Gourmet 1: Râme pane

Subtask 1

Dacă $M = 2$ atunci fie dăm cele mai mari două râme celor doi clienți, fie tăiem cea mai mare râmă in două. Vom alege cel mai bun caz dintre cele două.

Subtask 2

Subtask-ul acesta are aceeași soluție ca subtask-ul 3.

Subtask 3

Vrem să rezolvăm următoarea problemă ajutătoare: pentru un X număr natural fixat, vrem să știm dacă este posibil să împărțim râmele astfel încât să avem în final satisfacția X.

Răspunsul este afirmativ în cazul în care $\sum_{i=1}^{i\ N}[r_i/X] \geq M$.

Putem începe cu $X = 0$. La fiecare pas vom verifica dacă putem obține satisfacția $X + 1$, iar în caz afirmativ incrementăm valoarea lui X cu 1. Răspunsul va fi ultima valoare pentru care $\sum_{i=1}^{i\ N}[r_i/X] \geq M$. Complexitatea finală este de $O(VALMAX \cdot N)$.

Subtask 4

Observăm că soluția anterioară face o căutare secvențială a celei mai bune valori X. Putem înlocui aceasta cu o căutare binară a valorii X pe intervalul $[0, VALMAX]$. Complexitate finală: $O(log(VALMAX) \cdot N)$.

Gourmet 2: Prăjitură cu mujdei

Subtask 1

Vom încerca sa umplem vasul A si sa il varsam in B, până ramane un litru in A. Se poate demonstra ca este posibil daca si numai daca B este impar.

Subtask 2

Concurenții pot encoda manual operatiile.

Subtask 3

Calculam $dp(i, j)$ = daca putem (si cum putem) obtine exact i litrii n A si j litrii in B. Putem calcula matricea dp folosind algoritmi precum BFS, DFS sau Lee, plecand de la $dp(0, 0)$.

Reconstituim solutia plecand de la $dp(X, i)$ sau $dp(i, X)$.

Subtask 4

Fie $d = Gcd(A, B)$. Este clar ca daca X nu se divide cu d, atunci nu exista solutie. Fie $X' = \frac{X}{d}$.

Din algoritmul lui Euclid extins, exista doua constante a și b a.î.:

$$a * A + b * B = d$$

Amplificam ecuatia cu X':

$$(a * X') * A + (b * X') * B = X$$

Reducem ecuatia modulo B:

$$(a * X') * A \equiv X (mod\ B)$$

Asadar, este suficient sa umplem vasul A de $a * X'\ mod\ B$ ori, si sa il varsam in B. Cand B este plin, il golim.

Observatii:

- Daca $A > B$, il putem inversa pe A cu B.
- Exista si alte solutii, bazate pe DFS, care functioneaza. Aceastea efectueaza operatiile descrise mai sus, deci sunt corecte.

Gourmet 3: Răcituri cu pireu

Subtask 1

Știm că avem doar răcituri, iar reducerea nu se aplică niciodată. Fiecare răcitură poate fi aleasă cu o probabilitate de 1/2, deci valoarea medie a prețului plătit va fi $1/2 \cdot \sum_{i=0}^{i<N} r_i$.

Subtask 2

Ne interesează să calculăm $dp(i)$ = numărul de moduri în care putem obține suma i.

În final răspunsul va fi $(\sum_{i=0}^{i \leq X} dp(i) + 1/2 \cdot \sum_{i=X+1}^{S} dp(i))/2^R$, unde S este suma maximă posibilă.

Pentru acest subtask putem calcula $dp(i)$ folosind metoda backtracking în complexitate $O(2^R)$.

Subtask 3

În cazul în care avem doar răcituri putem aplica o dinamică asemănătoare cu problema rucsacului.

Definim $DP(i,j)$ = numărul de moduri de a obține suma i folosind primele j răcituri.

Astfel $DP(i,j) = DP(i,j-1) + DP(i-r_j, j-1)$.

Considerăm $DP(i-r_j, j-1) = 0$ dacă $i > r_j$.

Desigur, $dp(i) = \sum_{j=0}^{j<R} DP(i,j)$.

Complexitate finală: $O(R \cdot S)$.

Subtask 4

Pentru fiecare tip de pireu p și pentru fiecare valoare i vom adăuga la $dp(i)$ suma:

$\sum_{k=1}^{k \leq p} dp(i-k)$.

De asemena, numitorul fracției devine $2^R \cdot \prod_{i=0}^{i<P} (p_i + 1)$.

Pentru acest subtask putem calcula dinamica în mod naiv în complexitate $O(R \cdot S + R \cdot P \cdot MP)$, unde MP este valoarea maximă a cantităților de pireu.

Subtask 5

Pentru ultimul subtask folosim aceeași dinamică, dar ne folosim de sume parțiale pentru a actualiza tabloul dp cu cantitățile necesare de pireu. Astfel complexitatea devine $O(R \cdot (S + P))$.

Gourmet 4: Cornișoni afumați

Subtask-urile 1 și 2

Aceste subtask-uri reprezintă cazuri particulare care se pot rezolva ușor pe foaie.

Subtask 3

Putem citi N și apoi să generăm un program care sortează elementele permutării printr-o metodă similară cu bubble-sort, fără să fie necesar să folosim loop-uri în limbajul dat. Dacă am avea un ”for” care execută o secvență de cod de N ori, atunci pur și simplu înșiruim acea secvență de N ori în programul dat.

Această soluție produce un program care sortează permutarea în $O(N^2)$.

Subtask 4

Putem implementa un algoritm în $O(N^2)$ care să sorteze permutarea. Un exemplu ar fi următorul algoritm care folosește bubble-sort:

```
0. IF_DIFF_GOTO A N 2
1. END
2. INC A
3. ASSIGN B Z
4. ASSIGN C B
5. INC C
6. IF_SAME_GOTO C N 0
7. ASSIGN D B
8. ASSIGN E C
9. PLOAD D
10. PLOAD E
11. IF_LESS_GOTO D E 13
12. PSWAP B C
13. INC B
14. IF_DIFF_GOTO Z N 4
```

Programul generat va fi același indiferent de valoare lui N, deci numărul de instrucțiuni generat va fi mereu mic. Complexitatea acestuia este $O(N^2)$.

Subtask 5

Acest subtask este destinat implementării unui algoritm de sortare în $O(N \cdot logN)$. Comisia nu are un astfel de algoritm, cu toate acestea am decis să introducem acest subtask pentru a recompensa creativitatea concurenților.

Subtask 6

Deoarece vectorul ce trebuie sortat este o permutare, sortarea poate avea lor în $O(N)$.

Să luăm ca exemplu un index oarecare x. Acesta se află într-un ciclu de lungime L. Să presupunem pentru moment că $L \geq 3$.

Fie $y = V[x]$ și $z = V[y] = V[V[x]]$.

Dacă interschimbăm pe $V[x]$ cu $V[y]$ folosind instrucțiunea PSWAP atunci la poziția x vom avea valoarea $V[y]$, iar la poziția y vom avea valoarea $V[x]$.

Cu alte cuvinte obținem $V[x] = V[y] = z$ și $V[y] = V[x] = y$. A doua relație ne spune că în urma aceste operații am reușit să fixăm un punct al permutării. Practic am redus lungimea ciclului în care se află x cu 1.

Dacă x se află într-un ciclu de lungime 2 atunci operația de mai sus fixează atât pe x cât și pe $V[x]$.

Prin urmare vom itera cu x prin toate elementele de la 0 la $N-1$, iar atâta timp cât $x \neq V[x]$ vom folosi PSWAP pentru a interschimba pe $V[x]$ cu $V[V[x]]$.

Algoritmul sortează fiecare ciclu de lungime L al permutării în $O(L)$, iar precum suma lungimilor ciclilor este egală cu N, complexitatea totală a algoritmului va fi $O(N)$.

O posibilă implementare este următoarea:

```
0. IF_SAME_GOTO N A 10
```

```
1. ASSIGN B A
2. PLOAD B
3. ASSIGN C B
4. PLOAD C
5. IF_SAME_GOTO A B 8
6. PSWAP A B
7. IF_SAME_GOTO Z Z 1
8. INC A
9. IF_SAME_GOTO Z Z 0
10. END
```

Aceasta implementeaza urmatorul algoritm:

```
A := 0
while A != N:
    while True:
        B = V[A]
        C = V[B]
        if (B == C):
            break
        swap(V[A], V[B])
    A += 1
```