Cashier Problem: a Greedy Algorithm and an optimal Solution

Prof.dr. Nicolae GIURGIȚEANU
Facultatea de Economie şi Administrarea Afacerilor, Universitatea din Craiova

We will remind briefly the cashier problem. A cashier has leeway a range of different fractional coins and has to pay a certain amount using the most reduced number of coins. If we mark the pay-desk monetary with $P = \{p_1,...,p_n\}$*, each* p_i *having as denomination* d_i *and with A the final sum, the cashier must determine a coins subset* $S = \{q_1,...,q_m\}$ *of P, so that* $A = \sum_{i=1}^{m} q_i d_i$*. In order to solve this problem, there are several solutions consisting in greedy algorithms. Although there is an optimal solution, in the present article we will highlight a greedy algorithm and an optimal solution for this problem. The solution known at the time being use a lot of memory and, in addition, is difficult to justify, occurring the risk of misunderstanding by the reader. Our solution is simpler and uses little memory.*
Keywords*: greedy algorithm, cashier problem, optimal solution*

Punerea problemei

Problema casierului constă în următoarele: Să presupunem că într-o casierie (un bancomat etc.) se găseşte o anumită sumă de bani. O persoană oarecare se prezintă la ghişeu şi solicită un anumit rest. Se pune problema să se restituie acest rest folosind un număr cât mai mic de monede.

O astfel de problemă se încadrează foarte bine în categoria problemelor de optimizare. Dar, se ştie că, în general, aceste probleme admit mai multe soluții şi, din acestea, se alege soluția optimă. Cu alte cuvinte, în aceste probleme, se cere găsirea unei soluții care realizează un minim sau un maxim al unei funcții, numită *funcție obiectiv*.

Un algoritm greedy este o tehnică de găsire a unei soluții optime pentru o anumită problemă. El rezolvă problema prin căutarea soluției optime folosind optime locale, adică el va căuta soluția pentru o subproblemă mai simplă, înaintând din aproape în aproape, către soluția problemei inițiale. Aspectul lacom, (greedy), al algoritmului apare prin faptul că el alege întotdeauna soluția optimă, pentru o subproblemă, încercând, ca în final, să obțină soluția optimă a întregii probleme, fără să se mai uite înapoi. Algoritmul greedy nu garantează obținerea soluției optime a problemei, el garantează doar faptul că aceasta are o soluție greedy.

Dacă algoritmul greedy se dovedeşte a fi corect, atunci timpul său de execuție creşte liniar cu dimensiunea problemei.

Un astfel de algoritm construieşte soluția problemei în mod iterativ. La fiecare iterație algoritmul foloseşte o regulă greedy, pentru efectuarea alegerii elementului dintr-o listă de elemente prioritare. Odată efectuată o alegere a unui element, acesta niciodată nu va mai fi luat în considerare pentru obținerea unei eventuale alte soluții mai bune şi acesta este motivul pentru care algoritmul se numeşte greedy. În general, această regulă greedy poate fi exprimată prin intermediul noțiunii de candidat posibil pentru construirea soluției. În acest fel, fiecare iterație elimină următorul, cel mai bun candidat din listă şi verifică dacă soluția permite alegerea unui alt candidat în vederea construirii soluției problemei. Algoritmul se termină doar atunci când lista de posibili candidați este vidă.

În cele ce urmează vom rezolva problema casierului printr-un algoritm care seamănă cu algoritmii greedy, dar foloseşte şi alte tehnici decât cele greedy. Algoritmul implementat în Java este prezentat în Figura 1, iar justificarea sa se va face ulterior.

Programul Java pentru rezolvarea problemei

```
//RestBancomat.java
import java.io.*;
public class RestBancomat{
    static int n,k,i,j,tot=0;
    static int c[][],r[][],d[];
    public static void main(String arg[]){
       int [] d={1,3,6,8,9,12};
       int m=d.length;
System.out.println("Dati numarul a carui descompunere o vreti ");
              n=readInt();
       int minC=0,lin=0,minR=0,minc=0,minr=0,kh;
       String h="  ";
       int c[][]=new int[m][n+1];
       int r[][]=new int[m][n+1];
       constMatrici(c,r,d,n,m);
       System.out.println("Pentru scrierea lui "+n+" se folosesc ");
       do{
              minR=r[m-1][n];
              minC=c[m-1][n];
              lin=m-1;
              k=m-1;
              while(c[k][n]==0)
                     k--;
              if(k==0){
                     lin=0;
                     minR=0;
                     minC=c[0][n];
              }
              else{
                     lin=k;
                     minR=r[k][n];
                     minC=c[k][n];
                     minc=c[k-1][n];
                     minr=r[k-1][n];
       if(!ExistadMmR(minc,minr,d,m,minC,minR)){
                            lin=k-1;
                            minR=minr;
                            minC=minc;
                     }
              }
       System.out.println(h+minC+((minC>1)?" monezi de tipul ":" moneda de tipul
")+d[lin]);
                     tot+=minC;
                     n=minR;
                     h="+ ";
       }while(n!=0);
       System.out.println("Un total de "+tot+" monezi");
    }
static boolean ExistadMmR(int c, int r,int d[],int m, int C,int R){
              int minR,minr;
              minR=R/d[0];
              for(int j=1;j<m;j++){
              if(R>=d[j])
                     if(minR>=(R/d[j]))
                            minR=R/d[j];
       }
       minr=r/d[0];
       for(int j=1;j<m;j++){
              if(r>=d[j])
                     if(minr>=(r/d[j]))
                            minr=r/d[j];
       }
       if(minR+C<=c+minr)
              return (true);
       else
              return (false);
    }
```

```
    static void constMatrici(int c[][],int r[][],int[] d,int n,int m){
       for(i=0;i<m;i++)
              for(j=0;j<n+1;j++){
                     c[i][j]=0;
                     r[i][j]=0;
              }
       for(i=0;i<m;i++){
              for(j=1;j<n+1;j++){
                     c[i][j]=j/d[i];
                     if(j>d[i])
                            r[i][j]=j-c[i][j]*d[i];
              }
       }
    }
    public static String citLinie( ) {
              BufferedReader stdin = new BufferedReader(
              new InputStreamReader(System.in));
              String str="";
              try {
                     str = stdin.readLine();
              }
              catch( IOException e) {
                     System.out.println("I/O error");
              }
              return str;
       }
       public static int readInt() {
              String str = citLinie() ;
              int i =0;
              try {
                     i = Integer.parseInt(str) ;
              }
              catch(NumberFormatException e) {
                     System.out.println("Format ilegal") ;
              }
              return i;
       }
}
```

Justificarea algoritmului

Ideea de bază din acest algoritm vine din programarea dinamică şi anume, vom creea două tablouri bidimensionale: unul cu elemente *c(i,j), j* de la *1* la *n* şi i de la *1* la *m*, ce vor da numărul maxim de monede necesare constituirii sumei *j*, folosind doar denominarea d_i, iar cel de-al doilea, cu elemente *r(i,j)*, ce ne vor furniza restul care rămâne neacoperit din suma *j* cu monede de tipul d_i. #n aces fel *c(2,5)* va conţine numărul maxim de monede d_2 necesar constituirii lui 5, iar *r(2,5)* va conţine restul neacoperit din suma *j* cu monede de tip d_2.

Acum, problema este foarte similară celei a coeficienţilor binomiali. Pentru aceasta avem nevoie de nişte valori iniţiale şi de o regulă prin care să calculăm coeficienţii *c(i,j)* şi r(i,j).

Este evident că valorile iniţiale sunt *0*, iar din descrierea făcută elementelor *c(i,j)* şi *r(i,j)* rezultă imediat un mod direct de calcul şi anume:

$c(i,j)=j \text{ div } d_i$,

$r(i,j)=j-c(i,j)*d_i$.

După construirea celor două tablouri, conform regulilor de mai sus, se stabilesc sumele minime, din fiecare tip de monedă d_i, pentru acoperirea lui *n*. Să observăm, din regulile de calcul ale coeficienţilor celor două tablouri, că numărul minim de monede necesar acoperirii lui *n* este dat de suma elementelor *c(i,j)* care ni-l dă pe *n*.

Să observăm că pentru un număr *j* fixat, $c(i,j) \geq c(i+1,j)$, oricare ar fi *i* de la *1* la *m-1*. În ceea ce priveşte resturile *r(i,j),* nu mai putem spune acelaşi lucru, sunt posibile oricare din situaţiile: $r(i,j) < r(i+1,j)$, $r(i,j) = r(i+1,j)$, $r(i,j) > r(i+1,j)$.

Pentru stabilirea numărului minim de monede care acoperă pe *n*, un număr natural dat, se

procedează astfel:
Se determină cel mai mic cât, *c* şi acesta se determină conform situațiilor 1 şi 2 de mai jos:
(1) dacă $r(i+1,n) \le r(i,n)$, ținând seama că $c(i+1,n) \le c(i,n)$, se alege minimul ca fiind $c=c(i+1,n)$;
(2) dacă $r(i+1,n)>r(i,n)$, atunci se verifică dacă există posibilitatea ca
a)$r(i+1,n)$ să fie acoperit cu un număr mai mic de monede decât $r(i,n)$;
b)$r(i,n)$ să fie acoperit cu un număr mai mic de monede decât $r(i+1,n)$.
Dacă se întâmplă cazul a), atunci se consideră minimul ca fiind $c=c(i+1,n)$, altfel se consideră minimul ca fiind $c=c(i,n)$.
După aceasta, în locul numărului inițial *n* se consideră restul care a contribuit la stabilirea lui c. Mai exact în primul caz $n = r(i+1,n)$, iar în cel de-al doilea caz

$$n = \begin{cases} r(i+1,n), & \text{subcaz a} \\ r(i,n), & \text{subcaz b} \end{cases}.$$

După care lucrurile se reiau cu stabilirea lui *c*. Acest lucru continuă până se epuizează toate posibilitățile, adică până cînd *n* devine *0*.
Observații. 1. Sigur că algoritmul se termină în timp, deoarece la fiecare pas se ia în considerare un rest care în ultimă instanță va ajunge 0.
2. Algoritmul este clar că este un algoritm greedy, el merge foarte bine pe unele cazuri, pentru altele însă nu furnizează soluția optimă. De exemplu n=48 şi sistemul de denominări 1, 16, 22, 39.

Soluție optimă
Cu toate că algoritmul de mai sus este unul de tip greedy, totuşi realizarea lui ne-a facilitat obținerea unui algoritm care să obțină soluția optimă întotdeauna. Acest nou algoritm determină toate soluțiile posibile, care nu sunt mai multe decât numărul tipurilor de monede. În final, din această mulțime se alege soluția cea mai bună, adică cea care acoperă suma cerută cu un număr minim de tipuri de monede.
Ideea de bază este următoarea:
1. se determină cea mai mare denominare, d[j], mai mică decât n, suma dată;
2. se determină numărul de astfel de monede care acoperă suma n şi în acest sens se foloseşte formula minC=n d[j].
3. Se procedează asemănător pentru restul acestei împărțiri minR.
Noul algoritm obținut, numai partea distinctă a sa, este prezentat mai jos.

```
   public static void main(String arg[]){
       int [] d={1,16,22,39};
       int m=d.length;
       boolean sol=false;
              System.out.println("Dati numarul a carui descompunere o vreti ");
              n=readInt();
       int minC=0,lin=0,minR=0,minc=0,minr=0,s1,l=0;
              s1=Integer.MAX_VALUE;
       String h="  ";
       System.out.println("Pentru scrierea lui "+n+" se folosesc ");
              for(j=m-1;j>0;j--){
                     if(n>=d[j]){
                            minC=n/d[j];
                            minR=n-minC*d[j];
                            CautaSolutie(minC,minR,m,d,sol,h);
                            if(s<s1){
                                   s1=s;
                                   l=j;
                            }
                     }
              }
              minC=n/d[l];
System.out.println(h+minC+((minC>1)?" monede de tipul ":" moneda de tipul ")+d[l]);
              minR=n-minC*d[l];
              h="+ ";
              sol=true;
              CautaSolutie(minC,minR,m,d,sol,h);
              System.out.println("Un total de "+s+" monede");
```

```
    }
public static void CautaSolutie(int minC,int minR,int m,int d[],boolean sol, String
h){
             int minc=0;
             s=minC;
             while(minR!=0){
                    for(i=m-1;i>=0;i--){
                           if(minR>=d[i]){
                                  minc=minR/d[i];
                                  minR-=minc*d[i];
                                  s+=minc;
                                  if(sol)
System.out.println(h+minc+((minc>1)?" monede de tipul ":" moneda de tipul ")+d[i]);
                           }
                    }
             }
      }
```

Bibliografie

Giurgițeanu Nicolae – Analiza şi Sinteza algoritmilor, Universitaria, 2006, 234 pag.