AiSD L1

Maurycy Borkowski

10.03.2021

Zadanie 1

Liczba wierzchołków T

```
G[] <- drzewo
def count(v):
    subtree_cnt = 1
    if G[v].L:
        subtree_cnt += count(G[v].L)
    if G[v].R:
        subtree_cnt += count(G[v].R)
    return subtree_cnt

count(root)
```

Maksymalna odległość T

```
ans = 0

def getHeight(v):
    if v.L is None and v.R is None:
        return 0
    height_left, height_right = 0, 0
    sons = 0
    if v.L:
        height_left = getHeight(v.L)
        sons += 1
    if v.R:
        height_right = getHeight(v.R)
        sons += 1
    ans = max(ans, height_left + height_right + sons)
    return max(height_left + 1, height_right + 1)

getHeight(root)
```

W obu algorytmach przechodzimy całe drzewo raz, $\mathcal{O}(n)$.

Zadanie 3

```
def topoSort(edges, n):
    order = []
    inDegree = [0]*n
    G = []*n

    for (u,v) in edges:
        inDegree[v]+=1
        G[u].append(v)

    heap = [v if inDegree[v] == 0 for v in range(n)]

    while queue:
        u = heap.getMin()
        order.append(u)
        heap.delMin()
        for v in G[u]:
            inDegree[v] -= 1
            if inDegree[v] == 0:
                heap.add(v)
    return order
```

Dla dowolnej krawędzi (u, v) mamy $inDegree[v] = 0 \implies inDegree[u] = 0$ ponieważ, $inDegree[v] = 0$ tylko jak przeprocesujemy wszystkie wierzchołki z krawędziami wchodzącymi do v, w tym (u, v) więc $u \prec v$ w zadanym porządku.

Mając do wyboru kilka wierzchołków w kolejce wybieramy najmniejszy by otrzymać porządek pierwszy leksykograficznie.

$\mathcal{O}(n \log n)$

Zadanie 4

```
countWays = [0]*n
visited = [0]*n
distances = [0]*n
G = [[]]*n


def prep(u, v):
    countWays[v] = 1
    visited[v] = 1

    dijsktra(distances, v)
    dfs(u)

    return countWays[u]


def dfs(w):
    visited[w] = True
    for son in G[w]:
        if dist[son] < dist[w]:
            if visited[son] != true:
                dfs(son)
            countWays[w] += countWays[son]
    return
```

W preprocessingu znajdujemy najkrótsze odległości dla wszystkich wierzchołków od v algorytmem Dijkstr'y.

$countWays[w]$ oznacza liczbę sensownych dróg wychodzących z w do v
Ścieżka będzie zliczana do senswonych w $countWays[]$ tylko wtedy, gdy głębiej w poddrzewie DFS dojdzie do v.

Liczba sensownych dróg z każdego wierzchołka to suma liczby senswonych dróg jego **sensownych synów**, to znaczy takich, którzy mogą być następnikami w sensownej ścieżce, $dist[son] < dist[father]$.

$\mathcal{O}(m \log n)$

Zadanie 5

```
def findPath(n):
    orderedVertices = topoSort(G)
    distances = [0]*(n+1)
    parent = [0]*(n+1)

    for v in orderedVertices:
       for u in G[v]:
          if distances[u] < distances[v] + 1:
             distances[u] = distances[v] + 1
             parent[u] = v

    def generatePath(v):
       result = [v]
       while parent[v] != 0:
         result.append(parent[v])
         v = parent[v]

       return reversed(result)

    bestVertex = argmax(distances)

    return generatePath(bestVertex)
```

Zauważmy, że dla dowolnej drogi w G wierzchołki na tej drodze będą leżały zgodnie z porządkiem topologicznym (jest to DAG).

Dla dowolnej drogi $v_0 \dots v_k$ najpierw sprocesujemy wszystkich rodziców każdego wierzchołka v_i i będziemy znali najdłuższe drogi kończące się w nich, a biorąc wartość największą z tego (i dodając 1) otrzymamy najdłuższą ścieżkę zakończoną w v_i

By odwtorzyć ścieżkę wystarczy pamiętać przodka w najdłuższej ścieżce do danego wierzchołka.

$\mathcal{O}(n + m)$

Zadanie 6

Oznaczenie:
L - mniejsze elementy usuwane
R - większe elementy usuwane

Lemat

Istnieje rozwiązanie optymalne takie, że:

$$L = \{a_1, \ldots, a_k\}$$

$$R = \{a_{n-k+1}, \ldots, a_n\}$$

oraz elementy ciągu usuwane są parami (a_i, a_{n-k+i})

Dowód. .

I $\quad \max(L) \leqslant \min(R)$

Jeżeli istnieją $a_i \in L$ oraz $a_j \in R$ tż. $a_i > a_j$ pokażemy, że możemy je zamienić.
Oznaczmy przez a_n, a_m elementy usuwane odpowiednio z a_i i a_j. Z treści:
$2a_i \leqslant a_n$ więc $2a_j < 2a_i \leqslant a_n$ z tego (a_j, a_n) też jest poprawną parą do usunięcia.
Podobnie, $2a_m \leqslant a_j < a_i$ więc (a_m, a_i) też jest poprawną parą do usunięcia.
Możemy tak zamieniać elementy (nienaruszając rozwiązania), aż spełnimy **I**.

II $\quad L = \{a_1 \ldots a_k\}$ i $R = \{a_{n-k+1}, \ldots, a_n\}$

Załóżmy, że istnieje $a_i \notin L$ oraz $a_j \in L$ tż. $i < j$. Zauważmy, że wtedy możemy usunąć element a_i z elementem usuwanym z a_j bo $a_i < a_j$.
Analogicznie pokazujemy dla R.

III $\quad (a_i, a_{n-k+i})$

Jeżeli a_1 nie możemy usunąć z a_{n-k+1} to żadnego innego elementu nie możemy z nim usunąć a wiemy, że z kimś jest usuwany. Indukcyjnie powtarzamy. □

Z lematu wiemy, że istnieje pewne rozwiązanie optymalne postaci:

$$\underbrace{***********}_{k}\text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}\underbrace{***********}_{k}$$

Jeśli znajdziemy pierwsze $a_1, \ldots, a_k$ elementów, które możemy usunąć z jakaś parą i stwierdzimy w 100%, że dla a_{k+1} nie znajdziemy elementu do usunięcia, to jest to rozwiązanie optymalne.

```
ans = 0
j = n/2 + 1
while j <= n:
    if 2*a[i] <= a[j]:
        ans += 1
        i += 1
    j += 1
```

$\mathcal{O}(n)$

Zadanie 7

```
def solve(edges, n, V):
    D = []
    dist = [[INF]*n]*n
    for (u,v,c) in edges:
        dist[u][v] = c
        dist[v][u] = c
    for i in range(n):
        dist[i][i] = 0

    def sumuj(u):
        suma = 0
        for i in range(u+1, n):
            for j in range(i+1,n):
                suma += dist[i][j]
        return suma

    for u in range(n,1,-1):
        if u <= k:
            D.append(sumuj(u))
        for i in range(n):
            for j in range(n):
                if dist[i][j] > dist[i][V[u]] + dist[V[u]][j]:
                    dist[i][j] = dist[i][V[u]] + dist[V[u]][j]

    D.append(sumuj(0))
    return reversed(D)
```

Wykonujemy tak na prawdę Algorytm Floyda-Warshalla z tym, że w zewnętrznej pętli najpierw procesujemy wierzchołki $v_{k+1} \dots v_n$ (w dowolnej kolejności), a później dokładamy kolejno wierzchołki $v_k, v_{k-1} \dots v_1$ z ich krawędziami i patrzymy czy możemy uzsykać lepsze najkrótsze ściezki przechodząc przez nowo dodany wierzchołek.
Na bieżąco liczymy wynikowe D_i

$\mathcal{O}(n^3)$