Amirali Farazmand

99522329

برای تمامی نود ها ما باید با مانند iterative BFS با مبدا آن نود تابعی را صدا میزنیم. در تابه همانند BFS هرگاه به نودی میرسیم که ویزیت نشده است، آن را به صف اضافه میکنیم و به ترتیب صف را dequeue میکنیم(دقیقا عینBFS)با این فرق که تعداد گام ها را هم به ازای ان شروع با آن نود ذخیره داریم و اضافه میکنیم در هر گام. هنگامی که به نودی میرسیم که ویزیت شده است یعنی که به کوتاه ترین دور با شروع از ان نود رسیدیم.اگر آن مقدار از متغیر مینیممی که بطور کلی برای کل گراف داریم(که با بینهایتدر ابتدا مقدار دهی شده است)کوچکتر باشد، مقدار آنرا در متغیر میگذاریم. در نهایت و با زدن این عملیات برای همه ی نود ها آن متغیر را برمیگردانیم(اگر بینهایت بود یعنی دور نداریم.).

پیچیدگی زمانی آن O(V * (E+V)) است چرا که گویا برای تمام نود ها (V)تا ، BFS زده ایم که از O(E+V) است.

الگوریتم گفته شده:

```
// Function to find the length of
// the shortest cycle in the graph
int shortest_cycle(int n)
{
    // To store length of the shortest cycle
    int ans = INT_MAX;

    // For all vertices
    for (int i = 0; i < n; i++) {

        // Make distance maximum
        vector<int> dist(n, (int)(1e9));

        // Take a imaginary parent
        vector<int> par(n, -1);

        // Distance of source to source is 0
        dist[i] = 0;
        queue<int> q;

        // Push the source element
        q.push(i);

        // Continue until queue is not empty
        while (!q.empty()) {

            // Take the first element
            int x = q.front();
            q.pop();

            // Traverse for all it's childs
            for (int child : gr[x]) {

                // If it is not visited yet
                if (dist[child] == (int)(1e9)) {
```

```
                    // Increase distance by 1
                    dist[child] = 1 + dist[x];

                    // Change parent
                    par[child] = x;

                    // Push into the queue
                    q.push(child);
                }

                // If it is already visited
                else if (par[x] != child and par[child] != x)
                    ans = min(ans, dist[x] + dist[child] + 1);
            }
        }
    }

    // If graph contains no cycle
    if (ans == INT_MAX)
        return -1;

    // If graph contains cycle
    else
        return ans;
}
```

[source]