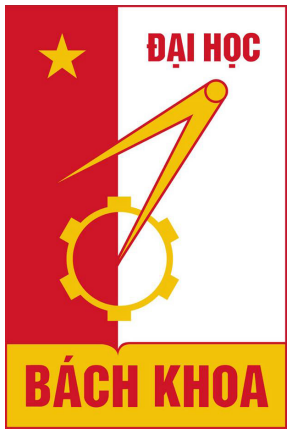
Thuật toán ứng dụng
Bài thực hành số 5.2: Các thuật toán trên đồ thị

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương



Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 1 tháng 6 năm 2021

Mục lục

Mục lục

MỞ ĐẦU

- Đồ thị là một cấu trúc gồm các đỉnh và cạnh, có thể mô hình hoá nhiều cấu trúc thực tế: hệ thống giao thông, mạng máy tính, mạng xã hội,...

MỞ ĐẦU

- Đồ thị là một cấu trúc gồm các đỉnh và cạnh, có thể mô hình hoá nhiều cấu trúc thực tế: hệ thống giao thông, mạng máy tính, mạng xã hội,...
- Vì vậy, nhiều bài toán thực tế được đưa về các bài toán trên đồ thị.

MỞ ĐẦU

Một số bài toán trên đồ thị:

- Các bài toán cổ điển:
  - Tìm đường đi ngắn nhất
  - Tìm cây khung nhỏ nhất
  - Tô màu đồ thị
  - Tìm khớp, cầu
  - Phát hiện chu trình
  - ...

MỞ ĐẦU

Một số bài toán trên đồ thị:

- Các bài toán cổ điển:
  - Tìm đường đi ngắn nhất
  - Tìm cây khung nhỏ nhất
  - Tô màu đồ thị
  - Tìm khớp, cầu
  - Phát hiện chu trình
  - ...
- Các bài toán learning, mining:
  - Mạng bạn bè trên mạng xã hội
  - Cấu trúc phân tử hóa học
  - Tương tác thuốc
  - Dự đoán dịch bệnh
  - Video có cấu trúc
  - ...

MỞ ĐẦU

Các thuật toán trên đồ thị:

- Duyệt đồ thị: DFS, BFS,...
- Tìm đường đi ngắn nhất: Dijkstra, Bellman-Ford, Floyd-Warshall,...
- Tìm cây khung nhỏ nhất: Prim, Kruskal,...
- Tìm thành phần liên thông của đồ thị: Tarjan, Kosaraju,...
- Các thuật toán learning và mining.
- ...

Mục lục

06. BUGLIFE

- Cho một đồ thị vô hướng.

06. BUGLIFE

- Cho một đồ thị vô hướng.
- Kiểm tra xem nó có phải là đồ thị hai phía hay không.

Thuật toán

- Dùng hai màu đen và đỏ để tô màu cho đồ thị.

Thuật toán

- Dùng hai màu đen và đỏ để tô màu cho đồ thị.
- Với mỗi thành phần liên thông của đồ thị, chọn một đỉnh bất kỳ và tô màu đỏ.

Thuật toán

- Dùng hai màu đen và đỏ để tô màu cho đồ thị.
- Với mỗi thành phần liên thông của đồ thị, chọn một đỉnh bất kỳ và tô màu đỏ.
- Với mỗi đỉnh đã được tô màu, xét các đỉnh kề với nó:

Thuật toán

- Dùng hai màu đen và đỏ để tô màu cho đồ thị.
- Với mỗi thành phần liên thông của đồ thị, chọn một đỉnh bất kỳ và tô màu đỏ.
- Với mỗi đỉnh đã được tô màu, xét các đỉnh kề với nó:
  - Nếu đỉnh kề chưa được tô màu, ta tô màu ngược lại với đỉnh đang xét.

Thuật toán

- Dùng hai màu đen và đỏ để tô màu cho đồ thị.
- Với mỗi thành phần liên thông của đồ thị, chọn một đỉnh bất kỳ và tô màu đỏ.
- Với mỗi đỉnh đã được tô màu, xét các đỉnh kề với nó:
  - Nếu đỉnh kề chưa được tô màu, ta tô màu ngược lại với đỉnh đang xét.
- Nếu có 2 đỉnh kề bất kỳ được tô cùng màu, trả về thông báo phát hiện bất thường.

Code

```
vector<int> a[N];
int color[N];

void dfs(int u) {
    for (int v : a[u]) {
        if (color[v] == -1) {
            color[v] = !color[u];
            dfs(v);
        }
    }
}
```

Code

```
    for (int i = 1; i <= n; ++i) color[i] = -1;
    for (int i = 1; i <= n; ++i) {
        if (color[i] == -1) {
            color[i] = 0;
            dfs(i);
        }
    }
    bool bipartite = true;
    for (int u = 1; u <= n; ++u) {
        for (int v : a[u]) {
            bipartite &= color[u] != color[v];
        }
    }
```

Mục lục

ICBUS

- Cho n thị trấn được đánh số từ 1 tới n.

ICBUS

- Cho n thị trấn được đánh số từ 1 tới n.
- Có k con đường hai chiều nối giữa các thị trấn.

ICBUS

- Cho n thị trấn được đánh số từ 1 tới n.
- Có k con đường hai chiều nối giữa các thị trấn.
- Ở thị trấn thứ i có thể mua vé với giá là c_i và đi qua tối đa d_i cạnh bất kỳ, xuất phát từ i.

ICBUS

- Cho n thị trấn được đánh số từ 1 tới n.
- Có k con đường hai chiều nối giữa các thị trấn.
- Ở thị trấn thứ i có thể mua vé với giá là c_i và đi qua tối đa d_i cạnh bất kỳ, xuất phát từ i.
- Tìm chi phí tối thiểu để đi từ thị trấn 1 tới thị trấn n.

Thuật toán

- **Bước 1:** Tính khoảng cách di chuyển ngắn nhất của tất cả các cặp đỉnh u, v bằng thuật toán BFS. Lưu vào mảng $dist[u][v]$.

Thuật toán

- **Bước 1:** Tính khoảng cách di chuyển ngắn nhất của tất cả các cặp đỉnh u, v bằng thuật toán BFS. Lưu vào mảng $dist[u][v]$.
- **Bước 2:** Tạo một đồ thị có hướng trong đó tồn tại cung (u, v) khi $dist[u][v] <= d[u]$ và cung này có trọng số là $c[u]$.

Thuật toán

- **Bước 1:** Tính khoảng cách di chuyển ngắn nhất của tất cả các cặp đỉnh u, v bằng thuật toán BFS. Lưu vào mảng $dist[u][v]$.
- **Bước 2:** Tạo một đồ thị có hướng trong đó tồn tại cung (u, v) khi $dist[u][v] <= d[u]$ và cung này có trọng số là $c[u]$.
- **Bước 3:** Tìm đường đi ngắn nhất từ 1 tới n trên đồ thị mới được tạo ra bằng thuật toán Dijkstra.

Thuật toán

- **Bước 1:** Tính khoảng cách di chuyển ngắn nhất của tất cả các cặp đỉnh u, v bằng thuật toán BFS. Lưu vào mảng $dist[u][v]$.
- **Bước 2:** Tạo một đồ thị có hướng trong đó tồn tại cung (u, v) khi $dist[u][v] <= d[u]$ và cung này có trọng số là $c[u]$.
- **Bước 3:** Tìm đường đi ngắn nhất từ 1 tới n trên đồ thị mới được tạo ra bằng thuật toán Dijkstra.
- Độ phức tạp thuật toán $O(n^2)$

Code

```
void calculate_dist() {
    ** Calculate dist[u][v] using BFS algorithm **
}
void find_shortest_path() {
    for (int i = 0; i <= n; i++) {
        ans[i] = MAX;
        visit[i] = 0;
    }
    ans[1] = 0;
    int step = n;
    while (step--) {
        int min_vertex = 0;
        for (int i = 1; i <= n; i++) {
            if (visit[i] == 0 &&
            ans[min_vertex] > ans[i]) {
                min_vertex = i;
            }
        }
```

Code

```
            visit[min_vertex] = 1;
            for (int i = 1; i <= n; i++) {
                if (dist[min_vertex][i]
                <= d[min_vertex]) {
                    ans[i] = min(ans[i],
                    ans[min_vertex] + c[min_vertex]);
                }
            }
        }
        cout << ans[n] << endl;
}
```

Mục lục

ELEVTRBL

- Xuất phát ở tầng s của tòa nhà có f tầng, cần đến tầng g.
- Có thể đi lên u tầng hoặc đi xuống d tầng mỗi lần bấm nút.
- **Yêu cầu:** Tìm số lần bấm nút nhỏ nhất.
- **Giới hạn:** Các giá trị $\leq 10^6$.

Nhận xét

- Tương tự bài WATERJUG.
  - f là sức chứa.
  - s là lượng nước ban đầu.
  - g là lượng nước cần đong.
  - Mỗi lần có thể đổ thêm u lít nước hoặc bớt d lít nước.
- Có thể mô hình hóa bằng đồ thị có hướng:
  - Mỗi tầng là 1 đỉnh của đồ thị.
  - Từ tầng x có cung nối trực tiếp đến $x + u$ và $x - d$ $(1 \leq x - d, x + u \leq f)$.

Thuật toán

- Sử dụng thuật toán BFS.

Code

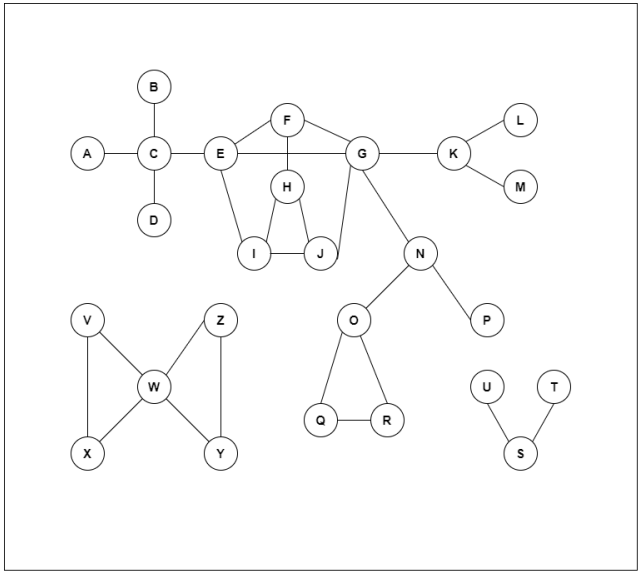
```
memset(flag, 0, sizeof(flag));
q.push(s);
flag[s] = 1;
bool has_sol = false;
while(!q.empty()) {
    p = q.front();
    q.pop();
    pu = p + u;
    pd = p - d;
    if ((pu == g) || (pd == g)) {
        cout << flag[p] << endl;
        has_sol = true;
        break;
    }
    if ((pu <= f) && (!flag[pu])) push(p, pu);
    if ((pd >= 1) && (!flag[pd])) push(p, pd);
}
if (!has_sol) cout << "use the stairs\n" << endl;
```

Mục lục

ADDEDGE

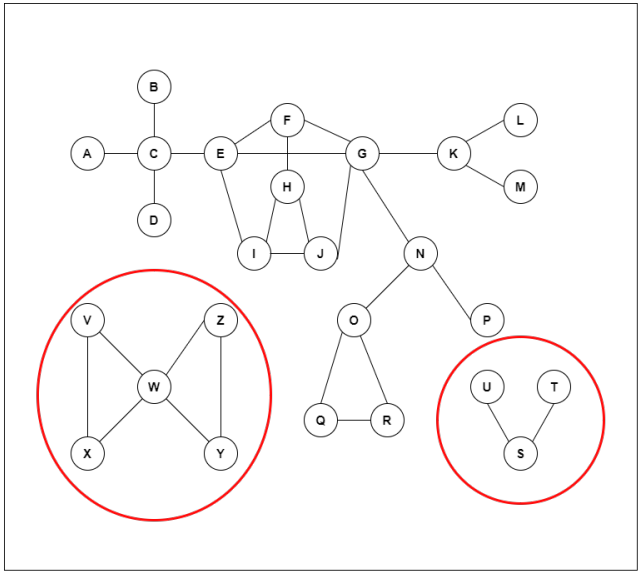
- Cho đồ thị vô hướng $G = (V, E), |V| = n, |E| = m$.
- Chu trình đơn là chu trình đi qua mỗi cạnh và mỗi đỉnh tối đa 1 lần, trừ đỉnh đầu và đỉnh cuối.
- **Yêu cầu:** Đếm số cặp đỉnh (u, v) không kề nhau mà khi thêm cạnh nối giữa chúng, đồ thị có thêm đúng 1 chu trình đơn.
- **Giới hạn:** $n, m \leq 10^5$.

Nhận xét



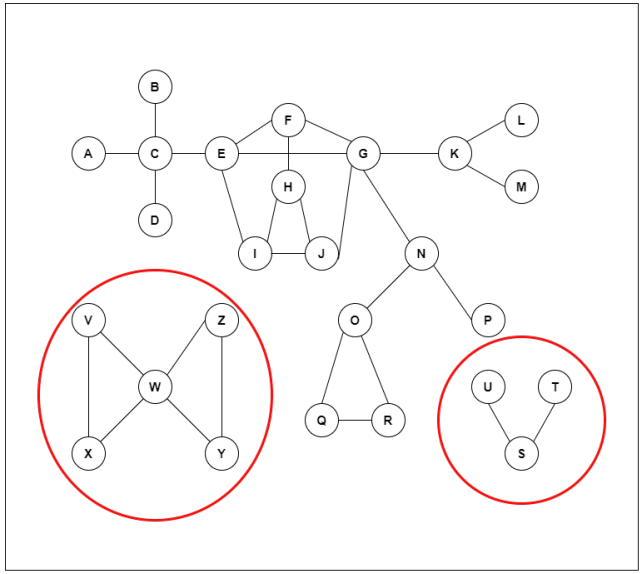
- Đồ thị có N_{cc} thành phần liên thông.

Nhận xét



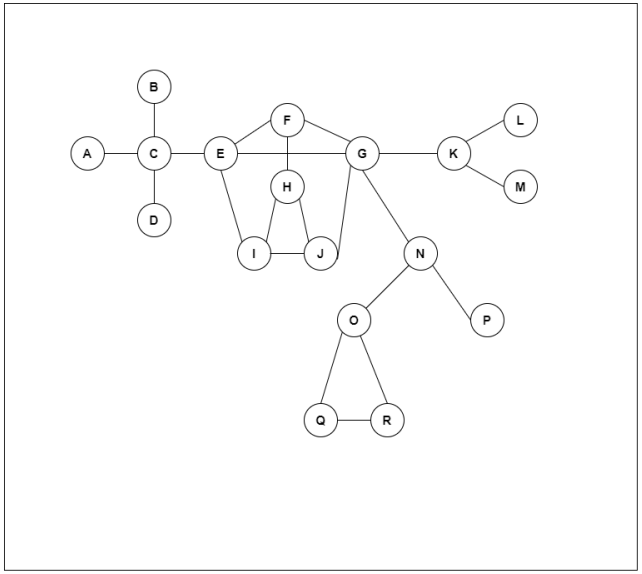
- Đồ thị có N_{cc} thành phần liên thông.

Nhận xét



- Đồ thị có N_{cc} thành phần liên thông.
- Nếu u, v không thuộc cùng thành phần liên thông, thêm cạnh (u, v) không tạo ra chu trình.

Nhận xét



- Đồ thị có N_{cc} thành phần liên thông.
- Nếu u, v không thuộc cùng thành phần liên thông, thêm cạnh (u, v) không tạo ra chu trình.
- **Hệ quả:** Có thể xem xét N_{cc} bài toán con độc lập, đồ thị của mỗi bài toán con là 1 thành phần liên thông.

Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.

Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.

Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.
- **Maximal cyclic subcomp (MCS)**: Là cyclic subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là cyclic subcomp nữa.

Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.
- **Maximal cyclic subcomp (MCS)**: Là cyclic subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là cyclic subcomp nữa.
- **Tree subcomp**: Là subcomp không chứa cạnh thuộc MCS bất kỳ.

Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.
- **Maximal cyclic subcomp (MCS)**: Là cyclic subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là cyclic subcomp nữa.
- **Tree subcomp**: Là subcomp không chứa cạnh thuộc MCS bất kỳ.
- **Maximal tree subcomp (MTS)**: Là tree subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là tree subcomp nữa.

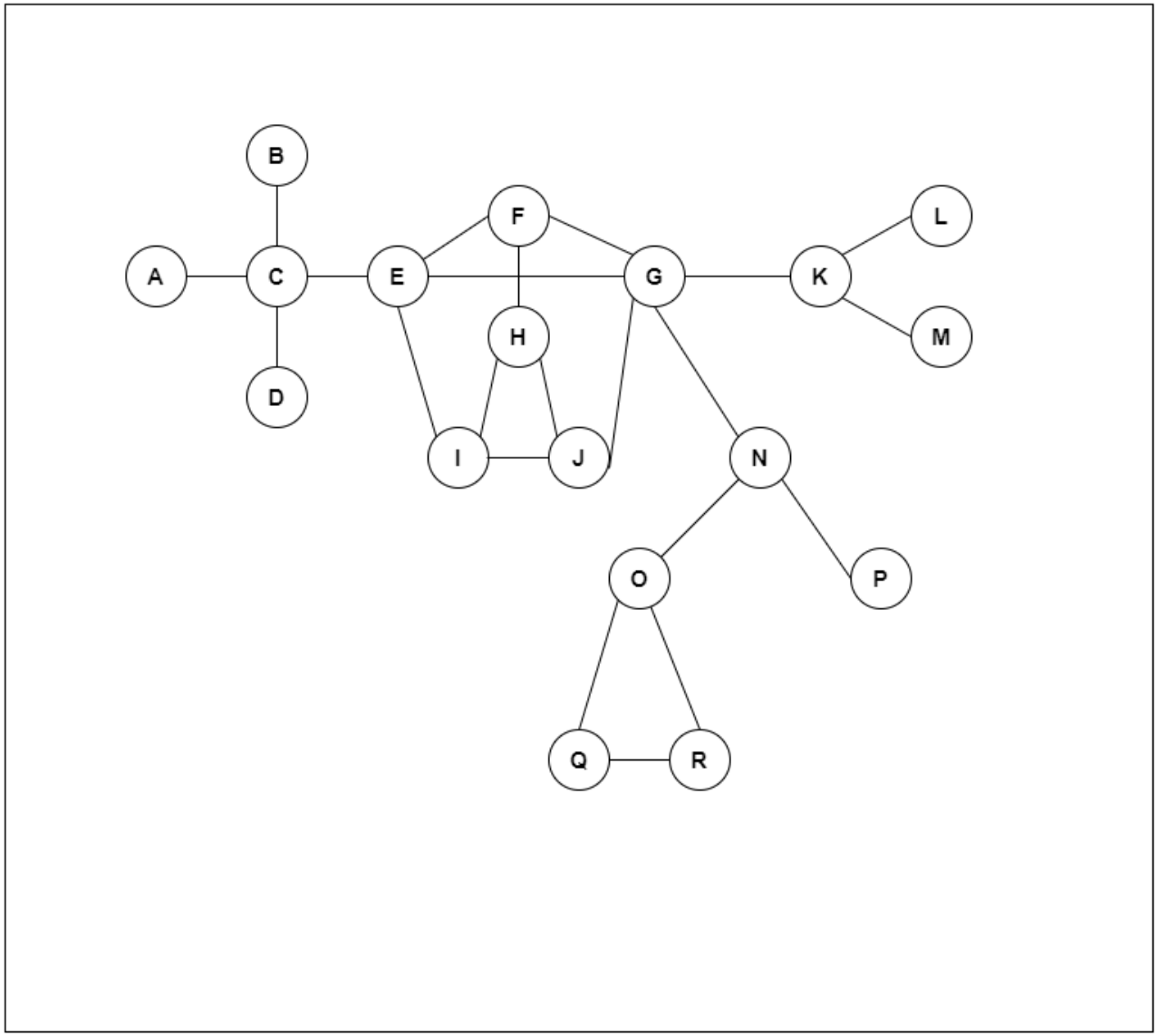
Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.
- **Maximal cyclic subcomp (MCS)**: Là cyclic subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là cyclic subcomp nữa.
- **Tree subcomp**: Là subcomp không chứa cạnh thuộc MCS bất kỳ.
- **Maximal tree subcomp (MTS)**: Là tree subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là tree subcomp nữa.
- MCS và MTS là subcomp đặc biệt. Hợp của tất cả MCS và MTS là đồ thị ban đầu.

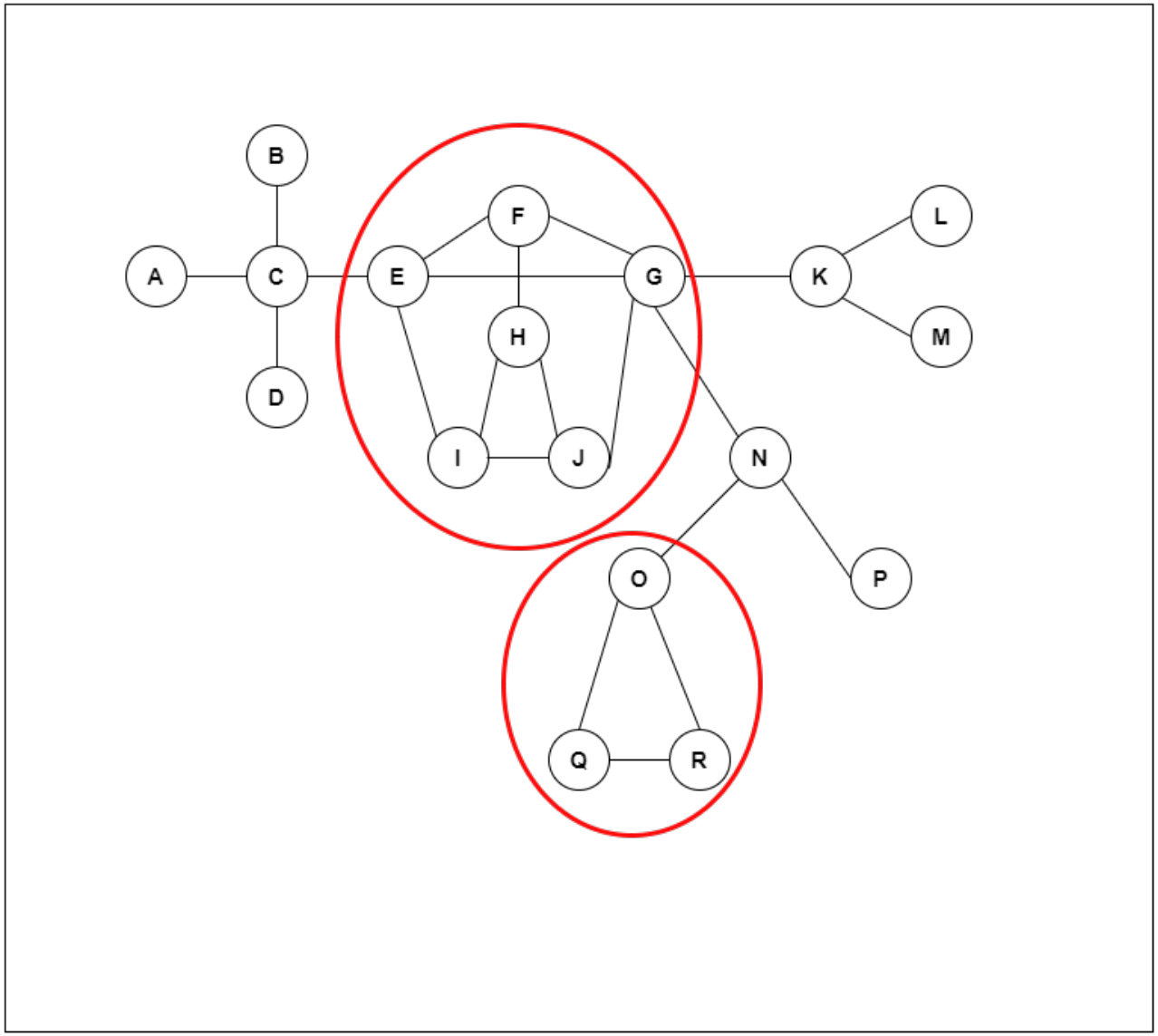
Nhận xét

- **Subcomp**: Là đồ thị con có đúng 1 thành phần liên thông của đồ thị đang xét.
- **Cyclic subcomp**: Là subcomp tồn tại chu trình giữa mọi cặp đỉnh u, v.
- **Maximal cyclic subcomp (MCS)**: Là cyclic subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là cyclic subcomp nữa.
- **Tree subcomp**: Là subcomp không chứa cạnh thuộc MCS bất kỳ.
- **Maximal tree subcomp (MTS)**: Là tree subcomp mà nếu thêm bất cứ đỉnh w nào, nó không còn là tree subcomp nữa.
- MCS và MTS là subcomp đặc biệt. Hợp của tất cả MCS và MTS là đồ thị ban đầu.
- **Gate**: Là điểm thuộc nhiều hơn 1 subcomp đặc biệt.

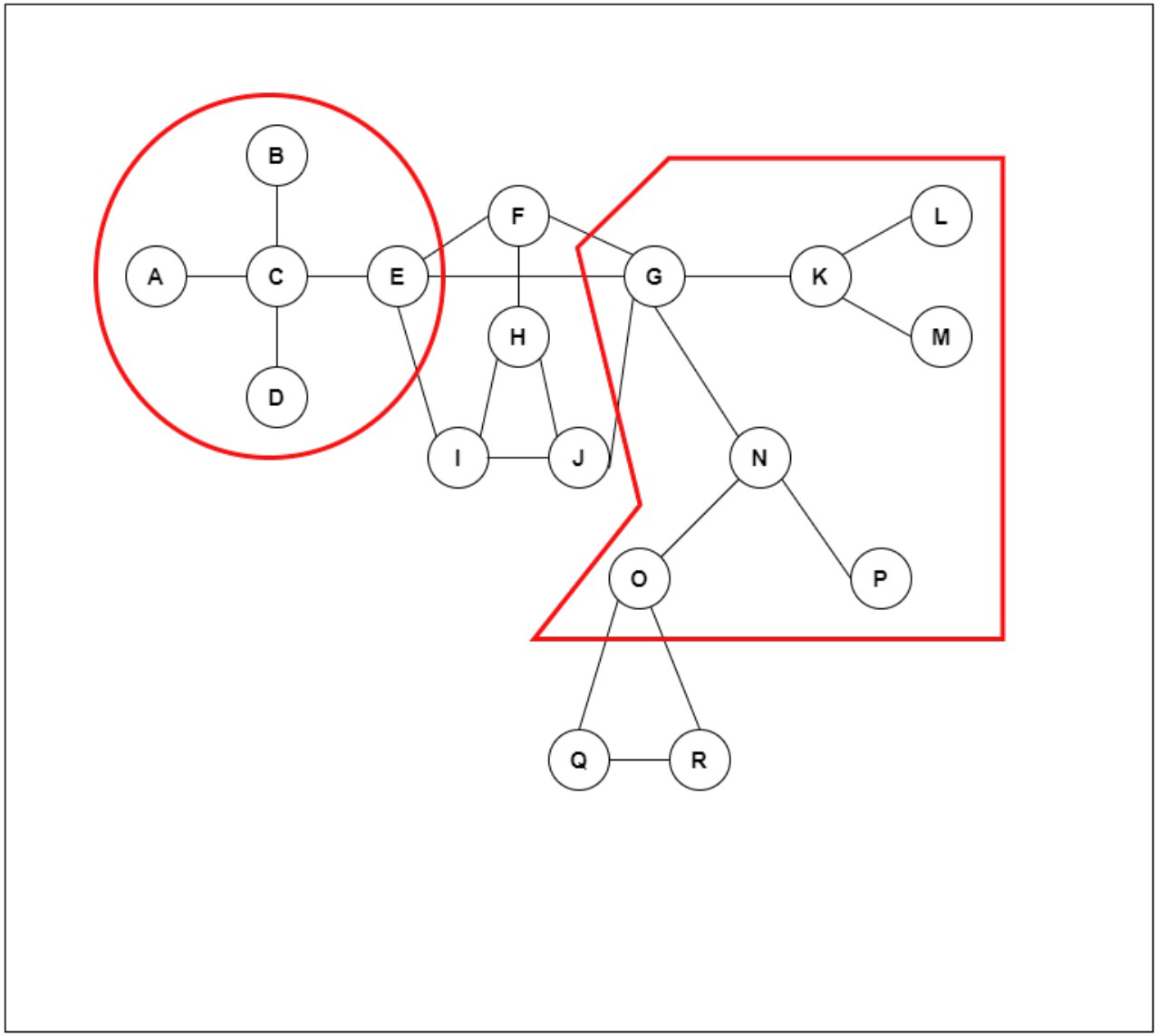
Nhận xét



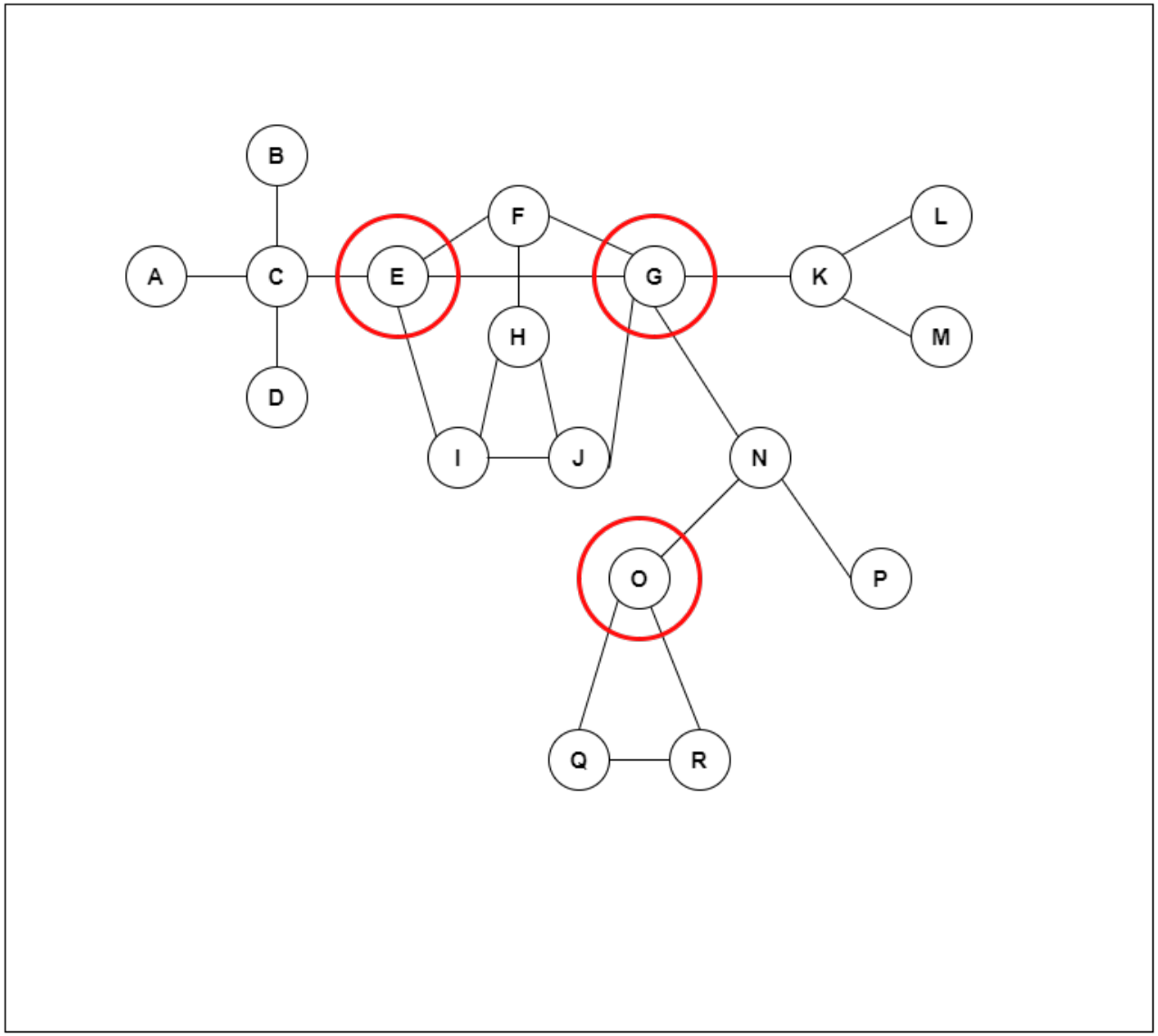
Nhận xét



Nhận xét



Nhận xét



Nhận xét

- Thêm cạnh nối giữa u và v không thuộc cùng 1 subcomp đặc biệt nào tạo ra nhiều hơn 1 chu trình đơn:
  - MCS - MCS
  - MCS - MTS
  - MTS - MTS

Nhận xét

- Thêm cạnh nối giữa u và v không thuộc cùng 1 subcomp đặc biệt nào tạo ra nhiều hơn 1 chu trình đơn:
  - MCS - MCS
  - MCS - MTS
  - MTS - MTS
- Thêm cạnh nối giữa u và v thuộc cùng 1 MCS tạo ra nhiều hơn 1 chu trình đơn.

Nhận xét

- Thêm cạnh nối giữa u và v không thuộc cùng 1 subcomp đặc biệt nào tạo ra nhiều hơn 1 chu trình đơn:
  - MCS - MCS
  - MCS - MTS
  - MTS - MTS
- Thêm cạnh nối giữa u và v thuộc cùng 1 MCS tạo ra nhiều hơn 1 chu trình đơn.
- Thêm cạnh nối giữa u và v thuộc cùng 1 MTS tạo ra đúng 1 chu trình đơn.

Ý tưởng

- Tìm tất cả các MTS, đếm số cặp (u, v) trên từng MTS rồi cộng lại:
- 1 cạnh thuộc MTS khi và chỉ khi cạnh đó là cầu của đồ thị.
  - Cách 1: Liệt kê tất cả cầu của đồ thị.
  - Cách 2: Xóa tất cả các cạnh thuộc MCS bất kỳ.
    - Các đỉnh không phải gate thuộc MTS trở thành cây có có 1 node.
    - Gate vẫn được kết nối với MTS.

Thuật toán

- Xét từng thành phần liên thông của đồ thị:
  - Dùng thuật toán Tarjan để liệt kê cầu của đồ thị.
  - Xây dựng rừng các MTS từ các cầu đã tìm được bằng thuật Disjoint Sets Union.
  - Với mỗi cây trong rừng MTS, số cặp đỉnh u, v thỏa mãn:

$$\frac{n \times (n-1)}{2} - (n-1) \tag{1}$$

Code - lưu trữ

```
void read_input() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> m;
    int u, v;
    for (int i = 0; i < m; i++) {
        cin >> u >> v;
        a[u].push_back(v);
        a[v].push_back(u);
    }
}
```

Code - khởi tạo

```
void init_value() {
    memset(low, 0, sizeof(low));
    memset(disc, 0, sizeof(disc));
    memset(parent, 0, sizeof(parent));
    for (int i = 1; i <= n; i++) {
        r[i] = -1;
    }
    timer = 0;
}
```

Code - Disjoint Sets

```
int root(int x) {
    if (r[x] < 0) {
        return x;
    } else {
        r[x] = root(r[x]);
        return r[x];
    }
}

void merge(int u, int v) {
    if (r[v] < r[u]) {
        swap(u, v);
    }
    r[u] += r[v];
    r[v] = u;
}
```

Code - Tarjan

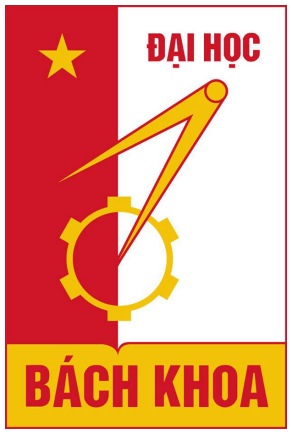
```
void tarjan(int u) {
    disc[u] = low[u] = ++timer;
    for (int v : a[u]) {
        if (disc[v] == 0) {
            parent[v] = u;
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (low[v] > disc[u]) {
                merge(root(u), root(v));
            }
        } else if (v != parent[u]) {
            low[u] = min(low[u], disc[v]);
        }
    }
}
```

Code - main

```
int main() {
    read_input();
    init_value();
    for (int i = 1; i <= n; i++) {
        if (disc[i] == 0) {
            tarjan(i);
        }
    }
    long long res = 0;
    for (int i = 1; i <= n; i++) {
        if (root(i) == i) {
            res += cal(-r[i]);
        }
    }
    cout << res << endl;
    return 0;
}
```

Thuật toán ứng dụng
Bài thực hành số 5.2: Các thuật toán trên đồ thị

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương



Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 1 tháng 6 năm 2021