SUMA – descrierea soluţiei

Soluţia I

Soluţia propusă utilizează metoda programării dinamice. Este implementat un algoritm de expandare tip Lee realizat cu o coadă alocată dinamic.
Astfel, fiecare cifră contribuie la expandarea soluţiilor precedente care au şansă de dezvoltare ulterioară. Vectorul **best** memorează la fiecare moment suma cea mai mare formată dintr-un număr dat de termeni.
Condiţia **nc-nr<=(n-p^.t-1)*3+2** (unde **nc** este numărul total de cifre care se distribuie, **nr** este numărul de ordine al cifrei curente, **n** este numărul total de termeni şi **p^.t** este numărul de termeni ai soluţiei curente) testează ca, prin crearea unui nou termen cu ajutorul cifrei curente, să mai existe şansa construirii cu cifrele rămase a unei soluţii cu **n** termeni.
Condiţia **nc-nr>=n-p^.t** testează ca, prin lipirea cifrei curente la ultimul termen al soluţiei curente, să mai existe şansa construirii cu cifrele rămase a unei soluţii cu **n** termeni.

```
type pnod=^nod;
     nod=record
       s:longint;   {suma}
       t,last:word; {nr. de termeni şi ultimul termen}
       next:pnod
     end;
var n,nc,i,k:longint; f:text;
    best:array[1..1000]of longint;
    p,u:pnod;
    c:char;
procedure citire; {determină numărul total de cifre}
var i,x:longint;
begin
   assign(f,'suma.in');reset(f);
   readln(f,n);
   for i:=1 to n do begin
     read(f,x);
     repeat inc(nc);x:=x div 10 until x=0
   end;
   close(f)
end;
procedure calc(nr:longint;cif:byte); {expandarea corespuny[toare cifrei curente}
var c,q:pnod; gata:boolean;
begin
  c:=u;gata:=false;
  repeat
    if (cif>0) and (nc-nr<=(n-p^.t-1)*3+2) and (best[p^.t]=p^.s) then begin
            new(u^.next);u:=u^.next;
            u^.s:=p^.s+cif;
            u^.t:=p^.t+1;
            u^.last:=cif
         end;
    if (p^.last<100)and(nc-nr>=n-p^.t) then begin
            new(u^.next);u:=u^.next;
            u^.s:=p^.s+p^.last*9+cif;
            u^.t:=p^.t;
            u^.last:=p^.last*10+cif;
         end;
    if p=c then gata:=true;
```

```
    q:=p;p:=p^.next;dispose(q)
  until gata;
end;
procedure optim; {recalcularea valorilor maxime memorate în vectorul best}
var i:longint;
    q:pnod; gata:boolean;
begin
     for i:=1 to n do best[i]:=0;
     q:=p;gata:=false;
     repeat
           if q^.s>best[q^.t] then best[q^.t]:=q^.s;
           if q=u then gata:=true;
           q:=q^.next
     until gata;
end;
begin
   citire;
   {reluarea citirii cifrelor, ignorând spaţiile}
   reset(f);readln(f);
   repeat read(f,c) until c<>' ';
   new(p);
   p^.s:=ord(c)-48;p^.t:=1;
   p^.last:=p^.s;
   best[1]:=p^.s;
   u:=p;
   for i:=2 to nc do begin
      repeat read(f,c) until c<>' ';
      calc(i,ord(c)-48);
      optim
   end;
   close(f);assign(f,'suma.out');rewrite(f);writeln(f,best[n]);close(f)
end.
```

Soluţia II

Problema se rezolvă prin metoda programare dinamică. Concatenam numerele şi obţinem un şir (să îl notăm a) de cifre (de lungime L maxim N*3).

Pentru fiecare poziţie p (p = 1..L) calculăm prima maximă pe care o putem obţine despărţind subşirul de până la p inclusiv în n sume (n = 1..N):

Obţinem relaţia:

smax[p][n] = min (

smax[p-1][n–1] + a[p], // punând ultima sumă formată doar din cifra a[i]

smax[p-2][n-1] + a[p–1]*10 + a[p], //punând ultima sumă din 2 cifre

smax[p-3][n-1]+a[p-2]*100+a[p-1]*10+a[p] // punând ultima sumă din 3 cifre

)

Trebuie avute în vedere cazurile limită când p = 1, p = 2 sau p = 3 şi cazurile în care a[p], a[p -1] sau a[p -2] sunt zero, moment în care nu putem forma o suma de lungimea respectivă, aşa ca excludem termenii din expresia de minim.

Pentru uşurinţa în implementare stocăm smax[p][n] = –infinit pentru cazul în care subşirul de până la p nu poate fi împărţit în mod corect în n sume, iar observând că recurenţa depinde doar de ultimele 3 linii, nu păstrăm decât pe acestea şi linia curentă pentru a nu avea probleme cu memoria.

Obţinem memorie O(N) şi timp de execuţie O(L*N) = O(N^2);