anticip – Soluţie

Problema se rezolvă prin programare dinamică.

Observăm că o adunare se împarte în mai multe blocuri continue (în funcţie de unii sumatori care anticipează transportul) care se efectuează în paralel (adică fiecare bloc începe adunarea în acelaşi timp). Timpul de execuţie al unei adunări este deci maximul dintre lungimile blocurilor care se formează.

Construim următoarea matrice:
`A[i][j][k]` = câte posibilităţi de a aduna primii `i` biţi, cu timpul maxim `j` (adică lungimea maximă a unui bloc) şi ultimul bloc de lungime `k` (bloc care poate continua), pentru `k>=1`
`A[i][j][0]` = câte posibilităţi de a aduna primii `i` biţi, cu timpul maxim `j` (adică lungimea maximă a unui bloc) şi ultimul bloc se termină după sumatorul `i`

Atunci când la bitul `i+1` se vor aduna biţii 0+1 sau 1+0, deoarece aceştia nu pot fi anticipaţi nu vor crea un nou bloc şi se va actualiza valoarea `A[i+1][max(j,k+1)][k+1]`.

Atunci când la bitul `i+1` se vor aduna biţii 0+0 sau 1+1, deoarece aceştia vor fi anticipaţi, se va crea un nou bloc după ei şi se va actualiza valoarea `A[i+1][max(j,k+1)][0]`.

Iniţializăm `A[1][1][1] = A[1][1][0] = 2` (2 dintre posibilităţi formează un bloc care se termină, pentru că sumatorul nu poate fi anticipat, iar celelalte 2 formează un bloc care poate continua).
Deci recurenţa este următoarea:
```
A[i+1][max(j,k+1)][k+1] += 2*A[i][j][k]
A[i+1][max(j,k+1)][0] += 2*A[i][j][k]
```

Răspunsul cerut se va afla astfel:
`R += j*(A[N][j][0] + A[N][j][1] + … + A[N][j][N]), 1<=j<=N.`

Se observă ca pentru a afla valorile pentru sumatorul `i+1` ne sunt necesare doar valorile pentru sumatorul `i`, deci putem lucra doar cu ultimele două “linii” din matrice, reducând astfel memoria.

Bineînţeles că vor trebui folosite numere mari, întrucât pentru `N=50`, rezultatul are aproximativ 40 cifre în baza 10.