A - Three

Bài này trông có vẻ khó nhưng thực ra khá đơn giản.

1. Giả sử đồ thị không có đỉnh nào có bậc lớn hơn 2. Lúc này hoặc là đồ thị có dạng thẳng hoặc là đồ thị có dạng vòng tròn.

- Nếu đồ thị có dạng thẳng, rõ ràng là không thể tồn tại kết quả, vì tổng khoảng cách từ điểm ở giữa đến 2 điểm còn lại băng khoảng cách của 2 điểm còn lại
- Nếu đồ thị có dạng vòng, kết quả chỉ tổn tại khi n là bội của 3 và lúc này với bất kì 1 điểm nào cũng tồn tại 2 điểm khác nhau cách nó n/3.

2. Nếu đồ thị tồn tại ít nhất một đỉnh có bậc lớn hơn 2, gọi một điểm bất kì có bậc lớn hơn 2 là X, với 3 đỉnh A, B, C bất kì kề với X thì chúng sẽ cách đều X:

- Nếu 2 trong số 3 điểm đấy kề với nhau(giả sử là A và B) thì lúc này A, B, X cách đều nhau và đều bằng 1.
- Ngược lại thì 3 điểm A, B, C cách đều nhau và đều bằng 2.

Để lấy 3 đỉnh kề cũng như kiểm tra 2 đỉnh có kề nhau hay không, chúng ta có thể xử dụng cấu trúc set trong C++ để dễ dàng hơn trong việc cài đặt.

B - Path

Gọi 2 điểm tồn tại 2 đường đi chẵn và lẻ là S và T.

Nhận thấy rằng, nếu tồn tại đỉnh U liên thông với S thì từ U cũng tồn tại 2 đường đi chẵn và lẻ tới T. Như vậy lúc này với mỗi thành phần liên thông ta chỉ quan tâm tới điểm kết thúc T còn điểm bắt đầu S thì có thể chọn ngẫu nhiên. Để đơn giản chúng ta sẽ chọn S là đỉnh có chỉ số nhỏ nhất của thành phần liên thông đấy luôn.

Ta có $f_{u,0}$ thể hiện có tồn tại con đường từ S tới u qua chẵn con đường hay không, tương tự với $f_{u,1}$ thể hiện có tồn tại con đường từ S tới u qua lẻ con đường hay không.

- Nếu $f_{u,0} = 1$ thì với mọi đỉnh v kề với u $f_{v,1} = 1$.
- Tương tự, nếu $f_{u,1} = 1$ thì với mọi đỉnh v kề với u $f_{v,0} = 1$.

Việc tính f có thể thực hiện bằng BFS và nếu bất kì $f_{v,0}$ hay $f_{v,1}$ nào đã bằng 1 thì chúng ta cũng không cần xét lại nữa.

Vậy nếu tồn tại 1 đỉnh u mà $f_{u,0} = f_{u,1} = 1$ thì đồ thị tồn tại 2 đỉnh có đường đi chẵn và đường đi lẻ tới nhau.

Độ phức tạp: O(n + m);

C - White Zone

Bài này thuộc kì thi chọn đội tuyển tỉnh Thanh Hóa năm 2019. Thuật toán mình lựa chọn sử dụng trong bài này là thuật toán tìm khớp, cầu. Do đó bạn nào chưa học về thuật toán này thì nên nghiên cứu trước khi đọc solution này.

Ta sẽ xây dựng một đơn đồ thị vô hướng dựa trên ma trận đã cho. Coi mỗi ô trắng trong ma trận là một đỉnh của đồ thị, 2 ô trắng kề cạnh nhau thì 2 đỉnh tương ứng của nó có cạnh chung. Đặc biệt đặt thêm 1 đỉnh phụ(gọi là S) có cạnh chúng tới tất cả các đỉnh mà tương ứng của nó là 1 ô ở rìa ma trận (đỉnh này sẽ giống như phần bên ngoài của ma trận).

Bài toán bây giờ trở thành: xóa một đỉnh khác S khỏi đồ thị sao cho nhiều đỉnh không liên thông được với S nhất.

Dễ thấy, việc chọn tối ưu nhất phải là một khớp của đồ thị. Vì nếu chọn xóa một đỉnh không phải khớp, tình liên thông của phần còn lại của đồ thị là không đổi. Và nếu chọn 1 khớp thì toàn bộ con của nó cũng sẽ không liên thông với S.

Vậy việc của chúng ta cần làm lúc này chỉ là tìm khớp có nhiều đỉnh con nhất.

Gọi f_u thuộc cây DFS gốc u và g_u là số đỉnh con của u.

- $f_u = 1 + \sum f_v$ Với mọi đỉnh v chưa được DFS kề với u.
- $g_u = 1 + \sum f_v$ với mọi đỉnh v mà $low_v \geq num_u$.

Kết quả bài toán là MAX g_u với mọi u.

Độ phức tạp: O(n $\times m$);