Disjoint Set Union

Phép hợp các tập phần tử rời nhau

Lý thuyết và thực hành

Tran Viet Khoa - HUSC

tvkhoa.husc@gmail.com

Mục lục

1 GIỚI THIỆU **3**

2 CÀI ĐẶT CẤU TRÚC **3**

2.1 Phương án Naïve 5

2.2 Cải tiến nén đường đi với `find_set(v)` 7

2.3 Cải tiến với phép hợp nhất `union_set(a, b)` 8

2.3.1 Hợp nhất theo kích thước (số nút) của cây 8

2.3.2 Hợp nhất theo độ sâu của cây 10

3 ỨNG DỤNG CỦA DSU **12**

3.1 Một số phép toán thống kê 12

3.2 Bài toán online - thành phần liên thông 14

4 BÀI TẬP ĐỀ NGHỊ **17**

4.1 Bài C 17

4.2 Bài E 17

4.3 Bài toán: Các thùng nước 17

4.4 Bài toán: Phá đường 17

1 GIỚI THIỆU

Bài viết này đề cập đến một cấu trúc dữ liệu gọi là Disjoint Set Union hay viết tắt là DSU và được liệt kê vào dạng cấu trúc cây. Ngoài ra nó còn được gọi bằng một tên khác là Union Find bởi vì cấu trúc này có hai phép toán (hàm) chính, đó là:

- Phép kết hợp (Union) hai tập hợp bất kỳ.
- Phép tìm (Find) một phần tử có trong một tập hợp hay không?

Nội dung lý thuyết và thực hành dựa trên hai tài liệu chính [1]đó là:

- http://codeforces.com/edu/course/2/lesson/7
- https://cp-algorithms.com/data_structures/disjoint_set_union.html

Phiên bản ban đầu của DSU còn có thêm một thao tác nữa, cho nên giao diện (interface) cơ bản của cấu trúc này được xem có ba thao tác cơ bản sau:

- `make_set(v)` - Tạo tập hợp mới với phần tử v.
- `union_sets (a, b)` - Hợp nhất hai tập hợp được chỉ định (tập hợp chứa phần tử a và tập hợp chứa phần tử b).
- `find_set (v)`- Trả về phần tử đại diện (còn gọi là thủ lĩnh) của tập hợp chứa phần tử v và nó có thể thay đổi trong quá trình hợp nhất. Phần tử đại diện dùng để kiểm tra hai phần tử bất kỳ a và b có cùng thuộc một tập hay không.

Các hàm trên đều thực hiện với thời gian trung bình gần như $O(c)$ với c là hằng số cho nên nó là một cấu trúc dữ liệu cực kỳ hiệu quả cho lớp các bài toán sau:

1. Thuật toán Kruskal cho tìm cây khung nhỏ nhất.
2. Tìm kiếm trên cây.
3. Thực hiện một số bài toán có tính online [2] trên đồ thị như: Kiểm tra thành phần liên thông, tìm cầu, tìm điểm khớp.
4. Kết hợp với các cấu trúc khác như RMQ, LCA.

2 CÀI ĐẶT CẤU TRÚC

DSU có nhiều cách cài đặt, tuy nhiên cài đặt dưới dạng cây (Tree) là tốt nhất cho nên DSU có thể xem như một trong những cấu trúc dữ liệu cây (BIT, segment Tree, Treap, Sqrt Tree, Randomized Heap), mỗi cây là một tập hợp và phần tử gốc (root) được xem là phần tử đại diện của tập hợp.

Xét ví dụ như hình 1 biểu diễn tập hợp với cây như sau:

Nhìn vào hình, ta thấy:

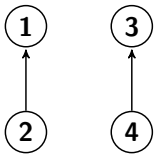
1. Ban đầu mọi phần tử bắt đầu như một tập hợp có một phần tử duy nhất và nó chính là gốc (phần tử đại diện) cho tập hợp. Ta có 4 tập.

[1]Tác giả chỉ viết lại thành bản tiếng việt và có hiệu chỉnh nội dung theo ý cá nhân của mình, mọi người đọc và gửi ý kiến phản hồi theo địa chỉ email ở trang đầu bài viết.

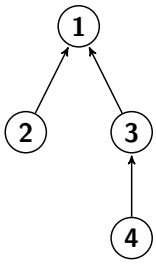
[2]Thuật ngữ của dân lập trình thi đấu nghĩa là dữ liệu của đồ thị thay đổi trong quá trình truy vấn, ngược lại là offline dữ liệu không thay đổi trong quá trình truy vấn

(a) Khởi tạo

(b) Hợp 1 - 2



(c) Hợp 3 - 4



(d) Hợp 1 - 3

Hình 1: Mô hình cây cho DSU

2. Sau đó ta kết hợp tập hợp chứa phần tử 1 và tập hợp chứa phần tử 2 (1 được chọn làm đại diện). Ta có 3 tập.
3. Tiếp theo ta kết hợp tập hợp chứa phần tử 3 và tập hợp chứa phần tử 4 (3 đại diện). Ta có 2 tập.
4. Và ở bước cuối cùng, ta kết hợp tập hợp chứa phần tử 1 và tập hợp chứa phần tử 3 (1 đại diện). Ta có 1 tập

Để thực hiện, điều này có nghĩa là chúng ta sẽ phải duy trì một mảng cha (parent) lưu trữ một tham chiếu đến tổ tiên trực tiếp của nó trong cây.

Để nắm bắt tốt hơn về cấu trúc này ta bắt đầu với bài toán thứ nhất như sau:

Bài tập 1: Disjoint Set Union

Hãy cài đặt cấu trúc DSU với hai phép toán truy vấn sau:

- union u v hợp hai tập hợp chỉ định bởi hai phần tử u và v.
- get u v kiểm tra xem u và v có cùng một tập hay không?.

Input

- Dòng đầu tiên chứa hai số nguyên dương n, m thỏa $1 \leq n, m \leq 10^5$ là số lượng các phần tử và số truy vấn.

- m dòng tiếp theo mô tả hai dạng truy vấn trên và $1 \leq u, v \leq n$.

Output

- In kết quả cho truy vấn dạng *get* u v là **YES** nếu u, v cùng một tập và **NO** nếu ngược lại.

Samples

input

```
4 4
union 1 2
union 1 3
get 1 4
get 2 3
```

output

```
NO
YES
```

Link nộp bài: http://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/problem/A

LỜI GIẢI:

2.1 Phương án Naïve

Phương án này cực kỳ đơn giản như định nghĩa của cấu trúc dữ liệu này. Sử dụng một mảng duy nhất gọi là parent chứa phần tử đại diện như phần giới thiệu trên. Phép tạo tập hợp `make_set(v)` cho v làm phần tử đại diện.

Để kết hợp hai tập hợp `union_set(a, b)`, trước tiên chúng ta tìm đại diện của tập hợp mà a được đặt và đại diện của tập hợp có b. Nếu các đại diện giống hệt nhau, mà chúng tôi không phải làm gì, các tập hợp đã được hợp nhất. Nếu không, chúng ta có thể chỉ định một cách đơn giản rằng một trong các đại diện là cha của đại diện kia (chọn a)- do đó kết hợp hai cây.

Cuối cùng là việc thực hiện hàm tìm phần tử v (`find_set (v)`), chúng ta chỉ cần leo lên đỉnh tổ tiên của đỉnh v cho đến khi chúng ta đến gốc, tức là một đỉnh sao cho tham chiếu đến tổ tiên dẫn đến chính nó. Thao tác này dễ dàng thực hiện một cách đệ quy.

Mã nguồn như sau:

ANaive.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> p;
void Init(int n) {
   p.resize(n+1);
   for (int i = 1; i <= n; i++) {
      p[i] = i;
   }
}
int find_set(int x) {
   if (x == p[x])
      return x;
   return find_set(p[x]);
}
void union_set(int x, int y) {
   x = find_set(x);
   y = find_set(y);
   if (x != y) {
      p[y] = x;
   }
}
int main() {
   ios::sync_with_stdio(0); cin.tie(0);
   int n, m; cin >> n >> m;
   Init(n);
   while (m--) {
      string s;
      int x, y;
      cin >> s >> x >> y;
      if (s == "union") union_set(x, y);
      else {
         if (find_set(x) == find_set(y)) cout << "YES\n";
         else cout << "NO\n";
      }
   }
   return 0;
}
```

Độ phức tạp của hàm `find_set(v)` trong trường hợp này có độ phức tạp xấu nhất là $O(n)$ khi cây thoái hóa thành dãy xảy ra khi ta hợp nhất tập hợp các phần tử có trật tự liên tiếp nhau. Dẫn đến mã nguồn trên sẽ bị TLE khi gặp n, m khoảng lớn hơn 10^4.

Bài tập khuyến khích

Bài 1. Vẽ cây DUS cho các phép hợp sau:

n=10;

union_set(1, 2);

union_set(3, 4);

union_set(8, 9);

union_set(9, 10);

union_set(5, 7);

union_set(8, 7);

union_set(1, 10);

Bài 2. Vẽ cây DUS cho các phép hợp sau:

n=10;

union_set(1, 2);

union_set(2, 3);

union_set(3, 4);

union_set(4, 5);

union_set(5, 6);

union_set(6, 7);

union_set(7, 8);

union_set(8, 9);

union_set(9, 10);

2.2 Cải tiến nén đường đi với `find_set(v)`

Giải pháp này được áp dụng để nâng cao hiệu quả của hàm `find_set(v)` tránh trường hợp bị suy biến do phép hợp nhất tạo ra.

Nếu chúng ta gọi `find_set(v)` cho một số đỉnh v, chúng ta thực sự tìm thấy đại diện p cho tất cả các đỉnh mà chúng ta truy cập trên đường đi giữa v và đại diện thực tế p. Bí quyết là làm cho các đường dẫn cho tất cả các nút đó ngắn hơn, bằng cách đặt trực tiếp đỉnh cha của mỗi đỉnh được truy cập thành p.

Bạn có thể xem hoạt động trong hình 2 sau đây. Ở bên trái có một cái cây, và ở bên phải có cây nén sau khi gọi `find_set(7)`, nó rút ngắn đường dẫn cho các nút đã truy cập 7, 5, 3 và 2.

Mã nguồn cải tiến của hàm `find_set(v)`:

```
int find_set(int x) {
   if (x == p[x])
      return x;
   return p[x]=find_set(p[x]);
}
```

(a) find_set(7) (b) Nén đường

Hình 2: Mô hình cây cho DSU nén đường đi

Việc cải tiến đơn giản này mang lại hiệu quả mà ta mong đợi, bạn đã accept được cho bài toán thứ nhất ở trên với độ phức tạp $O(logn)$ (không chứng minh ở đây). Bằng cách gọi đệ quy ta tìm được các phần tử đại diện, sau đó quá trình làm rỗng stack sẽ gán phần tử này cho các móc nối parent[] ở các phần tử đã đi qua. Thuật toán này có thể viết bằng lặp như sau:

```
int find_set(int x) {
   while (x !=p[x] ){
      p[x] = p[p[x]]; // Skip one level
      x = p[x]; // Move to the new level
   }
   return x;
}
```

Bài tập khuyến khích

Bài 3. Vẽ lại cây DUS cho hình 2 trên sau khi gọi hàm `find_set(8)`

2.3 Cải tiến với phép hợp nhất `union_set(a, b)`

Ngoài phép cải tiến trên người ta còn đưa ra thêm một giải pháp cải tiến nữa tiếp cận trên hàm `union_set(a, b)`

2.3.1 Hợp nhất theo kích thước (số nút) của cây

Phép hợp chọn đỉnh đại diện của cây mới dựa theo tiêu chí số lượng (size) của các nút trong hai cây con, cây nào nhiều nút hơn sẽ được chọn làm đại diện.

Để thực hiện điều đó, ta khai báo thêm một mảng (sz) là kích thước của mỗi tập và được khởi tạo giá trị 1 lúc ban đầu cho mỗi cây.

(a) kích thước hai cây con

(b) Hợp theo kích thước (size)

Hình 3: Phép hợp dựa vào số lượng nút của cây

ASize.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> p, sz;
void Init(int n) {
   p.resize(n+1); sz.resize(n+1);
   for (int i = 1; i <= n; i++) {
      p[i] = i;   sz[i] = 1;
   }
}
int find_set(int x) {
   if (x == p[x])
      return x;
   return find_set(p[x]);
}
void union_set(int x, int y) {
   x = find_set(x); y = find_set(y);
   if (x != y) {
      if (sz[x] < sz[y])
         swap(x, y);
      sz[x] += sz[y];
      p[y] = x;
   }
}
int main()
{
   ios::sync_with_stdio(0); cin.tie(0);
   int n, m; cin >> n >> m;
   Init(n);
   while (m--) {
      string s;
      int x, y; cin >> s >> x >> y;
      if (s == "union") union_set(x, y);
      else {
         if (find_set(x) == find_set(y)) cout << "YES\n";
         else cout << "NO\n";
      }
   }
   return 0;
}
```

Bài tập khuyến khích

Bài 4. Vẽ cây DUS cho các phép hợp có cải tiến hợp theo kích thước sau:

n=10;

union_set(1, 2);

union_set(4, 3);

union_set(3, 5);

union_set(5, 6);

union_set(6, 7);

union_set(7, 8);

union_set(9, 10);

union_set(1, 10);

Chú ý: Không sử dụng phép nén đường đi trong bài tập này.

2.3.2 Hợp nhất theo độ sâu của cây

Phép hợp chọn đỉnh đại diện của cây mới dựa theo tiêu chí độ sâu (depth hay còn gọi là rank) của hai cây con, cây nào có độ sâu lớn hơn sẽ được chọn làm đại diện.

Tương tự phép cải tiến theo kích thước, ở đây ta khai báo một mảng để lưu độ sâu của cây (depth hoặc rank) với khởi tạo ban đầu là 0, tức là ở mức nút gốc chưa có phần tử nào có giá trị độ sâu bằng không (một số cài đặt có thể gán bằng 1).

(a) union_set(4, 6) (b) Hợp hai cây

Hình 4: Phép hợp dựa vào độ sâu của cây

ADepth.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> p, depth;
void Init(int n) {
   p.resize(n+1); depth.resize(n+1);
   for (int i = 1; i <= n; i++) {
      p[i] = i;
      depth[i] = 0;
   }
}
int find_set(int x) {
   if (x == p[x])
      return x;
   return find_set(p[x]);
}
void union_set(int x, int y) {
   x = find_set(x); y = find_set(y);
   if (x != y) {
      if (depth[x] < depth[y])
         swap(x, y);
      p[y] = x;
      if (depth[x]==depth[y])
         ++depth[x];
   }
}
int main()
{
   ios::sync_with_stdio(0); cin.tie(0);
   int n, m; cin >> n >> m;
   Init(n);
   while (m--) {
      string s; int x, y;
      cin >> s >> x >> y;
      if (s == "union")
         union_set(x, y);
      else {
         if (find_set(x) == find_set(y)) cout << "YES\n";
         else cout << "NO\n";
      }
   }
   return 0;
}
```

Cả hai giải pháp hợp nhất đều có cùng tính hiệu quả nếu không áp dụng kết hợp với phép nén đường đi thì cả hai đều có độ phức tạp $O(logn)$ và cài đặt cũng gần như nhau.

Bài tập khuyến khích

Bài 5. Vẽ cây DUS cho các phép hợp có cải tiến hợp theo độ sâu (depth, rank) sau:

n=10;

union_set(1, 2);

union_set(4, 3);

union_set(3, 5);

union_set(5, 6);

union_set(6, 7);

union_set(7, 8);

union_set(9, 10);

union_set(1, 10);

Chú ý: Không sử dụng phép nén đường đi trong bài tập này.

Trong trường hợp cả hai giải pháp `union_set(v)` cùng kết hợp với giải pháp nén đường đi của thuật toán `find_set(v)` phép xử lý cho mỗi truy vấn với độ phức tạp gần như hằng số. Cụ thể được chứng minh trong một bài báo của tác giả Tarjan năm 1975 và Kurt Mehlhorn và Peter Sanders năm 2008 với độ phức tạp $O(\alpha(n))$, trong đó hàm $\alpha(n)$ là hàm nghịch đảo Akkerman có độ tăng rất chậm với $n \leq 10^{600}$ thì $\alpha(n)$ có giá trị không quá 4.

3 ỨNG DỤNG CỦA DSU

3.1 Một số phép toán thống kê

Bài tập 2: Disjoint Set Union 2

Hãy cài đặt cấu trúc DSU với hai phép toán truy vấn sau:

- union u v hợp hai tập hợp chỉ định bởi hai phần tử u và v.
- get v tìm tập hợp chứa v và tìm: phần tử nhỏ nhất, lớn nhất và số lượng các phần tử trong tập chứa v.

Input:

- Dòng đầu tiên chứa hai số nguyên dương $n,\ m$ thỏa $1 \leq n, m \leq 3.10^5$ là số lượng các phần tử và số truy vấn.

- m dòng tiếp theo mô tả hai dạng truy vấn trên và $1 \leq v \leq n$.

Output:

- In kết quả cho truy vấn dạng *get* v và in các giá trị cần tìm gồm: phần tử nhỏ nhất, phần tử lớn nhất và số phần tử của tập.

Examples

input

```
5 11
union 1 2
get 3
get 2
union 2 3
get 2
union 1 3
get 5
union 4 5
get 5
union 4 1
get 5
```

output

```
3 3 1
1 2 2
1 3 3
5 5 1
4 5 2
1 5 5
```

Link nộp bài: http://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/problem/B

LỜI GIẢI:

Khai báo thêm hai mảng dùng để chứa số nhỏ nhất của tập và số lớn nhất của tập, riêng số lượng các phần tử của tập thì dùng luôn mảng size trong phép hợp cài tiến với kích thước.

Mã nguồn đã accept:

Bsol.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> p, sz, Min, Max;
void Init(int n) {
   p.resize(n+1);
   sz.resize(n+1);
   Min.resize(n+1);
   Max.resize(n+1);
   for (int i = 1; i <= n; i++) {
      p[i] = i; sz[i] = 1; Min[i]= i; Max[i]=i;
   }
}
int find_set(int x) {
   if (x == p[x]) return x;
   return p[x]=find_set(p[x]);
}

void union_set(int x, int y) {
   x = find_set(x);
   y = find_set(y);
   if (x != y) {
      if (sz[x] < sz[y])
         swap(x, y);
      sz[x] += sz[y];
      p[y] = x;
      Min[x] = min(Min[x], Min[y]);
      Max[x] = max(Max[x], Max[y]);
   }
}

int main() {
   ios::sync_with_stdio(0); cin.tie(0);
   //freopen("dl.inp", "r", stdin);
   int n, m;   cin >> n >> m;
   Init(n);
   while (m--) {
      string q; cin >> q;
      if(q == "union"){
         int x,y; cin >> x >> y;
         union_set(x, y);
      }
      else{
         int x; cin >> x;
         int g = find_set(x);
         cout << Min[g] << " "<<Max[g]<< " "<<sz[g]<<'\n';
      }
   }
   return 0;
}
```

3.2 Bài toán online - thành phần liên thông

Bài tập 3: Xóa cạnh đồ thị và kiểm tra liên thông

Cho đồ thị vô hướng $G = (V, E)$. Hãy thực hiện hai phép toán truy vấn sau trên G

- cut u v Xóa cạnh nối hai đỉnh u và v ra khỏi đồ thị.
- ask u v Kiểm tra xem u và v có thuộc cùng một thành phần liên thông hay không?

Sau khi thực hiện hết các thao tác truy vấn trên chắc chắn G sẽ không còn cạnh nào. Vui lòng in kết quả của các truy vấn **ask**.

Input:

- Dòng đầu tiên chứa ba số nguyên dương $n,\ m,\ k$ thỏa $1 \leq n \leq 50000, 0 \leq m \leq 100000, m \leq k \leq 150000$ là số lượng đỉnh, cạnh của đồ thị kèm với k truy vấn.

- m dòng tiếp theo gồm hai số nguyên dương u_i, v_i mô tả cạnh nới giữa hai đỉnh của G thỏa $1 \leq u_i, v_i \leq n$. Đỉnh đánh số từ 1 và G không có cạnh lặp, khuyên.

- k dòng tiếp theo mô tả hai dạng của truy vấn và mỗi cạnh chỉ thực hiện việc cắt bỏ (cut, remove) đúng một lần.

Output:

- In kết quả cho truy vấn dạng *ask* u v là YES nếu hai đỉnh cùng một thành phần liên thông và in NO nếu ngược lại.

Examples

input

```
3 3 7
1 2
2 3
3 1
ask 3 3
cut 1 2
ask 1 2
cut 1 3
ask 2 1
cut 2 3
ask 3 1
```

output

```
YES
YES
NO
NO
```

Link nộp bài: http://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/problem/D

LỜI GIẢI:

Đây là một bài toán tìm kiếm trên đồ thị rất cơ bản, có thể giải quyết nó bằng DFS hoặc BFS. Tuy nhiên, do đặc thù online (đã giải thích ở trên) nên phép cài đặt DUS hoàn toàn thuận lợi cho bài toán này. Việc gỡ bỏ một cạnh và đếm hoặc xác định thành phần liên thông, số thành phần liên thông cũng chính là bài toán xác định cạnh gỡ bỏ đó có phải là cầu hay không.

Dữ kiện của bài toán cho phép người giải bằng nhiều thuật toán ví dụ như DFS hay BFS. Riêng với cách giải với DUS thì phần dữ liệu cho đồ thị không cần thiết và người dùng đọc kỹ phần chú ý của đề (Sau khi thực hiện hết các thao tác truy vấn trên chắc chắn G sẽ không còn cạnh nào), do vậy quy trình

ngược lại đó là với tập rỗng thay vì xóa là vẽ cạnh, cho nên bài toán này giải với DUS sẽ thực hiện các truy vấn với chiều ngược lại.

ADepth.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> p, sz;
void Init(int n) {
   p.resize(n+1); sz.resize(n+1);
   for (int i = 1; i <= n; i++) {
      p[i] = i; sz[i] = 1;
   }
}
int find_set(int x) {
   if (x == p[x]) return x;
   return p[x]=find_set(p[x]);
}
void union_set(int x, int y) {
   x = find_set(x); y = find_set(y);
   if (x != y) {
      if (sz[x] < sz[y])   swap(x, y);
      sz[x] += sz[y];
   p[y] = x;
   }
}

int main() {
   ios::sync_with_stdio(0); cin.tie(0);
   int n, m, q; cin>>n>>m>>q;
   Init(n);
   //Bỏ qua dữ liệu đồ thị.
   for(int i=0; i<m; i++){
      int u,v; cin>>u>>v;
   }
   string s[q]; int u[q],v[q];
   for(int i=0; i<q; i++)
      cin>>s[i]>>u[i]>>v[i];
   vector<string>ans;
   for(int i=q-1; i>=0; i--){
      if(s[i]=="ask"){
         int pu=find_set(u[i]);
         int pv=find_set(v[i]);
         ans.push_back((pu==pv?"YES":"NO"));
      }
      else union_set(u[i],v[i]);
   }
   if(ans.size()) reverse(ans.begin(),ans.end());
   for(auto it: ans)
      cout<<it<<"\n";
   return 0;
}
```

4 BÀI TẬP ĐỀ NGHỊ

4.1 Bài C

https://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/problem/C

4.2 Bài E

https://codeforces.com/edu/course/2/lesson/7/1/practice/contest/289390/problem/E

4.3 Bài toán: Các thùng nước

https://oj.vnoi.info/problem/ioibin

4.4 Bài toán: Phá đường

https://oj.vnoi.info/problem/qhroad

HẸN GẶP LẠI Ở BÀI DUS KẾ TIẾP