```
let menu = ["Home", "Algorithms", "CodeHub", "VNOI Statistics"];
```

Disjoint Set Union - Cấu trúc các tập không giao nhau

# Disjoint Set Union - Cấu trúc các tập không giao nhau

**Contents**



Disjoint Set Union (DSU) là một cấu trúc dữ liệu hỗ trợ các thao tác sau:

- Tìm xem X thuộc tập hợp nào.
- Gộp 2 tập hợp A B lại làm một.

Xem đề bài IOIBIN để hiểu rõ hơn các thao tác ở trên.

DSU còn có các tên gọi khác như Disjoint-Set Data Structure, Union-Find.

Ta xem mỗi tập hợp là một cây, như vậy DSU là một rừng gồm nhiều cây. Để cho đơn giản thì ban đầu mỗi tập hợp chỉ có một phần tử, và ta quy ước nếu X là gốc của cây thì `X.cha = X`.

Ta có thể cài đặt phép hợp và phép tìm tập như sau (mã giả):

```
function Tìm(X)
    Nếu X.cha == X
        return X
    Ngược lại
        return Tìm(X.cha)

function Hợp(X, Y)
    A = Tìm(X)
    B = Tìm(Y)
    Nếu A == B
        return
    A.cha = B
```

Có thể thấy thao tác tìm trong cách cài đặt trên không hiệu quả, mỗi lần tìm có độ phức tạp là $O(N)$. Nên ta có các cải tiến sau.

## Cải tiến nén đường

Sau khi tìm được tập hợp chứa X (gọi là A), ta có thể gán lại `X.cha = A` để các lần tìm sau không cần phải lặp lại việc tìm kiếm đã được thực hiện trước đó.

Thao tác tìm mới như sau:

```
function Tìm(X)
    Nếu X.cha == X
        return X
    Ngược lại
        X.cha = Tìm(X.cha)
        return X.cha
```

## Cải tiến hợp dùng hạng

Trong cải tiến này ta tìm cách để giảm độ sâu của cây. Sử dụng một biến `rank` để quản lý độ sâu cây (lưu ý là chỉ quản lý một cách tương đối, do đã sử dụng kĩ thuật nén đường ở trên nên không thể biết chính xác độ sâu cây), ta sẽ gán lại cây nhỏ hơn làm con của cây lớn hơn. Thao tác hợp mới:

```
function Hợp(X, Y)
    A = Tìm(X)
    B = Tìm(Y)
    Nếu A == B
        return

    Nếu A.rank == B.rank
        Tăng A.rank

    Nếu A.rank < B.rank
        A.cha = B
    Ngược lại
        B.cha = A
```

## Cài đặt

Trong thực tế ta sử dụng 2 mảng `cha` và `rank` để code đơn giản.

```
int cha[N], rank[N];

void init() {
    for (int i=0; i<N; i++) {
        cha[i] = i;
        rank[i] = 0;
    }
}

int find(int u) {
    if (cha[u] != u) cha[u] = find(cha[u]);
    return cha[u];
}

void join(int u, int v) {
    u = find(u);
    v = find(v);
    if (u == v) return;
    if (rank[u] == rank[v]) rank[u]++;
    if (rank[u] < rank[v]) cha[u] = v;
    else cha[v] = u;
}
```

# Disjoint Set ngẫu nhiên

Ngoài cách làm trên, còn có thể cài đặt phép hợp bằng cách chọn ngẫu nhiên một trong 2 đỉnh để làm gốc:

```
#include <stdlib.h> // rand()

void join(int u, int v) {
    u = find(u);
    v = find(v);
    if (rand() & 1) cha[u] = v;
    else cha[v] = u;
```