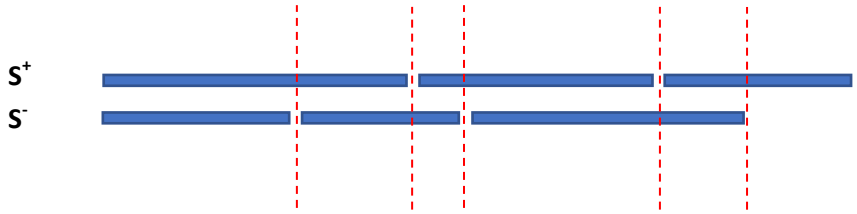
Descriere soluție – bețe

Propunător: Mihail-Cosmin Pit-Rada

Două bețe de lungimi **a**, respectiv **b**, cu **a ≠ b**, vor fi înlocuite cu un băț de lungime **a - b** sau **-a + b**, după cum **a ≥ b** sau **a < b**. Bazându-ne pe această observație, se poate deduce ușor că forma rezultatului final va fi **± L[1] ± L[2] ± … ± L[N]**. Grupând mai departe termenii pozitivi și cei negativi în două sume S^+ și S^-, rezultatul va avea forma $S^+ - S^- \geq 0$. Practic avem o partiție a valorilor **L[]** în două submulțimi.

Un alt aspect este că și invers, pornind de la o partiție cu două submulțimi, există o "planificare" de împerechere a bețelor astfel încât să se obțină la final un set de bețe de lungime totală $S^+ - S^-$ (vom accepta că pot rămane mai multe bețe, însă una dintre mulțimi va fi epuizată, nemaiputându-se face împerecheri). Vizual, ne putem convinge destul de ușor că există tăieturi verticale convenabile (conform figurii de mai jos), indiferent în ce ordine am înșirui bețele în cadrul mulțimii din care fac parte.



Putem concluziona că operațiunea de împerechere/tăiere a bețelor este echivalentă cu partiționarea lor în două mulțimi disjuncte. Așadar ne interesează identificarea unei partiții ce minimizează $S^+ - S^-$. Mai avem că $S^+ + S^- = S$ (suma tuturor lungimilor inițiale). Deducem că $S^+ - S^- = S - 2*S^- \geq 0$, deci va trebui să maximizăm S^-, echivalent cu a determina o submulțime a cărei sumă este cât mai apropiată de **S/2**.

Abordarea 1:

Determinarea celei mai apropiate sume de **S/2** se poate face cu un algoritm de tip **Lee** în manieră clasică, folosind un vector în care se marchează din aproape în aproape sumele ce pot fi atinse folosind primele **k** bețe. Acest algoritm ar avea complexitate $O(S \cdot N) = O(N^2 \cdot MAX\_VALUE)$. Vom căuta o implementare mai eficientă.

Abordarea 2:

Întrucât valorile **L[i]** sunt mici, este de așteptat să avem duplicate. Ar fi de preferat să reducem duplicatele. De exemplu, dacă avem valoarea **x** cu o multiplicitate de 7, putem obține doar 8 sume: **0·x**, **1·x**, **2·x**, … **7·x.** Aceleași sume le putem obține și din valorile **1·x**, **2·x**, **4·x**, fiecare cu ordinul de mutiplicitate 1. Astfel, vom putea înlocui cele 7 valori **x,** cu cele 3 valori de mai sus. Acest procedeu se poate generaliza ușor, **f** valori **x** fiind înlocuite de aproximativ **log(f)** valori, folosind de exemplu puteri ale lui 2 și eventual un rest. Aplicând această tehnică, reducem semnificativ numărul elementelor și se poate folosi în continuare abordarea 1.

Abordarea 3:

Ideea este să folosim simultan toate valorile egale și să nu le tratăm individual ca valori distincte. Vom folosi următoare definiție:

```
dp[t][k] = câte valori egale cu k mai am la dispoziție, după ce formez suma t,
           din valori mai mici sau egale cu k
```

Prin convenție, valorile strict negative înseamnă că suma **t** nu poate fi atinsă. Se deduce ușor:

```
dp[t][k] = frecvența inițială a valorii k, dacă dp[t][k - 1] ≥ 0
dp[t][k] = dp[t - k][k] - 1
dp[x][0] = (x ≠ 0) ? -1 : 0
```

Recurența de față se poate implementa cu un efort de calcul proporțional cu $\frac{S}{2} \cdot NUM\_DISTINCT\_VALUES$, echivalent cu $O(N \cdot MAX\_VALUE^2)$.

Abordarea 4:

Se poate folosi un vector caracteristic **v[]** pentru sumele atinse. Inițial **v[0] = 1**, apoi pentru fiecare valoare distinctă **a[i]** se parcurg descrescător sumele și pentru fiecare sumă atinsă **j** se marchează, spre dreapta, toate sumele nemarcate încă **j + a[i] * k**, dacă **1<=k<=freq[a[i]]** și **j + a[i] * k<= S/2**. Important este ca să se oprească marcarea atunci când se întâlnește o sumă deja marcată. Astfel se obține complexitatea $O(N \cdot MAX\_VALUE^2)$.