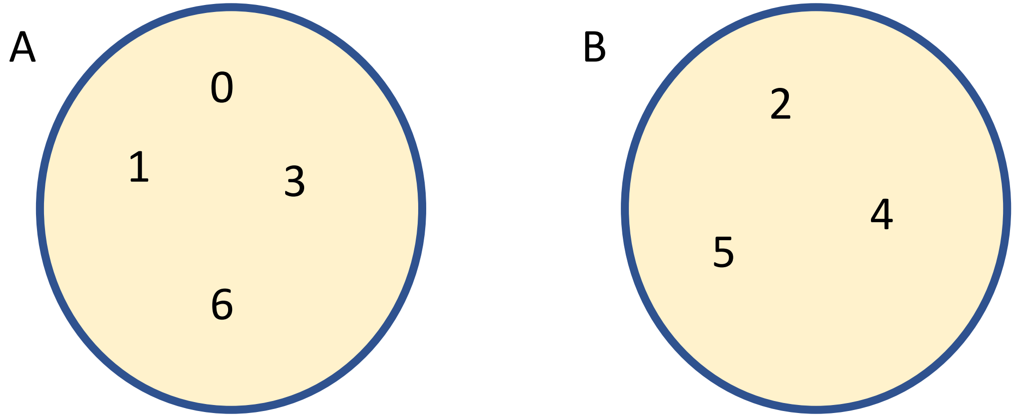
União e Busca

Union and Find

Mais uma árvore?

Não!!! Agora vamos falar de conjuntos

Vocês sabem ou lembram o que seria um conjunto disjunto?



Considere a situação de agrupar os elementos

Mas estando cada um em sua partição

O objetivo é agrupar os elementos combinando suas partições, que são disjuntas

Sabendo que:

Os elementos $x_1, x_2, ..., x_n$ pertence a um universo, denominado $U = \{x_1, x_2, ..., x_n\}$

E as k partições (S) que compõem uma coleção $C = \{S_1, S_2, ..., S_k\}$, possuem as seguintes propriedades:

- Qualquer partição está contida no universo U, ou seja: $S_i \subseteq U \; \forall_i$
- A união de todas as partições, representa a cobertura, e corresponde ao universo: $S_1 \cup S_2 \cup \cdots \cup S_k = U$
- E dessa forma, a interseção entre qualquer duas partições sempre será vazia: $S_i \cap S_j = \emptyset \; \forall \; i \; /= j$

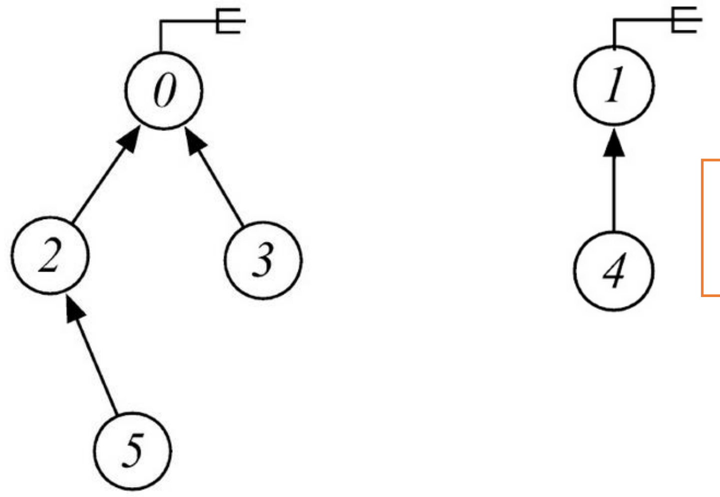
Como o objetivo é agrupar as partições disjuntas (unir), então é natural imaginar a necessidade de a qualquer momento querermos identificar qual elemento pertence a qual partição (buscar)

E daí surge o nome da estrutura que conheceremos: Union and Find, ou traduzindo União e Busca

Bom, como vamos trabalhar com partições, vamos conhecer um pouco sua representação

Para cada partição será eleito um elemento qualquer, que será um representante da partição

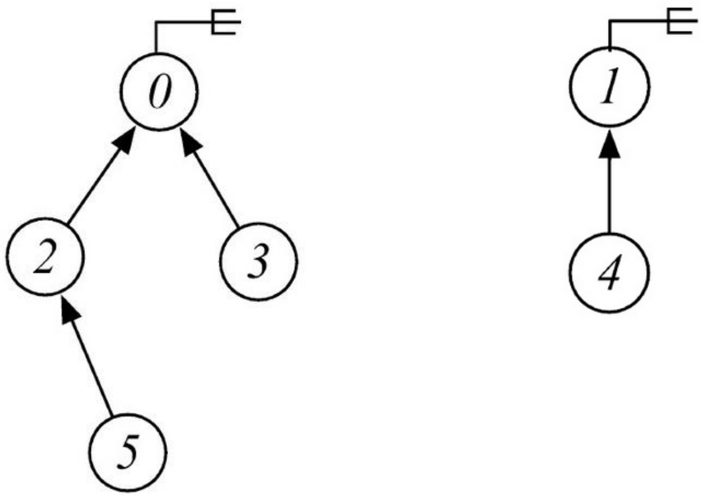
E para melhor identificar quem é o elemento representante da partição, que também será utilizado no processo de busca, uma das representações é a árvore reversa



a raiz é o representante da partição, identificado pela ausência de pai

Fonte: CELES; CERQUEIRA; RANGEL (2017)

Mas há outras formas também de bem representar, uma delas são por vetores:



Os valores negativos indicam o número de nós da árvore para aquele representante

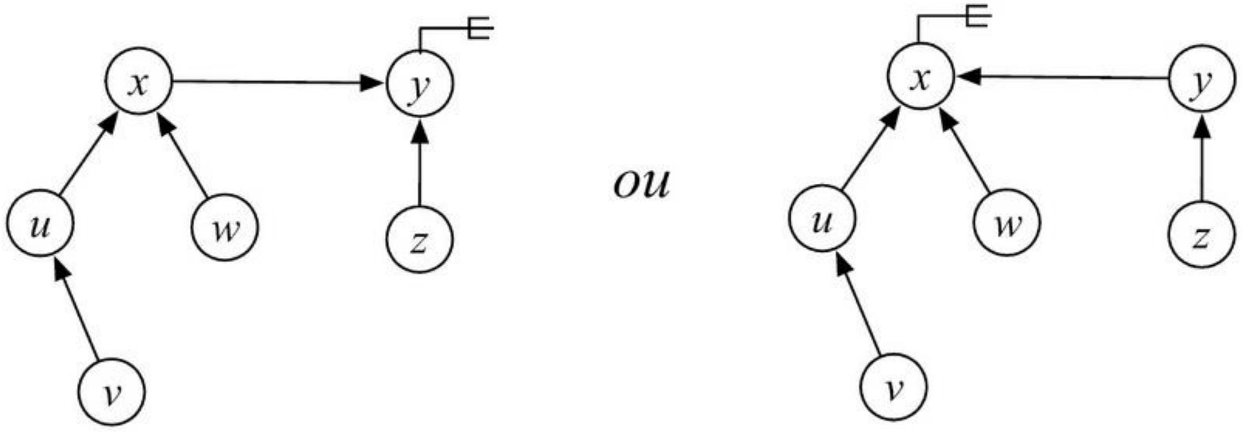
-4	-2	0	0	1	2
0	*1*	*2*	*3*	*4*	*5*

Fonte: CELES; CERQUEIRA; RANGEL (2017)

Então para o processo de busca em uma árvore reversa, basta caminharmos nas ramificações até a raiz

E para fazer o processo de união?

Para este basta associar as duas árvores reversas, unindo-as em uma só, em que uma recebe a outra como pai:



Fonte: CELES; CERQUEIRA; RANGEL (2017)

Dessa forma, o processo de união necessita do processo de busca para ocorrer

Pois será necessário saber quem são os representantes de cada árvore, para que possa ser feito a associação delas

E em termos de complexidade, uma é equivalente a da outra, O (h).
Então, para garantir a eficiência é interessante ter o controle da altura da árvore, pois se não for balanceada sua altura pode ser n.

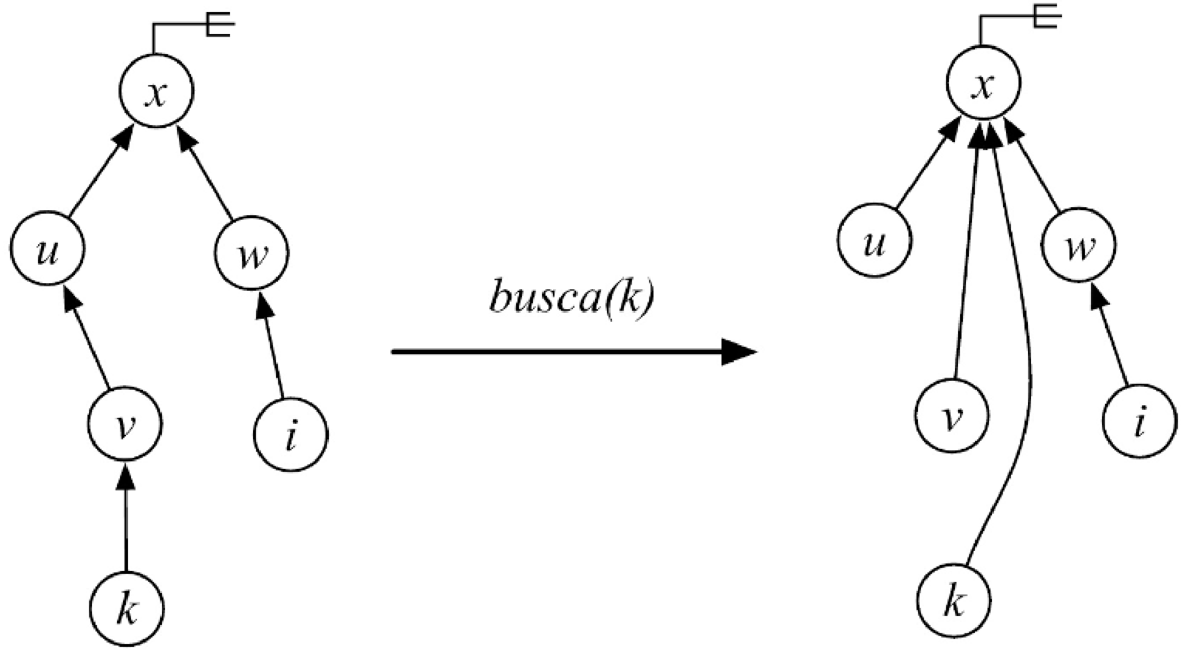
Segundo CELES; CERQUEIRA; RANGEL (2017), a união ela pode ocorrer forma balanceada por duas estratégias: por número de nós ou por altura.

Por número de nós a raiz da árvore com menor quantidade de elementos se torna filha da outra raiz;

Já por altura, a raiz da árvore de menor altura se tornar filha da outra raiz.

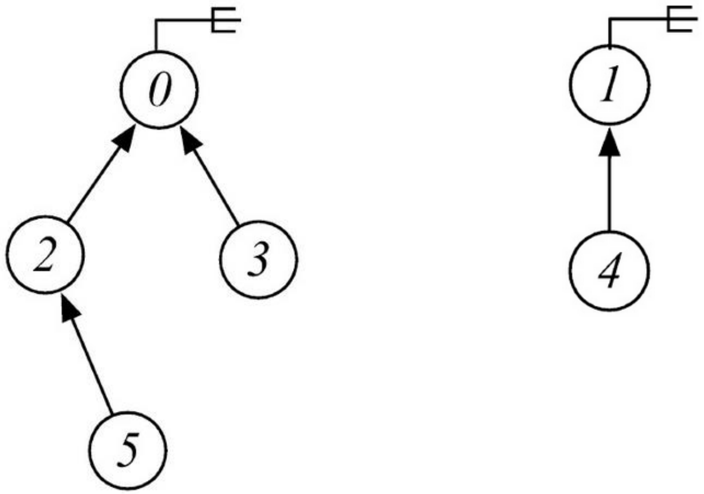
Em ambos os casos podemos observar que o custo computacional é O (log n)

Outra estratégia seria desmembrar a ramificação da árvore reversa, fazendo com que todos os elementos “apontem” para o representante da partição (pai)



Fonte: CELES; CERQUEIRA; RANGEL (2017)

Bom, mas como vimos a representação dessa estrutura pode ser realizada por meio de vetores



Os elementos que representam raízes guardam um valor negativo, indicando que não têm pai e representam a quantidade de elementos indexados para este representante

-4	-2	0	0	1	2
0	*1*	*2*	*3*	*4*	*5*

Fonte: CELES; CERQUEIRA; RANGEL (2017)

Como implementar?

Faremos uso da codificação apresentada por CELES; CERQUEIRA; RANGEL (2017)

```c
#include<stdlib.h>
#include<stdio.h>

struct UniaoBusca {
    int n;
    int * v;
};
```

`UniaoBusca* ub_cria (int n);` *Constrói a estrutura*

`void ub_libera ( UniaoBusca* ub);` *Libera a estrutura*

`int ub_busca ( UniaoBusca* ub, int x);` *Busca um representante da partição que x pertence*

`int ub_uniao ( UniaoBusca* ub, int x, int y);` *Realiza a união entre partições x e y*

```c
UniaoBusca* ub_cria (int n){

    int i;
    UniaoBusca* ub = ( UniaoBusca*) malloc(sizeof ( UniaoBusca));
    ub->n = n;
    ub->v = (int *) malloc(ub->n* sizeof (int ));

    for (i=0; i<ub->n; ++i)
        ub->v[i] = -1;

    return ub;

}
```

Nesta representação cada elemento está em uma partição individual, por isto sua representação é dada por -1, logo cada elemento é representado por si

```c
void ub_libera ( UniaoBusca* ub){
    free(ub->v);
    free(ub);
}
```

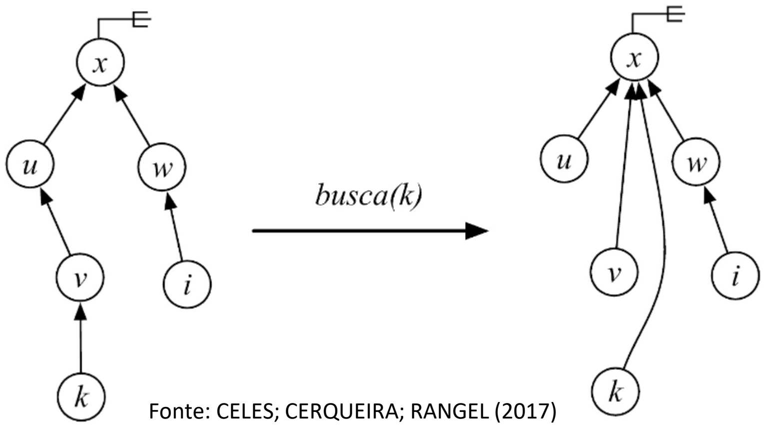
Libera a memória alocada para o vetor e para o ponteiro da estrutura

```c
int ub_busca ( UniaoBusca* ub, int x){

    int r = x;

    while (ub->v[r] >= 0)
        r = ub->v[r];

    while (ub->v[x] >= 0) {
        int p = ub->v[x];
        ub->v[x] = r;
        x = p;
    }

    return r;
}
```

Procura a raiz (o representante) da árvore que x pertence

Aproveita o processo de busca e realiza o processo de compressão do caminho, fazendo com que todos na árvore a que x pertence "apontem" para a o mesmo representante

Retorna o representante da partição que x pertence



Fonte: CELES; CERQUEIRA; RANGEL (2017)

O processo de união é realizado utilizando o critério do número de nós: a raiz da árvore com menos filhos passa a ser filha da outra árvore (partição)

```
int ub_uniao ( UniaoBusca* ub, int x, int y){
```

Busca-se as partições que x e y pertence

```
    x = ub_busca(ub,x);
    y = ub_busca(ub,y);
    if (x == y)
        return x;
```

Verifica-se se é a mesma, se for, nada é realizado

Os representantes das partições são negativos, para representarem que são raízes das árvores e os valores que possuem representam a quantidade de elementos da árvore, logo justifica-se a comparação <=

```
    if (ub->v[x] <= ub->v[y]) {
        ub->v[x] += ub->v[y];
        ub->v[y] = x;
        return x;
    }
```

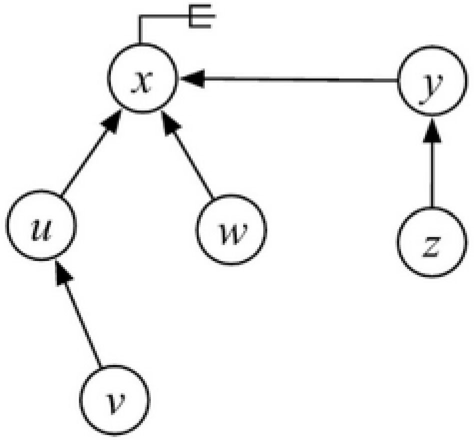
Atualiza a quantidade de nós da partição

E realiza a união das partições

```
    else {
        ub->v[y] += ub->v[x];
        ub->v[x] = y;
        return y;
    }

}
```

Atualiza a quantidade de nós da partição

E realiza a união das partições



Fonte: CELES; CERQUEIRA; RANGEL (2017)

Por hoje é só

Analisem o conteúdo e vejam a implementação no SIGAA

Próxima Aula
Trabalharemos na Disciplina:

Grafos (representação)

Fontes e Referências:

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José. **Introdução a estruturas de dados: com técnicas de programação em C**. Elsevier Brasil, 2017.