Conjuntos Disjuntos (Union-Find)

Estrutura de Dados Avançada — QXD0015

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2023

Introdução

- **Problema:** Manter uma coleção de conjuntos disjuntos dinâmicos $\mathcal{C} = \{S_1, S_2, \ldots, S_n\}$ (os conjuntos podem mudar no decorrer do tempo).

Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Problema:** Manter uma coleção de conjuntos disjuntos dinâmicos $\mathcal{C} = \{S_1, S_2, \ldots, S_n\}$ (os conjuntos podem mudar no decorrer do tempo).

- Dois conjuntos da coleção $\mathcal{C}$ podem ser unidos. A união é a única operação de modificação dos conjuntos.

Introdução

- **Problema:** Manter uma coleção de conjuntos disjuntos dinâmicos $\mathcal{C} = \{S_1, S_2, \ldots, S_n\}$ (os conjuntos podem mudar no decorrer do tempo).

- Dois conjuntos da coleção $\mathcal{C}$ podem ser unidos. A união é a única operação de modificação dos conjuntos.

- Cada conjunto é indentificado por um representante. Um representante é algum membro do conjunto. Frequentemente, não importa quem é o representante.

Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Problema:** Manter uma coleção de conjuntos disjuntos dinâmicos $\mathcal{C} = \{S_1, S_2, \ldots, S_n\}$ (os conjuntos podem mudar no decorrer do tempo).
- Dois conjuntos da coleção $\mathcal{C}$ podem ser unidos. A união é a única operação de modificação dos conjuntos.
- Cada conjunto é indentificado por um representante. Um representante é algum membro do conjunto. Frequentemente, não importa quem é o representante.
- Se nós perguntarmos quem é o representante de um conjunto duas vezes sem modificar o conjunto entre essas duas perguntas, nós devemos obter a mesma resposta.

Conjuntos Disjuntos

Universidade Federal do Ceará
Campus Quixadá

- Seja $S = \{x_1, \ldots, x_n\}$ um conjunto com N elementos.

Conjuntos Disjuntos

- Seja $S = \{x_1, \ldots, x_n\}$ um conjunto com N elementos.
- Dado um elemento $x_i \in S$, a estrutura de dados Conjuntos Disjuntos deve suportar três operações:

Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Seja $S = \{x_1, \ldots, x_n\}$ um conjunto com N elementos.
- Dado um elemento $x_i \in S$, a estrutura de dados Conjuntos Disjuntos deve suportar três operações:

(1) `Make-Set(x):` cria um novo conjunto S_i cujo único elemento é x. Após esta operação, x é o representante do conjunto S_i.

Conjuntos Disjuntos

- Seja $S = \{x_1, \ldots, x_n\}$ um conjunto com N elementos.
- Dado um elemento $x_i \in S$, a estrutura de dados Conjuntos Disjuntos deve suportar três operações:

(1) `Make-Set(x):` cria um novo conjunto S_i cujo único elemento é x. Após esta operação, x é o representante do conjunto S_i.

- A realização desta operação sobre os N elementos de S, resulta em uma coleção $\mathcal{C} = S_1, S_2, \ldots, S_N$ de N conjuntos, cada um com exatamente um elemento.

Conjuntos Disjuntos

- Seja $S = \{x_1, \ldots, x_n\}$ um conjunto com N elementos.
- Dado um elemento $x_i \in S$, a estrutura de dados Conjuntos Disjuntos deve suportar três operações:

(1) `Make-Set(x):` cria um novo conjunto S_i cujo único elemento é x. Após esta operação, x é o representante do conjunto S_i.

- A realização desta operação sobre os N elementos de S, resulta em uma coleção $\mathcal{C} = S_1, S_2, \ldots, S_N$ de N conjuntos, cada um com exatamente um elemento.
- Cada conjunto unitário S_i recebe um elemento distinto de S. Logo, $S_i \cap S_j = \emptyset$ para todo $i \neq j$. Isto faz os N conjuntos serem disjuntos.

Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

(2) `Union(x,y):` une os conjuntos que contém x e y, diga S_x e S_y, em um novo conjunto que é a união destes dois.

Conjuntos Disjuntos

(2) `Union(x,y):` une os conjuntos que contém x e y, diga S_x e S_y, em um novo conjunto que é a união destes dois.

- Supomos que os dois conjuntos são disjuntos antes da operação.

Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

(2) `Union(x,y):` une os conjuntos que contém x e y, diga S_x e S_y, em um novo conjunto que é a união destes dois.

- Supomos que os dois conjuntos são disjuntos antes da operação.
- O representante do conjunto resultante é qualquer membro de $S_x \cup S_y$.

Conjuntos Disjuntos

(2) `Union(x,y):` une os conjuntos que contém x e y, diga S_x e S_y, em um novo conjunto que é a união destes dois.

- Supomos que os dois conjuntos são disjuntos antes da operação.
- O representante do conjunto resultante é qualquer membro de $S_x \cup S_y$.
- Como os conjuntos na coleção devem ser disjuntos, conceitualmente, destruímos os conjuntos S_x e S_y, removendo-os da coleção. Na prática, nós frequentemente adicionamos os elementos de um dos conjuntos no outro conjunto.

Conjuntos Disjuntos

Universidade Federal do Ceará
Campus Quixadá

(2) `Union(x,y):` une os conjuntos que contém x e y, diga S_x e S_y, em um novo conjunto que é a união destes dois.

- Supomos que os dois conjuntos são disjuntos antes da operação.
- O representante do conjunto resultante é qualquer membro de $S_x \cup S_y$.
- Como os conjuntos na coleção devem ser disjuntos, conceitualmente, destruímos os conjuntos S_x e S_y, removendo-os da coleção. Na prática, nós frequentemente adicionamos os elementos de um dos conjuntos no outro conjunto.

(3) `Find-Set(x):` retorna o representante do conjunto contendo x.

Observações

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- O representante retornado pela operação `Find-set` é arbitrário. O que realmente importa é que `Find-Set(a) == Find-Set(b)` é `true` se e somente se a e b estão no mesmo conjunto da coleção.

Observações

- O representante retornado pela operação `Find-set` é arbitrário. O que realmente importa é que `Find-Set(a) == Find-Set(b)` é `true` se e somente se a e b estão no mesmo conjunto da coleção.

- A operação `Union` adiciona relações à coleção $\mathcal{C}$. Se quisermos relacionar os elementos a e b, então primeiro vemos se a e b já estão relacionados. Para isso, checamos se `Find-Set(a) == Find-Set(b)`. Se esta comparação resultar em `false`, então invocamos `Union(a,b)`.

Observações

- O representante retornado pela operação `Find-set` é arbitrário. O que realmente importa é que `Find-Set(a) == Find-Set(b)` é `true` se e somente se a e b estão no mesmo conjunto da coleção.

- A operação `Union` adiciona relações à coleção $\mathcal{C}$. Se quisermos relacionar os elementos a e b, então primeiro vemos se a e b já estão relacionados. Para isso, checamos se `Find-Set(a) == Find-Set(b)`. Se esta comparação resultar em `false`, então invocamos `Union(a,b)`.

- Como os N conjuntos iniciais construídos após N chamadas da função `Make-Set` são disjuntos, cada invocação da operação `Union` reduz o número de conjuntos da coleção $\mathcal{C}$ em um.

Observações

- O representante retornado pela operação `Find-set` é arbitrário. O que realmente importa é que `Find-Set(a) == Find-Set(b)` é `true` se e somente se a e b estão no mesmo conjunto da coleção.

- A operação `Union` adiciona relações à coleção $\mathcal{C}$. Se quisermos relacionar os elementos a e b, então primeiro vemos se a e b já estão relacionados. Para isso, checamos se `Find-Set(a) == Find-Set(b)`. Se esta comparação resultar em `false`, então invocamos `Union(a,b)`.

- Como os N conjuntos iniciais construídos após N chamadas da função `Make-Set` são disjuntos, cada invocação da operação `Union` reduz o número de conjuntos da coleção $\mathcal{C}$ em um.

- Após $N - 1$ operações `Union`, portanto, somente um conjunto resulta na coleção, esse conjunto é o próprio conjunto S dado como entrada.

Estruturas de dados para Conjuntos disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Alguns usos:

- Algumas aplicações envolvem o agrupamento de N elementos em uma coleção de conjuntos disjuntos, ou seja, uma partição dos elementos em classes de equivalência.

- Algoritmos em grafos:
  - componentes conexas
  - árvore geradora mínima

- Equivalência de tipos em compiladores

Implementação usando Florestas de Conjuntos Disjuntos

Florestas de Conjuntos Disjuntos

- Lembre que o problema não requer que a operação `Find-Set` retorne um nome específico, ele requer somente que:
    - invocações de `Find-Set` em dois elementos retorne a mesma resposta se e somente se eles estão no mesmo conjunto.

Florestas de Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Lembre que o problema não requer que a operação `Find-Set` retorne um nome específico, ele requer somente que:
  - invocações de `Find-Set` em dois elementos retorne a mesma resposta se e somente se eles estão no mesmo conjunto.

- Uma ideia seria usar uma árvore para representar cada conjunto. Assim, a raiz da árvore pode ser usada como representante do conjunto.

Florestas de Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Lembre que o problema não requer que a operação `Find-Set` retorne um nome específico, ele requer somente que:
  - invocações de `Find-Set` em dois elementos retorne a mesma resposta se e somente se eles estão no mesmo conjunto.

- Uma ideia seria usar uma árvore para representar cada conjunto. Assim, a raiz da árvore pode ser usada como representante do conjunto.
  - Uma coleção de árvores é chamada em teoria dos grafos de floresta. Daí o nome **floresta de conjuntos disjuntos** dado a esta abordagem.

Florestas de Conjuntos Disjuntos

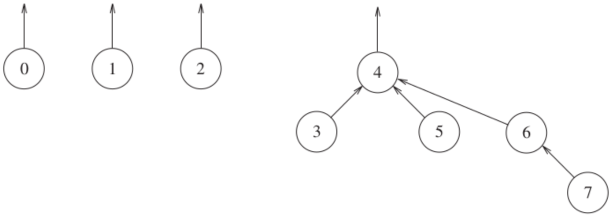
- Em uma floresta de conjuntos disjuntos, cada nó de árvore aponta somente para o seu pai. Além do ponteiro `parent`, cada nó armazena também o valor do elemento do conjunto.

Florestas de Conjuntos Disjuntos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Em uma floresta de conjuntos disjuntos, cada nó de árvore aponta somente para o seu pai. Além do ponteiro `parent`, cada nó armazena também o valor do elemento do conjunto.

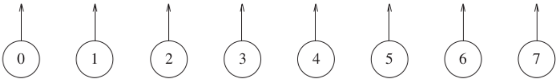
- Exemplo: Dada uma coleção de conjuntos disjuntos $\mathcal{C} = \{S_1, S_2, S_3, S_4\}$, com $S_1 = \{0\}$, $S_2 = \{1\}$, $S_3 = \{2\}$ e $S_4 = \{3, 4, 5, 6, 7\}$ uma possível representação destes conjuntos usando florestas de conjuntos disjuntos é ilustrada abaixo.



Operação `Make-Set(x)`

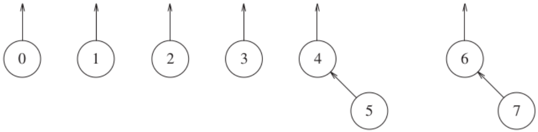
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- A operação `Make-Set(x)` cria uma árvore com somente um nó. Esta operação tem complexidade $O(1)$.
- Exemplo: Para o exemplo anterior, temos 8 árvores inicialmente na coleção criadas invocando Make-Set oito vezes com os valores $0, 1, \ldots, 7$.

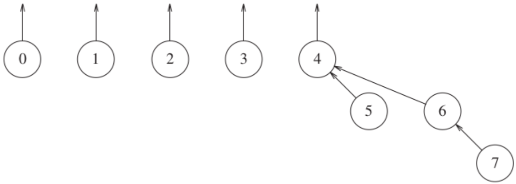


Operação `Union(x,y)`

- A operação `Union(x,y)` faz com que a raiz de uma árvore aponte para a raiz da outra. Complexidade: $O(1)$.



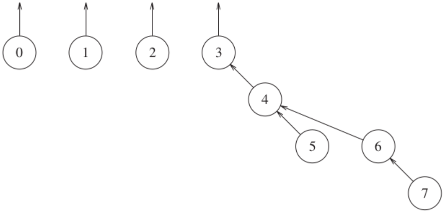
Florestas de seis conjuntos disjuntos



Após invocar `Union(5,6)`

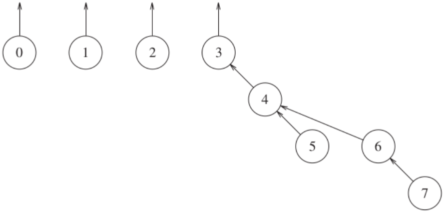
Operação `Find-Set(x)`

- Realizamos uma operação `Find-Set(x)` seguindo os ponteiros `parent` até encontrarmos a raiz da árvore.



Operação `Find-Set(x)`

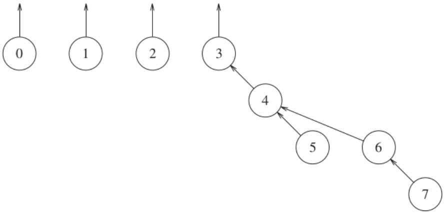
- Realizamos uma operação `Find-Set(x)` seguindo os ponteiros `parent` até encontrarmos a raiz da árvore.



- Complexidade: $O(h)$, onde h é a altura da árvore, que é $\lfloor \lg n \rfloor + 1$ no melhor caso e n no pior caso. Assim, a complexidade de pior caso da operação `Find-Set(x)` é $O(n)$.

Operação Find-Set(x)

- Realizamos uma operação `Find-Set(x)` seguindo os ponteiros `parent` até encontrarmos a raiz da árvore.



- Complexidade: $O(h)$, onde h é a altura da árvore, que é $\lfloor \lg n \rfloor + 1$ no melhor caso e n no pior caso. Assim, a complexidade de pior caso da operação `Find-Set(x)` é $O(n)$.

- O tempo de execução é computado para uma sequência de m operações `Make-Set(x)`, `Find-Set(x)` e `Union(x,y)`. Neste caso, m operações consecutivas poderiam demorar $O(mn)$ no pior caso. Tempo de execução quadrático para uma sequência de operações é geralmente inaceitável.

Pseudocódigos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

MAKE-SET(x)

$x.parent = x$

FIND-SET(x)

$y = x$
while $y.parent \neq y$
    $y = y.parent$
return y

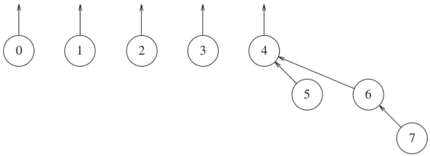
UNION(x_1, y_1)

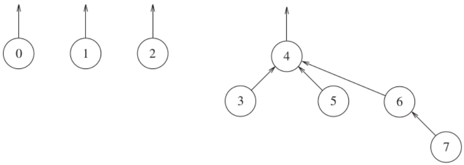
$x =$ FIND-SET(x_1)
$y =$ FIND-SET(y_1)
$y.parent = x$

Heurística: `Union-by-Size`

Union-by-size (União por tamanho)

- Heurística Union-by-size: Para cada nó, nós mantemos um novo campo inteiro chamado **size**, que guarda o número de nós da árvore enraizada naquele nó. Na union-by-size, fazemos a raiz da árvore que tem o menor size apontar para a raiz da árvore que tem o maior size durante uma operação `Union(x,y)`.





`Union-by-size` — Pseudocódigo

Universidade Federal do Ceará
Campus Quixadá

$\textsc{Make-Set}(x)$

$x.parent = x$
$x.size = 1$

$\textsc{Find-Set}(x)$

$y = x$
while $y.parent \neq y$
    $y = y.parent$
return y

$\textsc{Union}(x_1, y_1)$

$x = \textsc{Find-Set}(x_1)$
$y = \textsc{Find-Set}(y_1)$
if $x.size \geq y.size$
    $x.size = x.size + y.size$
    $y.parent = x$
else
    $y.size = y.size + x.size$
    $x.parent = y$

Análide de Complexidade: Union-by-Size

Teorema: Considere uma sequência de n operações MAKE-SET seguidas de m operações Union e Find-Set em conjuntos disjuntos, empregando union-by-size. Seja x um nó qualquer do universo considerado e T_x a árvore que armazena x. Após a última operação, T_x tem altura da ordem $O(\lg |T_x|)$, onde $|T_x|$ é o número de nós da árvore T_x.

Demonstração:

Análide de Complexidade: Union-by-Size

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Teorema: Considere uma sequência de n operações `MAKE-SET` seguidas de m operações `Union` e `Find-Set` em conjuntos disjuntos, empregando `union-by-size`. Seja x um nó qualquer do universo considerado e T_x a árvore que armazena x. Após a última operação, T_x tem altura da ordem $O(\lg |T_x|)$, onde $|T_x|$ é o número de nós da árvore T_x.

Demonstração:

- A prova consiste em mostrar que $2^{h(T_x)-1} \leq |T_x|$, onde $h(T_x)$ é a altura da árvore T_x. Nesse caso, $h(T_x) = O(\lg |T_x|)$, e o resultado segue.

Análide de Complexidade: Union-by-Size

Universidade Federal do Ceará
Campus Quixadá

Teorema: Considere uma sequência de n operações `MAKE-SET` seguidas de m operações `Union` e `Find-Set` em conjuntos disjuntos, empregando `union-by-size`. Seja x um nó qualquer do universo considerado e T_x a árvore que armazena x. Após a última operação, T_x tem altura da ordem $O(\lg |T_x|)$, onde $|T_x|$ é o número de nós da árvore T_x.

Demonstração:

- A prova consiste em mostrar que $2^{h(T_x)-1} \leq |T_x|$, onde $h(T_x)$ é a altura da árvore T_x. Nesse caso, $h(T_x) = O(\lg |T_x|)$, e o resultado segue.

- A prova de que $2^{h(T_x)-1} \leq |T_x|$ é feita por **indução em m**, o número de operações `Union` e `Find-Set` realizadas.

Análide de Complexidade: Union-by-Size

Teorema: Considere uma sequência de n operações `MAKE-SET` seguidas de m operações `Union` e `Find-Set` em conjuntos disjuntos, empregando `union-by-size`. Seja x um nó qualquer do universo considerado e T_x a árvore que armazena x. Após a última operação, T_x tem altura da ordem $O(\lg |T_x|)$, onde $|T_x|$ é o número de nós da árvore T_x.

Demonstração:

- A prova consiste em mostrar que $2^{h(T_x)-1} \leq |T_x|$, onde $h(T_x)$ é a altura da árvore T_x. Nesse caso, $h(T_x) = O(\lg |T_x|)$, e o resultado segue.

- A prova de que $2^{h(T_x)-1} \leq |T_x|$ é feita por **indução em m**, o número de operações `Union` e `Find-Set` realizadas.

- **Caso Base:** Para $m = 0$, o teorema é válido; tem-se construída a floresta de n árvores, cada uma com um elemento.

Análide de Complexidade: Union-by-Size

- Pela **hipótese de indução**, antes da m-ésima operação, o teorema está correto para todos os nós x.

  Ou seja, $2^{h(T_x)-1} \leq |T_x|$ para todo nó x.

  É preciso provar que é válido após a m-ésima operação.

Análide de Complexidade: Union-by-Size

- Pela **hipótese de indução**, antes da m-ésima operação, o teorema está correto para todos os nós x.

  Ou seja, $2^{h(T_x)-1} \leq |T_x|$ para todo nó x.

  É preciso provar que é válido após a m-ésima operação.

Passo indutivo:

- **Caso 1:** Se `Find-Set(x)` é a m-ésima operação, não há mudança na estrutura e vale o teorema.

Análide de Complexidade: Union-by-Size

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Pela **hipótese de indução**, antes da m-ésima operação, o teorema está correto para todos os nós x.

  Ou seja, $2^{h(T_x)-1} \leq |T_x|$ para todo nó x.

  É preciso provar que é válido após a m-ésima operação.

Passo indutivo:

- **Caso 1:** Se `Find-Set(x)` é a m-ésima operação, não há mudança na estrutura e vale o teorema.

- **Caso 2:** Então, suponha que a m-ésima operação é `Union(x,y)`.
  Sem perda de generalidade, pode-se assumir que $|T_x| \leq |T_y|$. Nesse caso, o conjunto x será ligado ao conjunto y, criando uma nova árvore T.

Análide de Complexidade: Union-by-Size

- Claramente, $h(T) = \max\{1 + h(T_x), h(T_y)\}$. Logo,

$$\begin{aligned}
h(T) - 1 &= \max\{1 + h(T_x), h(T_y)\} - 1 \\
h(T) - 1 &= \max\{1 + h(T_x) - 1, h(T_y) - 1\} \\
2^{h(T)-1} &= 2^{\max\{1+h(T_x)-1, h(T_y)-1\}} \\
&= \max\{2^{1+h(T_x)-1}, 2^{h(T_y)-1}\} \\
&\leq \max\{2|T_x|, |T_y|\}.
\end{aligned}$$

A última desigualdade segue por hipótese de indução.

Análide de Complexidade: Union-by-Size

- Claramente, $h(T) = \max\{1 + h(T_x), h(T_y)\}$. Logo,

$$\begin{aligned} h(T) - 1 &= \max\{1 + h(T_x), h(T_y)\} - 1 \\ h(T) - 1 &= \max\{1 + h(T_x) - 1, h(T_y) - 1\} \\ 2^{h(T)-1} &= 2^{\max\{1+h(T_x)-1, h(T_y)-1\}} \\ &= \max\{2^{1+h(T_x)-1}, 2^{h(T_y)-1}\} \\ &\leq \max\{2|T_x|, |T_y|\}. \end{aligned}$$

  A última desigualdade segue por hipótese de indução.

- Por outro lado, sabe-se que a operação `Union(x,y)` gera uma árvore T tal que $|T_x| + |T_y| = |T|$, e considerou-se $|T_x| \leq |T_y|$.

Análide de Complexidade: Union-by-Size

- Claramente, $h(T) = \max\{1 + h(T_x), h(T_y)\}$. Logo,

$$\begin{aligned}
h(T) - 1 &= \max\{1 + h(T_x), h(T_y)\} - 1 \\
h(T) - 1 &= \max\{1 + h(T_x) - 1, h(T_y) - 1\} \\
2^{h(T)-1} &= 2^{\max\{1+h(T_x)-1, h(T_y)-1\}} \\
&= \max\{2^{1+h(T_x)-1}, 2^{h(T_y)-1}\} \\
&\leq \max\{2|T_x|, |T_y|\}.
\end{aligned}$$

  A última desigualdade segue por hipótese de indução.
- Por outro lado, sabe-se que a operação `Union(x,y)` gera uma árvore T tal que $|T_x| + |T_y| = |T|$, e considerou-se $|T_x| \leq |T_y|$.
- Pode-se concluir que $2|T_x| \leq |T|$ e $|T_y| \leq |T|$. Assim, tem-se que

Análide de Complexidade: Union-by-Size

- Claramente, $h(T) = \max\{1 + h(T_x), h(T_y)\}$. Logo,

$$\begin{aligned}
h(T) - 1 &= \max\{1 + h(T_x), h(T_y)\} - 1 \\
h(T) - 1 &= \max\{1 + h(T_x) - 1, h(T_y) - 1\} \\
2^{h(T)-1} &= 2^{\max\{1+h(T_x)-1, h(T_y)-1\}} \\
&= \max\{2^{1+h(T_x)-1}, 2^{h(T_y)-1}\} \\
&\leq \max\{2|T_x|, |T_y|\}.
\end{aligned}$$

  A última desigualdade segue por hipótese de indução.

- Por outro lado, sabe-se que a operação `Union(x,y)` gera uma árvore T tal que $|T_x| + |T_y| = |T|$, e considerou-se $|T_x| \leq |T_y|$.
- Pode-se concluir que $2|T_x| \leq |T|$ e $|T_y| \leq |T|$. Assim, tem-se que

$$\begin{aligned}
2^{h(T)-1} &\leq \max\{2|T_x|, |T_y|\} \\
2^{h(T)-1} &\leq \max\{|T|, |T|\} \\
2^{h(T)-1} &\leq |T|. \qquad \blacksquare
\end{aligned}$$

Análide de Complexidade: Union-by-Size

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O seguinte resultado segue imediatamente do teorema anterior:

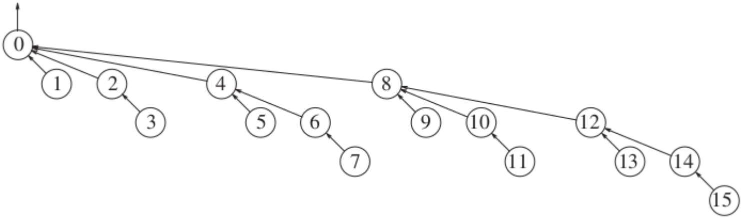
Corolário: Considere uma sequência de n operações `MAKE-SET` seguidas de m operações `Union` e `Find-Set` em conjuntos disjuntos, empregando `union-by-size`.

O tempo de execução de uma operação `Find-Set` é $O(\lg n)$ e uma sequência de m operações `Union` e `Find-Set` demora tempo total $O(m \lg n)$ no pior caso.

Exemplo de pior caso: Union-by-Size

Universidade Federal do Ceará
Campus Quixadá

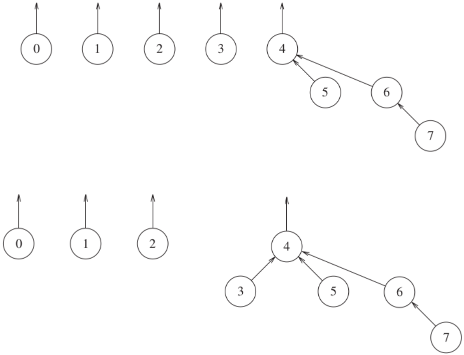
- A figura abaixo mostra a pior árvore possível depois de 16 operações de união na coleção inicial $\mathcal{C} = \{\{i\}: 0 \leq i \leq 15\}$ e ela é obtida se todas as uniões são realizadas entre árvores com o mesmo tamanho.



Heurística: `Union-by-Rank`

Union by rank (União por rank)

- Heurística Union by rank: Para cada nó, nós mantemos um novo campo inteiro chamado **rank**, que é um limitante superior na altura do nó. Na união por rank, fazemos a raiz da árvore que tem o menor rank apontar para a raiz da árvore que tem o maior rank durante uma operação Union(x,y).



`Union by rank` — Pseudocódigo

MAKE-SET(x)

```
x.parent = x
x.rank = 1
```

FIND-SET(x)

```
y = x
while y.parent ≠ y
        y = y.parent
return y
```

UNION(x_1, y_1)

```
x = FIND-SET(x1)
y = FIND-SET(y1)
if x.rank > y.rank
        y.parent = x
else
        x.parent = y
        if x.rank == y.rank
                y.rank = y.rank + 1
```

`Union by rank` – Análise de Complexidade

- Muito parecida com as análises feitas até agora.

Exercício 1: Prove que todo nó tem rank no máximo $\lfloor \lg n \rfloor$.

Exercício 2: Usando o enunciado do Exercício 1, dê uma prova simples de que as operações em uma floresta de conjuntos disjuntos usando union-by-rank executam em tempo $O(m \lg n)$.

Exercício 3: Dê uma sequência de m operações Make-Set, Union e Find-Set, n das quais são operações Make-Set, que leva $\Omega(m \lg n)$ quando usamos union by rank.

Heurística: Compressão de Caminhos

Compressão de Caminhos (path compression)

Universidade Federal do Ceará
Campus Quixadá

- Vimos duas heurísticas que melhoram o desempenho da função `Union`. Veremos agora como tornar a função `Find-Set` mais eficiente.

Compressão de Caminhos (path compression)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Vimos duas heurísticas que melhoram o desempenho da função `Union`. Veremos agora como tornar a função `Find-Set` mais eficiente.

- A operação `Find-Set` consiste em um percurso do nó desejado até a raiz da árvore. A ideia básica para diminuir o tempo de busca é compactar o caminho percorrido, de forma que as próximas buscas percorram um caminho menor.
    - A compactação é realizada durante o próprio procedimento de busca.

Compressão de Caminhos (path compression)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

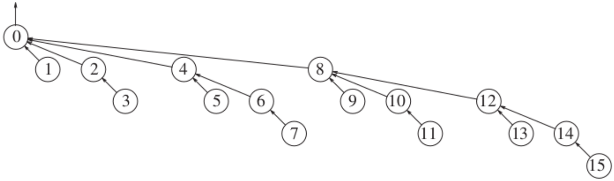
- Vimos duas heurísticas que melhoram o desempenho da função `Union`. Veremos agora como tornar a função `Find-Set` mais eficiente.

- A operação `Find-Set` consiste em um percurso do nó desejado até a raiz da árvore. A ideia básica para diminuir o tempo de busca é compactar o caminho percorrido, de forma que as próximas buscas percorram um caminho menor.
  - A compactação é realizada durante o próprio procedimento de busca.

- Dada uma execução da função `Find-Set(x)`, o efeito da `compressão de caminho` é que cada nó y no caminho que vai de x até a raiz da árvore tem o seu ponteiro $y.parent$ modificado para apontar para a raiz da árvore.

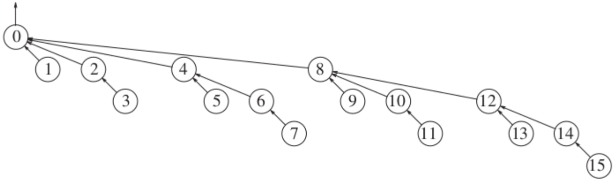
Compressão de Caminhos – Exemplo

Qual o efeito de executar `Find-Set(14)` usando compressão de caminho na árvore abaixo?

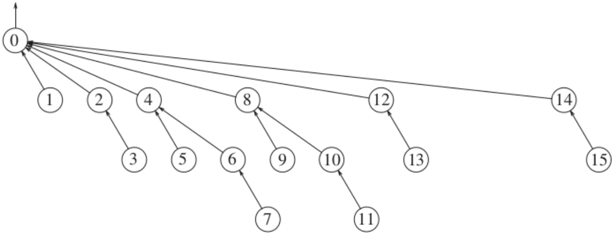


Compressão de Caminhos – Exemplo

Qual o efeito de executar `Find-Set(14)` usando compressão de caminho na árvore abaixo?



Resultado:



`Find-Set` com compressão de caminhos

Find-Set com compressão de caminhos

Find-Set(x)

if $x \neq x.parent$
    $x.parent = \text{Find-Set}(x.parent)$
return $x.parent$

Find-Set com compressão de caminhos

$\textsc{Find-Set}(x)$

```
if x ≠ x.parent
    x.parent = FIND-SET(x.parent)
return x.parent
```

Pode ser provado que, com compressão de caminhos (sem nenhuma melhoria na função Union), o tempo de execução de qualquer sequência de n operações Make-Set, f operações Find-Set e até $n-1$ operações Union é

$$\Theta(n + f(1 + \log_{2+f/n} n)).$$

União por Rank e Compressão de Caminhos

União por Rank e Compressão de Caminhos

$\textsc{Find-Set}(x)$

if $x \neq x.parent$
    $x.parent = \textsc{Find-Set}(x.parent)$
return $x.parent$

$\textsc{Union}(x_1, y_1)$

$x = \textsc{Find-Set}(x_1)$
$y = \textsc{Find-Set}(y_1)$
if $x.rank > y.rank$
    $y.parent = x$
else
    $x.parent = y$
    if $x.rank == y.rank$
        $y.rank = y.rank + 1$

$\textsc{Make-Set}(x)$

$x.parent = x$
$x.rank = 1$

União por Rank e Compressão de Caminhos

Observações de Cormen et al.:

Considerando $n - 1$ operações `Union` e m operações `Find-Set`.

- Quando usamos a união por rank e a compressão de caminho, o tempo de execução do pior caso é $O(m\alpha(n))$, onde $\alpha(n)$ é uma função de crescimento muito lento. (inversa da função de Ackermann)

- Em qualquer aplicação concebível de uma estrutura de dados de conjuntos disjuntos, $\alpha(n) \leq 4$; assim, podemos considerar o tempo de execução como linear em relação a m em todas as situações práticas.

Implementação em C++

Implementação

Conjuntos disjuntos com union by rank e compressão de caminhos

- A fim de facilitar a implementação, vamos supor que os n elementos que compõem os conjuntos disjuntos são os inteiros não negativos pertencentes ao conjunto $\{0, 1, \ldots, n-1\}$.
- Assim, podemos usar um array `S[0..n-1]` para armazenar os n nós da floresta de conjuntos disjuntos.
- Cada nó da floresta será um `struct Node` que terá dois campos: `parent` e `rank`. Assim, o array `S[0..n-1]` é um array de `struct Node`. O nó `S[i]` corresponde ao elemento i da floresta, para $0 \leq i \leq n-1$.

DisjointSets.h

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
#ifndef DSETS_H
#define DSETS_H
#include <vector>

/**
 * Implementation of the disjoint-sets data structure
 * using path compression and union by rank
 */
class DisjointSets {
public:
    explicit DisjointSets( int numElements );
    int findSet( int x );
    void unionSets( int x, int y );
private:
    struct Node { int parent; int rank; };
    std::vector<Node> sets;
};
```

DisjointSets.h – Continuação

```
/**
 * Construct the disjoint sets object.
 * numElements is the initial number of disjoint sets.
 */
DisjointSets::DisjointSets( int numElements ) {
    sets.resize( numElements );
    for( int i = 0; i < numElements; ++i ) {
        sets[i].parent = i;
        sets[i].rank = 1;
    }
}

/**
 * Performs a find-Set with path compression.
 * Returns the representative of the set containing x.
 */
int DisjointSets::findSet( int x ) {
    if ( sets[x].parent != x )
        sets[x].parent = findSet( sets[x].parent );
    return sets[x].parent;
}
```

DisjointSets.h – Final do arquivo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
/**
 * Union of two disjoint sets using union by rank
 */
void DisjointSets::unionSets( int x, int y ) {
    int xroot = findSet(x);
    int yroot = findSet(y);
    if( sets[xroot].rank > sets[yroot].rank )
        sets[yroot].parent = xroot;
    else {
        sets[xroot].parent = yroot;
        if(sets[xroot].rank == sets[yroot].rank)
            ++sets[yroot].rank;
    }
}

#endif
```

Universidade Federal do Ceará
Campus Quixadá

Exercícios

Exercício

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

1. Mostre a estrutura de dados de conjuntos disjuntos resultante e as respostas retornadas pelas operações `FIND-SET` no programa abaixo. Considere a estrutura de dados de conjuntos disjuntos representada como uma floresta de conjuntos disjuntos com union by rank e compressão de caminho.

```
for(int i = 1; i <= 16; i++)
  MAKE-SET(i)
for(int i = 1; i <= 15; i = i + 2)
  UNION(i, i+1)
for(int i = 1; i <= 13; i = i + 4)
  UNION(i, i+2)
UNION(1,7)
UNION(11,13)
UNION(1,10)
x = FIND-SET(2)
y = FIND-SET(15)
```

Exercícios

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

2. Escreva uma versão não recursiva de `Find-Set` com compressão de caminho.

3. Dê uma sequência de m operações Make-Set, Union e Find-Set, na qual n são operações Make-Set, que demore o tempo $O(m \lg n)$ quando usamos somente union by rank.

4. Suponha que queiramos acrescentar a operação `Print-Set(x)`, à qual é dado um nó x e que imprime todos os membros do conjunto de x em qualquer ordem. Mostre como podemos acrescentar apenas um único atributo a cada nó em uma floresta de conjuntos disjuntos de modo que `Print-Set(x)` demore tempo linear em relação ao número de membros do conjunto de x e que os tempos de execução assintóticos das outras operações permaneçam inalterados. Suponha que podemos imprimir cada membro do conjunto no tempo $O(1)$.

FIM