MỤC LỤC

Chuyên đề:

CẤU TRÚC DỮ LIỆU NÂNG CAO
DISJOINT – SET - UNION

Nhóm tác giả: **Hà Đại Tôn**

Đơn vị công tác: **Trường THPT Chuyên Hạ Long**

(Chuyên đề đạt giải Ba)

PHẦN I: MỞ ĐẦU

1. Lý do chọn

Đề thi học sinh quốc gia, các kì thi Online, …trong các năm gần đây bài tập về đồ thị luôn chiếm một tỉ lệ lớn. Các bài toán ngày càng nâng cao độ khó về thuật toán và cấu trúc dữ liệu. Có rất nhiều các cấu trúc dữ liệu nâng cao ngày càng xuất hiện nhiều trong các đề thi. Trong đó, cấu trúc dữ liệu Disjoint – Set - Union (DSU) là một cấu trúc dữ liệu cơ bản và quan trọng. Tuy nhiên, Tài liệu viết về DSU vẫn còn rất hạn chế đặc biệt là tiếng Việt. Do vậy, tôi đã chọn chủ đề DSU làm đề tài nghiên cứu.

2. Mục đích nghiên cứu

Mục đích tôi viết chuyên đề này vừa là để tổng hợp lại những kiến thức, sưu tầm lại hệ thống bài tập về DSU phục vụ cho công tác dạy đội tuyển.

3. Thời gian địa điểm

Đề tại được tôi thực hiện trong năm học tại trường THPT Chuyên Hạ Long.

4. Đòng góp mới về mặt thực tiễn

Chuyên đề đã trình bày hai kĩ thuật cải tiến mới trong cài đặt DSU là: nén lộ trình và hợp bởi thứ hạng. Qua đó, giảm độ phức tạp của hai truy vấn của DSU xuống còn $O(logn), O(\alpha(n))$.

Bên cạnh những ứng dụng phổ biến của DSU như xác định và quản lý các thành phần liên thông trong đồ thị, thành phần liên thông của một bức ảnh, thuật toán Kruskal, SKKN đã áp dụng DSU để giải các bài toán kinh điển như RMQ, tìm thành phần song liên thông, kiểm tra chu trình của đồ thị, bước nhảy trên đoạn con. Qua đó, cho thấy DSU còn có thể được sử dụng thay thế một cách hiệu quả cho cấu trúc dữ liệu nổi tiếng là IT với kĩ thuật Lazy Propagation.

Các bài toán trong các đề thi HSGQG luôn được cập nhật một cách liên tục và có phổ kiến thức rất rộng. Vì vậy, ngoài việc trang bị cho học sinh những kiến thức nền cơ bản thì cần rèn luyện cho học sinh kĩ năng phân loại, đoạn nhận thuật toán.

Để học sinh có thể dễ dạng hơn trong việc đoán nhận dạng toán sử dụng DSU. Trong chuyên đề này tôi đã phân dạng bài tập thành ba dạng chính: Dạng toán liên quan đến thành phần liên thông của đồ thị, dạng toán liên quan đến cây khung nhỏ nhất và một số bài toán khác.

Các bài tập được giới thiệu trong bài viết này đều được lựa chọn kĩ lưỡng có đủ cả test chấm, lời giải và chương trình kèm theo. Vì vậy, chuyên đề sẽ là một tài liệu phục vụ thiết thực cho việc giảng dạy đội tuyển ôn thi học sinh giỏi quốc gia, khu vực, các kì thi online,....

PHẦN II: NỘI DUNG

CHƯƠNG 1: TỔNG QUAN

1.1. Bài toán (*)

Tìm đường đi giữa hai đỉnh bất kì trên đồ thị là bài toán cơ bản nhất nhưng lại đóng một vài trò vô cùng quan trọng trong lý thuyết đồ thị, rất nhiều vấn đề của đồ thị đều bắt nguồn hoặc quy về bài toán này. Chúng ta xét một bài toán cụ thể như sau:

Cho một đồ thị (G) ban đầu đồ thị này rỗng. Bài toán yêu cầu thực hiện q truy vấn có dạng i, u, v:

- Nếu $i = 1$ thêm cạnh (u, v); (v, u) và đồ thị (G).
- Nếu $i = 2$ trả lời hai đỉnh u, v có đường đi đến nhau hay không?

Thuật toán:

Với truy vấn $i = 1$ thông thường được thực hiện với độ phức tạp $O(1)$ nên truy vấn này tạm thời được xem là đã có cánh thực hiện tối ưu. Tuy nhiên, với truy vấn loại $i = 2$ là bài toán tìm đường đi giữa hai đỉnh u, v. Đây là một bài toán cơ bản nhất của đồ thị. Chúng ta có thể sử dụng thuật toán DFS (Depth-first search) hoặc BFS (Breadth-first search). Cả hai thuật toán này đều cho độ phức tạp $O(n)$ (n là số đỉnh của đồ thị).

Vì vậy, nếu sử dụng thuật toán DFS hoặc BFS thì độ phức tạp của thuật toán là $O(q.n)$. Do vậy, thuật toán không khả thi với những trường hợp q, n lớn. Điều này thúc đẩy việc cải tiến thuật toán để giảm độ phức tạp thuật toán xuống $O(q.\log(n))$ hoặc $O(q)$.

Trong phần tiếp theo của chuyên đề, tôi sẽ trình bày cấu trúc dữ liệu Disjoint – Set – Union (DSU) thực hiện bài toán này với độ phức tạp $O(q.\log(n))$ hoặc $O(q.\alpha(n))$.

1.2. Định nghĩa

Trong khoa học máy tính, Cấu trúc dữ liệu Disjoint – Set - Union (hợp của các tập hợp không giao nhau) là một cấu trúc dữ liệu quản lý các phần tử đã được

phân chia thành các tập con không giao nhau. DSU cung cấp các phép toán: thêm các tập con mới, kết hợp các tập con lại với nhau, xác định các phần tử có trong cùng một tập hợp hay không.

Cấu trúc dữ liệu Disjoint – Set - Union có nhiều ứng dụng trong các thuật toán đồ thị đặc biệt là thuật toán Kruskal để tìm cây khung nhỏ nhất.

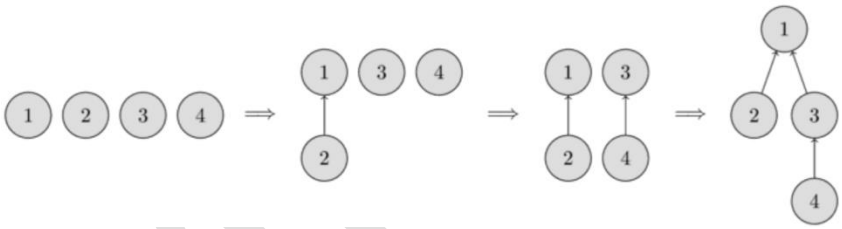
1.3. Các phép toán cơ bản

- Make_set(v): Tạo ra một tập hợp mới để chứa phần tử mới v.
- Union_set(a, b): Hợp nhất hai tập hợp (tập hợp chứa phần tử a và tập hợp chứa phần tử b).
- Find_set(v): Trả về phần tử đại diện của tập hợp mà chứa phần tử v. Phần tử đại diện này được lựa chọn cho mỗi tập hợp và phần tử này có thể thay đổi và được chọn lại sau phép toán Union_set. Phần tử đại diện này được sử dụng để kiểm tra hai phần tử có cùng một tập hợp hay không.

1.4. Cài đặt DSU hiệu quả với cấu trúc cây

Chúng ta sẽ lưu các tập hợp dưới dạng rừng các cây. Mỗi cây sẽ tương ứng với một tập hợp và gốc của cây sẽ là phần tử đại diện của cây đó.

Hình dưới đây minh họa một cách tạo cây



Hình 1. *Trước tiên, mỗi phần tử được cài đặt thành một cây mà gốc chính là phần tử đó. Sau đó chúng ta kết hợp cây chứa đỉnh 1 và cây chứa đỉnh 2 thành cây gốc 1, kết hợp cây chứa đỉnh 3 với cây chứa đỉnh 4 thành cây gốc 3. Cuối cùng chúng ta kết hợp cây gốc 1 với cây gốc 3 thành cây gốc 1.*

a) Cách cài đặt đơn gian (naive)

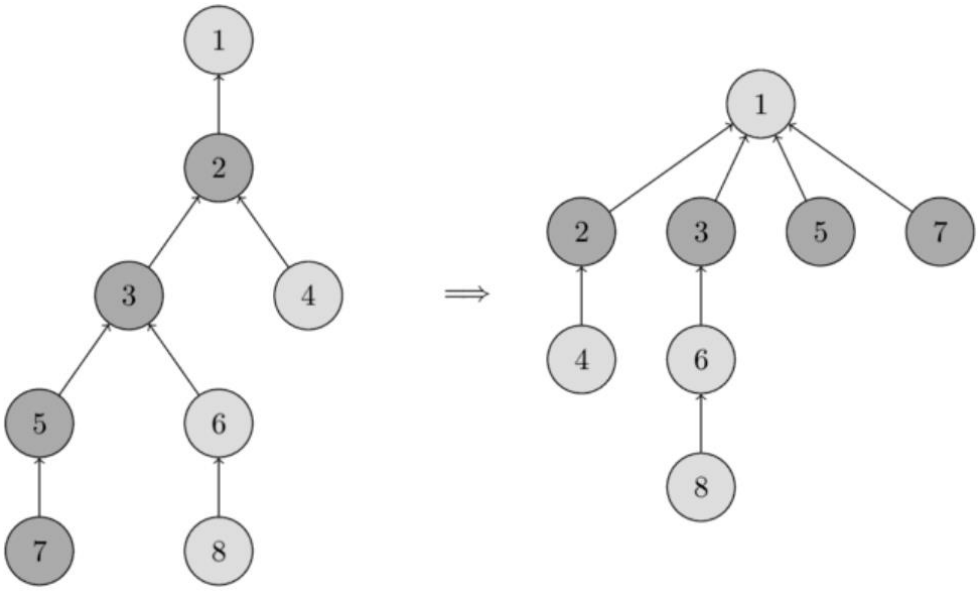
```
void make_set(int v) {
  parent[v] = v;
}
int find_set(int v) {
    if (v == parent[v])
        return v;
    return find_set(parent[v]);
}
void union_sets(int a, int b) {
    a = find_set(a);
```

```
        b = find_set(b);
        if (a != b)
            parent[b] = a;
    }
```

Tuy nhiên, cách cài đặt này là không hiệu quả, chúng ta có thể dễ dàng nhận thấy trong trường hợp cây suy biến thành chuỗi dài thì hàm find_set(v) có độ phức tạp $O(n)$. Sau đây, tôi sẽ trình bày hai cách tối ưu để làm việc hiệu quả hơn.

b) Tối ưu hóa bằng phương pháp nén lộ trình

Bước tối ưu này được thiết kế để tăng tốc hàm find_set(). Nếu chúng ta gọi hàm find_set(v) của một đỉnh v nào đó, thực chất là chúng ta tìm gốc p cho tất cả các đỉnh trên đường đi giữa v và p. Mẫu chốt của bước nén lộ trình ở đây là làm cho các đường đi của các node này ngắn hơn, bằng cách cài đặt cha của mỗi đỉnh đã được thăm trực tiếp là p.



Hình 2. *Minh họa bước nén lộ trình khi gọi hàm Find_set(7). Bến trái là một cây, bên phải nén lộ trình khi gọi hàm Find_set(7), ở đây đã rút ngắn đường đi của các đỉnh.*

Sau đây là cài đặt mới của hàm Find_set():

```
int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
```

Hàm này được thực hiện trước tiên là tìm gốc p của v, rồi sau đó gán lần lượt (theo kiểu stack) gốc trực tiếp của các đỉnh trên đường đi từ v đến p là p. Cách cải

tiến đơn giản này giúp làm giảm độ phức tạp của hàm Find_set() xuống trung bình là $O(logn)$.

c) Tối ưu hóa hàm Union_set bằng size/rank

Trong phần này, chúng ta sẽ tối ưu hàm Union_set(). Nói chính xác hơn chúng ta phải xác định cây nào phải gắn vào cây nào. Trong cách cài đặt đơn gian (naive) đã trình bày ở trên cây hai luôn gắn vào cây một. Trong thực hành cách cài đặt này có thể dẫn đến cây là một chuỗi có độ dài O(n). Trong bước cài tối ưu này chúng ta sẽ lựa chọn cận thận việc lựa chọn cây nào phải ghép vào cây nào.

Có nhiều phương pháp thích nghi (heuristics) có thể được sử dụng. Phương pháp thông dụng nhất hay được dùng là hai cách tiếp cận dưới đây: Cách tiếp cận thứ nhất là chúng ta sử dụng kích thước của cây (còn được gọi là rank), cách tiếp cận thứ hai là chúng ta sử dụng chiều sâu của cây (còn được gọi là size).

- Cách thực hiện của hàm Union_set() dựa trên size của cây:

```
void make_set(int v) {
    parent[v] = v;
    size[v] = 1;
}
void union_set(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (size[a] < size[b])
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

- Cách thực hiện của hàm Union_set() dựa trên chiều sâu của cây:

```
void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}
void union_set(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
```

```
        }
    }
```

Cả hai cách cài đặt này yêu cầu bộ nhớ và độ phức tạp là tương đương vì vậy khi cài đặt dùng một trong hai đều được.

- Độ phức tạp: Như đã đề cập ở trên, nếu chúng ta kết hợp cả hai giải pháp tối ưu là nén lộ trình và hợp nhất bằng size/rank thì chúng ta có thể đạt đến độ phức tạp hằng số cho mỗi truy vấn. Độ phức tạp của mỗi truy vấn đã được chứng mình là $O(\alpha(n))$, trong đó $\alpha(n)$ gần như không lớn hơn 4 với những giá trị n phù hợp ($n \leq 10^{600}$).

1.5. Áp dụng DSU giải bài toán (*)

Như đã trình bày ở **mục 1.1.1** bài toán **(*)** được giải với độ phức tạp $O(q.\log(n))$ là một giải pháp chưa tối ưu. Sau đây tôi sẽ trình bày thuật toán kết hợp với DSU để giảm độ phức tạp của bài toán.

Thuật toán:

- Với truy vấn $i = 1$ dùng hàm $Union\_set()$ với cách cài hợp bởi size
- Với truy vấn $i = 2$ dùng hàm $Find\_set()$ với kĩ thuật nén lộ trình

Độ phức tạp: Với sự kết hợp của cả hai kĩ thuật trên cho thuật toán với độ phức tạp $O(q.\alpha(n))$.

CHƯƠNG 2: ỨNG DỤNG CỦA DSU

2.1. Xác định và quản lý các thành phần liên thông của đồ thị

Đây là một trong những ứng dụng phổ biến nhất của DSU. Thông thường bài toán được định nghĩa như nhau: Ban đầu chúng ta có một đồ thị rỗng, chúng ta phải thêm các đỉnh và các cạnh vô hướng và trả lời các truy vấn có dạng (a, b)- "các đỉnh a và b có cùng thành phần liên thông hay không?"

2.2. Xác định các thành phần liên thông trong một bức ảnh

Bài toán: Có một bức ảnh kích thước $n \times m$ điểm ảnh với hai màu trắng và đen. Hãy xác định kích thước của các thành phần liên thông (các vùng ảnh có điểm ảnh màu đen) trong bức ảnh.

Thuật toán: Xét trên tất cả các điểm ảnh đen, với mỗi điểm ảnh đen chúng ta xét 4 lân cận của điểm ảnh và nếu lân cận là điểm ảnh đen thì gọi hàm Union_set(). Do đó, chúng ta sẽ có DSU với n.m nút tương ứng với các điểm ảnh. Kết quả cuối cùng ta thu được rừng các cây của DSU mà mỗi cây chính là một thành phần liên thông rời rạc.

2.3. Nén bước nhảy trên một đoạn/Tô màu các dãy con offline

Bài toán: Có một tập các đỉnh, mỗi đỉnh có đầu ra với một đỉnh khác. Với DSU chúng ta có thể tìm được điểm cuối cùng trên đường đi từ một đỉnh bắt đầu bất kì qua tất cả các cạnh trong thời gian hằng số.

Một ví dụ cho ứng dụng này là bài toán tô màu cho các dãy con. Chúng ta có một dãy có độ dài L. Ban đầu các phần từ của dãy được tô màu 0. Với mỗi truy vấn (l, r, c), ta tô lại màu cho mỗi đoạn con $[l; r]$ màu c. Sau q truy vấn như trên hãy tìm ra màu cuối cùng của mỗi phần tử.

Thuật toán:

- Tạo DSU, mỗi phần tử tạo một liên kết đến một phần tử khác chưa được tô màu, ban đầu mỗi phần từ được liên kết với chính nó. Khi mỗi đoạn được tô màu, tất cả các ô trong đoạn đó được liên kết đến ô sau đoạn đó.
- Để giải bài toán này, chúng ta đảo ngược lại các truy vấn từ cuối đến đầu. Khi đó, với mỗi truy vấn (l, r, c) ta chỉ phải tô màu các ô trên đoạn $[l, r]$ chưa được tô màu vì tất cả các ô đã được tô màu đều là màu cuối cùng của ô đó. Để lặp lại nhanh trên các ô chưa được tô màu chúng ta dùng DSU. Chúng ta tìm ô bên trái nhất trên đoạn $[l, r]$ chưa được tô màu và tô màu cho nó. Với con trỏ, chúng ta di chuyển ô trống kề tiếp sang bên phải.
- Ở đây chúng ta có thể dùng DSU với nén lộ trình (path compression) nhưng chúng ta không thể dùng Union by rank/size. Do đó, độ phức tạp sẽ là $O(\log(n))$ cho mỗi Union_set

Cài đặt:

```
for (int i = 0; i <= L; i++) {
    make_set(i);
}
for (int i = m-1; i >= 0; i--) {
    int l = query[i].l;
    int r = query[i].r;
    int c = query[i].c;
    for (int v = find_set(l); v <= r; v = find_set(v))
    {
        answer[v] = c;
        parent[v] = v + 1;
    }
}
```

Tối ưu hàm Union_set bằng rank/size: Chúng ta có thể dùng Union by rank/size, nếu chúng ta lưu ô kế tiếp chưa được sơn lại trong một mảng end[]. Thì chúng ta có thể hợp hai tập hợp lại thành một bằng thích nghi (heuristic) vì vậy độ phức tạp của Union_set là $O(\alpha(n))$.

2.4. Xác định khoảng cách đến điểm đại diện

Một vài ứng dụng cụ thể của DSU là duy trì khoảng cách từ đỉnh bất kì đến đỉnh đại diện (độ dài đường đi trên cây từ một node bất kì đến gốc). Nếu chúng ta không dùng nén lộ trình thì khoảng cách chính là số lần gọi đệ quy. Nhưng điều này là không hiệu quả. Tuy nhiên, chúng ta có thể dùng nén lộ trình bằng cách lưu thêm khoảng cách đến cha cho mỗi node.

Trong cài đặt, chúng ta sẽ dùng cấu trúc pair cho mảng parents[] và hàm Function_set() sẽ trả về hai giá trị là phần tử đại diện (gốc) và khoảng cách đến phần tử đại diện (gốc).

```
void make_set(int v) {
    parent[v] = make_pair(v, 0);
    rank[v] = 0;
}
pair<int, int> find_set(int v) {
    if (v != parent[v].first) {
        int len = parent[v].second;
        parent[v] = find_set(parent[v].first);
        parent[v].second += len;
    }
    return parent[v];
}
void union_sets(int a, int b) {
    a = find_set(a).first;
    b = find_set(b).first;
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = make_pair(a, 1);
        if (rank[a] == rank[b])
            rank[a]++;
    }
}
```

2.5. Cây khung nhỏ nhất

Một trong những ứng dụng thường được đề cập đến của cấu trúc dữ liệu Disjoint – Set - Union là áp dụng trong thuật toán Kruskal tìm cây khung nhỏ nhất.

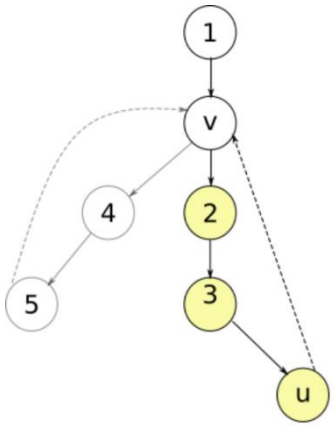
Thuật toán Krukal:

- Sắp xếp các cạnh theo thứ tự không giảm của trọng số các cạnh
- Chọn cạnh có trọng số nhỏ nhất. Kiểm tra cạnh này nếu được thêm vào cây khung đã được tạo trước đó thì có tạo ra chu trình hay không? Nếu tạo ta chu trình thì loại bỏ cạnh này đi. Nếu không tạo thành chu trình thì dùng cấu trúc dữ liệu DSU để hợp nhất cây.
- Lặp lại bước 2 cho đến khi chọn được $n-1$ cạnh tạo thành cây khung.

2.6. Thành phần song liên thông

Có nhiều cách để tìm thành phần song liên thông của đồ thị, cách phổ biến nhất là duyệt theo kiểu Tarjan. Tuy nhiên, trong phần này sẽ trình bày ứng dụng của cấu trúc dữ liệu DSU với kĩ thuật nén lộ trình để tìm thành phần song liên thông của đồ thị.

Thuật toán: Ta thấy các đỉnh cùng nằm trên một chu trình đơn sẽ cùng thuộc một thành phần song liên thông. Do vậy, khi ta duyệt DFS đỉnh u và có cạnh nối ngược lên v, ta thấy các đỉnh $(v, 2, 3, u)$ tạo nên một thành phần song liên thông. Vì vậy, ta sẽ sử dụng DSU để hợp nhất các đỉnh này lại. Mỗi thành phần sau khi hợp nhất được đại diện bởi đỉnh có bậc nhỏ nhất trên cây DFS. Tuy nhiên, do đỉnh v có thể thuộc TP song liên thông khác nên ta chỉ hợp nhất từ đỉnh 2 đến đỉnh u. Sau khi DFS xong, mỗi tập hợp trong disjoint-sets là một TP song liên thông, cộng thêm 1 đỉnh là đỉnh cha của gốc TP song liên thông đó.



Hình 3. *Minh họa các đỉnh $v, 2, 3, u$ được hợp lại thành một thành phần song liên thông.*

Cài đặt:

```
void dfs(int u, const dsk &ke) {
    visited[u] = true; root[u] = u;
```

```
    stack.push_back(u);
        for (int v: ke[u]) if (visited[v]) {
            v = find_set(active[v]);
            while (stack.back() != v) {
                root[find_set(stack.back())] = v;
                stack.pop_back();
            }
        }
        for (int v: ke[u]) if (!visited[v]) {
            active[u] = v;          dfs(v, ke);
        }
        if (stack.back() == u) stack.pop_back();
    }
```

2.7. Giải bài toán RMQ offline với độ phức tạp hằng số

Bài toán: Cho một mảng A gồm n phần tử các số nguyên. Bài toán yêu cầu trả lời Q truy vấn. Mỗi truy vấn yêu cầu đưa ra vị trí có giá trị nhỏ nhất trên đoạn từ i đến j của mảng A. Kí hiệu là $RMQ_A(i;j)$.

Thuật toán:

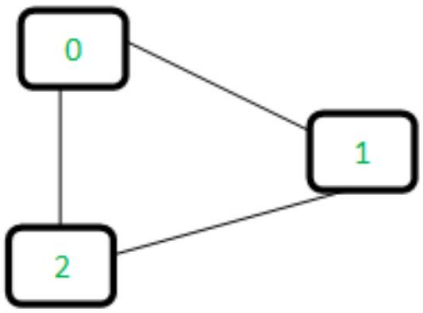
- Đặt $Parent[i] = j$ thỏa mãn j là vị trí đầu tiên về bên phải mà $a[j] < a[i]$, khởi tạo $Parent[i] = i$.
- Sắp xếp lại các truy vấn $(j;r)$ theo chỉ số r.
- Duyệt i từ 1 đến n với mỗi i trả lời tất cả các truy vấn $(l;r)$ mà $r = i$. Kết quả của mỗi truy vấn là sẽ là $a[find\_set(l)]$.
- Với hàm $find\_set()$ kết hợp với kỹ thuật nén lộ trình giúp cập nhật lại $Parent[k], k \in [l;r]$.

Chương trình:

```
stack<int> s;
for (int i = 0; i < n; i++) {
    while (!s.empty() && a[s.top()] > a[i]) {
        parent[s.top()] = i;          s.pop();
    }
    s.push(i);
    for (Query q : container[i]) {
        answer[q.idx] = a[find_set(q.L)];
    }
}
```

2.8. Kiểm tra đồ thị vô hướng có chu trình hay không

Bài toán: Cho một đồ thị vô hướng, các đỉnh không có cạnh nối mỗi đỉnh với chính nó. Kiểm tra đồ thì đã cho có chu trình hay không. Đồ thị dưới đây là một ví dụ về đồ thị có chu trình.



Hình 4. *Minh họa đồ thị có chu trình*

Thuật toán: Bài toán xác định chu trình của một đồ thị là một bài toán rất cơ bản. Đã có nhiều thuật toán xác định chu trình của đồ thị được trình bày. Tuy nhiên, trong chuyên đề này Tôi trình bày một thuật toán mới được trình bày trên https://www.geeksforgeeks.org/union-find/.

Ý tưởng của thuật toán được trình bày như sau:

- Duyệt qua tất cả các cạnh của đồ thị. Với mỗi cạnh (u, v) kiểm tra hai đỉnh u và v có cùng cha hay không?
- Nếu hai đỉnh u, v có cùng cha thì độ thi có chu trình,
- Nếu hai đỉnh u, v không cùng cha thì dùng hàm $Union\_set()$ để hợp nhất.

Chương trình:

```
    for(int i = 0; i < graph->E; ++i)
        {
            int x = find_set(parent, graph->edge[i].src);
            int y = find_set(parent, graph->edge[i].dest);
            if (x == y)
                return 1;
            Union_set(parent, x, y);
    }
```

CHƯƠNG 3. BÀI TẬP

3.1. Dạng toán liên quan đến thành phần song liên thông

Bài 1. Mạng máy tính an toàn (SAFENET2.CPP)

Có n máy tính đánh số từ 1 đến n và m dây cáp mạng, giữa 2 máy tính có thể có một hoặc nhiều đường dây cáp mạng nối chúng, không có cáp mạng nối một máy với chính nó. Hai máy tính có thể truyền dữ liệu cho nhau nếu có đường cáp nối trực tiếp giữa chúng hoặc truyền qua một số máy trung gian.

Một tập S các máy tính được gọi là hệ thống an toàn nếu dù một máy tính bất kỳ bị tấn công thì trong số những máy tính còn lại thuộc tập S vẫn có thể truyền được dữ liệu cho nhau. Xác định số lượng lớn nhất có thể các máy tính của tập S.

Dữ liệu vào:

- Dòng 1 chứa 2 số nguyên $n, m (1 <= n <= 3.10^4;\ 0 <= m <= 10^5)$
- m dòng tiếp theo ghi thông tin về các dây cáp mạng, gồm 2 chỉ số của 2 máy được dây đó nối trực tiếp.

Kết quả:

- Ghi một số nguyên duy nhất là số lượng máy tính lớn nhất tìm được.

Ví dụ:

SAFENET2.INP	SAFENET2.OUT
8 10 1 2 2 3 3 1 1 4 4 5 5 1 1 6 6 7 7 8 8 1	4

Thuật toán:

- Dễ dạng nhận thấy yêu cầu của bài toán chính là tìm thành phần song liên thông có nhiều đỉnh nhất.

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

3.2. Dạng toán liên quan đến thành phần liên thông của đồ thị

Bài 2. Xóa cạnh (Nguồn: Thầy Hiếu)

Cho đồ thị n đỉnh, m cạnh. Đỉnh thứ i có trọng số w_i. Cạnh thứ j nối giữa 2 đỉnh u_j và v_j.

Thực hiện 1 trong 2 truy vấn:

- $D\ i$: xóa cạnh thứ i của đồ thị
- $C\ u\ x$: Thay đổi trọng số đỉnh u thành x $(w_u = x)$

Yêu cầu: Sau mỗi truy vấn, hãy xác định vùng liên thông có tổng trọng số lớn nhất.

Dữ liệu: vào từ file CORRUPTED.INP

- Dòng đầu tiên chứa 3 số nguyên dương n, m, q
- Dòng thứ 2 chứa n số $w_1, w_2, \dots, w_n$ $(w_i \leq 10^9)$.
- m dòng tiếp, dòng thứ i chứa 2 số nguyên $u_i\ v_i$ $(1 \leq u_i, v_i \leq n)$.
- q dòng cuối, mỗi dòng chứa thông tin 1 truy vấn loại 1 hoặc loại 2.

Kết quả: ghi ra file CORRUPTED.OUT gồm q dòng, mỗi dòng một số nguyên là tổng trọng số lớn nhất của các đỉnh trong vùng liên thông tìm được.

CORRUPTED.INP	CORRUPTED.OUT
4 4 1 1 1 1 1 2 2 4 1 4 2 3 D 4 C 3 5 C 1 4 D 2 D 1	3 5 6 6 5

Ràng buộc:

- 30% số test có $n, m, q \leq 1000$
- 30% số test khác, trong test chỉ có truy vấn loại C
- 40% số test còn lại có $n, m, q \leq 2.10^5$

Nhận xét: Bài toán liên quan đến việc xác định và quản lý các thành phần liên thông nên chúng ta có thể nghĩ đến việc sử dụng DSU.

Vì phải đưa ra thành phần liên thông có tổng trọng số của các đỉnh là lớn nhất vì vậy chúng ta sẽ sử dụng thêm màng $f[]$ lưu tổng trọng số của mỗi thành phần liên và hàng đợt ưu tiên.

- Trước tiên, ta sẽ xác định xem có những truy vấn 'D' nào và các truy vấn này xõa đi những cạnh nào của đồ thị . Khi đó, các cạnh còn lại của đồ thị chắc chắn sẽ không bị thay đổi.

  Từ đây chúng ta sẽ sử dụng DSU để xây dựng trước một đồ thị.
- Đến truy vấn cuối cùng thì trọng số của một số đỉnh đã bị thay đổi từ các truy vấn $'C'$ trước đó.

  Ta sẽ giải quyết bài toán bằng cách thực hiện các truy vấn theo thứ tự ngược lại từ cuối về đầu.

Thuật toán:

- Dùng DSU để xây dựng đồ thị dựa trên các cạnh mà chắc chắn không bị xóa. Lưu ý trong quá trình xây dựng đồ thị thì xây dựng mảng $f[]$ lưu tổng trọng số của mỗi thành phần liên thông.
- Thực hiện các truy vấn kiểu $'C'$ để thay đổi trọng số của các đỉnh.
- Duyệt ngược các truy vấn từ cuối về đầu
  - Nếu gặp truy vấn $'D'$ thì lưu lại kết quả TPTL có trọng số max rồi thực hiện $Union\_set(u, v)$.
  - Nếu gặp truy vấn $'C'$ thì cập nhật đỉnh i với trọng số $c[u] - x$.

Chương trình, tets chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

Bài 3. Giao thông an toàn

Đất nước $Alpha$ lên kế hoạch xây dựng 2 hệ thống đường cao tốc và đường tàu điện ngầm. Có tất cả n thành phố. Kế hoạch được chia thành q thời điểm. Tới mỗi thời điểm, chính phủ sẽ xây thêm một đường cao tốc hoặc một đường tàu điện ngầm giữa 2 thành phố nào đó. Để đánh giá độ rủi ro, chính phủ rất cần biết hệ thống 2 đường có cân bằng hay không. Tại mỗi thời điểm, hệ thống 2 đường được coi là cân bằng nếu như u và v có thể đi đến nhau theo đường cao tốc thì u và v cũng có thể đi tới nhau theo đường tàu điện ngầm và ngược lại.

Yêu cầu: Cho thông tin q thời điểm, hãy xác định tính tới thời điểm đang xét, 2 hệ thống đường cao tốc và tàu điện có cân bằng hay không.

Dữ liệu: vào từ file BALANCED.INP

- Dòng đầu tiên chứa 2 số nguyên dương n, q
- Trong q dòng tiếp theo, dòng thứ i chứa 3 số nguyên dương x_i u_i, v_i xác định thông tin xây thêm đường ở thời điểm i. Với $x_i = 1$ tương ứng xây dựng thêm đường cao tốc giữa u_i và v_i. $x_i = 2$ tương ứng với việc xây thêm đường tàu điện ngầm giữa u_i và v_i

Kết quả: Ghi ra file BALANCED.OUT q dòng, dòng thứ i tương ứng ghi Yes/No tương ứng Y/N tương ứng hệ thống đường có cân bằng tại thời điểm i hay không.

Ví dụ:

BALANCED.INP	BALANCED.OUT
7 10 1 1 2 1 2 3 2 1 3 2 1 2 1 3 4 2 2 5 1 4 5 2 1 4 2 6 7 1 6 7	No No No Yes No No No Yes No Yes

Ràng buộc:

- Có 30% số test tương ứng 30% số điểm có $n \leq 20, q \leq 100$
- Có 30% số test khác tương ứng 30% số điểm có $n, q \leq 5000$
- 40% số test còn lại có $n \leq 10^5, q \leq 2.10^5$

Nhận xét: Bài toán có sử dụng các truy vấn và mỗi truy vấn cần kiểm tra hai đỉnh có đường đi đến nhau hay không? Tức là hai đỉnh có cùng thuộc một thành phần liên thông hay không? Vì vậy chúng ta có thể dùng DSU để kiểm tra điều này.

Thuật toán:

- Sub 1: Trâu, xét tường cặp và loang để kiểm tra hai đỉnh có đường đi đến nhau hay không?
- Sub 2: Tại mỗi thời điểm để hợp nhất hai cây chúng ta dùng Union_set(), kiểm tra hai đỉnh có cùng đường đi hay không dùng Find_set().
- Sub 3: Dùng queue lưu trữ những cạnh trong từng cây chưa thỏa mãn cây còn lại. Tăng dần. Nếu 2 queue rỗng →YES

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

Bài 4. Tô màu (PreVNOI 2016)

Một bức ảnh đen trắng được số hóa dưới dạng một bảng các điểm ảnh kích thước $m \times n$. Các hàng điểm được đánh số từ 1 tới m từ trên xuống dưới và các cột điểm được đánh số từ 1 tới n từ trái qua phải. Điểm ảnh nằm trên hàng i, cột j gọi là điểm (i, j). Mỗi điểm ảnh có màu đen (B) hoặc trắng (W).

Hai điểm ảnh được gọi là **thông nhau** nếu chúng cùng màu và đứng cạnh nhau trên cùng hàng hoặc cùng cột. Ta gọi một **miền** là một tập hợp tối đại các điểm cùng màu thỏa mãn: Giữa hai điểm bất kỳ của miền, ta có thể đi từ vị trí điểm này đến vị trí điểm kia qua một số phép di chuyển giữa hai ô thông nhau. Tính tối đại ở đây có nghĩa là việc bổ sung bất kỳ điểm ảnh nào vào **miền** sẽ làm cho tính chất trên bị vi phạm.

Giáo sư X đang muốn kiểm thử hoạt động cho một chương trình tô màu. Chương trình này có thể thực hiện những lệnh dạng $FloodFill(c, i, j)$: Tô màu c cho toàn bộ miền chứa điểm (i, j).

Bởi sau mỗi lệnh $FloodFill$, số lượng và sự phân bố các miền có thể thay đổi. Với một dãy lệnh $FloodFill$, giáo sư X muốn biết số lượng miền và kích thước miền lớn nhất (tính bằng số điểm ảnh) sau mỗi lệnh.

Dữ liệu: Vào từ file văn bản **FFILL.INP**

- Dòng 1 chứa hai số nguyên dương $m, n \leq 1000$ cách nhau bởi dấu cách
  m dòng tiếp theo, dòng thứ i chứa n ký tự $\in \{B, W\}$ liền nhau. Ký tự thứ j là màu của điểm (i, j)
- Dòng tiếp theo chứa số nguyên dương $q \leq 10^5$ là số lệnh $FloodFill$
  q dòng tiếp theo, mỗi dòng chứa ký tự $c \in \{B, W\}$ tiếp theo là hai số nguyên i, j cách nhau bởi
  dấu cách là tham số của một lệnh $FloodFill(c, i, j)$ mô tả như trên ($1 \leq i \leq m$; $1 \leq j \leq n$)

Kết quả: Ghi ra file văn bản FFILL.OUT

- q dòng, dòng thứ i chứa hai số cách nhau bởi dấu cách: Số thứ nhất là số miền trên ảnh, số thứ hai là kích thước miền lớn nhất sau lệnh $FloodFill$ thứ i.

Ví dụ:

Ffill.inp	Ffill.out
3 3 BWB WBW BWB 2 W 2 2 B 2 3	5 5 1 9

Ffill.inp	Ffill.out
2 4 WWBB WBWB 2 W 1 3 W 2 4	2 7 2 7

Ràng buộc:

- 50% số điểm ứng với $m, n, q \leq 200$.

Nhận xét: Khi thực hiện truy vấn tô màu một TPLT (ta chỉ xét truy vấn làm đổi màu TPLT), giả sử từ đen thành trắng, thì nó và tất cả các TPLT kề với nó (hiển nhiên tất cả đều có màu trắng) sẽ bị ghép lại thành một TPLT duy nhất có màu trắng. Do đó, ta phải tìm một CTDL có thể duy trì được các danh sách kề của mỗi TPLT.

Như vậy, bài toán của chúng ta có các yêu cầu sau:

i) Duy trì số TPLT còn lại và màu, kích thước của từng TPLT.
ii) Duy trì danh sách các TPLT kề với mỗi TPLT.
iii) Ghép 2 TPLT lại với nhau, cập nhật lại màu, kích thước và danh sách kề của TPLT mới. (ghép nhiều TPLT lại với nhau bản chất là thực hiện một số lần ghép 2 TPLT).

Thuật toán:

- Chuẩn bị dữ liệu: Trước khi nghĩ đến việc xử lý các truy vấn, ta sẽ biểu diễn bảng một cách rất 'tự nhiên': lưu lại các thành phần liên thông, chỉ số của TPLT mà từng ô thuộc vào, màu và kích thước của các TPLT đó.
- Với mỗi truy vấn tô màu c cho ô (i, j) ta tìm ô đại diện (node gốc) của miền liên thông mà ô (i, j) thuộc vào. Nếu màu của node gốc cũng là c thì bỏ qua truy vấn. Ngược lại, ta sẽ thực hiện như sau:
- Đổi màu của node gốc lại là c sau đó ghép tất cả các thành liên thông liên liền kề có màu là c lại với miền liên thông đang xét. Lưu ý, khi ta ghép $TPLT_A$ vào $TPLT_B$ thì:

  $size(TPLT_B) += size(TPLT_A)$
  $adj(TPLT_B).insert(adj(TPLT_A))$; *cập nhật danh sách kề của* $TPLT_B$
  Xóa $TPLT_A$
- So sánh kích thước của thành phần liên mới được tạo ra với giá trị max hiện tại và cập nhật max.

Ghi chú: Một kĩ thuật rất quan trọng nhưng ít được để ý của Disjoint-Set Unions (DSU). Đó là: Khi mỗi nút của DSU quản lý một tập hợp (ở đây là danh sách kề), để ghép 2 nút lại ta sẽ duyệt các phần tử của tập bé và đưa dần vào tập lớn.

Giả sử chi phí chuyển một phần tử từ tập này sang tập khác là O(1), thì độ phức tạp của **một truy vấn** có thể lên tới O(N); tuy nhiên **tổng độ phức tạp của tất cả các truy vấn** sẽ không vượt quá O(N * $\log_2$(N)). Thật vậy: xét một phần tử X và tập hợp S mà phần tử đó thuộc vào. Giả sử ta thực hiện ghép tập S với tập T. Khi đó có 2 trường hợp:

Nếu |S| > |T| thì ghép T vào S, X không bị di chuyển.

Nếu |S| <= |T| thì ghép S vào T, X có một lần di chuyển. Tuy nhiên, lúc này X sẽ thuộc tập T_new có kích thước bằng |T| + |S| >= 2 * |S|.

Như vậy sau mỗi bước di chuyển, tập hợp chứa X lớn hơn ít nhất gấp đôi tập cũ. Vậy X không thể bị di chuyển quá $\log_2(N)$ lần.

Tuy nhiên, để thuật toán chạy thực sự hiệu quả, ta cần phải lưu các danh sách kề sao cho mỗi phần tử chỉ xuất hiện đúng 1 lần. STL Set là một lựa chọn không tồi vì khá dễ code, tuy nhiên nó sẽ nâng chi phí di chuyển 1 phần tử lên $O(\log_2(N))$. Nếu kĩ năng code tốt và đủ tự tin, ta hoàn toàn có thể lưu danh sách kề bằng bảng băm hoặc Trie, giảm chi phí thêm/xóa phần tử xuống O(1).

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

Bài 5. Tái cơ cấu trúc (Nguồn: Thầy Bình)

Ngay cả những công ty thành công nhất cũng có thể trải qua một giai đoạn khủng hoảng khi phải thực hiện một quyết định khó khăn - tái cơ cấu, loại bỏ, hợp nhất các phòng ban, và những công việc khó chịu khác. Hãy xem xét mô hình sau của một công ty:

Con n người làm việc trong một công ty phần mềm lớn. Mỗi người đều thuộc một phòng nào đó. Đầu tiên, mỗi người có một dự án riêng và ở trong một phòng riêng (khởi đầu công ty có n phòng, mỗi phòng có đúng một người).

Tuy nhiên, thời điểm khắc nghiệt đã đến với công ty và công ty phải thuê một người quản lý khủng hoảng, nhiệm vụ của người này là xây dựng lại quá trình làm việc để tăng tính hiệu quả. Chúng ta sử dụng khái niệm một đội bao gồm nhiều người cùng làm việc. Người quản lý khủng hoảng có thể đưa ra các quyết định thuộc một trong hai loại:

- Nhập phòng của đội chứa nhân viên x với phòng của đội chứa nhân viên y thành một phòng lớn hơn với số nhân viên là số nhân viên của hai phòng nhập lại. Nếu như đội chứa nhân viên x và đội chứa nhân viên y là một thì không có điều gì xảy ra.
- Nhập tất cả các đội chứa nhan viên x, đội chứa nhân viên $x+1$, ..., đội chứa nhân viên y thành một đội lớn hơn. Ở đây x và y $(1 \le x \le y \le n)$ là số hiệu của hai nhân viên nào đó.

Đôi khi người quản lý khủng hoảng tự hỏi xem hai nhân viên x và y $(1 \le x, y \le n)$ có cùng một đội hay không?

Viết chương trình giúp người quản lý khủng hoảng thực hiện các điều trên.

Dữ liệu vào:

- Dòng đầu tiên chứa hai số nguyên n, q $(1 \le n \le 2 \cdot 10^5; 1 \le q \le 5 \cdot 10^5)$ là số nhân viên và số thao tác mà người quản lý khủng hoảng thực hiện.

- q dòng tiếp theo, mỗi dòng mô tả một thao tác gồm ba số $type, x, y$ với $type \in \{1,2,3\}$. Nếu $type = 1$ hoặc $type = 2$ thì đây là thao tác đưa ra quyết định thuộc một trong hai loại. Nếu $type = 3$ thì đây là thao tác xác định xem x, y có thuộc cùng một đội hay không? (chú ý rằng trong mọi trường hợp có thể xảy ra $x = y$)

Kết quả:

- Với các thao tác $type = 3$ in ra 'YES' hoặc 'NO' (không có dấu nháy) tùy theo hai người tương ứng có cùng một đội hay không?

Ví dụ:

RESTRUCT.INP	RESTRUCT.OUT
8 6 3 2 5 1 2 5 3 2 5 2 4 7 2 1 2 3 1 7	NO YES YES

Ràng buộc:

- Subtask 1(30%): $n, q \leq 1000$
- Subtask 2 (20%): Chỉ có quyết định loại 1; $n, q \leq 2.10^5$
- Subtask 3 (20%): Chỉ có quyết định loại 2; $n, q \leq 2.10^5$
- Subtask 3 (30%): $n, q \leq 2.10^5$

Thuật toán:

Ta sử dụng cấu trúc dữ liệu "Disjoint Set" (trong thuật toán Kruskal) để giải quyết bài toán này.

Xây dựng đồ thị với các đỉnh là các nhân viên (n đỉnh)

- Mảng int a[...] duy trì các thành phần liên thông của đồ thị
- Mảng int b[...] với ý nghĩa b[u]=v ($v \geq u$) với ý nghĩa là các đỉnh $u, u+1, \dots, v$ đã được nhập với nhau thành một thành phần liên thông

Khởi đầu $a[i] = b[i] = i$. Lần lượt xét hai loại thao tác:

Với thao tác 1 x y: Đơn giản chỉ là nhập đỉnh x với đỉnh y vào cùng một thành phần liên thông

```
void JoinA(x,y) {
    x=TimA(x), y=TimA(y);
    if (x!=y) a[x]=y;
}
```

Với thao tác 2 x y:

```
while (1) {
     z=TimB(x)+1; // z là đỉnh đầu tiên >x mà chưa được
     nhập vào x nhờ thao tác 2
    if (z>y) break;
    b[z-1]=z;
    JoinA(z-1,z);
    x=z;
}
```

Nhắc lại hai hàm TimA và TimB:

```
int TimA(u) {
    if (a[u]==u) return u;
    a[u]=TimA(a[u]);
    return a[u];
}
int TimB(u) {
    if (b[u]==u) return u;
    b[u]=TimB(b[u]);
    return b[u];
}
```

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

Bài 6. Kho báu (Nguồn: FreeContest)

Có N chiếc rương kho báu, chiếc rương thứ i có giá trị là C_i. Ban đầu, có K chìa khóa. Chiếc chìa khóa thứ i có thể được dùng để mở một trong hai chiếc rương A_i và B_i (lưu ý là không thể dùng chiếc chìa thứ i để mở cả hai rương A_i và B_i). Mỗi chiếc rương chỉ có thể được mở khóa một lần, và không nhất thiết phải sử dụng tất cả các chìa khóa.

Cho K truy vấn, truy vấn thứ i yêu cầu bỏ đi chiếc chìa khóa thứ P_i (các chìa khóa không được đánh số lại sau các truy vấn). Sau mỗi truy vấn, hãy cho biết: giả sử ta dùng các chìa khóa còn lại để mở rương, thì với cách mở khóa rương tối ưu, tổng giá trị kho báu lớn nhất thu được từ các rương được mở là bao nhiêu.

Dữ liệu:

- Dòng đầu tiên gồm ba số nguyên N, K ($2 \leq N \leq 200000, 1 \leq K \leq 200000$) — số rương kho báu, số chìa khóa đồng thời là số truy vấn.

- Dòng tiếp theo, gồm N số nguyên $C_1, C_2, \ldots, C_N$ ($1 \leq C_i \leq 10^9$) — giá trị của các rương kho báu.
- K dòng tiếp theo, dòng thứ i gồm hai số nguyên A_i và B_i ($1 \leq A_i, B_i \leq N, A_i \neq B_i$) — mô tả chiếc chìa khóa thứ i.
- Dòng tiếp theo, gồm K số nguyên phân biệt $P_1, P_2, \ldots, P_K$ ($1 \leq P_i \leq K$) — mô tả các truy vấn.

Kết quả:

- In ra K dòng, dòng thứ i gồm một số nguyên là tổng giá trị kho báu lớn nhất thu được sau khi thực hiện truy vấn thứ i.

Ví dụ:

TREASURE.INP	TREASURE.OUT
4 4 9 5 7 2 2 3 1 1 4 4 1 2 3	21 16 9 0

Giải thích:

- Sau truy vấn thứ nhất, còn lại các chìa khóa 1, 2 và 3. Ta có thể dùng chìa 1 để mở rương 2, chìa 2 để mở rương 3 và chìa 3 để mở rương 1. Tổng giá trị của các rương được mở là 9+5+7=21.
- Sau truy vấn thứ hai, còn lại các chìa khóa 2 và 3. Ta có thể dùng chìa 2 để mở rương 3, chìa 3 để mở rương 1. Tổng giá trị của các rương được mở là 9+7=16.
- Sau truy vấn thứ ba, chỉ còn lại chìa khóa 3. Ta dùng chìa khóa này để mở rương 1 với giá trị 9.
- Sau truy vấn thứ tu, ta không còn chiếc chìa khóa nào nên không mở được bất kì rương kho báu nào.

Ràng buộc:

- Subtask 1 (20% số điểm): $N,K \leq 16$,
- Subtask 2 (30% số điểm): $N,K \leq 2000$,
- Subtask 3 (50% số điểm): Không có ràng buộc gì thêm.

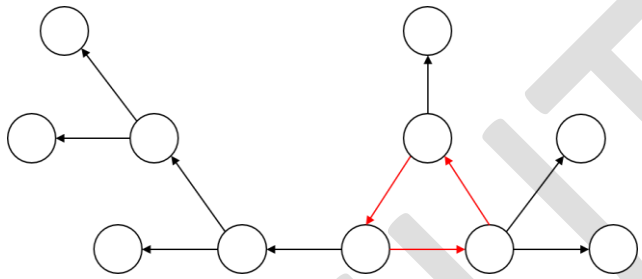
Thuật toán:

Trước hết, ta xét bài toán tìm cách mở rương tối ưu với đầy đủ K chìa khoá. Ta cần xây dựng một đồ thị N đỉnh, với đỉnh i đại diện cho chiếc rương thứ i. Với chìa khóa thứ i, ta thêm cạnh giữa đỉnh A_i và đỉnh B_i. Ta xem việc dùng chìa khóa

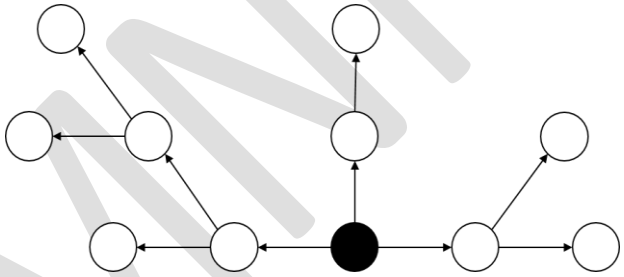
thứ i mở rương A_i hay B_i tương đương với việc định chiều cạnh i. Khi đó, chiếc rương thứ i sẽ được mở nếu ít nhất một cạnh đi vào đỉnh i.

Khi đó, xét từng thành phần liên thông (TPLT):

- Nếu TPLT có tồn tại chu trình $p_1,p_2,...,p_k$, thì ta có thể định hướng các cạnh trong chu trình theo chiều $p_1 \to p_2 \to p_3 \to ... \to p_k \to p_1$. Sau đó, ta dùng một thuật toán DFS, xuất phát từ các đỉnh trong chu trình, và khi ta thăm đỉnh v từ đỉnh u thì định hướng cạnh $u \to v$. Khi đó, các đỉnh trong TPLT đều có ít nhất một cạnh đi vào.



- Nếu TPLT hiện tại không có chu tình, ta xuất phát từ đỉnh đại diện cho kho báu có giá trị thấp nhất trong TPLT (gọi đỉnh này là r) và thực hiện thuật toán DFS tương tự như trên. Khi đó, các đỉnh trong TPLT trừ r đều có ít nhất một cạnh đi vào.



Do đó, đáp án sẽ là tổng giá trị các kho báu trừ đi tổng giá trị nhỏ nhất của các kho báu trong mỗi TPLT.

Ta trở lại với bài toán ban đầu. Thao tác xóa chìa khóa tương ứng với việc xóa cạnh trong đồ thị. Ta sẽ thực hiện các truy vấn theo thứ tự ngược lại: ban đầu, đồ thị không có cạnh nào, và mỗi truy vấn sẽ thêm một cạnh vào đồ thị.

Khi đó, ta có thể giải quyết bài toán bằng disjoint-set union (DSU). Với mỗi thành phần liên thông, ta sẽ lưu thêm thông tin về đỉnh có giá trị nhỏ nhất trong TPLT, và TPLT đó có chu trình hay không. Các bạn có thể xem bài giải mẫu để hiểu rõ hơn chi tiết cài đặt.

Độ phức tạp: $O(N + K \log K)$ hoặc $O(N + K\alpha(K))$ (với $\alpha(K)$ là nghịch đảo hàm Ackermann, có thể xem như hằng số), tùy vào cách cài đặt DSU.

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

3.3. Dạng toán liên quan đến cây khung

Bài 7. Hệ thống điện (Nguồn: FreeContest)

Đất nước Free Contest gồm N thành phố được đánh số từ 1 đến N. Có M đường dây dẫn có thể xây dựng được, đường dây dẫn thứ i kết nối hai thành phố U_i và V_i với chi phí xây dựng là W_i.

Chính phủ của đất nước Free Contest có kế hoạch xây dựng lưới điện quốc gia để cung cấp điện cho toàn bộ các thành phố. Họ dự định sẽ đặt hai trạm phát điện tại hai thành phố khác nhau, và xây dựng một số đường dây dẫn để các thành phố đều được cung cấp điện. Một thành phố u được cung cấp điện nếu như thành phố u được đặt trạm phát điện, hoặc có một đường dây dẫn nối thành phố u với một thành phố khác được cung cấp điện.

Chính phủ đã đề xuất Q phương án đặt hai trạm phát điện. Với phương án thứ i, hai trạm phát điện sẽ được đặt lần lượt tại hai thành phố A_i và B_i. Với mỗi phương án, họ cần tính tổng chi phí tối thiểu để xây dựng các đường dây dẫn sao cho các thành phố đều được cung cấp điện.

Bạn, một lập trình viên xuất sắc của đất nước Free Contest, được chính phủ tin cậy và giao cho nhiệm vụ này. Hãy hoàn thành nó một cách xuất sắc nhé!

Dữ liệu:

- Dòng đầu tiên gồm hai số nguyên N, M ($1 \leq N \leq 4000, 1 \leq M \leq 400000$) - số thành phố của đất nước Free Contest và số đường dây dẫn có thể xây dựng.
- M dòng tiếp theo, mỗi dòng gồm ba số nguyên U_i, V_i và W_i ($1 \leq U_i, V_i \leq N, U_i \neq V_i, 1 \leq W_i \leq 10^9$) mô tả đường dây dẫn thứ i. Dữ liệu vào đảm bảo, nếu xây dựng toàn bộ M đường dây, từ thành phố bất kì đều có thể truyền điện đến một thành phố khác thông qua các đường dây dẫn.
- Dòng tiếp theo gồm một số nguyên Q ($1 \leq Q \leq 200000$) - số phương án chính phủ đã đề xuất.
- Q dòng tiếp theo, mỗi dòng gồm hai số nguyên A_i và B_i ($1 \leq A_i, B_i \leq N, A_i \neq B_i$) mô tả phương án thứ i.

Kết quả:

- Với mỗi phương án, in ra một số nguyên duy nhất là tổng chi phí tối thiểu xây dựng các đường dây dẫn sao cho mỗi thành phố đều được cung cấp điện.

Ràng buộc:

- Subtask 1 (10% số điểm): $N, M \leq 15, Q \leq 100$
- Subtask 2 (25% số điểm): $Q = 1$
- Subtask 3 (40% số điểm): $Q \leq 3000$

- Subtask 4 (25% số điểm): Không có ràng buộc gì thêm

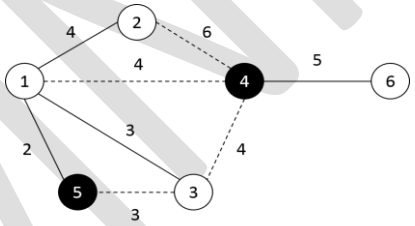
Ví dụ:

ELECTRIC.INP	ELECTRIC.OUT
6 8 1 2 4 1 3 3 1 4 4 5 2 4 6 5 3 4 4 6 5 2 4 5 6 4	14 13

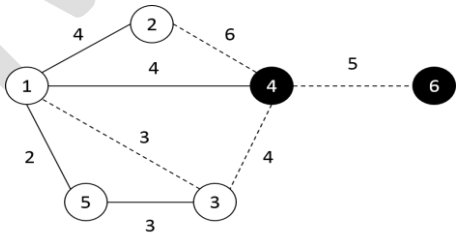
Giải thích:

Hình vẽ minh họa ví dụ thứ nhất (cạnh nét đứt biểu diễn các đường dây dẫn có thể xây dựng, cạnh nét liền biểu diễn các đường dây dẫn cần xây dựng, đỉnh màu đen biểu diễn thành phố được đặt trạm phát điện).

Phương án thứ nhất:



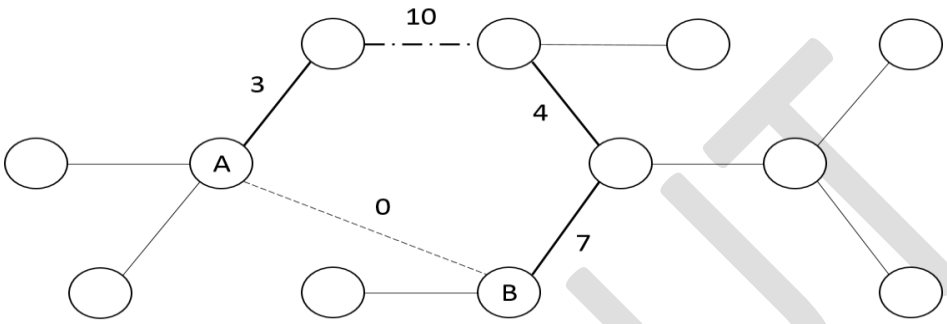
Phương án thứ hai:



Thuật toán:

Trước hết, với phương án đặt trạm điện tại hai thành phố A và B, ta có thể xem như chỉ đặt trạm điện tại thành phố A, và có thể xây dựng đường dây điện nối giữa A và B với chi phí 0. Từ đó, ta nghĩ ra được thuật toán cho hai subtask đầu tiên: bổ sung thêm cạnh $(A, B, 0)$ và tìm cây khung cực tiểu (minimum spanning tree) của đồ thị $M + 1$ cạnh bằng thuật toán Kruskal. Thuật toán trên có độ phức tạp $O(QM \log(N + M))$.

Để cải tiến thuật toán trên, ta cần hiểu bản chất của thuật toán Kruskal. Trước hết, ta sẽ tìm cây khung cực tiểu T của đồ thị M cạnh. Nhận xét rằng, khi bổ sung một cạnh có trọng số 0 giữa hai đỉnh A và B, cạnh này chắc chắn sẽ nằm trong cây khung tối thiểu. Khi đó, đường đi giữa A và B trong T và cạnh (A, B) sẽ tạo thành một chu trình. Do thuật toán Kruskal xét các cạnh theo trọng số từ nhỏ đến lớn, cạnh có trọng số lớn nhất trên đường đi giữa A và B sẽ bị bỏ ra khỏi cây khung cực tiểu.



Do đó, gọi W là tổng trọng số của các cạnh trong T, $maxW(u,v)$ là cạnh có trọng số lớn nhất trên đường đi giữa u và v trên T. Khi đó, với truy vấn i, đáp án sẽ là $W - maxW(A_i,B_i)$.

Đến đây, ta đã có lời giải độ phức tạp $O(M \log(N + M) + QN)$, đủ để vượt qua subtask 3. Để được trọn vẹn điểm bài này, ta cần thực hiện việc tính toán trước $maxW(u,v)$ với mọi cặp (u,v) trong $O(N^2)$ bằng cách DFS từ từng đỉnh. Khi đó, mỗi truy vấn có thể được trả lời trong $O(1)$.

Độ phức tạp: $O(M \log(N + M) + N^2 + Q)$.

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

Bài 8. Đơn giản hóa trang trại (Nguồn: FreeContest)

Nông dân John (FJ) đang tham gia một lớp học thuật toán buổi tối ở một trường đại học gần nhà và ông ta vừa học xong thuật toán tìm cây khung có tổng trọng số nhỏ nhất. Chính vì vậy, FJ nhận ra rằng thiết kế trạng trại của mình không hiệu quả như nó có thể và ông ta muốn đơn giản hóa sự bố trí trong trang trại của ông ta.

Trang trại được xây dựng như một đồ thị với các đỉnh đại diện cho các cánh đồng và các cạnh đại diện cho đường đi giữa các cánh đồng, mỗi đường đi có một độ dài xác định. FJ cũng ghi chú rằng với mỗi độ dài khác nhau có tối đa ba đường đi có độ dài này. FJ muốn bỏ một số đường đi trong trang trại của ông ta để nó trở thành một cây - có nghĩa là chỉ có một đường đi duy nhất giữa hai cánh đồng. Hơn nữa FJ lại muốn cây này trở thành cây khung có tổng trọng số nhỏ nhất.

Hãy giúp FJ tính toán tổng độ dài các cạnh trên cây khung có tổng trọng số nhỏ nhất được xây dựng từ đồ thị tạo bởi trang trại của ông ta và số cách khác nhau để tạo ra cây khung này.

Dữ liệu vào:

- Dòng đầu tiên gồm hai số nguyên N, M ($1 \le N \le 4.10^4, 1 \le M \le 10^5$) biểu diễn số đỉnh và số cạnh của đồ thị. Các đỉnh được đánh số từ 1 đến N
- M dòng tiếp theo, dòng thứ i ghi ba số a_i, b_i và L_i ($1 \le a_i, b_i \le N, 1 \le L_i \le 10^6$) biểu thị có một cạnh nối a_i với b_i có chiều dài L_i. Không có quá ba cạnh có cùng chiều dài.

Kết quả:

- Hai số nguyên. Số thứ nhất là tổng trọng số của cây khung nhỏ nhất và số thứ hai là số lượng cách khác nhau tạo ra cây khung nhỏ nhất (chỉ lấy phần dư khi chia cho $10^9 + 7$)

Ví dụ:

SIMPLIFY.INP	SIMPLIFY.OUT
4 5 1 2 1 3 4 1 1 3 2 1 4 2 2 3 2	4 3

Thuật toán:

Bài toán giải quyết được nếu như chúng ta đếm được số các cây khung tối tiểu khác nhau. Nhận xét rằng mỗi giá trị khoảng cách chỉ có tối đa ba cạnh có độ dài như vậy. Nên trong thuật toán Kruskal, chúng ta ghi nhận giá trị này có bao nhiêu cạnh tham gia cây khung nhỏ nhất. Các trường hợp sau xảy ra:

TH1: Giá trị chỉ có 1 cạnh tham gia cây khung: Lần lượt thử Disjoint Set từng cạnh có giá trị này một cách độc lập để tạo cây khung. Có bao nhiêu lần thao tác nhập được thực hiện thì kết quả sẽ nhân lên bấy nhiêu (1, 2 hoặc 3).

TH2: Giá trị có 2 cạnh tham gia cây khung: Lấy các tổ hợp chập 2 của số cạnh co giá trị thử Disjoint Set, có bao nhiêu cặp tạo cây cuối cùng thì nhân bấy nhiêu (1, 2 hoặc 3).

TH3: Giá trị có 3 cạnh tham gia cây khung: Nhân với 1.

Chương trình, test chấm:

https://www.dropbox.com/sh/o5unxwd62o6zvh4/AAAiXpSZNl7wpHGoobi34Wara?dl=0

3.4. Dạng khác

Bài 9. Tiệc khiêu vũ (Nguồn: Codefoce)

Mehrad muốn mời một số thiếu nữ đến cung điện để tham gia buổi khiêu vũ. Mỗi thiếu nữ có cân nặng là w_i và vẻ đẹp là b_i. Mỗi thiếu nữ cũng có thể có vài người bạn của riêng họ. Những thiếu nữ được chia thành từng nhóm bạn. Thiếu nữ x và thiếu nữ y được chia vào cùng một nhóm khi và chỉ khi tồn tại $a_1, a_2, \dots, a_k$ mà a_i và a_{i+1} là bạn $(1 \le i < k)$, $a_1 = x, a_k = y$.

Arpa cho phép Mehrad sử dụng giảng đường hoàng gia để tổ chức bữa tiệc. Giảng đường này chỉ chịu được cân nặng tối đa là w.

Merhad tham lam đến mức anh muốn mời các thiếu nữ mà cân nặng của họ không vượt quá w và vẻ đẹp của họ đạt lớn nhất. Tuy vậy, với nhóm bạn, Merhad chỉ có thể mời cả nhóm hoặc không mời quá 1 người. Nếu không, những thiếu nữ khác có thể sẽ bị tổn thương. Hãy giúp Merhad tìm vẻ đẹp lớn nhất của các thiếu nữ mà anh ta có thể mời mà không làm thiếu nữ nào tổn thương và cân nặng của họ không vượt quá w.

Dữ liệu vào:

- Dòng đầu tiên nhập vào các số n, m, w - số thiếu nữ, số cặp tình bạn và cân nặng tối đa mà giảng đường chịu được. $1 \le n \le 1000, 0 \le m \le \min\left(\frac{n(n-1)}{2}, 10^5\right), 1 \le w \le 1000$.
- Dòng thứ hai nhập vào n số $w_1, w_2, \dots, w_n$ $(1 \le w_i \le 1000)$ - cân nặng của mỗi thiếu nữ.
- Dòng thứ ba nhập vào n số $b_1, b_2, \dots, b_n$ $(1 \le b_i \le 10^6)$ - vẻ đẹp của mỗi thiếu nữ.
- m dòng tiếp theo là các cặp tình bạn, dòng thứ i chứa cặp số nguyên x_i và y_i $(1 \le x_i, y_i \le n, x_i \ne y_i)$, x_i và y_i là bạn. Lưu ý rằng A là bạn B thì B cũng là bạn A, các cặp bạn đôi một khác nhau.

Kết quả:

- In ra vẻ đẹp tối đa có thể của các thiếu nữ mà Mehrdad có thể mời để không ai bị tổn thương và tổng trọng lượng không vượt quá w.

Ví dụ:

PARTY.INP	PARTY.OUT
3 1 5 3 2 5 2 4 2 1 2	6

PARTY.INP	PARTY.OUY
4 2 11 2 4 6 6 6 4 2 1 1 2 2 3	7

Giải thích ví dụ:

- Ví dụ 1, ta có $\{1, 2\}$ là 1 nhóm bạn và $\{3\}$ là nhóm bạn riêng. Cách chọn tốt nhất là mời cả nhóm nhóm $\{1, 2\}$ cân nặng của họ bằng 5 và vẻ đẹp bằng 6.

- Ví dụ 2, ta có $\{1, 2, 3\}$ là 1 nhóm bạn và $\{4\}$ là nhóm bạn riêng. Merhad không thể mời của nhóm $\{1, 2, 3\}$ vì cân nặng của họ là 12 vượt quá 11.Nên cách chọn tốt nhất là 1 người ở nhóm 1 và người 4 – chọn người 1 và người 4, cân nặng của họ bằng 8 và vẻ đẹp bằng 7.

Thuật toán:

- Đầu tiên, chúng ta cần sắp xếp các cô gái thành các nhóm bạn như điều kiện của đề bài.Việc lập nhóm là quan trọng vì mỗi nhóm này riêng biệt và có cách chọn khác nhau trong từng nhóm khác nhau. Trong trường hợp nếu anh ta chọn cả nhóm thì chúng ta cần phải biết nhóm đấy có những ai hoặc nếu phải chọn người đại diện thì ta cũng cần phải biết nhóm đấy có những cô gái nào.
- Coi mỗi người bạn như 1 đỉnh của đồ thị thì ta dễ dàng sử dụng DSU để gộp các đỉnh có chung 1 gốc lại với nhau. Hai người bạn với nhau đều này đồng nghĩa với việc là 2 đỉnh này có cạnh chung nối đến với nhau.

  Đpt O(m+log(n)).
- Khi hoàn thành xong bước lập nhóm, ta nhận thấy rằng n có giới hạn khá bé $(n <= 1000)$ nên ta sẽ đoán rằng thuật toán sẽ gồm 2 for.
- Nhận ngay rằng cân nặng tối đa của bài toán đưa ra là $w <= 1000$.Ta có thể dần nghĩ ngay ra bài toán là một bài toán quy hoạch động dựa trên cân nặng.
- Yêu cầu của đề bài là tìm vẻ đẹp tối đa mà cân nặng lại không vượt quá w. Ta gọi $dp[i]$ là vẻ đẹp lớn nhất mà các thiếu nữ có thể đạt được mà không vượt quá i. Từ đó ta sẽ quy hoạch động để tìm ra $dp[w]$ là đáp án cần in ra. Dễ nhận thấy ngay công thức quy hoạch động sẽ là:
- $dp[i] = max(dp[i], dp[i - w[j]] + b[j])$
  + j là chỉ số của các thiếu nữ trong nhóm.
  + $i = w \rightarrow 0$.

Đpt: **O(2*w*n)**.

Bài 9. H - Height Preservation (Nguồn: ACM ICPC HCMC 2017)

Thực tế ảo (VR) là một trong những xu hướng mới gần đây. Khi dùng thiết bị VR, bạn sẽ được trải nghiệm trong một môi trường ảo về giáo dục, giải trí,…

Để nâng cao trải nghiệm của người dùng, ICPC thiết lập phòng VR để người dùng đi và khám phá môi trường ảo. Với phòng VR, người dùng có thể nhìn thấy cảnh ảo thông qua thiết bị VR và có thể cảm nhận độ dốc của địa hình ảo khi đi trong đó.

Phòng VR bao gồm m × n ô (m hàng và n cột). Chiều cao thực của mỗi ô có thể được điều chỉnh để chúng ta có thể mô phỏng địa hình ảo. Ô (i,j) có độ cao thực $S(i,j)$ và tương ứng với độ cao ảo $H(i,j)$ trong cảnh ảo. Ý tưởng chính của việc mô phỏng địa hình ảo là chỉ bảo toàn thứ tự tương đối của chiều cao ô trong mỗi hàng và mỗi cột:

- Trong mỗi hàng i $(1 \le i \le m), S(i,j) = S(i,k)$ nếu $H(i,j) = H(i,k)$ và $S(i,j) > S(i,k)$ nếu $H(i,j) > H(i,k)$.
- Trong mỗi cột j $(1 \le j \le n), S(i,j) = S(k,j)$ nếu $H(i,j) = H(k,j)$ và $S(i,j) > S(k,j)$ nếu $H(i,j) > H(k,j)$.

Yêu cầu: Cho chiều cao ảo của tất cả các ô, nhiệm vụ của bạn là xác định số lượng ít nhất các mức chiều cao thực khác nhau.

Dữ liệu đầu vào:

- Dòng đầu tiên chứa hai số nguyên dương: m và n, tương ứng với số lượng hàng và cột $(m \times n \le 10^6)$.
- Dòng thứ i trong m hàng chứa n số nguyên dương $H(i,j) \le 10^9$ tương ứng với chiều cao ảo của các ô trong hàng thứ i.

Kết quả:

- Đưa ra số nguyên là số số lượng ít nhất các mức chiều cao thực khác nhau trong phòng VR.

Thuật toán:

Trước hết, ta sẽ gộp các ô có giá trị giống nhau ở cùng một dòng hoặc cùng một cột vào một thành phần liên thông (ta gọi là siêu đỉnh). Việc tìm các cặp ô để gộp có thể thực hiện bằng cách duyệt từng dòng, sắp xếp các ô trên từng dòng theo thứ tự tăng dần, và thêm cạnh giữa hai ô trên cột j và cột j+1 của dòng hiện tại nếu số trên hai ô này bằng nhau (Với từng cột cũng làm tương tự). Việc gộp ô có thể làm bằng DSU hoặc DFS.

Sau đó, ta duyệt từng dòng, sắp xếp các ô trên dòng theo thứ tự tăng dần, rồi thêm cạnh một chiều nối từ siêu đỉnh chứa ô trên cột j đến siêu đỉnh chứa ô trên cột j+1 của dòng hiện tại nếu số trên cột j nhỏ hơn số trên côt j+1. Ta làm tương tự như vậy với từng cột.

Cuối cùng, ta sẽ được một DAG trên các siêu đỉnh. Đáp số cần tìm chính là số đỉnh trên đường đi dài nhất của DAG này. Bài toán tìm độ dài đường đi dài nhất trên DAG là một bài toán quy hoạch động kinh điển, có thể giải được trong O(N+M) (với N là số đỉnh trong DAG, M là số cạnh trong DAG).

Độ phức tạp: O(N*M*log(N+M)).

Chấm bài tại: https://hochiminh17.kattis.com/problems/heightpreservation

PHẦN III: KẾT LUẬN

Disjoint – Set – Union là một cấu trúc dữ liệu được sử dụng phổ biến trong các bài toán Tin học trong các cuộc thi HSG Quốc gia, ACM,… đặc biệt là các bài toán liên quan đến đồ thị. Những kĩ thuật cải tiến trong DSU giúp chúng ta trả lời các truy vấn: hai phần tử có cùng thuộc một tập hợp hay không, kết hợp các tập hợp rời nhau trong độ phức tạp $O(logn), O(\alpha(n))$.

Trong chuyên đề này, tôi đã trình bày nội dung lý thuyết căn bản nhất, một số ứng dụng phổ biến nhất về DSU. Đặc biệt là một số lớp bài toán thể hiện rõ được tầm quan trong của DSU.

Trong thời gian tới tôi hy vọng sẽ nhận được những đóng góp quý giá của quý Thầy Cô, đồng nghiệp để bài viết ngày càng được tốt hơn.

PHẦN IV: TÀI LIỆU THAM KHẢO:

[1]. Hồ Sĩ Đàm, Đỗ Đức Đông, Lê Minh Hoàng, Nguyễn Thanh Tùng, "Tài liệu chuyên tin – quyển 1, 2, 3", NXB Giáo dục Việt Nam, 2009.

[2]. Lê Minh Hoàng, "*Cấu trúc dữ liệu và giải thuật*", NXB Đại học Sư Phạm Hà Nội, 1999 - 2004.

[4]. Lê Minh Hoàng, "*Bài giảng Trại hè Tin học*", THPT chuyên Bắc Ninh , 2017.

[5]. https://cp-algorithms.com/data_structures/disjoint_set_union.html

[6]. https://kupc2017.contest.atcoder.jp/tasks/kupc2017_e

[7]. https://atcoder.jp/contests/cf16-tournament-round1-open/tasks/asaporo_c

[8]. https://www.geeksforgeeks.org/union-find/.