Descrierea Soluțiilor
Concursul Național “InfoPro”, Runda 2
Grupa A

1 Problema CntSubșirMax

Propunator: Tamio-Vesa Nakajima, University of Oxford

Problema ne cere să aflăm suma lungimilor subșirurilor maximale lexicografice a tuturor subsecvențelor unui șir de input.

Soluție în $O(N^3)$. Iterăm peste toate ($O(N^2)$ la număr) subsecvențele șirului dat în input. Pentru fiecare dintre ele, calculăm în $O(N)$ subșirul maximal lexicografic, cu ajutorul algoritmului 1.

Algorithm 1 Subșirul maxim lexicografic

Input: șirul s de lungime N

Fie st o stivă de caractere.
for $i \leftarrow 1 \dots N$ **do**
  while $empty(st)$ și $top(st) < s[i]$ **do**
    $pop(st)$
  end while
  $push(st, s[i])$
end for
return $size(st)$ ▷ Subșirul căutat este st

Soluție în $O(N^2)$. Iterăm capătul stâng al subsecvenței investigate. Apoi, putem aplica următorul algoritm pentru a calcula suma mărimilor subșirului maxim lexicografic pentru toate prefixele șirului fixat (algoritmul este asemănător cu cel din subtaskul anterior: observația cheie este ca acel algoritm calcula subșirul maxim lexicografic pentru toate sufixele șirului dat ca input pe parcurs). Pseudocodul acestui algoritm se găsește în algoritmul 2.

Algorithm 2 Suma mărimilor subșirului maxim lexicografic pentru toate prefixele

Input: șirul s de lungime N

Fie st o stivă de caractere.
Fre $r = 0$.
for $i \leftarrow 1 \dots N$ **do**
    while $empty(st)$ și $top(st) < s[i]$ **do**
        $pop(st)$
    end while
    $push(st, s[i])$
    $r \leftarrow r + size(st)$ ▷ st conține subșirul maxim lexicografic pentru $s[1 \dots i]$
end for
return r

Soluție în $O(N\Sigma)$ Parcurgem șirul de la dreapta la stânga. La fiecare poziție, considerând toate subsecvențele care încep pe acea poziție, calculăm pentru fiecare literă L două informații:

$$\begin{aligned} cnt[L] &= \text{pentru câte subsecvențe subșirul maxim lexicografic începe cu litera } L \\ lg[L] &= \text{suma lungimilor celor } cnt[L] \text{ subșiruri} \end{aligned}$$

Se observă că o poziție contribuie la soluție cu suma valorilor $lg[L]$ pentru toate literele. Pentru a gasi valorile $cnt[L]$ și $lg[L]$ pentru o poziție i putem să folosim informațiile de la poziția $i + 1$. Fiecare subșir de la $i + 1$ care începe cu literă mai mică sau egală decât L se va prelungi cu litera L. Valorile $cnt[]$ pentru litere mai mici sau egale dacât L devin 0 și se adună la $cnt[L]$ dar și la $lg[L]$ deoarece fiecare subșir devine cu o literă mai lung. Valorile $lg[]$ pentru litere mai mici sau egale dacât L devin 0 și se adună la $lg[L]$. La $lg[L]$ și $cnt[L]$ se mai adună o unitate (pentru subsecvența de lungime 1). La literele mai mari decât L valorile din $lg[]$ și $cnt[]$ rămân la i aceleași cu cele de la $i + 1$.

Soluție în $O(N)$. Abordarea este diferită față de celelalte subtaskuri. Aici ne întrebăm: pentru câte subsecvențe apare fiecare caracter în parte ca parte din subșirul maxim lexicografic? Suma acestor valori este răspunsul cerut.

Astfel, când apare un caracter în subșirul maxim lexicografic? Condiția necesară și suficientă este ca în dreapta caracterului să nu apară un caracter mai mare strict ca el. Astfel, să calculăm valori $d[i]$, unde $d[i]$ este numărul de caractere la dreapta lui i (incluzându-l pe $s[i]$) care sunt mai mici sau egale cu $s[i]$. Numărul de subsecvențe pentru care i apare în subșirul maxim lexicografic este dat atunci de

$$\underbrace{i}_{\text{moduri de a fixa capătul stâng}} \times \underbrace{d[i]}_{\text{moduri de a fixa capătul drept}}$$

Și deci răspunsul final este $\sum_{i=1}^{N} id[i]$.

Calcularea lui $d[i]$ se realizează cu ajutorul unei stive, precum este ilustrat în algoritmul 3.

Algorithm 3 Calculul lui d

Input: șirul s de lungime N

Fie st o stivă de întregi.
$push(st, N+1)$.
Fie d un șir de N întregi.
for $i \leftarrow N \dots 1$ **do**
  while $top(st) \neq N+1$ și $s[top(st)] < s[i]$ **do**
    $pop(st)$
  end while
  $d[i] = top(st) - i$
  $push(st, s[i])$
end for
return d

2 Problema Mex2D

Propunator: Bogdan Ciobanu, Hudson River Trading

Solutie în $O(N^4)$. Pentru aceast subtask se poate, spre exemplu, itera intregul dreptunghi si marcat elementele vizitate, iar apoi sa luam consecutiv numerele naturale si sa-l gasim pe primul nemarcat. Din principiul cutiei lui Dirichlet, valoarea maxima in matricea B va fi $N_1 \cdot N_2$.

Solutie in $O(N^3 \log N)$, $O(N^3)$ sau $O(N^2 log^2 N)$. Vom adauga crescator valorile in matrice. Vom considera deodata toate pozitiile unde apare o valoare v: $p_1, p_2, \dots p_K$. Ne dorim sa gasim frontiera acestor puncte, astfel incat dreptunghiurile formate din coltul stanga sus $(0,0)$ si punctele de pe frontiera sa nu contina vreo pozitie cu valoarea v, exceptand coltul dreapta jos. Aceasta frontiera se poate determina ordonand pozitiile dupa coloana crescator, iar apoi alegand fiecare punct cand descreste linia. Avand frontiera determinata, vrem sa gasim acele pozitii deasupra ei care nu au o valoare atributa inca in matricea B, pentru ca aceasta va fi chiar v. In functie de structura de date folosita se obtin diverse complexiti de timp, de diverse punctaje, insa nu stim vreuna care sa fie indeajuns de rapida pentru ultimul subtask.

Solutie in $O(N^2)$. Stim ca $B_{i_1,i_2} \geq \max\{B_{i_1,i_2-1}, B_{i_1-1,i_2}\}$, deci putem incepand cu aceasta valoare sa incrementam pana gasim una care lipseste in dreptunghi. Minoram, $\max\{B_{i_1,i_2-1}, B_{i_1-1,i_2}\} \geq \frac{1}{2}(B_{i_1,i_2-1} + B_{i_1-1,i_2})$. Presupunem pesimist ca vom face chiar $B_{i_1,i_2} - \frac{1}{2}(B_{i_1,i_2-1} + B_{i_1-1,i_2})$ pasi. Acum, daca insumam tot efortul depus in matrice, vom obtine o expresie telescopica din care se pastreaza doar valorile de pe ultima linie si ultima coloana, intr-o anumita masura. Insa, si acestea sunt valori crescatoare, care cresc pana la $N_1 \cdot N_2$, asadar intregul efort depus este $O(N^2)$. Ca sa verificam in $O(1)$ daca o valoare exista pe dreptunghiul prefix, vom retine pentru fiecare valoarea cea mai din stanga pozitie unde a aparut, de pe liniile vizitate pana acum.

3 Problema MakeBipartite

Propunator: Andrei Constantinescu

3.1 Solutie in $O(N(N+M))$

Se itereaza nodul $v \in V$ si se verifica daca graful $G \setminus \{v\}$ este bipartit. Acest lucru se poate face dupa cum urmeaza:

Notam cu G' graful $G \setminus \{v\}$. Observam ca un graf este bipartit daca si numai daca ii putem colora nodurile cu doua culori astfel incat oricare doua noduri vecine sa fie colorate diferit (aceasta fiind si definitia data in enunt). Apoi, efectuam o parcurgere in adancime (depth first search, DFS) pentru fiecare componenta conexa a lui G', colorand alternativ cu doua culori pe nivele nodurile descoperite. De cate ori gasim o muchie de intoarcere de la un nod a la un nod b, stramos al lui a, verificam daca culorile corespund, caz in care raspunsul va fi '0' pentru nodul v. **N.B.** Este important de mentionat ca intr-o parcurgere DFS toate muchiile sunt de doar doua tipuri: muchii de arbore si muchii de intoarcere, lucru care nu este adevarat, de exemplu, pentru o parcurgere BFS.

3.2 Solutie in $O((N+M)\log^2 N)$

Incepem prin a prezenta o tehnica numita *"Paduri cu Undo"* (unde Paduri se refera la Paduri de Multimi Disjuncte.

Fie G un graf neorientat. Asupra acestuia urmarim sa suportam rapid operatii de tipul `Union`, `Find` si `Undo`, unde primele doua au sensul cunoscut, iar cea din urma anuleaza efectul ultimei operatii de tip `Union`. Daca cunoasteti deja structura standard de Paduri, atunci cel mai simplu este sa urmariti codul complet in algoritmul 4.

Observati cum algoritmul aplica optimizarea uniunii dupa rang, insa algoritmul omite optimizarea compresiei caii—acest lucru nu este arbitrar! Complexitatea unei operatii de `Find` este aceeasi ca in cazul algoritmului obisnuit de Paduri fara compresia caii, si anume $O(\log N)$ pentru operatiile de `Union` si `Find`, si $O(1)$ pentru operatia de `Undo` (**Exercitiu:** gasiti un test pe care se atinge aceasta complexitate).

Urmatorul ingredient al solutiei il reprezinta o alta structura de date, relativ cunoscuta, dar fara un nume consacrat. Aceasta suporta operatiile `AddEdge` si `IsBipartite`, cu sensul intuitiv. Si de aceasta data codul este mai usor de inteles decat orice explicatie in cuvinte, asa ca va invitam sa cititi algoritmul 5. Observam din structura codului cum acesta poate suporta implicit si o a treia operatie de `UndoLastAddEdge`, daca am folosi Paduri cu Undo in loc de cele standard, insotite de alte cateva modificari minimale, lucru ce se va dovedi util pe viitor. **Remarca** Unii dintre voi veti remarca o oarecare similaritate cu algoritmul 2-SAT. Cunoasterea algoritmului nu este necesara pentru intelegerea solutiei, insa legatura este una foarte bine fondata, si pe care va recomandam sa o aprofundati daca doriti.

Punand impreuna tot ce avem pana acum, putem efectua operatii de `AddEdge`, `IsBipartite` si `UndoLastAddEdge` asupra oricarui graf dorim, toate in complexitate maxim $O(logN)$ per operatie. Ceea ce ne-am dori cu adevarat ar fi o operatie la fel de rapida de `RemoveEdge`, cu ajutorul careia

Algorithm 4 Paduri cu Undo

Fie f si h doua siruri indexate cu nodurile lui G.
Initial, $f[v] = v$ pentru toate nodurile v ale lui G, iar h este indentic nul.
Fie stk o stiva, initial goala.

Find(v)

if $f[v] = v$ **then**
    return v
else
    return Find($f[v]$)
end if

Union(v_1, v_2):

v_1, $v2$ = Find(v_1), Find(v_2)
if $v_1 = v_2$ **then**
    return false
end if
if $h[v_1] < h[v_2]$ **then**
    Interschimba v_1 si v_2.
end if
Adauga la stk tuplul $(v_1, v_2, f[v_2], h[v_1])$.
$f[v_2] = v_1$
if $h[v_1] = h[v_2]$ **then**
    $h[v_1] = 1 + h[v_1]$
end if
return true

Undo():

Extrage tuplul $(v_1, v_2, f_{v_2}, h_{v_1})$ din varful stk.
$f[v_2] = f_{v_2}$
$h[v_1] = h_{v_1}$

Algorithm 5 Verificare graf bipartit cu inserari de muchii

Fie G^* graful G unde fiecare nod $v \in G$ a fost inlocuit cu doua noduri: v si $\overline{v}$.
De asemenea, vom considera ca avem Paduri de Multimi Disjuncte implementate pentru G^*.
Fie b o variabila booleana, initial true.

AddEdge(v_1, v_2):
1: **if** Union(v_1, $\overline{v_2}$) = true sau Union(v_2, $\overline{v_1}$) = true **then**
2: b = false
3: **end if**

IsBipartite():
1: **return** b

problema de fata s-ar rezolva foarte usor: pe rand fixam cate un nod $v \in G$, iteram muchiile incidente lui v si le eliminam din G, apelam `IsBipartite`, adaugam la loc muchiile eliminate. Complexitatea in acest caz ar fi $O(N + M \log N)$, deoarece fiecare muchie ar fi eliminata si apoi readaugata de maxim doua ori. **Din pacate, o astfel de operatie este dificil de implementat eficient, si prin urmare nu o vom lua pe aceasta cale!**

Cu toate acestea, avem in continuare la dispozitie operatia `UndoLastAddEdge`, care, desi nu la fel de generala ca `RemoveEdge`, poate fi folosita ingenios pentru a obtine un efect similar. Mai exact, vom aplica un algoritm de tip Divide et Impera, prezentat in algoritmul 6. Algoritmul poate parea straniu la prima vedere, dar se preteaza natural datorita limitarii anterior mentionate. Invariantul pe care trebuie sa il consideram este ca functia `Solve(l, r)` este responsabila pentru calcularea raspunsului pentru fiecare dintre nodurile $l, l+1, \ldots, r$, raspuns pe care il stocheaza intr-un sir global *ans*. De asemenea, in momentul cand se intra in apelul $Solve(l, r)$, structura de date va fi adaugat in graf toate nodurile **mai putin** $l, \ldots, r$, si toate muchiile dintre nodurile adaugate (astfel ca un apel `IsBipartite` la momentul curent raspunde exact la intrebarea daca $G \setminus \{l, \ldots, r\}$ este bipartit, lucru folosit explicit in cazul $l = r$ pentru a popula sirul raspuns). Algoritmul se va apela initial cu parametrii $l = 1$, $r = N$.

Acum, sa calculam complexitatea timp a algoritmului nostru: Observam ca recursivitatea are $O(\log N)$ nivele, si ca intervalele pe care este apelat `Solve` pe fiecare nivel partitioneaza $\{1, 2, \ldots, N\}$ in intervale continue (de exemplu, pe primul nivel avem doar apelul `Solve(1, N)`, pe al doilea nivel avem apelurile `Solve(1, mid)` si `Solve(mid, N)`, si asa mai departe). Astfel, fiecare muchie va fi parcursa de cel mult 2 ori per nivel, cate o data pentru fiecare interval din partitie ce o contine. Similar, fiecare nod va fi parcurs maxim o data per nivel, din intervalul apelului `Solve(l, r)` ce il contine. Asadar, numarul total de operatii pe graf este de cate $O(N + M)$ per nivel, deci $O((N + M) \log N)$ in total. Cum fiecare operatie pe graf ruleaza in $O(\log N)$ datorita Padurilor cu Undo, complexitatea totala finala este $O((N + M) \log^2 N)$. **N.b. Daca am folosi si euristica compresiei caii pentru implementarea Padurilor, atunci complexitatea nu ar mai fi in mod evident $O(\log N)$ per operatie—*de ce?***

3.3 Solutie in $O(N + M)$

Desi solutia anterioara nu ar fi primit 100 *de puncte in concurs, datorita complexitatii timp prea ridicate, noi o consideram deosebit de importanta de inteles si internalizat, poate chiar mai importanta decat solutia ce urmeaza! Motivul este ca solutia anterioara prezinta tehnici generale ce*

Algorithm 6 Dynamic Connectivity Divide et Impera

Solve(l, r):

1: **if** $l = r$ **then**
2:     ans[l] = IsBipartite()   ▷ Raspunsurile sunt stocate in $ans[1], \ldots, ans[N]$.
3:     **return**
4: **end if**
5: Fie $mid = \lfloor \frac{l+r}{2} \rfloor$
6: Pentru fiecare muchie e cu un capat intr-unul din nodurile $l, \ldots, mid$ si celalalt in afara intervalului $[mid + 1, r]$, apeleam AddEdge(e).
7: Solve(l, mid)
8: Apeleam operatia Undo de atatea ori cate muchii au fost adaugate la pasul 6. Practic, prin aceasta eliminam muchiile adaugate la pasul respectiv.
9: Pentru fiecare muchie e cu un capat intr-unul din nodurile $mid + 1, \ldots, r$ si celalalt in afara intervalului $[l, mid]$, apeleam AddEdge(e).
10: Solve($mid + 1$, r)
11: Apeleam operatia Undo de atatea ori cate muchii au fost adaugate la pasul 9.

se aplica unei game largi de probleme, in timp ce prezenta studiaza atent structura particulara a problemei de fata si nu este aplicabila si in alte situatii.

In curand ...

Echipa. Setul de probleme pentru această rundă a fost pregatit de:

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