Metoda GREEDY

Curs 6

Continut

1. Prezentarea generala a metodei
2. Schema generala a metodei
3. Implementari ale metodei

Prezentarea generala a metodei

Algoritmii de tip greedy (backtracking si de programare dinamica) se aplica unor probleme a caror solutie poate fi exprimata sub forma unui vector de numere întregi (cu valori între 1 si n).

Intr-o problema de optimizare trebuie gasita solutia optima dintre toate solutiile posibile.

Alte clase de probleme cu solutie vectoriala sunt probleme de enumerare a tuturor solutiilor posibile si probleme de decizie, la care trebuie spus daca exista sau nu cel putin o solutie.

Prezentarea generala a metodei

Metoda Greedy se poate aplica unor probleme de optimizare cu solutie vectoriala, ca alternativa mai eficienta la o cautare exhaustiva (de tip 'backtracking').

Un algoritm Greedy este un algoritm iterativ (nerecursiv) care determina în fiecare pas k o componentã x[k] a vectorului solutie si nu mai revine ulterior la aceasta alegere.

Prezentarea generala a metodei

Numele metodei ('Greedy'= lacomie) sugereaza modul de lucru: – la stabilirea valorii lui x[k] se alege dintre candidatii posibili pe acela care este optim în pasul k, deci un optim local.

In general, aceasta alegere precipitata, grabita si care nu tine seama de valorile ce vor fi stabilite în pasii urmatori pentru x[k+1],..x[n] nu poate garanta solutia optima a problemei.

Prezentarea generala a metodei

In functie de specificul problemei, un algoritm greedy poate conduce la solutia optima sau la o solutie destul de buna, desi suboptimala.

Rezultatul unui algoritm greedy pentru o problema data depinde si de datele concrete ale problemei, sau chiar de ordinea introducerii lor.

Prezentarea generala a metodei

De exemplu, în problema exprimarii unei sume de bani printr-un numar minim de monede de valori date rezultatul (optim sau suboptim) depinde de valorile monedelor si de valoarea sumei.

Algoritmul greedy foloseste monedele în ordinea descrescatoare a valorilor lor, deci “se repede” la monedele de valoare maxima, care vor fi în numar mai mic pentru aceeasi suma.

Fie monede de valori 11,5 si 1:

– pentru suma 12 rezultatul algoritmului greedy va fi optim (doua monede de valori 11 si 1)

– dar pentru suma 15 rezultatul algoritmului greedy nu va fi optim (5 monede de valori 11,1,1,1,1 în loc de 3 monede de valori 5,5,5).

Schema generala a metodei

Pentru a exemplifica această metodă considerăm o mulţime A cu n elemente.

Problema care ar trebui rezolvată constă din determinarea unei submulţimi B a lui A.

Aceasta trebuie să îndeplinească anumite condiţii pentru a fi acceptată ca soluţie.

Dintre toate submulţimile acceptate (numite soluţii posibile), se va alege una singură numită soluţie optimă.

Schema generala a metodei

Dintre soluţiile posibile trebuie aleasă cea optimă tinând cont de proprietatea următoare:

Dacă B este soluţie posibilă şi $C \subset B$ atunci şi C este soluţie posibilă.

Multimea $\emptyset$ este întotdeauna soluţie posibilă.

Iniţial se porneşte de la mulţimea vidă.

Se alege într-un anumit fel un element din A, neales la paşii precedenţi.

Dacă acestă adăugare la soluţia parţial construită conduce la o soluţie posibilă atunci construim noua soluţie posibilă prin adăugarea elementului – procedura Greedy.

Schema generala a metodei

Descrierea formala a unui algoritm greedy general este:

function *greedy*(C) // C este multimea candidatilor

$S \leftarrow \emptyset$ // S este multimea in care construim solutia

while not *solutie*(S) **and** $C \neq \emptyset$ **do**

$x \leftarrow$ un element din C care maximizeaza/minimizeaza *select*(x)

$C \leftarrow C - \{x\}$

if *fezabil*($S \cup \{x\}$) **then** $S \leftarrow S \cup \{x\}$

if *solutie*(S) **then return** S

else return ”nu exista solutie”

Schema generala a metodei

La fiecare pas, incercam sa adaugam la aceasta multime pe cel mai promitator candidat, conform functiei de selectie.

Daca, dupa o astfel de adaugare, multimea de candidati selectati nu mai este fezabila, eliminam ultimul candidat adaugat, acesta nu va mai fi niciodata considerat.

Daca, dupa adaugare, multimea de candidati selectati este fezabila, ultimul candidat adaugat va ramane de acum incolo in ea.

De fiecare data cand largim multimea candidatilor selectati, verificam daca aceasta multime nu constituie o solutie posibila a problemei.

Schema generala a metodei

Daca algoritmul greedy functioneaza corect, prima solutie gasita va fi totodata o solutie optima a problemei.

Solutia optima nu este in mod necesar unica.

Aceeasi valoare optima pentru mai multe solutii posibile.

Implementari ale metodei

Maximizarea/minimizarea valorii unei expresii

- Se dau n numere întregi nenule $b_1, b_2, ..., b_n$ şi m numere întregi nenule $a_1, a_2, ..., a_m$.

Să se determine un subşir al şirului $b_1, b_2, ..., b_n$ care să maximizeze/minimizeze valoare expresiei:

$$E = a_1 * x_1 + a_2 * x_2 + ... + a_m * x_m$$

ştiind că $n > m$ şi că $x_i \in \{b_1, b_2, ..., b_n\}$, $b_i>0$, $\forall i=1,n$.

Implementari ale metodei

Maximizarea/minimizarea valorii unei expresii

Algoritmul de rezolvare este următorul: se ordonează cei doi vectori (a şi b) (cele două mulţimi de numere) crescător a şi b.

Apoi se grupează cele mai mari elemente pozitive din a cu elementele cele mai mari din b şi elementele cele mai mici din a cu elementele cele mai mici din b.

Implementari ale metodei

Maximizarea/minimizarea valorii unei expresii

Se citeste vectorul a [m] , b [n]

Se ordoneaza crescator a si b //avem o procedura schimba (int a, int b) care poate fi apelata in program

Se iau elementele pozitive din b si se grupeaza cu elementele cele mai mari din a

```
i=n, j=m, expresie = 0
cat timp a[j] > 0 si j > 0
        expresie = expresie +  b[i] * a[j]
        i=i-1 ,  j=j-1
k=j
i=1 , j=1
cat timp k>0
        expresie = expresie + b[i]*a[j]
        i=i+1, j=j+1, k=k-1
scrie expresie
```

Implementari ale metodei

Maximizarea/minimizarea valorii unei expresii

```
dati m pentru a 3
dati n pentru b 5
6
7
5
4
2
3
1
3
a este
5  6  7
b este
1   2   3   3   4
valoarea a lui a 7 cu valoare a lui b 4
valoarea a lui a 6 cu valoare a lui b 3
valoarea a lui a 5 cu valoare a lui b 3
valoarea expresiei este 61
```

```
dati m pentru a 5
dati n pentru b 5
-9
2
1
7
3
1
4
3
5
2
a este
-9  1  2  3  7
b este
1   2   3   4   5
valoarea a lui a 7 cu valoare a lui b 5
valoarea a lui a 3 cu valoare a lui b 4
valoarea a lui a 2 cu valoare a lui b 3
valoarea a lui a 1 cu valoare a lui b 2
valoarea expresiei este 46
--------------------------------
```

Implementari ale metodei

Problema spectacolelor

Într-o sală, într-o zi, trebuie planificate `n` spectacole. Pentru fiecare spectacol se cunoaşte ora de începere (`start[i]`) şi durata spectacolului (`durata[i]`). Se cere să se planifice un număr maxim de spectacole astfel încât să nu se suprapună.

Să considerăm `A` = mulţimea iniţială de spectacole şi `B` = mulţimea spectacolelor ce vor fi alese.

- Pentru a rezolva problema prin tehnica Greedy, prelucrarea care se va face asupra mulţimii `A` – este o ordonare crescătoare după ora de finalizare
- Apoi se iau spectacolele în ordine, astfel încât fiecare spectacol să înceapă după ce s-a terminat cel anterior lui.

Implementari ale metodei

Problema spectacolelor

Exemplu: Numărul total de spectacole n=6, cu cei doi vectori: start=(2,4,1,3,6,8) şi durata=(2,2,2,2,1,3).

Se vor obţine orele de terminare a spectacolelor: (4,6,3,5,7,11).

- În urma sortării după criteriul orelor de terminare a spectacolelor, se va obţine ordinea: 3,1,4,2,5,6.

- Se ia **spectacolul 3** (iniţial şi nu după ordonare!), care se termină la ora 3. Urmează spectacolul 1, care începe, însă, la ora 2, deci se sare peste el. Următorul este **spectacolul 4**, care începe la ora 3 şi se termină la ora 5. Urmează spectacolul 2, care nu se ia. În schimb, se ia **spectacolul 5**, care începe la ora 6, și apoi **spectacolul 6.** Ora de terminare a tututor spectacolelor va fi, aşadar, 11.

Implementari ale metodei

Problema spectacolelor

```
dati nr spectacole
6
spectacol nr 1
ora incepere 2
durata lui 2
spectacol nr 2
ora incepere 4
durata lui 2
spectacol nr 3
ora incepere 1
durata lui 2
spectacol nr 4
ora incepere 3
durata lui 2
spectacol nr 5
ora incepere 6
durata lui 1
spectacol nr 6
ora incepere 8
durata lui 3
soluti
spectacolul 3
spectacolul 4
spectacolul 5
spectacolul 6
```

Implementari ale metodei

Problema continuă a rucsacului

- Cu ajutorul unui rucsac de greutate maximă admisibilă GG trebuie să se transporte o serie de obiecte din n disponibile, având greutăţile G1, G2, ..., Gn, aceste obiecte fiind de utilităţile C1, C2, ..., Cn.
- Dacă pentru orice obiect i putem să luăm doar o parte $x_i \in [0,1]$ din el, atunci spunem că avem **problema continuă a rucsacului**
- În problema continuă a rucsacului, prin raportarea utilităţilor la greutăţi obţinem utilităţile pe unitate de greutate, astfel încât va trebui să ordonăm obiectele în funcţie de aceste raporturi şi să le încărcăm, pe cât posibil, în întregime în rucsac, până când acesta se umple.
- Astfel, reprezentând soluţia în vectorul x, vom pune la început x[i]:=1, până când greutatea rămasă disponibilă, notată GGr, nu mai permite punerea unui obiect în întregime. La fiecare pas actualizând greutatea rămasă disponibilă GGr, după formula: GGr:=GGr-G[i].
- Atunci, vom face x[i]:=GGr/G[i], ultimul obiect fiind "tăiat".

Implementari ale metodei

Problema continuă a rucsacului

```
nr obiecte 4
c[1]= 3
g[1]= 4
c[2]= 2
g[2]= 7
c[3]= 4
g[3]= 2
c[4]= 1
g[4]= 2
greut maxima este 7
am ordonat
c[3]= 4 g[3]= 2 -> 2
c[1]= 3 g[1]= 4 -> 0.75
c[4]= 1 g[4]= 2 -> 0.5
c[2]= 2 g[2]= 7 -> 0.285714
 Aleg c[3]  cu greutatea 2
 Greutatea ramasa este 5
 Aleg c[1]  cu greutatea 4
 Greutatea ramasa este 1
 Aleg din c[4] cu greutatea 2 doar 0.5-a parte
Obiectul c[2] cu greutatea 2 nu este selectat
 Greutatea ramasa este 0
x[3]= 1
x[1]= 1
x[4]= 0.5
x[2]= 0
```

Implementari ale metodei

Arborele parţial de cost minim

- Algoritmul de rezolvare pe care îl folosim se bazează pe tehnica Greedy.
- La început se ia nodul 1 şi se pune în arbore. Apoi, la fiecare din cei

`n-1` paşi (arborele va avea `n-1` muchii) se ia muchia de cost minim printre muchiile cu o extremitate în arborele deja creat şi cu alta în afara acestuia, apoi această muchie se adaugă arborelui.

- Pentru o alegere simplă a celei mai mici muchii cu respectiva proprietate, se ordonează mai întâi muchiile crescător după costuri.
- De aceea, chiar şi graful va fi dat sub forma unui şir de muchii, fiecare fiind caracterizată de extremităţile şi de costul ei.
- Verificarea sau stabilirea faptului că un anumit nod se află sau nu în arbore se va face cu ajutorul unui vector numit Marcat, de valori booleene

Implementari ale metodei

Arborele parţial de cost minim

```
struct muchie{
            int u,v;  //extremitati muchie, u primul nod, v al doilea nod
            int c;  //costul muchiei
};
muchie graf[20]; bool marcat[20]; int cost_min;
Pentru i=a la m
      citeste graf[i].u, citeste graf[i].v,  citeste graf[i].c
cost_min =0
//ordonam muchiile dupa cost
   pentru i=1 la m
            pentru j=i+1 la m
                        daca graf[i].c >graf[j].c  atunci schimba(graf[i], graf[j])
```

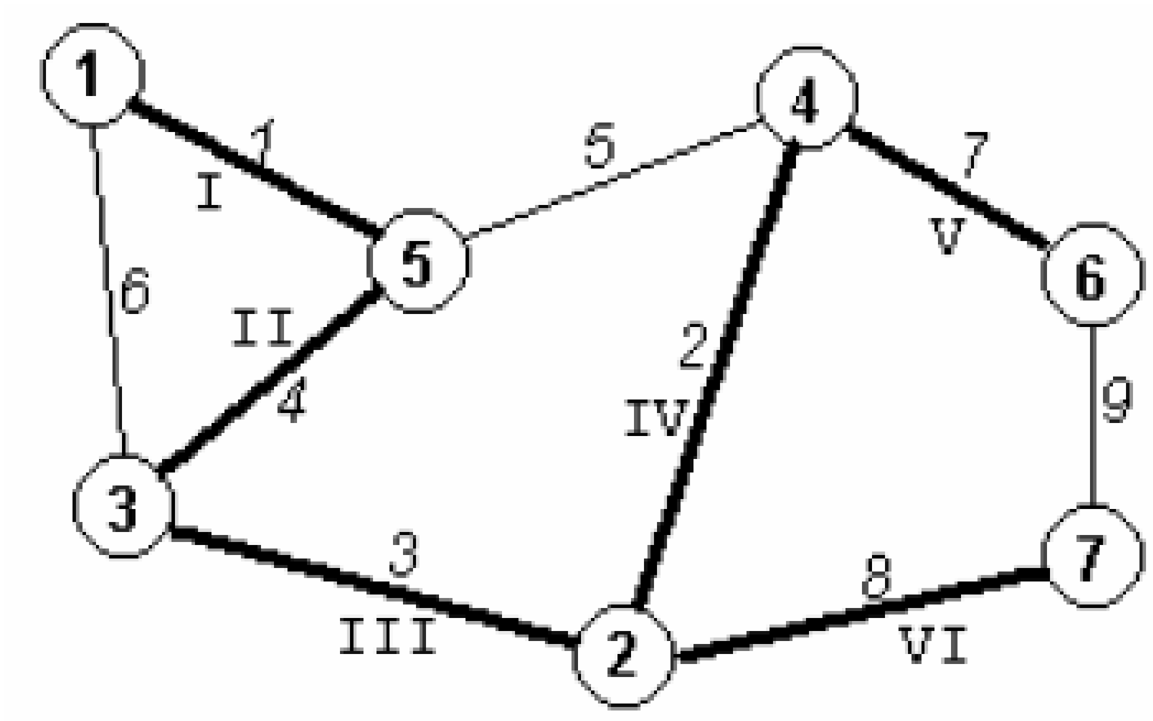
Implementari ale metodei

Arborele parţial de cost minim

```
//la inceput niciun nod nu e marcat ca vizitat
Pentru i=1 la n
        marcat[i]=false
marcat[i]=true
//aflam muchiile ,  vor fi n-1 muchii
pentru i=1 la n
        j=1
        cat timp (!  Marcat[graf[j].u] xor marcat[graf[j].v]  ) //xor pentru 1 1->0
                j++  //daca avem muchie marcata trecem la alta muchie , crestem j
        marcat[graf[j].u] = true
        marcat[graf[j].v] = true
        afisare "muchia de la" graf[j].u  "la" graf[j].v "cu cost" graf[j].c
        Cost_min=cost_min + graf[j].c
afisare cost_min
```

Implementari ale metodei
Arborele parţial de cost minim

Exemplu cu 7 noduri si 11 muchii



Implementari ale metodei

Arborele parţial de cost minim

```
dati datele muchiei:  3
 primul nod 3
 al doilea 5
 costul 4
dati datele muchiei:  4
 primul nod 5
 al doilea 4
 costul 4
dati datele muchiei:  5
 primul nod 4
 al doilea 2
 costul 2
dati datele muchiei:  6
 primul nod 3
 al doilea 2
 costul 3
dati datele muchiei:  7
 primul nod 4
 al doilea 6
 costul 7
dati datele muchiei:  8
 primul nod 6
 al doilea 7
 costul 9
dati datele muchiei:  9
 primul nod 2
 al doilea 7
 costul 8
 arborele este
muchia 1 - 5 de cost 1
muchia 3 - 5 de cost 4
muchia 3 - 2 de cost 3
muchia 4 - 2 de cost 2
muchia 4 - 6 de cost 7
muchia 2 - 7 de cost 8
cost minim 25
```

Implementari ale metodei

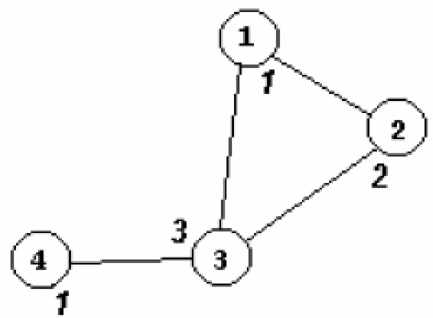
Problema colorării hărţilor

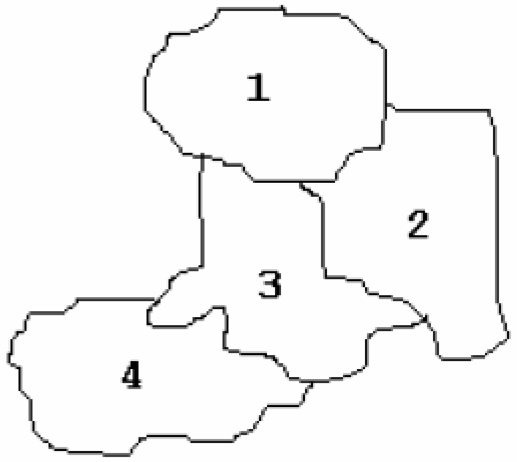
- N ţări sunt date precizându-se relaţiile de vecinatate. Se cere să se determine o posibilitate de colorare a hărţii (cu cele n ţări), astfel încât să nu existe ţări vecine colorate la fel.
- Avem un algoritm Greedy care va colora, pe rând, fiecare nod în cea mai mică (ca indice) culoare posibilă.
- Ordinea în care se iau nodurile grafului este memorată în vectorul a.
- **Exemplu: Avem 4 tari**

Harta poate fi reprezentată sub forma unui graf, cu culori

asociate nodurilor reprezentate prin cifre numere intregi.

Permutarea nodurilor este a=(1,2,3,4)





Implementari ale metodei

Problema colorării hărţilor

1.Citire matrice vecin[i][j] – doua tari sunt vecine deci v[i][j]=1 altfel este 0

2.Vecin [i][i] =0

3. Pentru i=1 la n

x[i] =1 // vectorul x pentru stabilirea culorii. Prima tara/nod –prima culoare

pentru j=1 la i-1

daca vecin[i][j] =1 si x[j] = x[i] atunci x[i]=x[i] +1

4. Pentru i=1 la n

scrie "tara " , i, "in culoarea", x[i]

Implementari ale metodei
Problema colorării hărţilor

```
dati nr de tari/noduri 4
este vecina tara 1 cu tara 2 ? da=1, nu=0 : 1
este vecina tara 1 cu tara 3 ? da=1, nu=0 : 1
este vecina tara 1 cu tara 4 ? da=1, nu=0 : 0
este vecina tara 2 cu tara 3 ? da=1, nu=0 : 1
este vecina tara 2 cu tara 4 ? da=1, nu=0 : 0
este vecina tara 3 cu tara 4 ? da=1, nu=0 : 1
o colorarea greedy a tarilor este
tara 1 in culoarea 1
tara 2 in culoarea 2
tara 3 in culoarea 3
tara 4 in culoarea 1


--------------------------------
Process exited after 35.41 seconds with return value 0
Press any key to continue . . .
```

Implementari ale metodei

Problema colorării hărţilor

```
dati nr de tari/noduri 6
este vecina tara 1 cu tara 2 ? da=1, nu=0 : 1
este vecina tara 1 cu tara 3 ? da=1, nu=0 : 1
este vecina tara 1 cu tara 4 ? da=1, nu=0 : 0
este vecina tara 1 cu tara 5 ? da=1, nu=0 : 0
este vecina tara 1 cu tara 6 ? da=1, nu=0 : 0
este vecina tara 2 cu tara 3 ? da=1, nu=0 : 1
este vecina tara 2 cu tara 4 ? da=1, nu=0 : 1
este vecina tara 2 cu tara 5 ? da=1, nu=0 : 0
este vecina tara 2 cu tara 6 ? da=1, nu=0 : 1
este vecina tara 3 cu tara 4 ? da=1, nu=0 : 1
este vecina tara 3 cu tara 5 ? da=1, nu=0 : 0
este vecina tara 3 cu tara 6 ? da=1, nu=0 : 0
este vecina tara 4 cu tara 5 ? da=1, nu=0 : 1
este vecina tara 4 cu tara 6 ? da=1, nu=0 : 1
este vecina tara 5 cu tara 6 ? da=1, nu=0 : 0
o colorarea greedy a tarilor este
tara 1 in culoarea 1
tara 2 in culoarea 2
tara 3 in culoarea 3
tara 4 in culoarea 1
tara 5 in culoarea 2
tara 6 in culoarea 2
```