CENTRUL DE PREGĂTIRE PENTRU PERFORMANȚĂ HAI LA OLIMPIADĂ!
DISCIPLINA **INFORMATICĂ**
JUDEȚUL **SUCEAVA**

Titlul lecției: **Metoda Greedy. Aplicații.**

Data: 25.02.2017

Profesor: Petrică Galan

Grupa a IX-a Seniori, **locul de desfășurare:** Colegiul Național „Ștefan cel Mare”, Suceava

Cuprins

I. Metoda Greedy

Prezentare

Greedy (lacom) este o tehnică de elaborarea a algoritmilor ce se bazează pe alegerea optimului local la fiecare etapă din construirea soluției, în speranța obținerii optimului global. Abordarea este una iterativă, fiecare alegere putându-se baza pe alegerile anterioare, dar nu pe cele viitoare. **O alegere o dată efectuată nu va fi reconsiderată** (este definitivă). Acest lucru diferențiază tehnica de altele (programare dinamică).

Aplicabilitate

Algoritmii de tip Greedy sunt ideali pentru probleme care au proprietatea de **optim local**, adică soluția optimă a problemei conține soluțiile optime ale subproblemelor similare cu problema inițială, dar de dimensiune mai mică. Spunem despre aceste probleme că au **substructură optimală**.

În general, pentru astfel de probleme, se cere determinarea elementelor unei submulțimi S a unei mulțimi A, care să îndeplinească anumite condiții. Algoritmul furnizează o singură soluție, de obicei sub forma unui tablou, care în general este optimă (uneori soluția obținută nu este optimă, iar în alte situații nu ajungem la soluție – metoda este euristică).

Soluția este construită prin selectarea dintr-o mulțime a elementelor care îndeplinesc anumite condiții. Pentru ca elementele selectate să aparțină soluției optime, se alege la fiecare pas candidatul optim pentru pasul respectiv. De multe ori, în practică, se ordonează mulțimea A după criteriul candidatului optim.

Pentru a fi siguri că algoritmul conduce la obținerea soluției optime, trebuie îndeplinite condițiile:

- Alegerea optimului local trebuie să conducă la obținere soluției optime;
- Soluția optimă conține soluțiile optime ale subproblemelor.

Algoritm

Pasul 1: Se inițializează mulțimea S cu mulțimea vidă
Pasul 2: Cât timp S nu este o soluție a problemei și mulțimea A are elemente
    Pasul 3: Se alege candidatul optim din mulțimea A
    Pasul 4: Se elimină candidatul optim din A
    Pasul 5: Dacă poate fi element al soluției, se adaugă acesteia și se revine la Pasul 2.
Pasul 6: Daca S este soluția problemei se afișează, altfel se afișează „Nu s-a găsit soluția”

Exemple de probleme ce admit implementări Greedy

- Planificarea optimă a unui set de activități;
- Ocuparea optimă a unui mijloc de transport;
- Plata unei sume de bani cu număr minim de bancnote;
- Grafuri (drumuri de cost minim, arbori parțiali de cost minim, etc).

II. Exemple de utilizare

1. Planificarea optimă a unui set de activități

Pentru fiecare activitate se cunoaște ora de început, ora de sfârșit și un număr de identificare. Repartizarea optimă presupune planificarea unui cât mai mare număr de activități. Din mulțimea activităților se alege la fiecare etapă activitatea cu ora de sfârșit cea mai mică. Ordonarea activităților după criteriul candidatului optim presupune ordonarea activităților crescător după ora de sfârșit.

Exemplu:

Activitate	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Început	**9**	**12**	**8**	**10**	**16**	**14**	**20**
Sfârșit	**11**	**13**	**10**	**12**	**18**	**16**	**22**

Ordonarea activităților după ora de sfârșit

Activitate	**3**	**1**	**4**	**2**	**6**	**5**	**7**
Început	**8**	**9**	**10**	**12**	**14**	**16**	**20**
Sfârșit	**10**	**11**	**12**	**13**	**16**	**18**	**22**

Planificare:

Activitate	**3**	**4**	**2**	**6**	**5**	**7**
Început	**8**	**10**	**12**	**14**	**16**	**20**
Sfârșit	**10**	**12**	**13**	**16**	**18**	**22**

```
#include<iostream>
#include<fstream>
using namespace std;

struct task{
    int start, stop, id;
};

void ordonez(task a[100], int n)
{
    task aux;
    for(int i=1;i<n;i++)
        for(int j=i+1;j<=n;j++)
            if(a[i].stop>a[j].stop)
            {
                aux=a[i];
                a[i]=a[j];
                a[j]=aux;
            }
}

void greedy(task a[100], int n, int s[100], int &m)
{
    s[1]=1;
```

```
    m=1;
    for(int i=2;i<=n;i++)
        if(a[i].start>=a[s[m]].stop)
            s[++m]=i;
}

int main()
{
    ifstream f("tasks.in");
    task a[100];
    int s[100], n, m;
    f>>n;
    for(int i=1;i<=n;i++)
    {
        f>>a[i].start>>a[i].stop;
        a[i].id=i;
    }
    ordonez(a, n);
    greedy(a, n, s, m);
    for(int i=1;i<=m;i++)
        cout<<a[s[i]].id<<'
'<<a[s[i]].start<<">"<<a[s[i]].stop<<endl;
}
```

2. Problema rucsacului

Se consideră că dispunem de un rucsac cu capacitatea G și de N obiecte, definite fiecare prin greutate și valoare, ce trebuie introduce in rucsac. Se cere o modalitate de a umple rucsacul cu obiecte , astfel încât valoarea totală să fie maximă. Putem lua fragmente dint-un obiect.

Exemplu:

Iniţial					
Obiect	**1**	**2**	**3**	**4**	**5**
Greutate	**5**	**4**	**4**	**8**	**10**
Valoare	**10**	**20**	**10**	**10**	**22**
Eficiență	**2**	**5**	**2,5**	**1,25**	**2,2**

Sortare					
Obiect	**2**	**3**	**5**	**1**	**4**
Greutate	**4**	**4**	**10**	**5**	**8**
Valoare	**20**	**10**	**22**	**10**	**10**
Eficiență	**5**	**2,5**	**2,2**	**2**	**1,25**

Soluţie					
Obiect	**2**	**3**	**5**	**1**	**4**
Greutate	**4**	**4**	**10**	**5**	**8**
Fracțiune	**1**	**1**	**1**	**0,4**	**0**

Vom reprezenta soluția problemei ca pe un vector. Vom ordona obiectele descrescător ținând cont de valoarea/greutate. Atâta timp cât obiectele încap în rucsac, le vom adaugă în întregime și putem întâlni una din următoarele situații:

- obiectele alese au o greutate totală egală cu a rucsacului;
- mai există loc în rucsac, dar nu mai încape nici un obiect întreg, caz în care adăugăm o fracțiune de obiect.

*Ce se întâmplă dacă obiectele nu pot fi fracționate?

```
#include<iostream>
#include<fstream>
using namespace std;

struct obiect{
    int id;
    float greutate, valoare, eficienta;
};

void ordonez(obiect a[100], int n)
{
    obiect aux;
    for(int i=1;i<n;i++)
        for(int j=i+1;j<=n;j++)
            if(a[i].eficienta<a[j].eficienta)
            {
                aux=a[i];
                a[i]=a[j];
                a[j]=aux;
            }
}

void greedy(obiect a[100], int n, int s[100], int &m, float x[100],
float G)
{
    m=0;
    for(int i=1;i<=n && G!=0;i++)
        if(G>a[i].greutate)
        {
            s[++m]=i;
            x[m]=1;
            G-=a[i].greutate;
        }
        else{
            s[++m]=i;
            x[m]=G/a[i].greutate;
            G=0;
        }
}
```

```
int main()
{
    ifstream f("obiecte.in");
    obiect a[100];
    int n, m, s[100];
    float G, x[100];
    f>>n>>G;
    for(int i=1;i<=n;i++)
    {
        f>>a[i].greutate>>a[i].valoare;
        a[i].eficienta=a[i].valoare/a[i].greutate;
        a[i].id=i;
    }
    ordonez(a, n);
    greedy(a, n, s, m, x, G);
    for(int i=1;i<=m;i++)
        cout<<"obiect "<<a[s[i]].id<<" de greutate
"<<a[s[i]].greutate<<" fractiune "<<x[i]<<endl;
}
```

III. Aplicaţii propuse

#398 Plopi2

Cerinţa

De-a lungul principalei străzi din orașul nostru există n plopi, pentru fiecare cunoscându-se înălțimea. Primarul orașului dorește ca plopii să aibă înălțimile în ordine descrescătoare. Pentru aceasta, este posibilă tăierea dintr-un plop a unei bucăți – este o tehnică ecologică, nevătămătoare, în urma căreia plopul nu are de suferit. Plopii nu pot fi înălțați în nici un fel.

Determinați numărul minim de plopi din care se va tăia și lungimea totală minimă a bucăților tăiate.

Date de intrare

Fișierul de intrare plopi2.in conține pe prima linie numărul de plopi n. Urmează n numere naturale nenule, separate prin spaţii, care pot fi dispuse pe mai multe linii, reprezentând înălțimile plopilor.

Date de ieşire

Fișierul de ieșire plopi2.out va conține pe prima linie numerele C T, separate prin exact un spațiu, reprezentând numărul minim de plopi din care se va tăia și lungimea totală minimă a bucăților tăiate.

Restricţii şi precizări

$2 \leq n \leq 100$

înălțimile plopilor vor fi mai mici decât 5000

Exemplu

plopi2.in

8

5 7 3 6 4 4 2 6

plopi2.out

5 11

Explicație

Vom tăia din plopii numerotați cu 2 4 5 6 8, astfel încât înălțimile să devină 5 5 3 3 3 3 2 2. Lungimea totală a bucăților tăiate este: 2 + 3 + 1 + 1 +4 = 11

#91 Masini

În curtea unui atelier de reparaţii auto, sunt n maşini care trebuie sa fie reparate. Deoarece nu sunt suficienţi mecanici, în fiecare moment de timp se poate lucra doar la o singură maşină.

Cerinţa

Cunoscând timpul necesar pentru repararea fiecărei maşini, scrieţi un program care calculează numărul maxim de maşini care pot fi reparate într-un interval de timp T.

Date de intrare

Pe prima linie a fişierului masini.in se găsec două numere naturale n şi T separate printr-un singur spaţiu, reprezentând numărul de maşini din curtea atelierului auto şi timpul total în care se va lucra. Pe linia a doua, separate prin câte un spaţiu, se găsesc n numere naturale t1, t2, …, tn, reprezentând timpii necesari pentru repararea fiecărei maşini.

Date de ieşire

Pe prima linie a fişierului masini.out se va găsi un număr natural k, reprezentând numărul maxim de maşini care pot fi reparate.

Restricţii şi precizări

1 < n, T <= 1000

numerele de pe a doua linie a fişierului de intrare vor fi mai mici sau egale cu 100

Exemplu

masini.in

5 10

6 2 4 8 2

masini.out

3

Explicație

Se vor repara masinile 2, 3 şi 5, cu timpii de reparaţie 2, 4 şi 2.

1004 Eureni

Cerința

Pentru cadourile pe care Moş Crăciun urmează să le cumpere copiilor cuminţi, Consiliul Polului Nord a alocat suma de S eureni. Ştiind că în comerţul polar se utilizează n+1 tipuri de bancnote de valori 1, e1, e2, e3,…, en şi faptul că Moşul trebuie să primească un număr minim de bancnote pentru suma aprobată, să se determine numărul de bancnote din fiecare tip utilizat în plata sumei şi numărul total de bancnote care i s-au alocat.

Date de intrare

Fișierul de intrare eureni.in conține pe prima linie numerele S n e.

Date de ieșire

Fișierul de ieșire eureni.out va conține mai multe linii: pe fiecare linie va fi scrisă valoare unei bancnote folosită în plata sumei S și numărul de bancnote folosite, separate printr-un spațiu, în ordinea descrescătoare a valorilor bancnotelor folosite. Pe ultima linie se va scrie numai numărul total de bancnote folosite.

Restricții și precizări

1 < S < 2 000 000 000

1 < n < 10

1 < e < 10

se presupune că există un număr nelimitat de bancnote de fiecare tip

Exemplu

eureni.in

107 4 5

eureni.out

25 4

5 1

1 2

7

Explicație

Sunt 5 tipuri de bancnote, cu valorile: 1, 5, 25, 125, 625 eureni. Pentru a plăti suma de 107 eureni se folosesc 4 bancnote de 25 eureni, 1 bancnotă de 5 eureni și 2 bancnote de 1 euren, în total 7 bancnote.

#400 Pachete

Cerinţa

Într-un depozit există un raft cu n+1 spații de depozitare, numerotate de la 1 la n+1. Primele n spatii de depozitare sunt ocupate cu n pachete numerotate cu valori între 1 și n, iar spațiul de depozitare n+1 este gol.

Administratorul depozitului decide mutarea pachetelor, astfel încât pentru orice i, pachetul numerotat cu i să se afle în spațiul de depozitare i. Pentru aceasta se va folosi spațiul de depozitare suplimentar, n+1, singura manevră validă fiind mutarea unui pachet dintr-un spațiu de depozitare în altul, cu condiția ca acesta să fie gol.

Determinați o succesiune de manevre prin care fiecare pachet să fie în spațiul corect.

Date de intrare

Fişierul de intrare pachete.in conţine pe prima linie numărul n, iar pe a doua linie n numere naturale separate prin spaţii. Al i-lea număr reprezintă numărul pachetului aflat în spațiul de depozitare i.

Date de ieşire

Fişierul de ieşire pachete.out va conţine pe prima linie numărul M, reprezentând numărul de manevre efectuate. Pe fiecare dintre următoarele M linii se descrie o manevră, prin două numere i j, cu semnificația: se ia pachetul din spațiul i și se mută în spațiul j.

Restricţii şi precizări

$1 \le n \le 100$

pentru fiecare manevră i j efectuată, $1 \le i,j \le n+1$, $i \ne j$, iar spațiul j trebuie să fie gol

numărul de manevre realizate nu trebuie să fie minim

Exemplu

pachete.in

```
5
2 5 4 3 1
```

pachete.out

```
7
1 6
5 1
2 5
6 2
3 6
4 3
6 4
```

Explicație

Pachetele se vor muta în felul următor.

2	5	4	3	1	_
_	5	4	3	1	2
1	5	4	3	_	2
1	_	4	3	5	2
1	2	4	3	5	_
1	2	_	3	5	4
1	2	3	_	5	4
1	2	3	4	5	_