بچرخ تا بچرخیم (امتیازی)

شما در دنیایی زندگی می‌کنید که دو گراف مرموز پیش روی شماست:

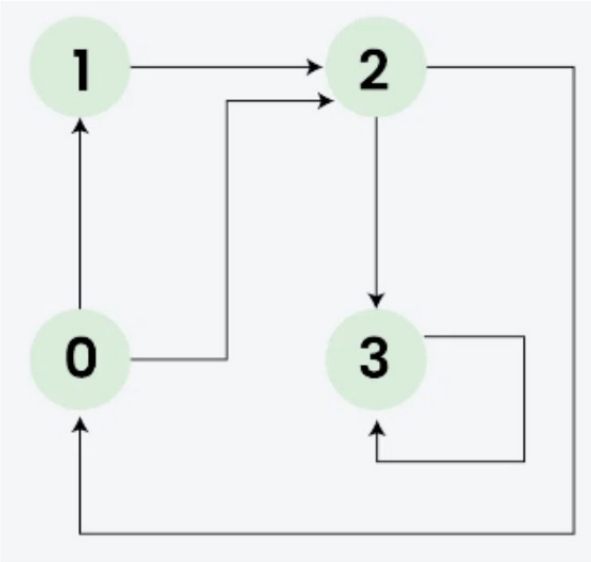
۱. **گراف جهت‌دار**: گرافی که یال‌های آن با فلش مشخص شده‌اند و مسیرها همیشه یک‌طرفه هستند.

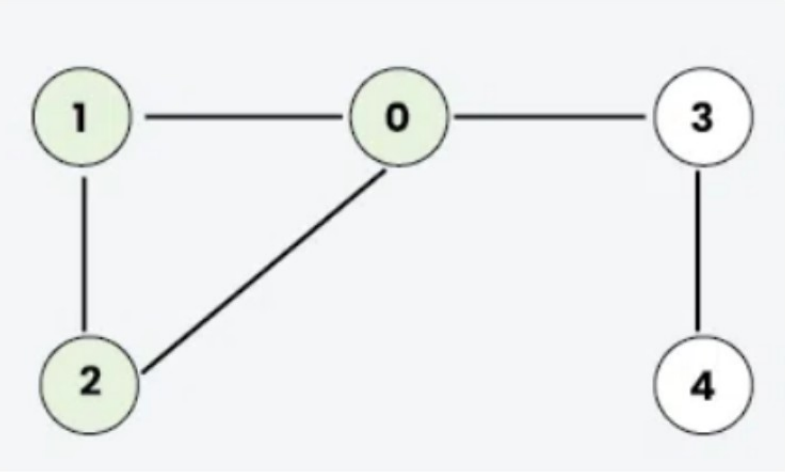
۲. **گراف بدون جهت**: گرافی که یال‌های آن دوطرفه بوده و شما می‌توانید در هر دو جهت از آن عبور کنید.

هر یک از این گراف‌ها ممکن است چرخه‌هایی داشته باشند؛ حلقه‌هایی که شما را در مسیرهایی تکراری گرفتار می‌کنند. وظیفه شما این است که:

۱. در **گراف جهت‌دار** بررسی کنید که آیا چرخه‌ای وجود دارد؟ اگر وجود دارد، آن را شناسایی کنید.

۲. در **گراف بدون جهت** نیز چرخه‌ای را که ممکن است مخفی شده باشد، پیدا کنید.





راهنمایی:

- در گراف جهت‌دار، ممکن است چرخه‌ها به گونه‌ای پنهان شده باشند که نیاز به بررسی ترتیب یال‌ها داشته باشید.
- در گراف بدون جهت، چرخه‌ها ممکن است با بازدید دوباره از یال‌هایی که قبلاً مشاهده کرده‌اید، پیدا شوند.

وظیفه شما: با استفاده از شبه کد برای هر کدام از گراف ها چرخه را تشخیص دهید الگوریتمی که برای گراف جهت دار استفاده میکنید میتواند با الگوریتمی که برای تشخیص چرخه در گراف بدون جهت استفاده میکنید متفاوت باشد.

آیا شما می‌توانید این گراف‌ها را به دقت بررسی کنید و از اسرار چرخه‌ها پرده بردارید؟ مراقب باشید، زیرا در گراف‌های بزرگ ممکن است به راحتی گم شوید!

برای تشخیص چرخه در گراف‌های جهت‌دار و بدون‌جهت، باید از دو الگوریتم متفاوت استفاده کرد، چون ساختار و تعریف چرخه در آن‌ها متفاوت است.

گراف جهت‌دار:

برای پیدا کردن چرخه در گراف جهت‌دار، از DFS و سه وضعیت برای گره‌ها استفاده می‌کنیم. برای هر گره یکی از وضعیت‌های زیر در نظر می‌گیریم.

۰ : هنوز سراغش نرفتیم

۱ : در مسیر DFS

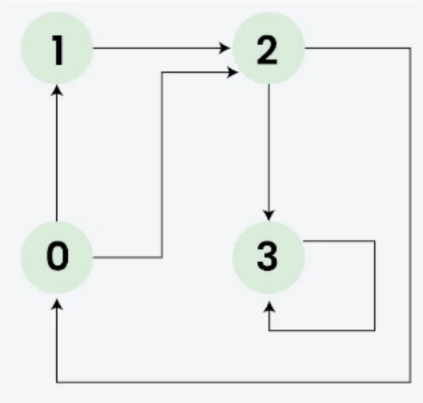
۲ : بررسی شده

اگر حین DFS به گره‌ای برسیم که دارای وضعیت ۱ (در حال بازدید) است، یعنی داریم در یک مسیر DFS به گره‌ای برمی‌گردیم که قبلاً وارد آن شده‌ایم ولی هنوز بیرون نیامده‌ایم، پس یک چرخه داریم.

```
func has_cycle_directed(graph, n):
   visited = [0] * n        // 3 states of visit for each node
   stack = []               // current path

   function dfs(u):
      visited[u] = 1
      stack.push(u)

      for neighbor in graph[u]:
         if visited[neighbor] == 0:
            if dfs(neighbor):
               return true
         else if visited[neighbor] == 1:
            // found cycle
            printCycle(stack, neighbor)
            return true

      stack.pop()
      visited[u] = 2
      return false
```

```
// maybe graph has some components
for i = 0 to n-1:
    if visited[i] == 0:
        if dfs(i):
            return true

return false
```

اجرای الگوریتم:



visited = {0, 0, 0, 0}

stack = {}

dfs روی گره 0 صدا زده می‌شود:

visited = {1, 0, 0, 0}

stack = {0}

dfs روی گره 1 صدا زده می‌شود:

visited = {1, 1, 0, 0}

stack = {0, 1}

dfs روی گره 2 صدا زده می‌شود:

visited = {1, 1, 1, 0}

stack = {0, 1, 2}

در اینجا برای گره ۰ که همسایه گره ۲ است، visited[0] == 1 را چک می‌کند و چرخه را تشخیص می‌دهد.

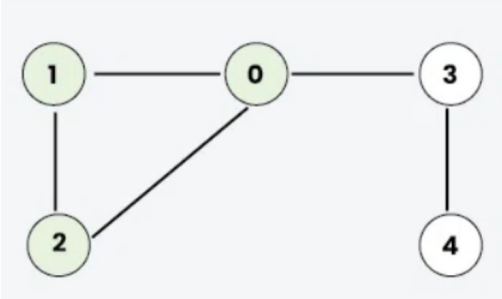
گراف بدون جهت:

در گراف بدون جهت، هر یال را می‌توان از دو طرف پیمود، یعنی هنگام بازگشت ممکن است به گره‌ای برسیم که همان والد باشد و این چرخه نیست. اما اگر به گره‌ای برسیم که بازدید شده و والد نیست، چرخه وجود دارد.

```
function has_cycle_undirected(graph, n):
    visited = [false] * n

    function dfs(u, parent):
        visited[u] = true
```

```
        for neighbor in graph[u]:
            if visited[neighbor] == 0:
                if dfs(neighbor, u):
                    return true
            else if neighbor != parent:
                // found cycle
                return true

        return false

    for i = 0 to n-1:
        if not visited[i]:
            if dfs(i, -1):
                return true

    return false
```

اجرای الگوریتم:



visited = {0, 0, 0, 0, 0}

dfs روی گره 0 صدا زده می‌شود:

visited = {1, 0, 0, 0, 0}

parent = -1

dfs روی گره 1 صدا زده می‌شود:

visited = {1, 1, 0, 0, 0}

parent = 0

برای گره ۰ که همسایه گره ۱ است، می‌بیند که visited[0] != 0 است ولی چون neighbor == parent است، چرخه تشخیص داده نمی‌شود.

dfs روی گره 2 صدا زده می‌شود:

visited = {1, 1, 1, 0, 0}

parent = 1

برای گره ۰ که همسایه گره ۲ است، می‌بیند که visited[0] != 0 است و همچنین neighbor != parent است، پس چرخه تشخیص داده می‌شود.