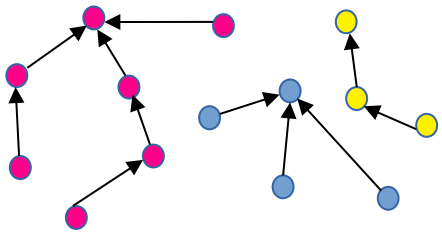
Система непресичащи се множества

Disjoint Set Union (DSU), или Система от непресичащи се множества, е начин за представяне на разбиване на дадено множество на негови подмножества и работа с тях чрез следните операции:

- **make_set(x)** – създава ново множество, съдържащо единствено x.
- **union_sets(x, y)** – обединява множествата, към които принадлежат съответно x и y.
- **find_set(x)** – намира към кое множество принадлежи x.

Съответно в началото започваме с набор елементи, всяко от които в собствено множество. За една стъпка можем да обединим две множества или да проверим принадлежността на елемент.

Ефективно представяне е като кореново дърво, наричайки корена представител на множеството. Този елемент определя еднозначно множеството поради липсата на повторения (по дефиниция), но може да бъде променен с течение на времето (примерно след обединение).
Така например, ако x и y са различни, но принадлежат на едно множество, то find_set(x) ще е равно на find_set(y).



<u>Наивна реализация:</u>

Ще използваме масив на родителите parent[], за да отбележим структурата на дървото.

Тогава функциите биха изглеждали така:

```
void make_set(int& parent[], int x){
    parent[x] = x;     //коренът е родител сам на себе си
}

int find_set(int parent[], int x){
    if (x == parent[x])    //ако сме стигнали до корена, то той е представител на множеството
        return x;
    return find_set(parent[x]);     //в противен случай, продължаване нагоре по дървото
}

void union_sets(int& parent[], int x, int y){
    x = find_set(x);    //намираме множеството на x
    y = find_set(y);  //намираме множеството на y
    if(x != y)   //ако са в различни множества, ги обединяваме
        parent [x] = y;
}
```

Този метод има един недостатък – докато make_set е със сложност O(1), то в най-лошия случай дървото може да се изроди в линеен списък и това да доведе до сложност O(n) за find_set, а оттам и за union_sets.

Оптимална реализация:

Нека запазим представянето чрез масив и функциите make_set и union_set. Промяната ще дойде във функцията find_set:

```
void make_set(int& parent[], int x){
   parent[x] = x;     //коренът е родител сам на себе си
}

int find_set(int& parent[], int x){
   if (x == parent[x])    //ако сме стигнали до корена, то той е представител на множеството
      return x;
   return parent[x] = find_set(parent[x]);     //в противен случай, продължаване нагоре по дървото
}

void union_sets(int& parent[], int x, int y){
   x = find_set(x);    //намираме множеството на x
   y = find_set(y);  //намираме множеството на y
   if(x != y)   //ако са в различни множества, ги обединяваме
      parent [x] = y;
}
```

Сега в хода на изпълнение на find_set мутираме дървото по такъв начин, че родител на всички върхове от клона, по който се движим, се оказва корена. Съответно след краен брой прилагания на find_set ще получим дърво с дълбочина едва единица, т.е. амортизираната сложност на find_set ще е O(1), а сложността на union_set пряко зависи от find_set, т.е. също ще бъде O(1).