AISD lista 4

Dominik Szczepaniak

April 28, 2024

1 Zadanie 1

Jednocześnie przy liczeniu DP będziemy jeszcze trzymać drugi co do wielkości element oprócz pierwszego.

Drugi oczywiście liczymy tak, że bierzemy drugi najmniejszy z poprzednich opcji i dodajemy do niego wartośc z aktualnego elementu.

Na końcu znajdziemy minimum z pierwszego i drugiego elementu, a później znowu przejdziemy przez dp z pierwszych elementów i dp z drugich elementów i weźmiemy minimum które nie jest minimum globalnym (wcześniej znalezionym).

Jeśli istnieją dwa najmniejsze wyniki to muszą mieć jakieś miejsce w tablicy w którym się kończą, więc jeśli będziemy pamiętać dwa wyniki, to niezależnie gdzie sie skończą dostaniemy oba.

2 Zadanie 2

skip

3 Zadanie 3

Podciąg (substring) to ciąg znaków które muszą występować w oryginalnej kolejności, czyli nie może być między nimi przerwy.

Podsekwencja (subsequence) to ciąg znaków które mogą mieć spacje między sobą.

Znaleźć taki najdłuższy wyraz który zawiera w sobie ciągi X i Y.

Na pewno X + Y będzie dobrym wynikiem.

Kiedy można to zmniejszać? Jeśli jakaś początkowa część X zawiera się w Y albo na odwrót.

Czyli jeśli X = "abcdggg" i Y = "ggdfgh" to możemy pozbyć sie dwóch g z Y i mieć :

"abcdgggdfgh"

W takim razie musimy sobie znaleźć jaka część Y zawiera się w X albo jaka część X zawiera się w Y. Możemy to zrobić binarnie.

Jeśli to jest podciąg to u gory

Jeśli podsekwencja to musimy znaleźć najdłuższy wspólny podciąg (longest common subsequence). W $O(n^2)$. później po prostu możemy dodać pozostałe litery z X i Y.

Jak znaleźć LIS?

```
    int L[m + 1][n + 1];

for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        if (i == 0 || j == 0)
            L[i][j] = 0;

        else if (X[i - 1] == Y[j - 1])
            L[i][j] = L[i - 1][j - 1] + 1;

        else
            L[i][j] = max(L[i - 1][j], L[i][j - 1]);
    }
}

return L[m][n];
```

4 Zadanie 4

Niech dp[i][j] oznacza minimalny koszt na przedziale od i do j wlacznie

Wtedy wynik to dp[0][n]

Jesli wielkosc przedzialu jest rowna 1 to koszt to c[i]

Jesli wielkosc przedzialu jest rowna 2 to koszt to c[i] + c[j]

Jesli wielkosc przedzialu jest rowna 3 to koszt to c[środek] + max(dp[i][środek-1], dp[środek+1][j])

dp[i][j] - minimalny wynik na przedziale i do j

dp[i][j] = x w przedziale i<=x<=j który minimalizuje wynik(c[x] + max(dp[i][x-1], dp[x+1][j]))

5 Zadanie 5

nie zapamietujemy nic wiecej oprocz ostatniego wiersza. odwolujemy sie modulo 2

6 Zadanie 6

Podzbiór jest niezależny jeśli nie ma żadnych dwóch takich wierzchołków które są połączone krawędzią.

dla każdego wierzchołka chcemy mieć maksymalny wynik dla jego poddrzewa jeśli

a) wybralibyśmy ten wierzchołek

w takim przypadku bierzemy sume z dzieci dla przypadku jeśli byśmy ich nie wybrali.

b) nie wybralibyśmy go

w takim przypadku bierzemy sume z dzieci dla lepszego z dwóch przypadków.

```
def licz(i, p):
    if dzieci[i].size() == 0:
        dp[i][0] = w[i]
        dp[i][1] = 0
    else:
        for j in dzieci:
            licz(j, i)
        dp[i][0] = w[i] + sum(dp[j][1] for j in dzieci[i])
        dp[i][1] = max(dp[j][0], dp[j][1] for j in dzieci[i])
```

wynik będzie w max(dp[korzen][0], dp[korzen][1])

jeśli mamy znaleźć dokładnie jakie wierzchołki wybraliśmy to dodatkowo do dp trzymamy sobie binary string który nam mówi które wierzchołki wybraliśmy, jeśli wybraliśmy to 1, jeśli nie to 0. z każdym kolejnym wyborem aktualizujemy string który wzięliśmy od synów.

Dlaczego działa? Ponieważ rozpatrujemy dp w którym mamy jakieś drzewo i rozszerzamy je o jakiś nowy wierzchołek. W takim przypadku albo weźmiemy ten wierzchołek albo go nie weźmiemy. W obu przypadkach wiemy jaki będzie najlepszy wynik.

Złożoność:

O(n) - dla każdego wierzchołka wejdziemy do niego tylko raz.

7 Zadanie 8

Dzielimy jakiś zbiór na trzy części tak aby suma w każdym z nich była taka sama.

W takim razie w każdym zbiorze suma elementów będzie równa $T = S/3$.

Możemy trzymać tablice dwuwymiarową T[i][j] która mówi nam czy możemy mieć dwa zbiory o sumie i oraz j.

Na początku T[0][0] dostaje 1

Teraz musimy przejść po każdym elemencie.

Wtedy jeśli T[i][j] = 1 to T[i+a[k]][j] = 1 oraz T[i][j+a[k]] = 1

Jesli przekroczymy S/3 to nie dodajemy i ignorujemy tą wartość (ponieważ tablica jest posortowana, to później nie będzie już liczb ujemnych, więc jeśli przekroczymy wartość S/3 to się nie cofniemy)

Na końcu sprawdzamy czy T[S/3][S/3] = 1, jeśli tak to mamy odpowiedź.

Mamy najpierw pętla która idzie po wszystkich elementach, a później dwie pętle które idą po wartościach od -C*k do C*k więc ostatecznie złożoność to $O(n * 2C * n * 2C * n) = O(n^3 * 4C^2)$.

Jeśli posortowalibyśmy na początku naszą tablicę, a później nowe wartości T[][] dodawali do vectora, a później przechodzili po tych wartościach w wek-

torze, to nasza złożoność będzie wynosić $O(n*2C*n*S/3) = O(n^2*2/3C*S) = O(n^2*CS)$, co jest potencjalnie o wiele lepsze, ponieważ S jest mniejsze niż C.

8 Zadanie 9

Mamy LCS gdzie
f[i][j] = f[i-1][j-1] + 1 jeśli a[i] == b[j]
max(f[i][j-1], f[i-1][j]) jeśli a[i] != b[j]

Oraz mamy LIS gdzie:
f[i] = max(f[j] + 1, 1) j < i, a[j] < a[i]

W LCIS chcemy miec f[i][j] jako LCIS dla pierwszych i elementów z a i końcu w elemencie b[j].

Jeśli a[i] != b[j] to nie możemy wybrać a[i], bo ma się kończyć w b[j], więc b[j] musi być wybrane, czyli
f[i][j] = f[i-1][j]

Jeśli a[i] == b[j] to sprawdzamy wszystkie podciągi które kończą się z b[k] (k < j) i sprawdzamy czy możemy dodać a[i] (czyli b[j])
f[i][j] = max(f[i-1][k] + 1) dla każdego k < j, b[k] < a[i]

f[i][j] = f[i-1][j] jeśli a[i] != b[j]
max(f[i-1][k] + 1, k<j, b[k] < a[i]) jeśli a[i] == b[j]

Ponieważ możemy pamiętać największy f[i-1][k] to możemy to zrobić w złożoności $O(n^2)$.

```
for (int i = 1; i <= n; ++i)
{
    int fmax = 0, pos = 0;
    for (int j = 1; j <= m; j++)
    {
        f[i][j] = f[i - 1][j];      // Exclude a[i] from LCIS
        pre[i][j] = pre[i-1][j];
```

```
            if (a[i] == b[j])
            {    // Add a[i] to LCIS
            // fmax is the maximum value of the O(n^3) solution f[i-1][
                if (f[i][j] < fmax + 1)
                {
                    f[i][j] = fmax + 1;
                    // Prepare for the output path, record the index of
                    // previous element of the LCIS ending with b[j] am
                    // first j elements of b, which is also an element
                    // first i elements of a.
                    pre[i][j] = pos;
                }
            }
            if (b[j] < a[i])
            {
                if (f[i - 1][j] > fmax)
                {
                // fmax is the maximum value of the O(n^3) solution f[i
                    fmax = f[i - 1][j];
                    pos = j;
                }
            }
        }
    }
```

ograniczenie k nic nie zmienia

9 Zadanie 10

Proste p przecinaja proste rownolegle l' i l" w dwoch punktach. Jesli punkt przeciecia 1 jest w (a, b), a drugi w (c, d) to proste sie przecinaja jesli a $<$ c oraz b $>$ d

Każda prosta zawiera dwa punkty które symbolizują przecięcie z l' i l".
$p1 -> p(x_{p_1}, y_{p_1}) oraz k(x_{k_1}, y_{k_1})$
$p2 -> p(x_{p_2}, y_{p_2}) oraz k(x_{k_2}, y_{k_2})$

$\vdots$

Proste p_1i p_2 przecinają się gdy:

$x_{p_1} < x_{p_1} oraz x_{k_1} > x_{k_2}$
lub
$x_{p_1} > x_{p_2} oraz x_{k_1} < x_{k_2}$
W takim razie wartości y nas w ogóle nie interesują.

Wartości dokładne x też są bez znaczenia, liczy się tylko ich porządek.

Zapiszmy teraz wszystkie pary przecięć jako pary (x_{p_1}, x_{k_1})

Ponieważ posortujemy wszystkie te pary względem pierwszego przecięcia zawsze będzie zachodzić, że ta po lewej jest mniejsza od tej po prawej.

Teraz jedyne co wystarczy zrobić to policzyć najdłuższy rosnący podciąg.

Robimy taki dp, że dp[i] oznacza najmniejszy element którym kończy się podciąg długości i.

Na początku dp[0] = -inf a reszta +inf

Jak idziemy po a[i] i chcemy wstawić a[i] na miejsce d[l] to d[l-1] $<$ a[i] oraz a[i] $<$ d[l]

Możemy to zrobić dwoma pętlami w $O(n^2)$

Dwie obserwacje:

1) Tablica d jest posortowana.

2) a[i] może zupdatować co najwyżej 1 wartość w d (bo może być tylko jedno miejsce w tablicy d, gdzie d[l-1] $<$ a[i] $<$ d[l])

Więc możemy znaleźć sobie to miejsce za pomocą binsearcha w O(nlogn).

Odwarzamy trzymając drugą tablicę która mówi skąd przyszliśmy (indeks)

b) Jeśli chcemy zapytać się ile jest LIS o podanej długości to wystarczy

W dp[i] trzymamy wszystkie takie wartości które były kiedyś liczbą którą kończył się LIS o długości i+1. Te wartości są trzymane jako pierwszy element pary. Drugi element pary reprezentuje liczbę możliwych opcji dla których LIS o długości i+1 kończy się elementem większym lub równym niż pierwszy element pary. Na tych wartościach robimy sumy prefiksowe.

Jeśli chcemy wiedzieć, ile mamy możliwości zakończenia najdłuższego rosnącego podciągu (LIS) długości m wartością y, wystarczy wykonać wyszukiwanie binarne dla indeksu i w tablicy dyn[m-1], gdzie pierwsza część pary jest ściśle mniejsza niż y. Wtedy liczba opcji to:

dyn[m-1].back().second - dyn[m-1][i-1].second: Różnica między liczbą LISów długości m-1 kończących się dowolną wartością a liczbą kończącą się wartoś-

cią ściśle mniejszą niż y (przechowywaną w indeksie i-1).

dyn[m-1].back(): Przypadek, gdy i jest równe 0, co oznacza, że w dyn[m-2] nie ma elementów ściśle mniejszych niż y. W takim przypadku wszystkie możliwe wartości można dołączyć do LIS, więc używamy po prostu całkowitej liczby LISów długości m-1 (przechowywanej w dyn[m-1].back()).

Weźmy przykład:
[1, 4, 2, 5, 3]

i = 0
dp[1] = 1, 1
i = 1
binsearchem wyszukujemy miejsce 2
dp[2] = 4, 1
i = 2
binsearchem wyszukujemy miejsce 2
dp[2] = 4, 1, 2, 1
i = 3
binsearchem wyszukujemy miejsce 3
binsearchem wyszukujemy miejsce zerowe (czyli większe od każdego elementu w historii) w dp[2]
więc bierzemy sumę prefiksową z całej tablicy
dp[3] = 5, 2
i = 4
binsearchem wyszukujemy miejsce 3
dp[3] = 5, 2, 3, 1

to jaką optymilizacje robimy to to, że zamiast trzymać tylko nasze wystąpienia w drugim elemencie trzymamy wystąpienia nasze i wszystkich mniejszych.

Czyli w dp[2] mamy 4, 2, 2, 1 i wtedy jak bierzemy 5 to będziemy mieć po prostu 5, 2

Później jak weźmiemy 3, to mamy, że 3 jest większa od 2 i mniejsza od 4, więc weźmiemy wartość z 2 i dodamy wynik z 5:
dp[3] = 5, 2, 3, 2+1=3 = 5, 2, 3, 3

Wynik to dp[3].back().second()

```
int findNumberOfLIS(vector<int>& nums) {
    if (nums.empty())
        return 0;

    vector<vector<pair<int, int>>> dp(nums.size() + 1);
    int max_so_far = 0;
    for (int i = 0; i < nums.size(); ++i) {
        // bsearch insertion point
        int l = 0, r = max_so_far;
        while (l < r) {
            int mid = l + (r - l) / 2;
            if (dp[mid].back().first < nums[i]) {
                l = mid + 1;
            } else {
                r = mid;
            }
        }

        // bsearch number of options
        int options = 1;
        int row = l - 1;
        if (row >= 0) {
            int l1 = 0, r1 = dp[row].size();
            while (l1 < r1) {
                int mid = l1 + (r1 - l1) / 2;
                if (dp[row][mid].first < nums[i]) {
                    r1 = mid;
                } else {
                    l1 = mid + 1;
                }
            }

            options = dp[row].back().second;
            options -= (l1 == 0) ? 0 : dp[row][l1 - 1].second;
        }
```

```
            dp[l].push_back({nums[i], (dp[l].empty() ? options : dp[l].l
            if (l == max_so_far) {
                max_so_far++;
            }
        }

        return dp[max_so_far-1].back().second;
    }
```