

ESTRUTURA DE DADOS - UNION FIND

José Gabriel
Thiago Pontes
Willian Tcheldon

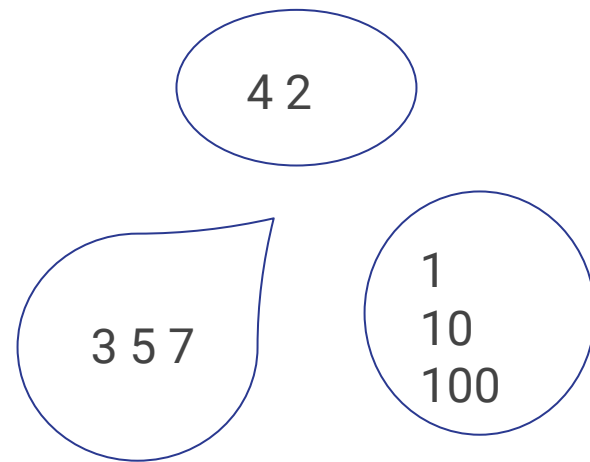
FUNDAMENTO MATEMÁTICO E SUA IMPLEMENTAÇÃO

Conjuntos Disjuntos - Fundamento Matemático

O Union-Find também é chamado de Disjoint-Set, por se basear no fundamento matemático de Conjuntos Disjuntos.

Esses conjuntos são caracterizados por não terem nenhum elemento em comum, ou seja não há intersecção entre eles.

Além disso, usaremos o conceito de Elemento Representante de um Conjunto, para nos auxiliar na implementação de operações envolvendo os Conjuntos.



Exemplo da Concorrência entre Empresas

Suponhamos que em uma certa época, houve um boom econômico, que provocou o surgimento de diferentes empresas start-up (Consideremos elas como vários Conjuntos Disjuntos, cujo Representante são elas mesmas);



Com o passar do tempo, as circunstâncias fizeram com que muitas empresas tiveram que se fundir, e filiar à outras, no intuito de evitar a falência, virando parte de um grande conjunto de um monopólio;

Exemplo da Concorrência entre Empresas

Para que os clientes de empresas que se fundiram, continuem com seus antigos cadastros, eles terão de encontrar qual o representante atual deste serviço para poderem entrar em contato e atualizarem seu cadastro.



E é para lidar com casos similares, que desenvolveu-se a Estrutura de Dados Union-Find.



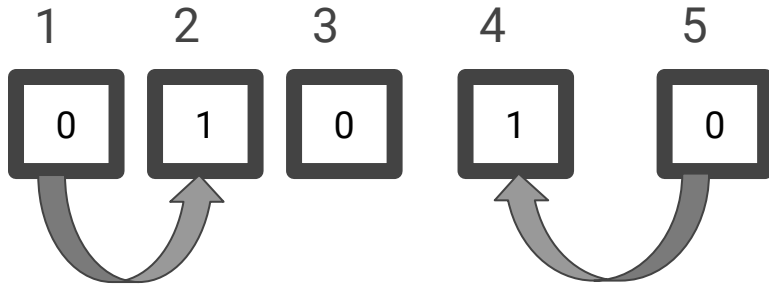
INTRODUZINDO A ESTRUTURA DE UNION-FIND

Requisitos a serem Considerados

```
#define MAX 100  
  
int parent[MAX], rank[MAX];
```

Dado um espaço limitado, contendo vários conjuntos, temos algo a considerar:

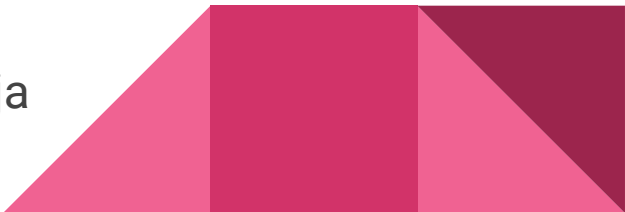
- Como demarcar os elementos representantes?!
- Como indicar que um determinado conjunto possui outros elementos dentro de si?!



TIPOS ABSTRATOS DE DADOS

```
void makeset(Conjunto *no, int valor);  
  
Conjunto* find(int i);  
  
union_sets(Conjunto *num, Conjunto *mon);
```

Apenas duas operações principais, e uma opcional caso seja necessário; Trivial

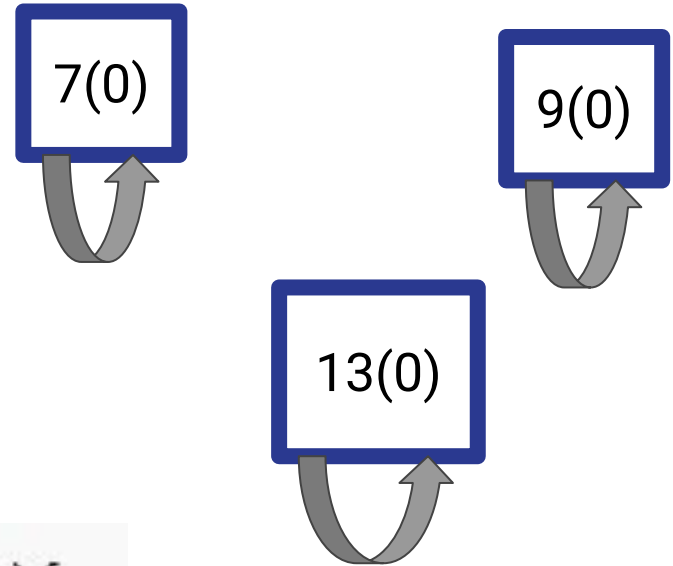


Struct

```
typedef struct Node{  
    int value;  
    struct Node *parent;  
    int rank;  
}Conjunto;
```

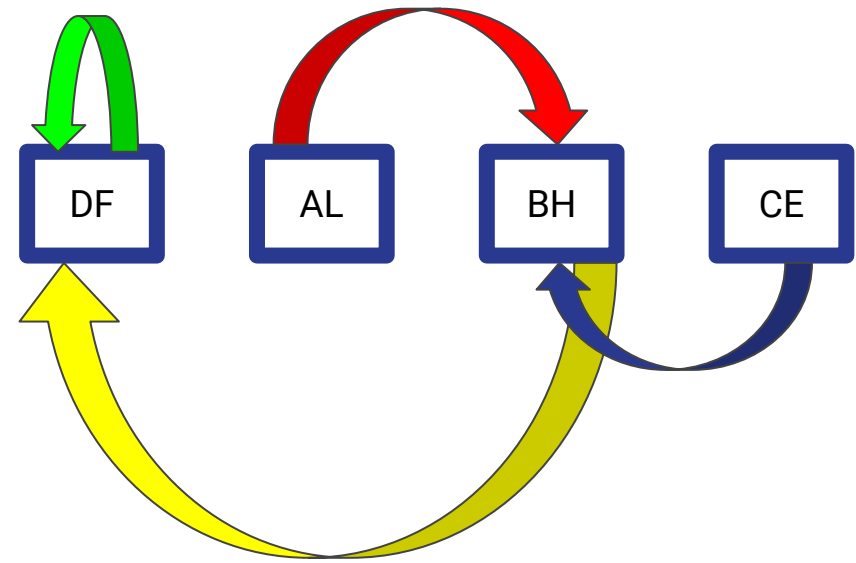
MakeSet

```
void makeset(Conjunto *no, int valor){  
    no->value=valor;  
    no->parent=no;  
    no->rank=0;  
}
```



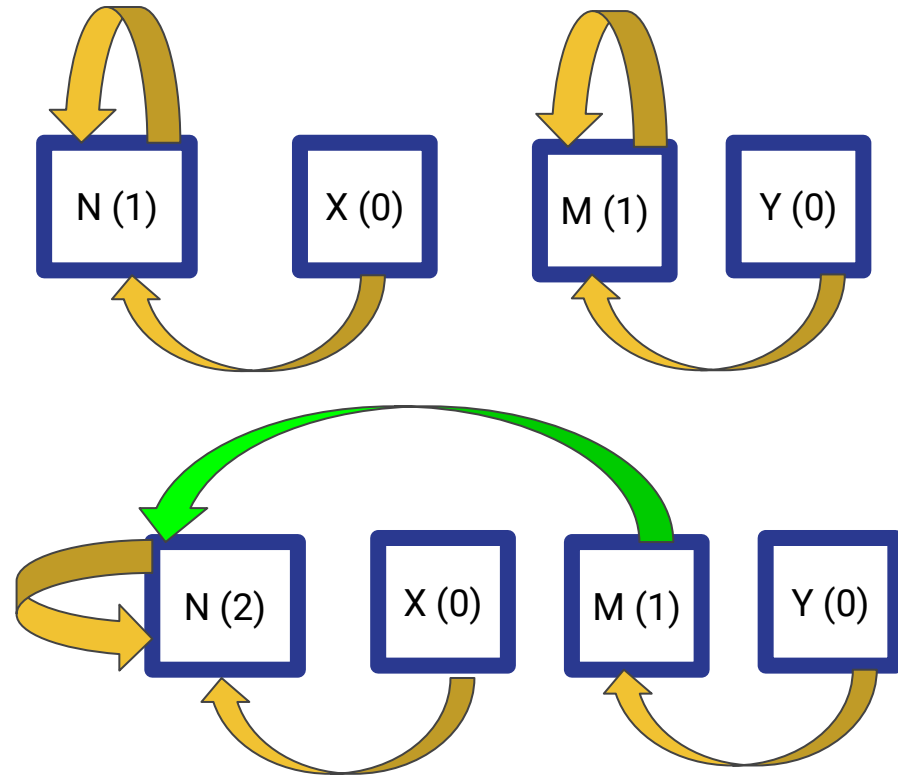
Find

```
Conjunto* find(Conjunto *a1) {  
    if (a1->parent != a1) {  
        a1->parent = find(a1->parent);  
    }  
    return a1->parent;  
}
```



Union

```
void union_sets(Conjunto *num, Conjunto *mon) {  
    Conjunto *uno = find(num);  
    Conjunto *duo = find(mon);  
  
    if (uno != duo) {  
        if (uno->rank > duo->rank) {  
            duo->parent = uno;  
        } else if (uno->rank < duo->rank) {  
            uno->parent = duo;  
        } else {  
            duo->parent = uno;  
            uno->rank++;  
        }  
    }  
}
```



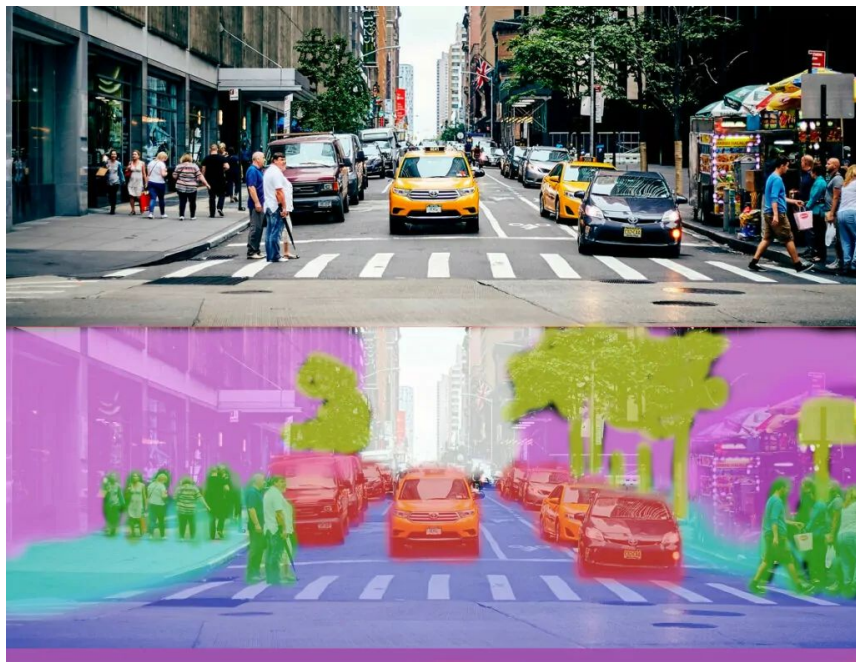
APLICAÇÕES DA ESTRUTURA

Eficiência

Graças à técnica de compressão de caminho, usada na chamada recursiva na função Find, e ao uso do rank na função Union, temos uma otimização de tempo magnífica:

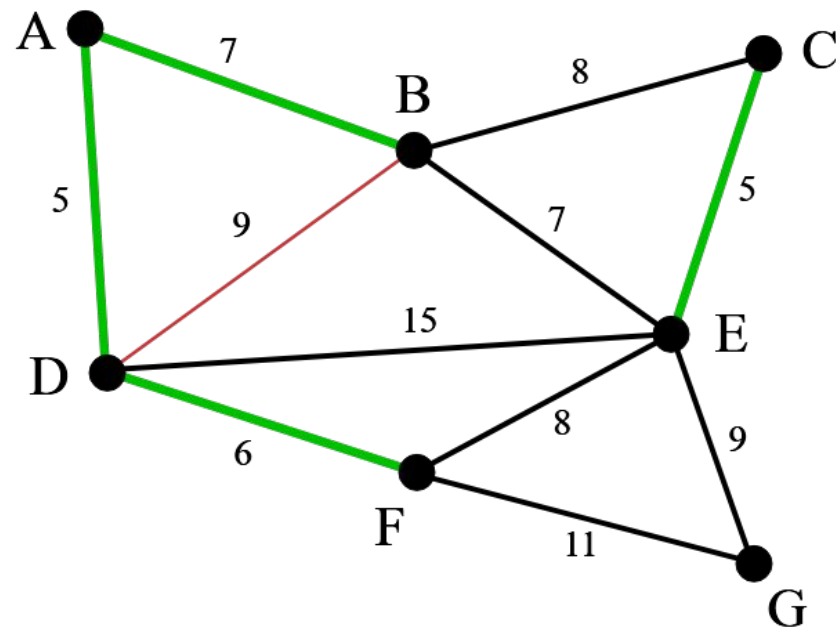
$$O(\alpha(n)) \sim O(1)$$





Reconhecimento de Imagem

Algoritmo de Kruskal



REFERÊNCIAS

- <https://www.youtube.com/watch?v=-VHbKJhu15k>
- https://www.youtube.com/watch?v=P_CPKrzIVTI
- <https://www.youtube.com/watch?v=OzFKSD2CA7E>
- <https://www.youtube.com/watch?v=I7EfCnsh2fc>
- <https://noic.com.br/materiais-informatica/curso/data-structures-02/>
- <https://pt.wikipedia.org/wiki/Uni%C3%A3o-busca>