j <= N; j++)
              OPT[j] = rec_OPT(j);
}


void print_info(void)
{
       int i;
       printf("Activitati\n");
       for (i = 1; i < N+1; i++)
       {
              printf("ACT[%d]= start:%d, finish:%d, weight:%d", i, ACT[i].start,
ACT[i].finish, ACT[i].weight);
              printf("\n");
       }
       printf("\nProgramare dinamica\n\n");
       printf("Valorile q\n");
       for (i = 1; i < N+1; i++)
       {
              printf("q[%d]= %d", i, q[i]);
              printf("\n");
       }
       printf("Valoare OPT\n");
       for (i = 1; i <= N; i++)
       {
              printf("OPT[%d]= %d", i, OPT[i]);
              printf("\n");
       }
       printf("Activitatile alese:");
       for (i = 0; i < nr_act_alese; i++)
       {
              printf("%d, ", act_alese[i]);
       }
       printf("\n");
       printf("Valoarea activitatilor alese = %d\n", VAL);

       printf("\nGreedy\n\n");
       printf("Activitatile alese :");
       for (i = 0; i < nr_act_alese_g; i++)
       {
              printf("%d, ", act_alese_g[i]);
       }
       printf("\n");
       printf("Valoarea activitatilor alese = %d\n", VAL_G);
}

void alegere_act(int j)
{
       int i;
       int max_weight;
       int ind_max;

       while (VAL <= OPT[j])
       {
        max_weight = 0;
        // alege activitatea cu ponderea cea mai mare (dintre activitatile
compatibile)
        for (i = 1; i < N+1; i++)
        {
            if (sel_act[i] == 1) continue;
            if (ACT[i].weight > max_weight)
            {
                max_weight = ACT[i].weight;
                ind_max = i;
            }
        }
        if (max_weight > 0)
        {
            sel_act[ind_max] = 1;
            act_alese[nr_act_alese] = ind_max;
            nr_act_alese++;
            VAL = VAL + max_weight;
        }
        // elimina activitatile necompatibile cu activitatea aleasa
        // activitate compatibila - incepe inainte si se termina inainte de
activitatea aleasa sau incepe dupa activitatea aleasa
        for (i = 1; i < N+1; i++)
        {
            if (i == ind_max || sel_act[i]==1) continue;
            if (!((ACT[i].start < ACT[ind_max].start) && (ACT[i].finish <=
ACT[ind_max].start) || (ACT[i].start >= ACT[ind_max].finish)))
            {
                sel_act[i] = 1;
            }
        }
        if (VAL == OPT[j]) break;
    }
}

void alegere_act_greedy(int j)
{
    int i;
    int ind;
    int sum;

    sum = 0;
    while (sum < N)
    {
        // alege activitatea in ordinea timpului de start
        ind = 0;
        for (i = 1; i < N + 1; i++)
        {
            if (sel_act_g[i] == 0)
            {
                ind = i;
                break;
            }
        }
        if (ind > 0)
        {
            sel_act_g[ind] = 1;
            act_alese_g[nr_act_alese_g] = ind;
            VAL_G = VAL_G + ACT[ind].weight;
            nr_act_alese_g++;
            sum = sum + sel_act_g[ind];
        }
        // elimina activitatile necompatibile cu activitatea aleasa
        for (i = 1; i < N + 1; i++)
        {
            if (i == ind || sel_act_g[i]==1) continue;
            if (ACT[i].start < ACT[ind].finish)
            {
                sel_act_g[i] = 1;
                sum = sum + sel_act_g[i];
            }
        }
    }
}
```