Lectia #1 Introducere

Problema Euler

Fie un arbore general cu radacina fixata. Arborele are N noduri numerotate de la 1 la N. O parcurgere euler a acestui arbore se face astfel: se tipareste radacina arborelui curent iar pentru fiecare din fii radacinii se afiseaza parcurgerea euler a subarborelui respectiv dupa care se afiseaza si radacina. De exemplu, pentru arborele cu 7 noduri, cu radacina in nodul 5 si cu lista de muchii (5 3) , (5 7), (3 6), (3 1), (3 2), (7 4), parcurgerea euler este 5 3 6 3 1 3 2 3 5 7 4 7 5.

Cerinta: Problema cere ca, dandu-se un numar N si o succesiune de numere, sa se spuna daca succesiunea reprezinta o parcurgere euler corecta a unui arbore cu N noduri si, in caz afirmativ, sa se reconstituie arborele.

Observatii:

- Intr-o parcurgere Euler corecta, pentru elementele de pe pozitii consecutive trebuie sa existe o relatie de tata-fiu / fiu-tata. Relatia de tata-fiu apare cand ajungem prima oara in nodul “fiu”, iar relatia de fiu-tata apare dupa ce a fost parcurs intreg subarborele fiului si urca inapoi in tata.
- Lungimea unei parcurgei Euler corecta este de 2n - 1.
- O parcurgere Euler corecta incepe si se termina cu nodul radacina.
- In mod evident, intr-o parcurgere Euler corecta toate nodurile apar cel putin o data.
- Cele 4 observatii sunt suficiente pentru a stabili daca sirul dat este o parcurgere Euler corecta sau nu.

Solutie:

Mentinem un vector t[], cu semnificatia: t[i] = tatal nodului i. Parcurgem sirul de numere, daca gasim 2 noduri consecutive pentru care t[a[i]] != a[i - 1] && t[a[i - 1]] != a[i], stim ca nu este o parcurgere corecta. Verificam si celelalte 3 conditii. Daca sunt indeplinite, afisam vectorul t[].

Problema Asmax

Se considera un arbore (graf neorientat, conex si aciclic) cu N varfuri, in care fiecare varf i are asociata o valoarea intreaga Vi. Se defineste un subarbore al arborelui dat, ca fiind un subgraf conex nevid al acestuia (care poate coincide chiar cu arborele dat). Se cere sa determinati un subarbore al unui arbore dat, pentru care suma valorilor asociate varfurilor continute in subarbore sa fie maxima.

Sample input:

11

-1 3 -2 -4 6 -1 -2 7 2 3 -6

4 1

1 3

1 2

4 5

4 6

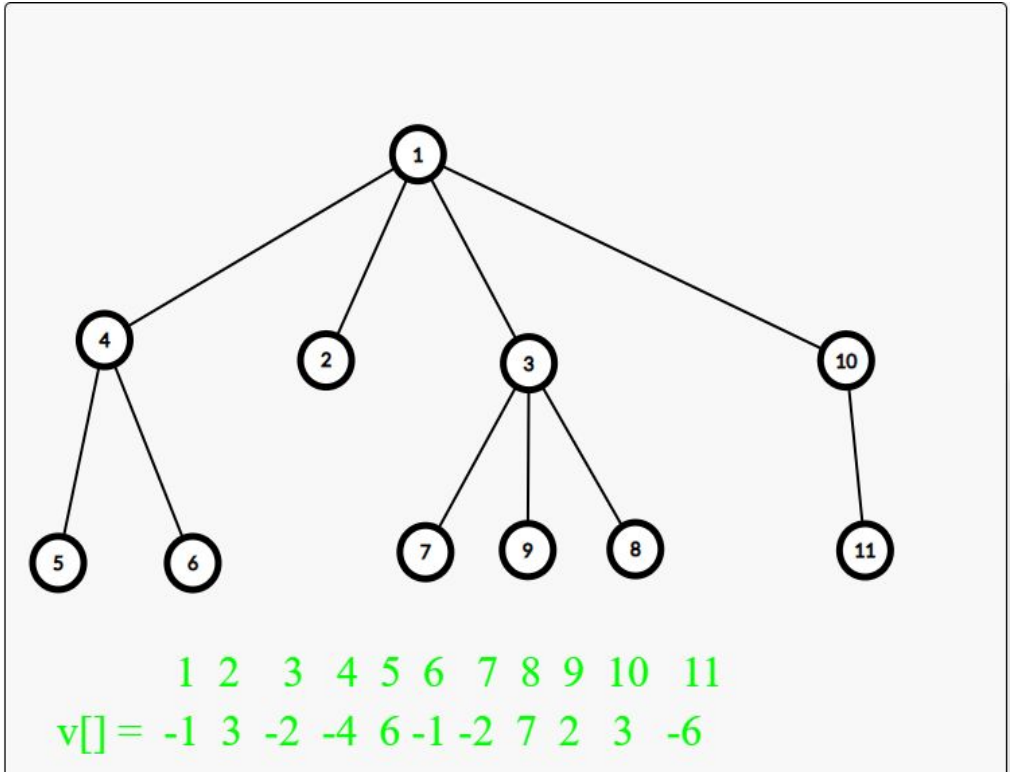
3 7

3 8

3 9

1 10

10 11



Solutie:

Parcurgem arborele si mentinem vectorul val[] cu semnificatia:

val[i] = suma maxima care se poate obtine din subarborele cu radacina in i.

Formula pentru val[nod]:

val[nod] = suma(val[fiu], cu proprietatea ca val[fiu] > 0) + v[nod] (valoarea asociata lui nod).

Raspunsul problemei: max(val[i], 1 <= i <= n).

Complexitate: O(n).

Problema TwinPerms

Georgian şi Ştefan, căutând în 3 camere din casă, au găsit două permutări gemene, `p` şi `q`, de mărime `N`. Atunci când cineva interschimba elementele de pe poziţiile `i` şi `j` dintr-o permutare, în paralel se interschimbau şi elementele de pe aceleaşi poziţii în cealaltă permutare. De exemplu, dacă cele două permutări sunt: `p = [2, 1, 3]` şi `q = [3, 2, 1]`, atunci prin interschimbarea elementelor de pe poziţiile `1` şi `3`, atunci permutările devin `p = [3, 1, 2]` şi `q = [1, 2, 3]`. Ei pot aplica oricâte interschimbări de acest fel.

Şi Georgian şi Ştefan vor să îşi minimizeze numărul de inversiuni al permutărilor fiecăruia, dar pentru că de fiecare dată când cineva schimbă ceva la o permutare, se schimbă şi în cealaltă, atunci ei s-au decis să facă astfel încât numărul de inversiuni în total din ambele permutări să fie cât mai mic.

Cerinta: Dându-se două permutări de mărime `N`, calculaţi suma minimă a numărului de inversiuni din cele două permutări după ce se aplică operaţia de interschimbare descrisa mai sus de oricâte ori.

Observatii:

Fie perechile de $(x, y) \in p$ si $(z, t) \in q$ cu $x < y$ si $z < t$ si $p[i_x] = x$, $q[i_x] = z$ si $p[i_y] = y$, $q[i_y] = t$.

- Daca $x < z$ si $y < t$, atunci perechile vor genera:
  - 2 inversiuni daca $i_x > i_y$.
  - 0 inversiuni daca $i_x < i_y$.
- Daca $x < z$ si $y > t$, atunci perechile vor genera:
  - 1 inversiune daca $i_x > i_y$.
  - 1 inversiune daca $i_x < i_y$.

Observatia 1:

Observam ca perechile din cazul al doilea vor genera o inversiune indiferent de ordinea indicilor.

Observatia 2:

De asemenea, observam ca putem elimina toate inversiunile pentru perechile care se incadreaza in primul caz.

Astfel, problema se reduce la eliminarea tuturor inversiunilor pentru perechile din primul caz si numararea inversiunilor ramase.

Solutie:

Sortam una dintre permutari, astfel eliminam toate perechile care se incadreaza in cazul 1. Gasim numarul de inversiuni ramase in cea de-a doua permutare.

Deoarece N <= 10^5, numararea tuturor inversiunior in $O(n^2)$ operatii nu este suficient de rapida. Astfel, vom folosi metode care gasesc numarul inversiunilor in complexitate O(n * log(n)), precum: MergeSort, Aib, Aint.

Problema Grigo

Fie o permutare P cu n elemente (cu indexare de la 1). Consideram o pozitie i in cadrul lui P ca fiind vizibila daca $P_i > P_j$ pentru orice $j < i$ (adica P_i e maximul dintre primele i elemente). Se dau n si m, urmati de un sir de m indici distincti intre 1 si n. Sa se calculeze cate permutari de n elemente exista astfel incat singurele lor pozitii vizibile sa fie cele date de sirul de lungime m.

Observatii:

Sortam sirul de indecsi (si il notam cu p).

1. Prima pozitie din permutare va fi intotdeauna vizibila, prin urmare, daca in sirul dat de indecsi nu exista valoarea 1, nu exista solutii (raspunsul e 0).
2. Consideram de acum ca exista valoarea 1 in sir. Prin urmare, putem descompune permutarea in felul urmator:

   $a_{p1}\ [b_{1,1}\ b_{1,2}\ \dots\ b_{1,t1}]\ a_{p2}\ [b_{2,1}\ b_{2,2}\ \dots\ b_{2,t2}]\ \dots\ \dots\ a_{pm}\ [b_{m,1}\ b_{m,2}\ \dots\ b_{m,tm}]$,

   unde a_{p1}, a_{p2} … a_{pm} sunt valorile ce se regasesc la indecsii din sirul dat

3. Pentru a satisface conditia din cerinta, trebuie ca $a_{pi} > a_j$ si $a_{pi} > b_{j,\_}$, pentru orice $j < p_i$, dar si $a_{pi} > b_{i,k}$, pentru orice $k <= t_i$.
4. $a_{pm} = n$, deoarece, din conditia de la 3, e mai mare decat toate celelalte elemente din permutare. Prin urmare, elementele din permutare de dupa a_{pm} ($[b_{m,1}\ b_{m,2}\ \dots\ b_{m,tm}]$) pot avea orice valori si orice ordine, deci pot fi calculate prin aranjamente: $A(n - 1, t_m)$.
5. Dupa ce am ales ultimele $t_m + 1$ elemente, ramanem cu o submultime a multimii de numere pe care o aveam. Avand in vedere ca ne intereseaza doar ordinea lor, nu si valorile propriu-zise, putem reaplica rationamentul, alegand pentru a_i cea mai mare valoare din submultimea actuala.

Solutie:

Pentru fiecare index din sirul dat vedem cate elemente sunt pana la indexul urmator (sau pana la finalul sirului), calculam t_i = A(index_actual + cate_pana_la_urmator - 1, cate_pana_la_urmator) si calculam produsul tuturor t_i.

Problema Zaharel

Avem o matrice de n x n (n <= 1000). Se dau m puncte din aceasta matrice. Fiecare punct poate fi fie rosu, fie albastru. Stim ca pe fiecare linie se gaseste cel putin un punct rosu, iar pe fiecare coloana se gaseste cel putin un punct albastru. Sa se determine 2 poligoane cu numar egal de varfuri (>= 2), unul dintre ele format doar din puncte rosii, iar celalalt doar din puncte albastre, astfel incat sa aiba acelasi centru de greutate. (poligoanele pot fi oricat de degenerate)

Observatii:

1. Avand in vedere ca poligoanele au acelasi numar de noduri, conditia ce tine de centrul de greutate poate fi formulata altfel: suma coordonatelor x ale varfurilor primului poligon este egala cu suma coordonatelor x ale varfurilor celui de-al doilea poligon (la fel si pentru y).
2. Problema poate fi privita, in acest caz, ca o problema de tip rucsac 2D (cand alegem un punct rosu adunam coordonatele, cand alegem un punct albastru scadem coordonatele; vrem sa ajungem la (0, 0)). Din considerente de memorie si timp, problema nu poate fi rezolvata aplicand rucsac direct (coordonatele sunt pana in 1000, sunt 10^5 puncte => aprox. $(10^3 * 10^5)^2$ operatii).
3. Pana acum nu am luat in calcul aparitia punctelor rosii pe fiecare linie (si cea a punctelor albastre pe fiecare coloane).
4. O idee care ar putea aparea, pe baza observatiei 3), e urmatoarea: pentru un punct de o anumita culoare ales deja, sa incercam sa egalam pentru culoarea cealalta coordonata pentru care are apare pe fiecare linie / coloana (ex.: daca alegem un punct rosu, sa alegem un punct albastru cu acelasi y (si stim sigur ca exista, din cerinta); daca alegem un punct albastru, sa alegem dupa un punct rosu cu acelasi x).
5. Daca repetam procesul mentionat anterior, vom ajunge la un moment dat la un punct pe care l-am mai vizitat (pentru ca, din orice nod, avem posibilitatea sa ajungem la un nod urmator, iar numarul de puncte e finit). Secventa de puncte parcurse care incepe de la punctul vizitat de 2 ori (de la prima lui vizitare) si se termina cu punctul care il viziteaza pe acesta 2-a oara e solutie a problemei.

Demonstratie:

Notam punctele rosii cu r, iar cele albastre cu a.

Atunci secventa parcursa arata astfel (presupunem ca incepem cu un nod rosu):

$r_1\ a_1\ r_2\ a_2\ r_3\ a_3 \dots \mid r_i\ a_i \dots r_n\ a_n\ (r_i)$ sau $r_1\ a_1\ r_2\ a_2\ r_3\ a_3 \dots r_i \mid a_i \dots a_{n-1}\ r_n\ (a_i)$

Daca pastram doar ciclul din secventa, atunci:

1. Un nod rosu urmat de un nod albastru vor contribui cu aceleasi valori la componentele y ale sumelor (intrucat asa le-am ales).
2. Analog, un nod albastru urmat de un nod rosu vor contribui cu aceleasi valori la componentele x ale sumelor (din acelasi motiv).

Prin urmare, sumele ambelor componente vor fi egale.

Problema Sub

Se dau doua liste de siruri de caractere. Sa se calculeze cate siruri de caractere apar drept subsiruri pentru cel putin un sir din prima lista, dar pentru niciun sir din a doua lista.

Observatii:

1. Fie A multimea subsirurilor sirurilor din prima lista, iar B multimea subsirurilor sirurilor din a doua lista. Problema ne cere practic sa calculam |A - B|.
2. Problema se reduce la a stoca in mod eficient multimea subsirurilor si a putea face update-uri pe aceasta (adaugam subsirurile din prima lista si le stergem daca apar in a doua lista).
3. O idee ar fi sa folosim hash-uri pentru a asocia subsirurilor numere / perechi / tupluri. Atunci problema se reduce la diferenta a doua multimi de lucruri care se pot compara mai usor.
4. O a doua varianta ar fi sa construim, pentru prima lista, o trie care sa salveze sufixele tuturor sirurilor. Astfel, salvam implicit toate subsirurile (multimea subsirurilor unui sir poate fi gandita si ca reuniunea multimilor prefixelor tuturor sufixelor).
5. Fiecarui nod din trie ii asociem si o valoare booleana (initial setata ca false, pentru ca sirul asociat nodului respectiv nu apare ca subsir in a doua lista, inca). Parcurgem sufixele sirurilor din a doua lista si le adaugam in trie setand bool-ul nodurilor parcurse / create la true. In final, nodurile din trie care au bool-ul setat la false reprezinta subsirurile care apar in sirurile din prima lista si nu apar in cele din a doua lista.