Descrierea soluțiilor, InfoPro Runda 1, Grupa A

1 Problema Balans

Propunator: Bogdan Ciobanu, Hudson River Trading

1.1 Subtask 1

In acest subtask nu avem operatia de eliminare de la inceputul sirului si este de ajuns sa se aplice un algoritm clasic de verificare a unei parantezari cu precizia ca sirul va ramane permanent parantezat gresit daca intalnim un ")" fara pereche (in timp ce un "(" mai are sanse sa-si gaseasca perechea in operatiile ce urmeaza).

1.2 Subtask 2

In acest subtsak este necesara mentinerea structurii de coada. In cazul simplu de fata putem aloca un sir cu atatea elemente cate operatii ne asteptam sa avem in cel mai rau caz si apoi sa lucram cu doi indici, unul care sa semnifice inceputul sirului si altul care sa semnifice finalul. Cand trebuie sa verificam starea sirului este de ajuns sa iteram elementele aflate intre acesti doi indici si sa aplicam solutia de la subtaskul precedent.

1.3 Subtask 3

Se poate folosi o structura de date pentru a interoga mai rapid intervalul delimitat de cei doi indici mentionati anterior. Daca mentinem la fiecare pozitie din sir valoarea $N1 - N2$, unde $N1$ este numarul de paranteze deschise pana acum, iar $N2$ este numarul de paranteze inchise pana acum, atunci este de ajuns sa vedem daca $N1 = N2$ la capat, iar pe parcurs nu avem niciodata ca $N1 < N2$.

1.4 Subtask 4

Putem adapta destul de usor algoritmul de la primul subtask intelegand mai bine implicatiile unei eliminari la inceputul sirului. Vom retine T ca fiind un numar auxiliar care inseamna ca va trebui sa asteptam macar pana vom elimina elementul cu indicele T ca sa avem vreo sansa ca sirul sa fie corect parantezat, din perspectiva operatiilor de eliminare specifice cozii. Distingem urmatoarele cazuri:

• Daca adaugam un "("

Atunci vom adauga elementul la capatul sirului.

• Daca adaugam un ")"

Atunci vom incerca sa-i gasim pereche in elementele existente in sir. Daca exista, atunci vom retine pentru perechea ei indicele la care se va inchide. Conceptual asta inseamna ca indata ce eliminam acel "(", parantezarea nu va fi corecta decat dupa ce vom elimina si perechea ei, anume caracterul curent. Altfel spus, cand vom elimina acel "(", vom stii ca T este macar egal cu pozitia perechii lui. Daca nu exista o pereche posibila, atunci prefixul nostru nu mai are sanse sa se rezolve pana nu eliminam acest ")", deci vom schimba T in consecinta.

• Cand se elimina un "("

Daca a avut o pereche, atunci va trebui sa asteptam macar pana si acel ")" este sters, folosind valoarea T.

• Cand se elimina un ")"

In acest caz nu trebuie sa luam nicio masura.

In cele din urma, sirul va fi corect parantezat daca am depasit pragul T de la care stim ca parantezarea este corecta din perspectiva operatiilor de eliminare si daca toate "(" care au mai ramas in coada au o pereche acum, pentru ca nu verificam acest lucru prin valoarea T.

2 Problema Monede

Propunator: Tamio-Vesa Nakajima, Oxford University

În problemă se dau N monede, fiecare cu probabilitatea p_i să fie pajură. Se cere să se determine probabilitatea ca, pentru Q intervale $[a_i, b_i]$, monedele din interval să aibe un număr impar de pajuri atunci când sunt aruncate.

Să zicem că moneda i este reprezentată de X_i, în felul următor:

$$X_i = \begin{cases} 1, & \text{dacă moneda } i \text{ este pajură} \\ 0, & \text{altfel} \end{cases}$$

Atunci, X_i este 1 cu probabilitate p_i, și 0 altfel. Definim $f_i(x) = xp_i + (1 - x)(1 - p_i)$. Astfel $f_i(x)$ este p_i când $x = 1$ și $1 - p_i$ când $x = 0$ – deci $f_i(x)$ este probabilitatea ca X_i să ia valoarea x.

Interogarea $[a_i, b_i]$ ne cere probabilitatea ca

$$\sum_{k=a_i}^{b_i} X_k = 1 \mod 2$$

Vom discuta acum soluțiile parțiale, și cea completă.

2.1 Soluție in $O(2^N Q)$

Pentru a rezolva problema în $O(2^N Q)$, vom rezolva fiecare interogare independent, în $O(2^N)$. Pentru a rezolva o interogare $[a, b]$, vom itera prin toate configurațiile monedelor din $[a, b]$, și vom aduna probabilitățile acelor configurații pentru care este un număr impar de pajuri. Mai exact:

```
1: r = 0.
2: for x_a ← 0,1 do
3:     for x_{a+1} ← 0,1 do
4:         ...                                   ▷ Mai multe for-uri
5:         for x_b ← 0,1 do
6:             if x_a + ... + x_b = 1 mod 2 then
7:                 r = r + f_a(x_a) ... f_B(x_b)
8:             end if
9:         end for
10:        ...                                   ▷ Mai multe end for-uri
11:    end for
12: end for
13: return r
```

2.2 Soluție în $O(N + Q \log N)$

(Această soluție este mai tehnica ca soluția oficiala, necesitând arbori de intervale. Soluția finală nu necesită această structură de date, dar necesită formula ce o deducem mai jos.)

Observăm că dacă știm ca A și B sunt numere aleatoare, fie 0 sau 1, iar ele au probabilitatea p_A (respectiv p_B) să fie 1, atunci numărul $A + B \mod 2$ este tot un număr aleator, fie 0 sau 1, iar probabilitatea lui să fie 1 este dată de expresia

$$p_A(1 - p_B) + (1 - p_A)p_B$$

Acest fapt rezultă din faptul că suma $A+B \mod 2$ este 1 dacă și numai dacă $A = 1, B = 0$ (care se întâmplă cu probabilitate $p_A(1 - p_B)$), sau $A = 0, B = 1$ (care se întâmplă cu probabilitate $(1 - p_A)p_B$).

Astfel, putem utiliza un arbore de intervale pentru a rezolva problema. Într-un interval din arbore, calculăm probabilitatea ca suma, modulo 2, a variabilelor X_i din interval este 1. Combinăm două intervale cu ajutorul formulei de înainte.

2.3 Soluție în $O(N + Q)$ când $p_i \neq \frac{1}{2}$

Să ne uităm mai in amănunt la formula de înainte:

$$p_A(1 - p_B) + (1 - p_A)p_B$$

Să definim $p_A \oplus p_B$ ca fiind egal cu expresia anterioară. Cum $p_A \oplus p_B$ este probabilitatea ca $A+B \mod 2$ să fie 1, inductiv putem vedea că probabilitatea

ca $X_a + \ldots X_b \mod 2$ să fie 1 este egala cu $p_a \oplus \ldots \oplus p_b$. Observăm următoarea identitate:

$$\begin{aligned} x \oplus y &= x(1-y) + (1-x)y \\ &= \frac{1-(1-2x)(1-2y)}{2} \end{aligned}$$

Astfel, definim funcția $f : \mathbb{R} \to \mathbb{R}$ prin $f(x) = 1 - 2x$, și observăm că inversa sa satisface $f^{-1}(x) = \frac{1-x}{2}$. Atunci devine clar că

$$x \oplus y = f^{-1}(f(x)f(y))$$

Se poate arăta inductiv (exercițiu!) că

$$p_a \oplus \ldots \oplus p_b = f^{-1}(f(p_a) \ldots f(p_b))$$

Astfel, putem calcula, la inițializare, în $O(N)$, un șir r_i, ce reprezintă un fel de "produs parțial", unde $r_i = f(p_1) \ldots f(p_i)$. Acesta se calculează cu recurența

$$\begin{aligned} r_0 &= f(p_0) \\ r_k &= r_{k-1} f(p_k) \qquad (k > 0) \end{aligned}$$

Observație: cum $p_i \neq 1/2$, rezultă că $f(p_i) \neq 0$, și deci $r_i \neq 0$. Astfel, convenind că $r_{-1} = 1$, putem deduce că

$$\begin{aligned} f(p_a) \ldots f(p_b) &= \frac{f(p_0) \ldots f(p_b)}{f(p_0) \ldots f(p_{a-1})} \\ &= \frac{r_b}{r_{a-1}} \end{aligned}$$

(Acest raport este bine definit doar pentru că $r_i \neq 0$). Asta implică că rezultatul pentru $[a, b]$, adică $p_a \oplus \ldots \oplus p_b$, este dat de

$$p_a \oplus \ldots \oplus p_b = f^{-1}\left(\frac{r_b}{r_{a-1}}\right)$$

Lucru care se poate calcula ușor.

2.4 Soluție în $O(N + Q)$

Observație: dacă în intervalul $[a, b]$ apare cel puțin o monedă cu probabilitate $1/2$, atunci rezultatul pentru acel interval este $1/2$. Pentru a vedea de ce, observăm că putem schimba ordinea în care sunt aruncate monedele fară a schimba probabilitatea ca numărul final de pajuri să fie impar (respectiv par). Astfel considerăm că ultima monedă aruncată este cea cu probabilitatea $1/2$.

În cazul acesta, indiferent de paritatea numărului de pajuri de până atunci, e echiprobabil ca ultima monedă să mențină acea paritate, sau nu (adica să fie cap sau pajură respectiv). Din asta rezultă că observăm un număr impar de pajuri cu probabilitate 1/2.

Astfel, putem împărții problema în mai multe bucăți, în funcție de monedele cu probabilitate 1/2. În interiorul unei secvențe care conține doar monede cu probabilitate diferită de 1/2, vom răspunde ca în soluția din secțiunea precedentă. Altfel, răspunsul este mereu 1/2.

3 Problema Semafor

Propunator: Andrei Constantinescu, Oxford University

Problema se poate reformula ca o problema de grafuri:

Se da un graf **orientat** $G(V, E)$ format din N noduri si M muchii. Fiecare muchie este colorata fie cu **rosu**, fie cu **albastru** (asa cum sugereaza si enuntul). Un *ciclu de lungime* k este o succesiune de noduri din V de forma $n_0, n_1, \ldots, n_k = n_0$, legate intre ele prin muchii $m_0, m_1, \ldots, m_{k-1} \in E$, astfel incat oricare ar fi $0 \leq i < k$ avem ca muchia m_i uneste nodurile n_i si $n_{(i+1)\%k}$. Spunem ca un astfel de ciclu este *alternant* daca k este par si oricare ar fi $0 \leq i < k$ avem ca m_i si $m_{(i+1)\%k}$ au colori diferite. Sa se spuna pentru fiecare nod din V daca exista un ciclu alternant ce il contine.

Incercati sa va ganditi de ce conditiile de echilibrat si nemonoton se traduc in cele de mai sus!

3.1 Subtask 1 (15 puncte)

Limitele $N \leq 8$ si $M \leq 22$ sunt foarte mici, asa ca un algoritm de tip backtracking poate rezolva problema: Se itereaza $v_0 \in V$ si o culoare $c_0 \in \{rosu, albastru\}$ si se incearca sa se gaseasca un ciclu alternant ce pleaca din v_0 si al carei prime muchii are culoarea c_0. Acest lucru se poate face cu o functie recursiva:

```
BACKTRACKING(v, c, v_0, c_0, pasi) intoarce bool
  Daca pasi = 2*N
    return false
  Pentru fiecare muchie e = (v, v') de culoare c' != c
    Daca v' = v_0 si c' != c_0
      return true
    Daca BACKTRACKING(v', c', v_0, c_0, pasi + 1) = true
      return true
  return false
```

Functia se apeleaza initial cu tuplul de valori $(v_0, c_0, v_0, c_0, 0)$. Observatia fundamentala pe care am folosit-o aici este ca daca avem un ciclu alternant C format din **strict** mai mult de $2N$ noduri, atunci se va intampla la un moment dat daca am parcurge ciclul nod cu nod sa ajungem de doua ori in acelasi nod intrand in el cu aceeasi culore (**de ce?** *indiciu: exista N noduri in G, si in*

fiecare se poate intra in 2 *moduri, cate un mod pentru fiecare culoare, deci* $2N$ *modalitati in total, apoi se aplica Principiul Cutiei*), deci putem obtine un ciclu mai scurt C' plecand de la C si eliminand portiunea dintre cele doua aparitii repetate. Asadar, este suficient sa consideram doar cicluri de lungime cel mult $2N$, de unde si variabila *pasi* din codul de mai sus.

3.2 Subtask 2 (17 puncte)

Algoritmul prezentat anterior este mult prea lent pentru $N \leq 600$ si $M \leq 3000$. Totusi, diferenta este o optimizare relativ simpla, bazata pe urmatoarea observatie: Daca ignoram pe moment variabila *pasi*, presupunand ca toate apelurile functiei s-ar face cu $pasi = 0$, atunci observam ca daca functia noastra de backtracking a ajuns o data intr-un tuplu (v, c, v_0, c_0), atunci daca va ajunge ulterior intr-un alt tuplu de aceasta forma putem opri imediat cautarea, deoarece stim deja ca ea nu va gasi nicio solutie (altfel am fi intors solutia inca din primul apel si nu am fi ajuns sa il facem si pe al doilea). Codul modificat arata in felul urmator:

```
DFS(v, c, v_0, c_0) intoarce bool
  vis[v, c] = true
  Pentru fiecare muchie e = (v, v') de culoare c' != c
    Daca v' = v_0 si c' != c_0
      return true
    Daca vis[v', c'] = false
      Daca DFS(v', c', v_0, c_0) = true
        return true
  return false
```

Am folosit un tablou bidimensional *vis* de dimensiuni $N \times 2$, ce porneste initializat cu `false` pentru fiecare pereche noua (v_0, c_0). Valoarea vis$[v, c]$ este `true` daca si numai daca pentru perechea (v_0, c_0) curenta am ajuns deja in starea (v, c) cel putin o data. Deoarece se trece de un numar constant de ori prin fiecare nod si prin fiecare muchie pentru o pereche (v_0, c_0) fixate, complexitatea finala este $O(N \cdot M)$.

Daca va intrebati de ce se numeste functia `DFS`, continuati sa cititi pana la solutia de 100 de puncte.

3.3 Subtask 3 (19 puncte)

(Solutia acestui subtask este destul de tehnica, si, am putea argumenta, nenaturala. Cu toate acestea, comisia a inclus subtaskul in ideea de a obtine o mai buna diferentiere a punctajelor la varf)

Cheia subtaskului este intelegerea indeaproape a restrictiei:

> **Subtask 3 (19 puncte):** Se garantează că pentru oricare 51 de orașe distincte alese dintre cele **N**, există printre acestea două orașe **a** și **b**, cu proprietatea următoare: dacă există un traseu de la **a** la

b (indiferent de tipul de drumuri care îl compun) atunci nu exista un traseu de la **b** la **a**.

Definim o *componenta tare conexa (CTC)* a grafului ca fiind o submultime $V' \subseteq V$ de noduri astfel incat oricare ar fi doua noduri $a, b \in V'$, avem ca exista un *drum* (traseu) de la a la b (automat, si unul de b la a, *de ce?*). In plus, V' trebuie sa fie *maximal*, adica sa nu fie inclus intr-un V'' mai mare ce respecta aceleasi proprietati. Se poate demonstra ca CTC-urile lui G formeaza o *partitie* a multimii nodurilor (adica sunt disjuncte doua cate doua, iar runiunea lor este V, deci **fiecare nod face parte din fix o singura componenta tare conexa** - de ce? *indiciu: reducere la absurd*).

Date fiind acestea, restrictia de mai sus transmite ca toate componentele tare conexe ale lui G sunt de marime cel mult 50. Asadar, putem folosi unul dintre algoritmii cunoscuti pentru determinarea componentelor tare conexe in complexitate $O(N + M)$, cum ar fi Algoritmul lui Kosaraju sau Algoritmul lui Tarjan (vezi si problema de pe Infoarena). O data cunoscute componentele tare conexe, putem aplica algoritmul din Subtaskul 2 pe fiecare componenta in parte (deoarece o data ce un traseu iese dintr-o componenta tare conexa, el nu se va mai putea niciodata intoarce in ea, deci este suficient sa tratam fiecare componenta independent).

Complexitatea totala se compune din $O(N+M)$, datorat algoritmului pentru determinarea CTC-urilor, la care se adauga un numar de operatii egal cu $\sum N_i \cdot M_i \leq 50 \sum M_i = 50M$, unde am notat cu (N_i, M_i) numarul de noduri si, respectiv, numarul de muchii interne componentei tare conexe cu numarul de ordine i. Si mai exact, daca am nota cu K marimea maxima a unei CTC (aici $K \leq 50$), atunci complexitatea totala ar fi $O(N + K \cdot M)$, suficienta pentru subtaskul de fata.

3.4 Subtask 4 (20 puncte)

(Un subtask inedit, cu o solutie in spiritul celei de 100 de puncte, dar mai simpla conceptual. A fost introdus pentru a facilita tranzitia spre solutia de 100)

Deoarece din fiecare nod iese exact o muchie albastra, atunci cand suntem in mijlocul unui traseu nu incape loc de indoiala ce muchie albastra vom folosi pentru a iesi din nodul curent, deoarece exista una singura din care putem alege, oriunde am fi. Astfel, putem simplifica graful initial G, si sa construim un nou graf G', dupa cum urmeaza: de cate ori avem o muchie $a \to b$ in G, adaugam muchia $a \to (b\%N + 1)$ in G'. Practic, aceasta noua muchie corespunde unei succesini de doua muchii din G, anume $a \to b \to (b\%N + 1)$, prima fiind rosie, iar a doua albastra. Observati ca graful G' nu mai are muchiile colorate. Acum, un ciclu alternant in G corespunde oricarui ciclu din G' (datorita observatiei anterioare), deci problema noastra se reduce la a afla pentru ce noduri din G' exista un ciclu de lungime cel putin 1 ce incepe si se termina in el. *Lasam ca tema cititorului sa demonstreze urmatoarea observatie ce practic incheie solutia pentru acest subtask*:

> Pentru un nod $v \in G'$ exista un ciclu de lungime cel putin 1 plecand din v si intorcandu-se tot in v daca si numai daca exista o muchie $v \to v$ in G' sau componenta tare conexa ce il contine pe v in G' are marime cel putin 2.

Atfel, algoritmul se rezuma la:

1. Construim graful G'
2. Calculam componentele tare conexe ale lui G', asa cum s-a explicat pentru subtaskul anterior.
3. Verificam conditia de mai sus pentru toate nodurile $v \in G'$.

Complexitatea totala este $O(N + M)$.

3.5 Subtask 5 (29 puncte)

Subtaskul (si implicit toata problema) este o variatiune a ideii de la Subtaskul 4. De data aceasta construim graful G' astfel:

- Pentru fiecare nod $v \in G$, in graful G' vom avea doua noduri v_1, v_2 in graful G'.
- Pentru fiecare muchie rosie $e = (a, b)$ din G, in G' vom avea muchia $a_1 \to b_2$ (necolorata).
- Pentru fiecare muchie albastra $e = (a, b)$ din G, in G' vom avea muchia $a_2 \to b_1$ (necolorata).

Este probabil putin mai greu de observat, dar, si de aceasta data, este adevarat un fapt similar:

> Un nod $v \in G$ este afla pe un ciclu alternant daca si numai daca fie v_1, fie v_2 se afla pe un ciclu de lungime cel putin 1 in G'.

Nu vom da o demonstratie formala a cestui fapt, dar una intuitiva este cu siguranta bine venita:

Aceasta idee este des intalnita in jargonul concursurilor de algortimica, sub numele de "dublarea nodurilor", deoarece fiecare nod din G corespunde cu doua noduri din G'. Observati cum dintr-un nod de tipul $v_1 \in G'$ pot iesi doar muchii colorate cu rosu in graful initial, iar dintr-un nod de tipul $v_2 \in G'$ pot iesi doar muchii coloarate cu albastru in graful initial, deci tipurile nodurilor de pe un drum din G' corespund tipurilor muchiilor unui drum din G, si viceversa. De asemenea, toate muchiile din G' sunt intre noduri de tipuri diferite (graful este, prin abuz de limbaj, unul *bipartit*). Aceste observatii puse cap la cap practic arata cum constructia de mai sus asociaza fiecarui traseu alternant din G un traseu in G'.

Stiind acestea, putem acum trage urmatoarea concluzie, similara cu cea de la Subtaskul 4:

> Pentru un nod $v \in G$ exista un ciclu alternant plecand din v si intorcandu-se tot in v daca cel putin una din componentele tare conexe ale lui v_1 sau v_2 in G' au marime cel putin 2.

(Observati cum nu mai e nevoie sa specificam nimic legat de muchiile de forma $v \to v$ in G', deoarece G' este bipartit)

Algoritmul final este urmatorul, si ruleaza tot in complexitate $O(N + M)$, suficient pentru 100 de puncte:

1. Construim graful G'.
2. Calculam componentele tare conexe ale lui G', asa cum s-a explicat pentru subtaskurile anterioare.
3. Verificam conditia de mai sus pentru toate nodurile $v \in G'$ (mai exact, o verificam atat pentru v_1, cat si pentru v_2).

Nota Probabil ca va veti intreba cum s-ar gandi cineva la aceasta constructie pentru G'? Ei bine, cei mai dibaci dintre voi posibil sa fi observat o legatura intre constructia lui G' si algoritmul folosit pentru rezolvarea Subtaskului 2 (functia `DFS`) - mai exact, algoritmul `DFS` (eng. Depth First Search) de acolo realizeaza parcurgerea in adancime a grafului `G'`, fara sa il construiasca explicit, tranzitiile $(v, c) \to (v', c')$ de acolo fiind, de fapt, doar muchii ale lui G' deghizate! Exact asa se poate ajunge cu gandul la solutie: plecam de la solutia ineficienta cu DFS, observam ca de fapt parcurgem un graf, observam conditia legata de componentele sale tare conexe ale grafului, iar restul vine de la sine odata cu experienta.

Nota 2: Daca va intrebati de ce se numeste problema Semafor - nu aveti voie sa stiti, secret *oltenesc*!

Echipa care a pregătit setul din probleme pentru această rundă a fost formată din:

- prof. Adrian Panaete, Colegiul "A. T. Laurian", Botosani
- prof. Zoltan Szabó, Liceul Tehnologic "Petru Maior" Reghin / ISJ Mureș Tg. Mureș
- Andrei Constantinescu, student Oxford University, Balliol College
- Bogdan Ciobanu, software engineer, Hudson River Trading
- George Chichirim, student Oxford University, Keble College
- Tamio-Vesa Nakajima, student Oxford University, University College