Un prim algoritm de tip "brute force"

Putem observa ca exista anumite subtask-uri (1 & cel mai probabil si 2) care pare că doresc sa fie rezolvate printr-o simulare efectiva a procesului din problema. Acest lucru este datorat faptului ca in acele subtaskuri avem suficient timp sa generam fiecare partitie a numarului N de prieteni. O prima solutie de tip simulare ar fi urmatoarea: consideram dp[mask][nr_coloane_parcurse]. mask apartine [0, 2^N-1], nr_coloane_parcurse apartine [0, M]. ca fiind numarul de partitionari a multimii totale de prieteni care dupa nr_coloane_parcurse coloane ajung sa convearga catre prietenii din "mask". (cu proprietatea extra ca toti prietenii din "mask" au primii "nr_coloane_parcurse" biti egali) Initial se porneste prin a pune pentru oricare "mask" apartine [0, 2^N-1] valoarea 1 in dp[mask][0] ca sa semnalizam faptul ca initial doar prietenii din "mask" sunt considerati. Mai apoi ca sa facem tranzitia de la dp[mask][coloana] -> dp[nou mask][coloana + 1] ceea ce trebuie sa facem este sa numaram cati prieteni "i" inclusi in "mask" sunt de parere ca obiectivul nou (de la coloana "coloana + 1") trebuie vizitat si cati considera ca acesta nu trebuie vizitat. De asemenea sa se memoreze mask1 inclus in mask, mask0 inclus in mask care considera noile masti pentru prietenii care considera ca ar trebui / nu ar trebui ca obiectivul sa fie inclus. Iar acum in functie care categorie are mai multe elemente (in caz de egalitate conform enuntului problemei va fi aleasa categoria cu "1") se va incrementa fie dp[mask0][coloana + 1], fie dp[mask1][coloana + 1] cu dp[mask][coloana]. Complexitatea algoritmului este O(M * N * 2^N) iar a memoriei este O(M * 2^N). Este important de notat faptul ca memoria poate fi optimizata la O(2^N) avand in vedere faptul ca putem memora doar ultimele doua coloane ale matricei "dp". ChatGPT Textul corectat cu adăugarea diacriticelor și îmbunătățiri minore de stil este: Un prim algoritm de tip "brute force" Putem observa că există anumite subtask-uri (1 și cel mai probabil și 2) care par că doresc să fie rezolvate printr-o simulare efectivă a procesului din problemă. Acest lucru este datorat faptului că în acele subtaskuri avem suficient timp să generăm fiecare partiție a numărului N de prieteni. O primă soluție de tip simulare ar fi următoarea: considerăm dp[mask][nr_coloane_parcurse], unde mask aparține [0, 2^N-1], iar nr_coloane_parcurse aparține [0, M], ca fiind numărul de partiționări ale mulțimii totale de prieteni care, după nr_coloane_parcurse coloane, ajung să converge către prietenii din "mask" (cu proprietatea extra că toți prietenii din "mask" au primii "nr_coloane_parcurse" biți egali). Inițial se pornește prin a pune

pentru oricare "mask" aparține [0, 2^N-1] valoarea 1 în dp[mask][0] ca să semnalizăm faptul că inițial doar prietenii din "mask" sunt considerați. Mai apoi, ca să facem tranziția de la dp[mask][coloană] -> dp[nou_mask][coloană + 1], ceea ce trebuie să facem este să numărăm câți prieteni "i" incluși în "mask" sunt de părere că obiectivul nou (de la coloana "coloană + 1") trebuie vizitat și câți consideră că acesta nu trebuie vizitat. De asemenea, să se memoreze mask1 inclus în mask, mask0 inclus în mask, care consideră noile măști pentru prietenii care consideră că ar trebui / nu ar trebui ca obiectivul să fie inclus. Iar acum, în funcție de care categorie are mai multe elemente (în caz de egalitate, conform enunțului problemei, va fi aleasă categoria cu "1"), se va incrementa fie dp[mask0][coloană + 1], fie dp[mask1][coloană + 1] cu dp[mask][coloană]. Complexitatea algoritmului este O(M * N * 2^N), iar a memoriei este O(M * 2^N). Este important de notat faptul că memoria poate fi optimizată la O(2^N), având în vedere faptul că putem memora doar ultimele două coloane ale matricei "dp".

Soluționarea problemei pentru obținerea punctajului maxim: Bineînțeles că soluția de punctaj maxim este într-o direcție complet diferită față de cea pe care am prezentat-o anterior deoarece (N <= 4000) iar algoritmul nostru anterior avea o complexitate exponențială în funcție de N. Inițial voi prezenta soluția iar mai apoi am să discut despre complexitatea ei. Vom construi un Trie cu preferințele celor N prieteni (Vedem fiecare șir de preferință al fiecărui prieten ca un număr binar de lungime M). În fiecare nod al Trie-ului vom menține de asemenea un contor "cnt" care simbolizează câți prieteni au prefixul de lungimea aferentă nodului din Trie egal cu drumul parcurs de la rădăcină către nod. De asemenea vom avea înăuntrul fiecărui nod un vector de dimensiune "cnt + 1" care să se numească "dp". Până acum haideți să analizăm complexitatea memoriei a structurii pe care am menționat-o. Este evident că aceasta este O(N * M) deoarece fiecare dintre cei N prieteni adaugă O(M) poziții în vectorii "dp" din nodurile pe care le parcurge. De asemenea, în frunzele Trie-ului vom menține un vector numit "inds" care să simbolizeze care dintre cei N prieteni are șirul său de preferințe egal cu drumul parcurs de la rădăcină către frunza curentă. Ce va simboliza dp de Trie Node [c]? Ei bine, va simboliza faptul că până acum am fixat primii "câți biți sunt de la rădăcină la nod" biți în preferința prietenilor și că dintre cei Trie node -> cnt total prieteni care corespund acestui nod noi avem o submulțime de cardinal exact "c". De notat este faptul că noi am fixat doar cardinalul mulțimii, urmând să fixăm elementele efective ale mulțimii ulterior. De aceea inițializăm Root Node [c] cu 1 și

NU cu comb(n, c). Mai apoi vom efectua un dfs pe arborele nostru Trie care va propaga dp-ul nostru din rădăcină către frunze. Să presupunem că nodul nostru curent este frunză: atunci este ușor, doar trebuie să ne plimbăm prin vectorul "inds" al acestei frunze și să incrementăm sol[element din inds] cu dp[c] * comb(cnt - 1, c - 1) deoarece noi știm că "element din inds" va fi printre cele "c" elemente ale soluției noastre, doar că așa cum am menționat, noi am fixat doar cardinalul acestei mulțimi și NU și care sunt elementele ei, de aceea va trebui să mai alegem celelalte c-1 elemente din cele cnt-1 rămase (deoarece "element din inds" este deja ales). Să presupunem că nodul curent din Trie NU este o frunză: atunci acesta are doi copii: Node0, Node1. Să presupunem că ne uităm la dp Trie Node [c], atunci va trebui să partiționăm c în c0 și c1 astfel încât c = c0 + c1 și 0 <= c0 <= Node0 -> cnt, 0 <= c1 <= Node1->cnt. Dacă c1 >= c0 atunci va trebui să adăugăm în Node1-> dp[c1] valoarea Trie Node->dp[cnt] * comb(Node0-> cnt, c0) deoarece atunci când vom "da drop" la cei c0 prieteni din soluție îi putem alege. Cei c1 prieteni rămași încă NU sunt aleși, doar cardinalul lor este. Acum: care este complexitatea acestei mulțimi: avem suma(nod aparține Trie | nod->copil0->cnt * nod->copil1->cnt) + complexitatea memoriei = suma(nod aparține Trie | nod->copil0->cnt * nod->copil1->cnt) + O(N * M) = O(N * N) + O(N * M) = O(M * (N + M)). De ce este suma(nod aparține Trie | nod->copil0->cnt * nod->copil1->cnt) = O(N * N)? Deoarece dacă ne uităm atent, fiecare pereche (i, j) de prieteni are să contribuie în maxim într-un singur nod din Trie la formula nod->copil0->cnt * nod->copil1->cnt și anume la lca-ul lor. Și cum sunt O(N * N) astfel de perechi complexitatea este O(N*N). Complexitatea finală: O(M * (N + M)) Am foarte multe comentarii utile în codurile mele deci vă încurajez cu tărie să le inspectați:

Codul meu pentru soluția brute force: https://pastebin.com/TuebbWHN

Codul meu pentru soluția de 100 de puncte: https://pastebin.com/2wqdLcJZ