Problem: Bicoloring (BFS) — Bicoloring problem in Onlinejudge

গ্রাফকে কি দুটি রঙে রঙ করা যায়? বাইকালারিং সমস্যার সহজ সমাধান

আমরা সবাই হয়তো "ফোর কালার ম্যাপ থিওরেম" (***Four Color Map Theorem***) -এর কথা শুনেছি। ১৯৭৬ সালে কম্পিউটারের সাহায্যে এটি প্রমাণিত হয়। এই উপপাদ্যটি বলে যে, “যেকোনো মানচিত্রকে মাত্র চারটি রঙ দিয়ে এমনভাবে রঙ করা যায় যাতে পাশাপাশি দুটি অঞ্চলের রঙ কখনো এক না হয়।”

আজ আমরা একই রকম, কিন্তু আরও সহজ একটি সমস্যা নিয়ে আলোচনা করবো। আমাদের দেখতে হবে, একটি গ্রাফকে মাত্র দুটি রঙ (যেমন: সাদা এবং কালো) দিয়ে রঙ করা সম্ভব কি না, যেন কোনো দুটি পাশাপাশি বা adjacent নোডের রঙ এক না হয়। এই সমস্যাটিকেই "গ্রাফ বাইকালারিং" (Graph Bicoloring) বা "দ্বি-রঙিনকরণ" বলা হয়।

বাইকালারিং এবং বাইপার্টাইট গ্রাফ **(Bicoloring & Bipartite Graphs):**

একটি গ্রাফকে যদি দুটি রঙে রঙ করা যায়, তবে সেই গ্রাফটিকে আমরা "বাইপার্টাইট গ্রাফ" (Bipartite Graph) বা "দ্বিবিভক্ত গ্রাফ" বলি।

Bipartite Graph কী? সহজ কথায়, একটি গ্রাফ Bipartite হবে যদি আমরা এর সমস্ত নোড বা ভার্টেক্সকে (vertex) দুটি আলাদা দলে (ধরা যাক, "দল ক" এবং "দল খ") ভাগ করতে পারি, যেন: ১. প্রতিটি এজ (edge) দ্বারা "দল ক"-এর একটি নোডকে "দল খ"-এর একটি নোডের সাথে যুক্ত করে। ২. "দল ক"-এর কোনো দুটি নোড নিজেদের মধ্যে সংযুক্ত থাকবে না। ৩. "দল খ"-এরও কোনো দুটি নোড নিজেদের মধ্যে সংযুক্ত থাকবে না।

যদি আমরা "দল ক"-এর সব নোডকে সাদা রঙ এবং "দল খ"-এর সব নোডকে কালো রঙ দিই, তবেই আমাদের বাইকালারিংয়ের শর্ত পূরণ হয়ে যায়!

কখন গ্রাফ বাইকালারিং সম্ভব নয়**?**

একটি গ্রাফকে কখন দুটি রঙে রাঙানো সম্ভব হয় না, তার একটি খুব সহজ এবং গুরুত্বপূর্ণ শর্ত আছে।

শর্ত**:** যদি কোনো গ্রাফে **"**বিজোড় দৈর্ঘ্যের চক্র**" (Odd-Length Cycle)** থাকে**,** তবে সেই গ্রাফ বাইকালারিং সম্ভব নয়।

একটি উদাহরণ দেখা যাক। ধরি, তিনটি নোড (০, ১, ২) একটি ত্রিভুজ তৈরি করেছে (০-১, ১-২, ২-০)। এটি একটি ৩-দৈর্ঘ্যের চক্র (বিজোড়)।

- ধরি, নোড ০-কে আমরা সাদা রঙ দিলাম।
- যেহেতু ০ এবং ১ সংলগ্ন, নোড ১-কে অবশ্যই কালো রঙ দিতে হবে।
- যেহেতু ১ এবং ২ সংলগ্ন, নোড ২-কে অবশ্যই সাদা রঙ দিতে হবে।
- কিন্তু, নোড ২ আবার নোড ০-এর সাথেও যুক্ত! এখানে নোড ০ এবং নোড ২, দুটির রঙই সাদা হয়ে যাচ্ছে। এটি একটি দ্বন্দ্ব বা (Conflict)!

সুতরাং, একটি ত্রিভুজ (৩-চক্র), পেন্টাগন (৫-চক্র) বা যেকোনো বিজোড় চক্র থাকলেই গ্রাফটি বাইকালারিং করা যাবে না।

সমাধান অ্যালগরিদম **(BFS):**

আমরা কীভাবে বুঝব যে একটি গ্রাফে বিজোড় চক্র আছে কি না? এর জন্য আমরা খুব জনপ্রিয় একটি গ্রাফ ট্রাভার্সাল অ্যালগরিদম, ব্রেডথ-ফার্স্ট সার্চ **(Breadth-First Search** বা **BFS)** ব্যবহার করতে পারি।

ধাপগুলো নিচে বর্ণনা করা হলো:

১. শুরু**:** প্রথমে, সব নোডকে uncolored হিসেবে চিহ্নিত করি। আমরা প্রতিটি নোডের রঙ সংরক্ষণের জন্য একটি color অ্যারে নেব। এখানে: `-1` মানে uncolored `0` মানে রঙ gray 1 মানে রঙ black

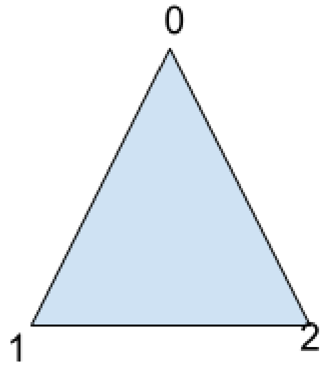
২. **BFS** শুরু**:** যেকোনো একটি নোড (যেমন, নোড ০) থেকে BFS শুরু করি। নোড ০-কে `color[0] = 0` করি। নোড ০-কে একটি queue তে যোগ করি।

৩. **Traversal:** যতক্ষণ queue খালি না হয়, ততক্ষণ নিচের কাজগুলো করি: queue থেকে একটি নোড u বের করি। u-এর বর্তমান রঙ হলো `currentColor = color[u]`। এর প্রতিবেশীদের রঙ হওয়া উচিত `neighborColor = 1 - currentColor` (অর্থাৎ, ০ হলে ১, আর ১ হলে ০)। এখন u-এর প্রতিটি neighbor v-এর জন্য পরীক্ষা করি: যদি **v** রঙবিহীন হয় অর্থাৎ **`color[v] == -1`:** খুবই ভালো! v-কে `neighborColor` দিই (`color[v] = neighborColor`) এবং তাকে queue-তে যোগ করি। যদি **v** আগে থেকেই রঙিন হয়**:** এখানে আমাদের conflict পরীক্ষা করতে হবে। যদি **`color[v]`** এবং **`color[u]`** একই হয় **(`color[v] == currentColor`):** খুবই খারাপ! দুটি পাশাপাশি নোডের রঙ এক হয়ে গেছে। এর মানে গ্রাফটি বাইকালারিং সম্ভব নয়। আমরা সাথে সাথে অ্যালগরিদম থামিয়ে দেব এবং "NOT BICOLORABLE" বলব। যদি `color[v]` সঠিক উল্টো রঙ (`neighborColor`) হয়, তবে কিছু করার নেই, সব ঠিক আছে।

৪. ফলাফল**:** যদি সম্পূর্ণ BFS কোনো conflict ছাড়াই শেষ হয়, তার মানে গ্রাফটিকে সফলভাবে দুটি রঙে রাঙানো গেছে। ফলাফল হবে "BICOLORABLE"।

<u>স্যাম্পল ইনপুট (ত্রিভুজ) ওয়াকথ্রু:</u>

ইনপুট:
3
3
0 1
1 2
2 0



১. `color` অ্যারে = `[-1, -1, -1]`
২. নোড ০ থেকে শুরু করি। `color[0] = 0`। queue: `[0]`। `color` = `[0, -1, -1]`

৩. queue থেকে `0` বের করি (`u = 0`, `currentColor = 0`, `neighborColor = 1`)।
৪. `0`-এর প্রতিবেশী 1: রঙবিহীন। `color[1] = 1`। queue: `[1]`। color = `[0, 1, -1]`
৫. `0`-এর প্রতিবেশী 2: রঙবিহীন। `color[2] = 1`। কিউ: `[1, 2]`। color = `[0, 1, 1]`
৬. queue থেকে 1 বের করি (`u = 1`, `currentColor = 1`, `neighborColor = 0`)।
৭. 1-এর প্রতিবেশী `0`: `color[0]` (যা `0`) `currentColor` (যা 1)-এর সমান নয়। ঠিক আছে।
৮. 1-এর প্রতিবেশী 2: `color[2]` (যা 1) `currentColor` (যা 1)-এর সমান!
৯. **Conflict!** দুটি সংলগ্ন নোড (১ এবং ২) একই রঙ (1) পেয়েছে।
১০. অ্যালগরিদম থামবে।

আউটপুট: `NOT BICOLORABLE.`

Pseudocode:

```
FUNCTION isBicolorable(n, adjList):

    DECLARE colors[n]
    FOR i FROM 0 TO n-1:
        colors[i] = -1
    DECLARE queue

    colors[0] = 0  // নোড 0-কে প্রথম রঙ (রঙ 0) দিই।
    queue.push(0)

    WHILE queue is not empty:
        u = queue.pop()
        // u-এর রঙ যা, তার neighbour তার উল্টোটা হবে। (০->1,1->0)
        neighborColor = 1 - colors[u]

        FOR each neighbor v in adjList[u]:

            IF colors[v] == -1:
                colors[v] = neighborColor  // v-কে উল্টো রঙটি দিই।
                queue.push(v)

            ELSE IF colors[v] == colors[u]:
                RETURN false

    RETURN true

// ---Main Program---

WHILE true:
    READ n
    IF n == 0:
        BREAK
```

```
READ e
DECLARE adjList[n]

FOR i FROM 0 TO e-1:
    READ u, v

    adjList[u].add(v)
    adjList[v].add(u)
IF isBicolorable(n, adjList) == true:
    PRINT "BICOLORABLE."
ELSE:
    PRINT "NOT BICOLORABLE."
```