**PROIECT INDIVIDUAL**

**LA INFORMATICĂ**

**TEMA: TEHNICA GREEDY**

Realizat de Prisăcaru Ruxanda,

eleva clasei a 11-a „D”

Profesor: Guțu Maria

IPLT „Spiru Haret”, mun. Chișinău 2019

**Informații generale**

►Metoda Greedy este una din cele mai directe tehnici de proiectare a algoritmilor care se aplică la o varietate largă de probleme. In general, această metodă se aplică problemelor de optimizare. Specificul acestei metode constă în faptul că se construiește soluția optimă pas cu pas, la fiecare pas fiind selectat în soluție elementul care pare „cel mai bun/optim” la momentul respectiv, în speranța că această alegere locală va conduce la **optimul globa**l.

►Algoritmii Greedy sunt foarte eficienți, dar nu conduc în mod necesar la o soluție optimă și nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm Greedy trebuie însotit de o demonstrație a corectitudinii sale . Demonstrația faptului că o anumită problemă are proprietatea alegerii Greedy se face de obicei prin inducție matematică.

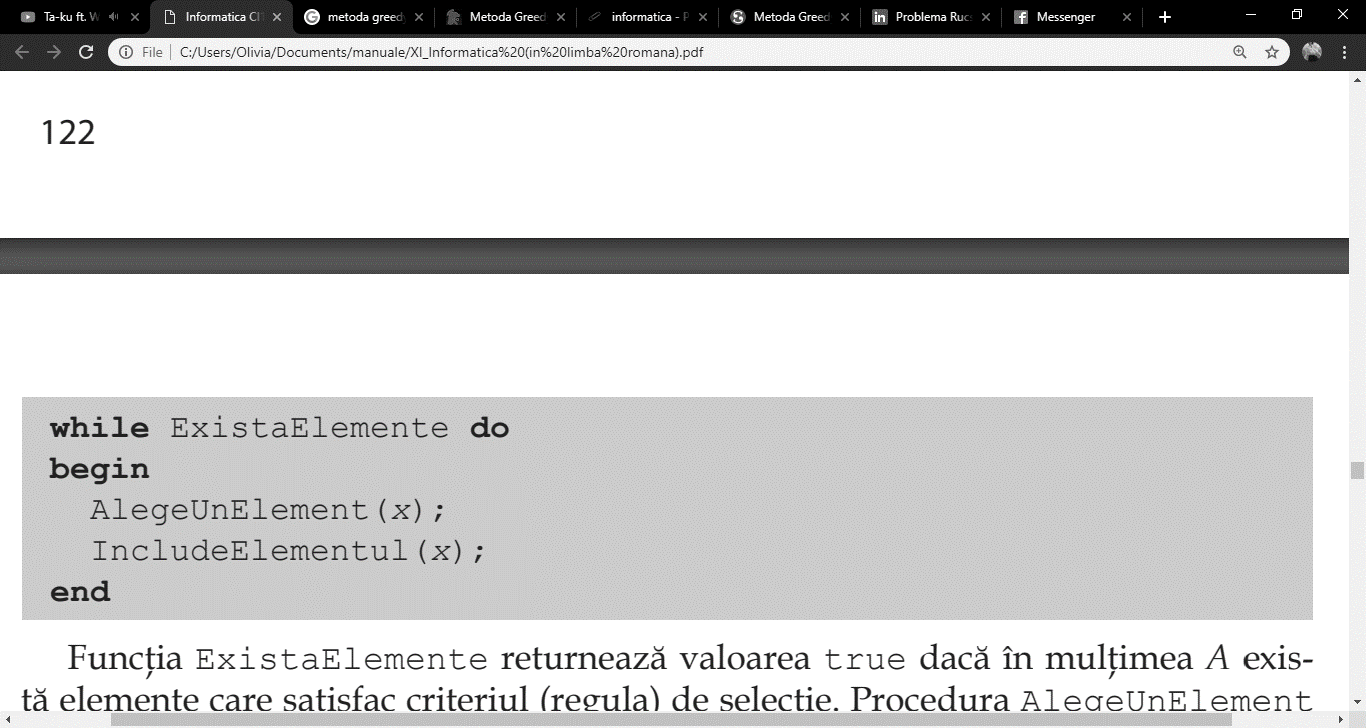
Metoda **Greedy** se aplică problemelor pentru care se dă o **mulţime A** cu **n** elemente şi pentru care trebuie determinată o **submulţime** a sa, **S** cu **m** elemente, care îndeplinesc anumite condiţii, numite și **condiții de optim**.

**Algoritmul în limbaj natural al metodei de programare Greedy are urmatoarea structură:**

* se dă o mulţime A
* se cere o submulţime S din multimea A care să:
* să îndeplinească anumite condiţii interne (să fie acceptabilă)
* să fie optimală (să realizeze un maxim sau un minim).

●În principiu, problemele de acest tip pot fi rezolvate prin metoda trierii, generând consecutiv cele *2n* submulţimi *Ai* ale mulţimii *A*. Dezavantajul metodei trierii constă în faptul că timpul cerut de algoritmii respectivi este foarte mare. Pentru a evita trierea tuturor submulţimilor *Ai* , *Ai ⊆A*, în metoda Greedy se utilizează un criteriu (o regulă) care asigură alegerea directă a elementelor necesare din mulţimea A. De obicei, criteriile sau regulile de selecţie nu sînt indicate explicit în enunţul problemei şi formularea lor cade în sarcina programatorului. Evident, în absenţa unor astfel de criterii metoda Greedy nu poate fi aplicată.

Schema generală a unui algoritm bazat pe metoda Greedy poate fi redată cu ajutorul unui ciclu:



Funcţia *ExistaElemente* returnează valoarea true dacă în mulţimea *A* există elemente care satisfac criteriul (regula) de selecţie. Procedura *AlegeUnElement* extrage din mulţimea *A* un astfel de element x, iar procedura *IncludeElementul* înscrie elementul selectat în submulţimea *B*. Iniţial *B* este o mulţime vidă. După cum se vede, în metoda Greedy soluţia problemei se caută prin testarea consecutivă a elementelor din mulţimea *A* şi prin includerea unora din ele în submulţimea *B*.

**Nota Bene:** Într-un limbaj plastic, submulţimea *B* încearcă să „înghită” elementele „gustoase” din mulţimea *A*, de unde provine şi denumirea metodei (greedy − lacom, hrăpăreţ).

**PROBLEME REZOLVATE:**

**1...**

***Suma maximă.* *Se dă o mulţime X={x1, x2, . . ., xn } cu elemente reale.* *Să se determine o submulţime a lui X astfel încât suma elementelor submulţimii să fie maximă.***

k:=0;

**for** i:=1 **to** n **do**

**if** x[i]>0 **then**

**begin**

k:=k+1;

s[k]:=x[i]

**end**;

**for** i:=1 **to** k **do**

**write**(s[i]:5:2,' ');

**readln**;

**end**.

**Program** suma\_maxima;

**var** s,x:**array**[1..20] **of** real;

i,k,n:integer;

**begin**

**write**('Numarul de elemente n = ');

**readln**(n);

**for** i:=1 **to** n **do**

**begin**

**write**('x[',i,']= ');

**readln**(x[i]);

**end**;

***Observaţie*** : evident că pentru a maximiza suma unui şir de numere acestea trebuie să fie, în primul rând, pozitive. Deci condiţia de alegere a unui element din şir ca să facă parte din mulţimea soluţie este ca acesta să fie pozitiv. Dacă, în plus, am adăuga şi condiţia suplimentară ca mulţimea soluţie să conţină un număr dat **m** de elemente (m<=n) atunci apare necesară ordonarea elementelor din mulţimea **X** în ordine descrescătoare, astfel ca la fiecare alegere să adăugăm la soluţie un element cu valoare maximă. În acest caz algoritmul se termină când în mulţimea soluţie au fost introduse numărul cerut de elemente.

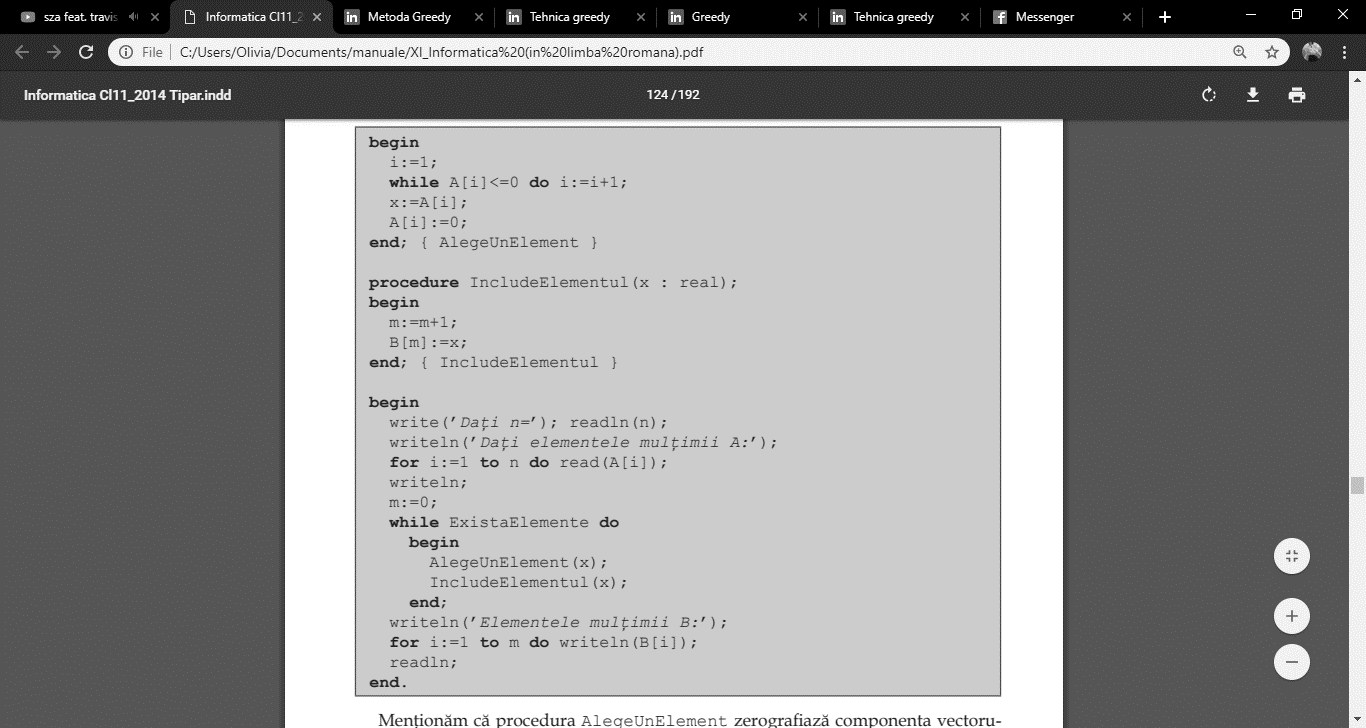
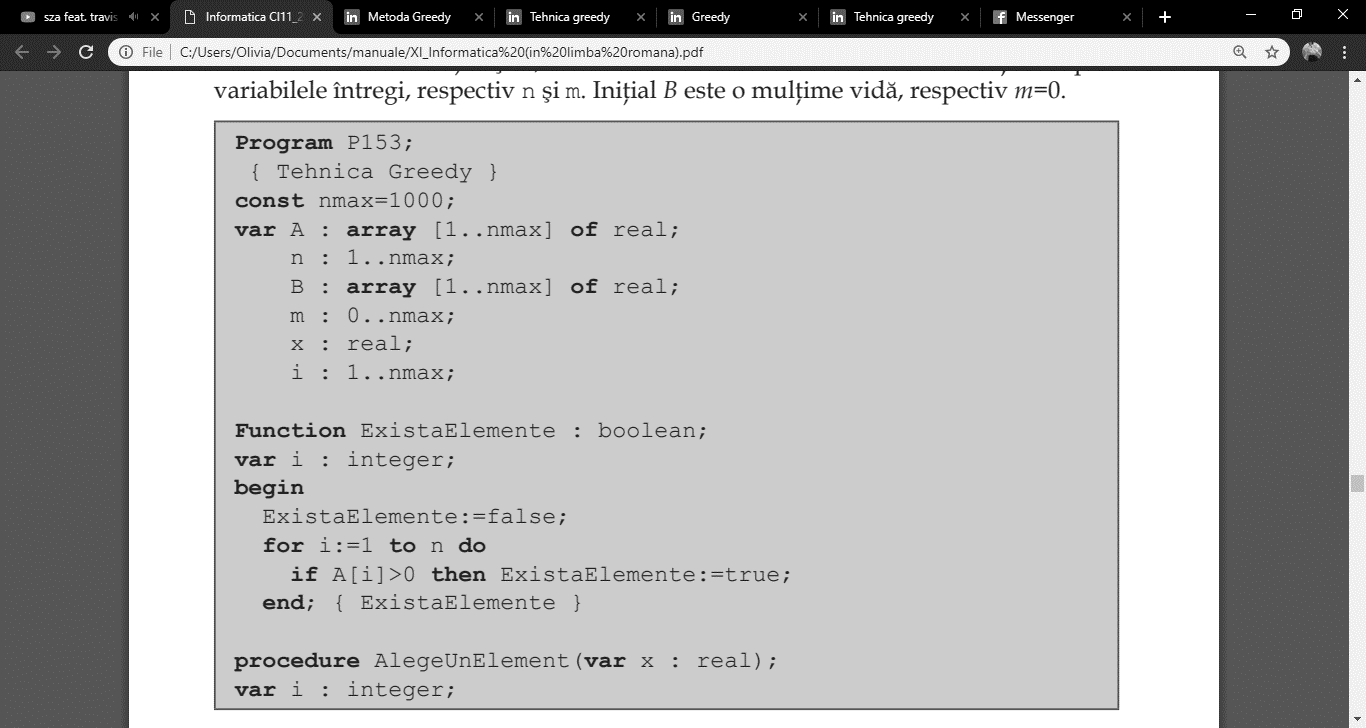
Pentru ***rezolvarea problemei*** reprezentăm atât mulţimea **X** cât şi mulţimea soluţiilor S sub forma a doi vectori de numere reale. Alegerea unui element din **X** se face in ordine, de la **1** **la n**. Funcţia **POSIBIL(B, x)** se reduce la comparaţia **x[i]>0**, iar procedura **ADAUG(B, x)** va consta din adăugarea unui element **x[i]>0** la vectorul **S** în funcţie de contorul **k**.

***Se consideră mulţimea A={a1, a2, ..., ai , ..., an} elementele căreia sînt numere reale, iar cel puţin unul din ele satisface condiţia ai >0. Elaboraţi un program care determină o submulţime B, B⊆A, astfel încît suma elementelor din B să fi e maximă. De exemplu, pentru A={21,5; -3,4; 0; -12,3; 83,6} avem B={21,5; 83,6}.***

**2**

UI

Se observă că dacă o submulţime B, B A, conţine un element b≤0, atunci 7suma elementelor submulţimii B \{b} este mai mare sau egală cu cea a elementelor din B. Prin urmare, regula de selecţie este foarte simplă: la fi ecare pas în submulţimea B se include un element pozitiv arbitrar din mulţimea A.



În programul ce urmează mulţimile A şi B sînt reprezentate prin vectorii (tablourile unidimensionale) A şi B, iar numărul de elemente ale fiecărei mulţimi – prin variabilele întregi, respectiv n şi m. Iniţial B este o mulţime vidă, respectiv m=0.

***Problema continuă a rucsacului*** Se dă un rucsac de o anumită capacitate, greutate și un numar de n obiecte specificandu-se masa obiectelor. Se cere un program care să determine variantă de introducere a obiectelor în rucsac astfel încât să încapă cât mai multe obiecte.

**3**

Algoritm de rezolvare:

* Citirea greutății fiecarui obiect,
* Citirea capacității rucsacului.
* Iniţializăm obiectele.
* Se ordonează obiectele crescător în funcție de greutatea lor.
* Se inţializează volumul disponibil cu volumul obiectului.
* Se verifică dacă fiecare obiect încape în rucsac astfel:Dacă volumul obiectului e mai mic sau egal decât volumul disponibil al rucsacului atunci acesta încape în rucsac și din volumul disponibil al rucsacului scădem volumul obiectului.
* Dacă a rămas o zonă din rucsac neocupată atunci afişăm volumul rămas neocupat, în caz contrar înseamnă că nu am putut introduce nici un obiect în rucsac.

*max\_ob* – numărul maxim de obiecte care pot fi puse în rucsac

*N* - numărul de obiecte disponibile.

**Program** rucsac;

**Var** g:array [1..10] of **integer**;

i,n,Gm,R, aux : **integer**;

ok : **boolean**;

**begin**

**writeln**('nr obiecte'); readln(n);

**writeln**(‘capacitate rucsac'); readln(R);

**writeln**(' Obiectele de luat în rucsac:' );

**for** i:=1 to n do

**read** (g[i]);

*{ sortarea vectorului }*

ok:=false;

**while**(ok=false) **do**

**begin**

ok:=true;

**fo**r i:=1 to n-1 **do**

**if** g[i]>g[i+1] **then**

**begin**

aux:=g[i];

g[i]:=g[i+1];

g[i+1]:=aux;

ok:=false;

**end;**

**end;**

*{ verifică dacă fiecare obiect încape în rucsac }*

**writeln;**  
 for i:=1 to n do write( g[i], '\*');

Gm:=0 ; i:=1;

**while** ( Gm +g[i]<=R ) **do**

**begin**

Gm:=Gm+g[i];

i:=i+1;

**end;**

**writeln**('sunt‘ ,i-1,‘ obiecte cu greutate‘ , Gm,‘) ;

**writeln** ( ‘ a ramas‘ , R-Gm,‘ loc liber‘ ) ;

**end.**

*v\_dis* - volumul rămas disponibil din rucsac.

*O* - obiectele pe care vreau să le iau.

*Greutate* - greutatea fiecarui obiect.

***Divizorii naturali. Fiind dat numărul natural k > 1, se cere să se determine cel mai mic număr natural n având exact k divizori naturali proprii (diferiţi de 1 şi n).***

4 k:=0;

for i:=1 to n do

if x[i]>0 then

begin

k:=k+1;

s[k]:=x[i]

end;

for i:=1 to k do

write(s[i]:5:2,' ');

readln;

end.

4

**begin**

**write**('Numarul de divizori k > 1 '); readln(k);

**write**('Cel mai mic numar care are exact ',k,' divizori este ');

n:=k+2; s:=0;

**while** s = 0 **do**

**begin**

VERIF(n,k,v);

**if** v = true t**hen**

**begin**

**write**(n); s:=1;

**end;**

n:=n+1;

**end;**

**readln;**

**end.**

**Program** k\_divizori\_naturali;

**var** v:boolean;

k,n,s,i:integer;

**procedure** VERIF(n,k:integer;var v:boolean);

**var** j,i:integer;

**begin**

i:=0;

**for** j:=2 **to** n-1 **do**

**if** n mod j = 0 **then**

i:=i+1;

**if** i = k **then**

v:=true

**else**

v:=false;

**end;**

5

***Problema spectacolelor*** Într-un oraş de provincie se organizează un festival de teatru. Oraşul are o singură sală de spectacole, iar la festival şi-au anunţat participarea mai multe trupe. Aşadar, în sală, într-o zi, trebuie planificate N spectacole. Pentru fiecare spectacol se cunoaşte intervalul în care se desfăşoară: [ora\_inceput, ora\_sfarsit]. Se cere să se planifice un număr maxim de spectacole care, bineînţeles, nu se pot suprapune.

Pentru descrierea algoritmului convenim că spectacolele sunt codificate cu numere întregi din mulţimea **{1,2,…N}** iar ora de început şi sfârşit al fiecărui spectacol este exprimată în minute scurse de la miezul nopţii

O planificare optimă a spectacolelor presupune alegerea unui număr maxim **k** de spectacole **i1, i2,...,ik** unde **i1, i2,...,ik∈{1,2,…N},** şi care îndeplinesc condiţia că spectacolul **ij+1** începe după terminarea spectacolului **ij**.

**Vom construi o soluţie după următorul algoritm:**

1. Sortăm spectacolele după ora terminării lor;
2. Primul spectacol programat este cel care se termină cel mai devreme;
3. Alegem primul spectacol dintre cele care urmează în şir după ultimul spectacol programat care îndeplineşte condiţia că începe după ce s-a terminat ultimul spectacol programat;
4. Dacă tentativa de mai sus a eşuat (nu am găsit un astfel de spectacol) algoritmul se termină; altfel se programează spectacolul găsit şi algoritmul se reia de la **3.**

Algoritmul poate fi descris folosind diferite structuri de date:

* tablouri cu două linii şi **N** coloane în care să memorăm ora de început şi sfârşit al spectacolului de pe coloana **j**
* un vector de înregistrări în care să memorăm atât numărul de ordine al spectacolului cât şi ora de început şi sfârşit al lui. Vom aborda algoritmul folosind a doua variantă.

**procedure** greedy;

**var** ;integer;

**begin**

**writeln**(’Ordinea spectacolelor este:’);

ultim:=1;

nr:=1;

**write**(v[1].ord,’ ’);

for i:=2 to n do

if v[i].ora\_inc>v[ultim].ora\_sf then

**begin**

**write**(v[i].ord,’ ’);

ultim:=i;

Inc(nr);

end;

**writeln**(‘Se pot juca ’, nr, ‘ spectacole’); **end**;

**begin**

citire; sortare; greedy;

**end.**

**procedure** citire;

**var** hh, mm, i:integer;

**begin**

**write**(‘Numarul de spectacole:’);

**readln**(n);

for i:=1 to n do

**begin**

**write**(‘Spectacolul, i, incepe la:’);

**readln**(hh,mm);

v[i]. ora\_inc:=hh\*60+mm;

**write** (‘Spectacolul, i, se termina la:’);

**readln**(hh,mm);

v[i].ora\_sf:=hh\*60+mm;

v[i].ord:=i;

**end**;

**end;**

**Program** spectacole;

**Type** spectacol=record

ora\_inc, ora\_sf:integer;

ord:integer;

**end;**

**Var** v:array[1..30] of spectacol;

n, ultim, nr:integer;

**procedure** sortare;

**var** i,j :integer; aux:spectacol;

**begin**

for i:=1 to n-1 do

for j:=i+1 to n do

if v[j].ora\_sf < v[i].ora\_sf then

begin

aux:=v[j];

v[j]:=v[i];

v[i]:=aux;

end;

**end;**

**CONCLUZII:**

►Metoda Greedy constă în a alege pe rând câte un element urmând sa îl introducă eventual în soluția optimă dar nu caută să aleagă soluția conform criteriului de optimizare. Astfel, ea nu furnizează soluția optimă pentru orice problema. De aceea aceasta metoda este mai favorabila numai în cazul rezolvării problemelor de optimizare.

►Metoda Greedy este una dintre cele mai directe tehnici de proiectare a algoritmilor care poate fi aplicată la o gamă largă de probleme. Insa cu regret, metoda Greedy poate fi aplicată numai atunci cînd din enunţul problemei poate fi dedusă regula care asigură selecţia directă a elementelor necesare din mulţimea A.

**Avantaje:**

1. Poate fi aplicata multor probleme
2. Determinarea celor mai scurte drumuri in grafuri (Dijkstra)
3. Determinarea arborelui minimal de acoperire (Prim, Kruskal)
4. Codificare arborilor Huffmann
5. Planificarea activităților
6. Problema spectacolelor și problema fracționară a rucsacului
7. Algoritmii sunt polinomiali
8. Necesită mai puțin timp.

**Dezavantaje:**

1. Algoritmii Greedy nu conduc în mod necesar la o solutie optimă.
2. Nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare.
3. Poate fi aplicată numai atunci cînd din enunțul problemei poate fi dedusă regula care asigură selecția directă a elementelor necesare din mulțimea A.
4. Există probleme pentru care nu se cunosc astfel de algoritmi. Mai mult, pentru cele mai multe probleme, nu se cunosc algoritmi Greedy.

**DATE BIBLIOGRAFICE:**

* [file:///C:/Users/Documents/manuale/XI\_Informatica%20(in%20limba%20romana).pdf](file:///C:\Users\Documents\manuale\XI_Informatica%20(in%20limba%20romana).pdf)
* <https://ru.scribd.com/doc/43454385/Metoda-Greedy#download>
* <https://sites.google.com/site/eildegez/home/clasa-xi/prezentarea-metodei-greedy>
* <https://www.infoarena.ro/metoda-greedy-si-problema-fractionara-a-rucsacului>
* <http://www.worldit.info/articole/algoritmica-articole/metoda-greedy/>