E. Count Paths

Eric Luis López Tornas C-411

Contents

1 Definiendo el Problema **3**

2 Primera Solución **3**
- 2.1 Idea . . . 3
- 2.2 Algoritmo . . . 3
- 2.3 Lema . . . 4
- 2.4 Corolario . . . 5
- 2.5 Lema . . . 5
- 2.6 Teorema . . . 5
- 2.7 Complejidad Computacional . . . 5

3 Segunda Solución **6**
- 3.1 Idea . . . 6
- 3.2 Notación . . . 6
- 3.3 Algoritmo . . . 6
- 3.4 Teorema . . . 7
- 3.5 Teorema . . . 8
- 3.6 Complejidad temporal . . . 9
- 3.7 Optimización . . . 9

4 Anexos **11**
- 4.1 Código de la primera solución . . . 11
- 4.2 Código de la segunda solución . . . 12
- 4.3 Código optimizado de la segunda solución . . . 13

1 Definiendo el Problema

Se te da un árbol que consiste en n vértices, numerados del 1 al n. Cada vértice está coloreado con algún color, denotado por un número entero entre 1 y n.

Un camino simple en el árbol se denomina **hermoso** si:

- Consiste en al menos 2 vértices.
- El primer y el último vértice del camino tienen el mismo color.
- Ningún otro vértice en el camino tiene el mismo color que el primer vértice.

Cuenta el número de caminos simples hermosos en el árbol. Ten en cuenta que los caminos son considerados no dirigidos (es decir, el camino de x a y es el mismo que el camino de y a x).

2 Primera Solución

2.1 Idea

Por cada v en T se ejecuta un algoritmo de tipo BFS el cual calcula la cantidad de caminos hermosos donde v es uno de sus extremos. Para el momento que se termina el algoritmo se habran registrado exactamente el doble de los caminos hermosos presentes en T.

2.2 Algoritmo

Por cada s en la lista V(T), se aplica un BFS con la siguiente modificación:

- Antes de entrar al ciclo que itera sobre la cola Q, Se guardan en Q los vertices abyacentes a s y se reporta a s como explorado.
- Al sacar un vertice v de Q se verifica su v.color, si s.color = v.color, se aumenta en uno el contador de la cantidad de caminos hermosos en los que participa s y se marca a v como explorado sin guardar niguno de sus vertices abyacentes en Q. Si s.color $\neq$ v.color se guarda en Q todos los vecinos de v no estén o hayan pasado Q.

La condicción de parada de este algoritmo es la misma que la del BFS básico. En el resto del documento nos referiremos a la versión propuesta del BFS como BFS^*

2.3 Lema

Sea T un árbol y s un vertice de $V(T)$ El algoritmo $BFS^*(s)$ confecciona implicitamente un arbol abarcador de T, T', que está compuesto por todos los vertices v de T a los que se puede llegar desde s sin pasar por un vertice del mismo color de s.
Demostración:
Se demostrará que solo se puede acceder a $v \in V(T)$ por el recorrido de $BFS^*(s)$ si v puede ser alcanzado desde s sin pasar por otro vertice del mismo color que s.

La demostración se hará por inducción sobre la distancia de los vertices de T a s, la cual es finita ya que T es un árbol. Tomaremos como momento de referencia de la incorporación de un vertice v a T' el mometo que v entra a la cola Q.

Caso base d = 1:
Por la contrucción BFS^* todo v_i vecino de s se guardara en la cola Q, puesto que entre v_i y s no existe tercer vertice que este entre ellos, luego se cumple la propiedad para d=1.

Hipótesis de inducción:
Supóngase que la propiedad se cumple para todo vertice de T con $1 \leq d\ n$.

d = n:
Sea $v \in V(T)$ con distancia a s igual n. El camino de s a v tiene no un vertice interior z con el mismo color de s o tiene al menos uno:

- v puede ser alcanzado partiendo de s sin pasar por un vertice del color de s
- El camino s a v tiene un vertice interior z con el mismo color de s

De cumplirse lo primero, entonces v tiene un vecino w con un color distinto a $s.color$ a distancia n-1 de s al cual se puede llegar desde s sin pasar por otro vertice del mismo color que s. Necesariamente $w \neq s$ ya que v está a distancia al menos 2 de s. Como w tiene distancia menor que n entonces por hipótesis de inducción w es alcanzado por el recorrido $BFS(s)^*$, es decir $w \in T'$, luego en algún punto w entrado y salido de cola Q. Puesto que T es un árbol v no tiene otro vecino diferente de w con una distancia a s menor o igual que n-1, por consiguiente w va a ser el primer y único vertice que guardara a w en Q por tanto $v \in T'$.

Si en cambio existe un vertice interior z con z.color = s.color, necesariamente está a distancia a lo sumo n-1. Luego por hipótesis de inducción $BFS(s)^*$ no accedio a z, luego todo posible descendiente de z no podrá ser alcanzado por el $BFS(s)^*$, Por tanto v no es alcanzado por $BFS(s)^*$.
Luego la propiedad se cumple para todo vertice con d=n.‖

De la demostración anterior y la construcción de BFS^* obtenemos el siguiente resultado.

2.4 Corolario

Sea T un árbol y $v \in T'$ El algoritmo $BFS(s)^*$ calcula el número de caminos hermosos que tienen a s como uno de sus extremos a s.

2.5 Lema

Sea p un camino hermoso con vertices extremos v y w y sea $A(v)$ y $A(w)$ los conjuntos de caminos hermosos que tienen como extremo a v y a w respectivamente. Entonces se cumple que $\mid A(v) \cap A(w) \mid = 1$.

Demostración:
Puesto que T es un árbol solo puede existir un camino entre v y w.

Como resultado de las proposiciones anteriores, podemos asegurar la correctitud del algoritmo con el siguiente teorema:

2.6 Teorema

Sea T un árbol. Si para todo $v \in T$ se ejecuta $BFS(v)^*$ exactamente una vez, se habrán registrado el doble de los caminos hermosos de T.

Demostración:
Si p es un camino hermoso, solo puede ser registrado durante la ejecución de BFS^* en uno de sus vértices extremos. Dado que BFS^* se ejecuta una vez por cada vértice, y este recorrido solo puede registrar caminos hermosos, al finalizar el algoritmo se habrá contado a p exactamente dos veces.

2.7 Complejidad Computacional

El algoritmo ejecuta n veces el algoritmo BFS^*, el cual tiene una complejidad de $O(n)$, igual que el BFS básico. Por lo tanto, la complejidad temporal del algoritmo propuesto es $O(n^2)$.

3 Segunda Solución

3.1 Idea

Se utiliza un recorrido tipo DFS sobre T para construir una DP de abajo hacia arriba, utilizando una matriz fq de tamaño $n \times c$, donde c es la cantidad de colores presentes en T.
En el momento en que un vértice v sea completamente explorado por el recorrido DFS, se habrá actualizado el valor de $fq[v]$ y calculado la cantidad de caminos hermosos en los que participa v en el subárbol abarcador del DFS del cual v es padre. Una vez que el recorrido DFS haya finalizado, se habrán calculado todos los caminos simples hermosos presentes en T.

3.2 Notación

Debido a la frecuencia con la que utilizamos los siguientes términos, emplearemos las siguientes notaciones: T' se referirá al subárbol abarcador que forma el DFS sobre T y $T'(v)$ al subárbol de T' cuyo vértice padre v en T'.

fq: `fq`$[i][j]$ es igual a la cantidad de vértices w de color j que se encuentran en $T'(i)$ y a los cuales se puede acceder por un camino simple que comienza en i, donde el único vértice de color j es w

3.3 Algoritmo

En el momento que el DFS descubré v, se asigna 1 a la posición que le corresponde al color de v en fq$[v]$ y se itera sobre los k hijos de v en T'. Por cada uno de ellos se continua el recorrido DFS. Para el momento que haya terminado de explorar el hijo w_i. Se recorren todos los colores c en `fq`$[w_i]$ se actualizando `fq`$[v][c]$ y el total de caminos de la siguiente manera:

- Si el color de v es c al total de caminos suma de `fq`$[w_i][c]$.
- Si el color de v es diferente de c, al total de caminos se le suma `fq`$[v][c]$ * `fq`$[w_i][c]$ y posteriormente se le suma de `fq`$[w_i][c]$ a `fq`$[w_i][c]$).

3.4 Teorema

Una vez que el vértice v ha sido completamente explorado por el algoritmo DFS, el valor de $\texttt{fq}[v][c]$ estará correctamente calculado.

Demostración:
La demostración se realiza por inducción sobre la posición i del vértice v en la secuencia de vértices completamente explorados por el DFS.

Caso base $i = 1$: El vértice v_i es una hoja tanto en T' como en T; de lo contrario, no sería el primero en ser completamente explorado. Por la construcción del algoritmo, se tiene que:

$$\texttt{fq}[v_i][c] = \begin{cases} 0 & \text{si } c \neq v_i.\texttt{color}, \\ 1 & \text{si } c = v_i.\texttt{color}. \end{cases}$$

Por lo tanto, para $i = 1$, $\texttt{fq}[v_i][c]$ está correctamente calculado.

Hipótesis de inducción: Supongamos que para todo vértice con $1 \leq i < m$, $\texttt{fq}[v_i]$ está correctamente calculado.

Sea v_m el m-ésimo vértice que será completamente explorado. Por la correctitud del DFS, cada hijo w_i de v_m en $T'(v)$ ha sido completamente explorado antes de iterar por $\texttt{fq}[w_i]$. Luego por hipótesis de inducción, para el momento que se itera sobre $\texttt{fq}[w_j]$ estará correctamente calculado todo valor c.

El vértice v_m es el primer vértice de su color en cualquier camino contenido en $T'(v)$ que comience desde él. Por lo tanto, la asignación $\texttt{fq}[v_m][v_m.\texttt{color}] = 1$ cumple con la definición de $\texttt{fq}$. Para cualquier vértice z de color c distinto al de v_m, si z es alcanzable directamente desde v_m sin pasar por ningún otro vértice de color c, entonces z se encontrará en uno de los subárboles cuya raíz es un hijo de v_m en $T'(v)$. En consecuencia, z será alcanzable directamente desde el hijo w_j de v_m que es la raíz del subárbol que contiene a z. Por lo tanto z ya está registrado en $\texttt{fq}[w_j][c]$ por definición de $\texttt{fq}$. Así, el valor de $\texttt{fq}[v_m][c]$ es la suma de $\texttt{fq}[w_i][c]$ para los k hijos de v_m.

Puesto que $\texttt{fq}[v_m][c]$ está correctamente calculado para el m-ésimo vértice, por inducción, la propiedad se cumple para todos los vértices explorados en cualquier momento del DFS.

3.5 Teorema

Una vez que el vértice v ha sido completamente explorado por el algoritmo DFS, se habrán calculado correctamente todos los caminos simples hermosos que están completamente contenidos en $T'(v)$.

Demostración:
La demostración se realizará por inducción sobre la posición i que ocupa el vértice v en la secuencia de vértices completamente explorados por el DFS.

Caso base $i = 1$: El vértice v_1 es una hoja tanto en T' como en T; de no ser así, no sería el primero en ser completamente explorado. Según la construcción del algoritmo, el número de caminos que se contarán es cero, lo cual es la respuesta correcta.

Hipótesis de inducción: Supongamos que para todo vértice con $1 \leq i \leq m-1$, se han calculado correctamente todos los caminos simples hermosos que están completamente contenidos en $T'(v_i)$.

Sea v_m el m-ésimo vértice completamente explorado. Por la correctitud del DFS, si w_j es hijo de v_m, habrá sido completamente explorado para el momento de recorrer $\texttt{fq}[w_j]$ en el segundo ciclo. Luego, se cumple la hipótesis de inducción, y se habrán calculado correctamente los caminos hermosos contenidos en $T'(w_j)$. Además, por el teorema anterior, en este punto $\texttt{fq}[w_j][c]$ fue correctamente calculado.

De lo anterior, se tiene que al terminar de explorar v_m se habrán calculado al menos los caminos hermosos contenidos en $T'(v_m)$ que no contienen a v_m. Luego, falta demostrar que también se calculan aquellos en los que participa v_m y que están contenidos en $T'(v_m)$.

Para calcular todos los caminos hermosos que tienen a v_m como extremo, es necesario conocer cuántos vértices en $T'(v_m)$ tienen el mismo color x de v_m y son accesibles directamente desde v_m. Esta cantidad se puede obtener, por cada subárbol hijo $T'(w_j)$, mediante $\texttt{fq}[W_j][x]$. El total de caminos hermosos de este tipo en $T'(v_m)$ es igual a la suma de todos los $\texttt{fq}[W_j][x]$.

En cambio, para los caminos hermosos cuyos extremos no comparten el color x de v_m, necesariamente ambos extremos estarán en subárboles distintos, ya que, de no ser así, el camino ya habría sido calculado por la hipótesis de inducción. Para cada color $c \neq x$, sean $T'(v_i)$ y $T'(v_j)$ subárboles hijos de $T'(v_m)$. Por la definición de $\texttt{fq}$, cada vértice contado en $\texttt{fq}[w_i][c]$ formará un camino hermoso distinto con cada vértice contado en $\texttt{fq}[w_j][c]$. La cantidad de caminos hermosos de este tipo que se pueden formar entre $T'(v_i)$ y $T'(v_j)$ es igual a $\texttt{fq}[w_i][c] \times \texttt{fq}[w_j][c]$, por el principio de la multiplicación.

Al realizar esta operación para cada par no ordenado de hijos de v_m y sumar sus resultados, se obtiene la cantidad de caminos hermosos con extremos de color c en los que participa v_m. Por construcción, para el momento en que se analiza el color c, justo después de haberse explorado por completo el hijo w_i, en $\texttt{fq}[u][c]$ estará almacenada la suma de los $\texttt{fq}[w_j][c]$ correspondientes a los $i-1$ hijos anteriores de v_m que ya fueron explorados. Por tanto, $\texttt{fq}[v][c] \times \texttt{fq}[w_i][c]$ será igual a la cantidad de caminos hermosos con extremos de color c en los que participa v_m que se forman con un extremo en $T'(w_i)$ y el otro extremo en cualquiera de los $i-1$ árboles hijos anteriores a $T'(w_i)$. Al terminar el ciclo más externo, se habrán calculado correctamente todos los caminos hermosos en $T'(v_m)$.

3.6 Complejidad temporal

Sea m_i la cantidad de hijos del i-ésimo vertice explorado, la complejidad temporal del algoritmo es O(c($m_1 + m_2 + ... + m_n$) + n), puesto que ningún vertice en T comparte hijos $m_1 + m_2 + ... + m_n = n$. Luego la complejidad temporal del algoritmo es $O(cn)$.

3.7 Optimización

Podemos optimizar este algoritmo utilizando árboles AVL para representar fq. La idea es que, al momento en que el DFS haya terminado de procesar el vertice w_j, hijo de v, se verifique si el fq[u] que corresponde a u es menor que el que corresponde a v. De ser así, se intercambian las referencias de los mismos. Posteriormente, se recorren todos los elementos i de fq[v], es decir se itera por todos los colores presentes en $T'(v)$. Para cada i se actualiza $fq[u][i]$ adiccionando el valor$fq[v][i]$ (Puede para ese entonces en $fq[u]$ no hubiera presencia del color i), el proposito de esto es ir fusionando todos los fq de menor a mayor, explotando el hecho que por construcción, en todo paso el conjunto fq más pequeño a fusionar es disjunto al conjunto mayor.

AVL: Complejidades de las Operaciones
Sea G un árbol AVL con n vértices. Se cumple que las operaciones de inserción y búsqueda tienen una complejidad de $O(\log n)$, mientras que el recorrido de todos los elementos del árbol tiene una complejidad de $O(n)$.

Complejidad temporal
La complejidad temporal del algoritmo anterior es $O(n \cdot (log(n))^2)$.

Demostración

Podemos asumir que el costo de toda operación de inserción y búsqueda que se haga en cualquier AVL dentro del algoritmo es $O(\log n)$, ya que todos tendrán a lo sumo n vértices. Siempre que se haga la mezcla de dos arboles fq esto se pude ver como la mezcla de dos conjuntos disjuntos. Por construcción del algoritmo al mezclar dos conjuntos, se mueve el conjunto más pequeño hacia el conjunto más grande. Si el tamaño del conjunto más pequeño es m, entonces el tamaño del conjunto resultante es al menos $2m$. Por lo tanto, un elemento que ha sido movido t veces estará en un conjunto de tamaño al menos 2^t, y dado que el tamaño máximo de un conjunto es n, cada elemento será movido como máximo $log(n)$ veces. Luego por cada uno de los n vertice se hacen a lo sumo $log(n)$ insercciones de costo $log(n)$ luego el el algoritmo tiene una complejidad temporal de $O(n \cdot (log(n))^2)$.

4 Anexos

4.1 Código de la primera solución

```python
from collections import deque

def BFS(edges, c, u, visited):

    count = 0
    q = deque([])
    visited[u] = True

    for w in edges[u]:
        q.append(w)
        visited[w] = True
    while q:
        v = q.popleft()

        if c[v] == c[u]:
            count += 1
        else:
            for w in edges[v]:
                if not visited[w]:
                    visited[w] = True
                    q.append(w)
    return count

t = int(input())

for _ in range(t):
    n = int(input())
    c = list(map(int, input().split()))
    edges = [[] for _ in range(n)]

    for _ in range(n - 1):
        u, v = map(int, input().split())
        edges[u - 1].append(v - 1)
        edges[v - 1].append(u - 1)

    total_count = 0
    for u in range(n):
        visited = [False] * n
        total_count += BFS(edges, c, u, visited)

    print(total_count//2)
```

Listing 1: Código solución

4.2 Código de la segunda solución

```python
from collections import defaultdict

def dfs(u, parent, colors, adj, fq):
    global count
    for v in adj[u]:
        if v != parent:
            dfs(v, u, colors, adj, fq)
    fq[u][colors[u] - 1] = 1
    for c in range(0, len(colors)):
        for v in adj[u]:
            if v != parent:
                if c == colors[u] - 1:
                    count += fq[v][c]
                else:
                    count += fq[v][c] * fq[u][c]
                    fq[u][c] += fq[v][c]

t = int(input())
for _ in range(t):
    count = 0
    n = int(input())
    colors = list(map(int, input().split()))
    fq = [[0] * len(colors) for _ in range(n)]
    adj = defaultdict(list)
    for _ in range(n - 1):
        u, v = map(int, input().split())
        adj[u-1].append(v-1)
        adj[v-1].append(u-1)


    dfs(0, -1, colors, adj, fq)
    print(count)
```

Listing 2: Código solución

4.3 Código optimizado de la segunda solución

```python
from collections import defaultdict

def dfs(u, parent, colors, adj, fq):
    global count
    fq[u][colors[u]] = 1

    for v in adj[u]:
        if v == parent:
            continue
        dfs(v, u, colors, adj, fq)
        if len(fq[v]) > len(fq[u]):
            fq[v], fq[u] = fq[u], fq[v]  # swap de diccionarios
        for p in fq[v].items():
            count += fq[u].get(p[0], 0) * p[1]
            fq[u][p[0]] = fq[u].get(p[0], 0) + p[1]

        fq[u][colors[u]] = 1

t = int(input())
for _ in range(t):
    count = 0

    n = int(input())
    adj =  [[] for _ in range(n)]
    fq = [defaultdict(int) for _ in range(n)]
    colors  = list(map(int,input().split()))

    for _ in range(n - 1):
        u, v = map(int, input().split())
        adj[u-1].append(v-1)
        adj[v-1].append(u-1)

    dfs(0, -1, colors, adj, fq)
    print(count)
```

Listing 3: Código solución