Análise de Algoritmos
Aula 17

Prof. Murilo V. G. da Silva

DINF/UFPR

(material da disciplina: André Guedes, Renato Carmo, Murilo da Silva)

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

- Requerimento:

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

- Requerimento: os seguintes problemas devem admitir soluções eficientes.

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

- Requerimento: os seguintes problemas devem admitir soluções eficientes.

União de classes (union)

- Entrada: Dois elementos $c_1, c_2 \in C$
- Saída: A partição

$$P - \{C_1, C_2\} \cup \{C_1 \cup C_2\},$$

onde C_1 e C_2 são as classes originais de c_1 e c_2 em P.

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

- Requerimento: os seguintes problemas devem admitir soluções eficientes.

União de classes (union)

- Entrada: Dois elementos $c_1, c_2 \in C$
- Saída: A partição

$$P - \{C_1, C_2\} \cup \{C_1 \cup C_2\},$$

onde C_1 e C_2 são as classes originais de c_1 e c_2 em P.

Obs: Iremos nos referir as partes do conjunto como classes

Análise de Hopcroft-Ullman para Union-find

Queremos uma estrutura de dados para armazenar uma partição P de um conjunto finito C com n elementos.

- Requerimento: os seguintes problemas devem admitir soluções eficientes.

União de classes (union)

- Entrada: Dois elementos $c_1, c_2 \in C$
- Saída: A partição

$$P - \{C_1, C_2\} \cup \{C_1 \cup C_2\},$$

onde C_1 e C_2 são as classes originais de c_1 e c_2 em P.

Obs: Iremos nos referir as partes do conjunto como classes

Identificação da Classe (find)

- Entrada: Um elemento $c \in C$.
- Saída: A classe de P à qual c pertence.

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor PAI indexados por $[1..n]$.

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor PAI indexados por $[1..n]$.
2. os elementos de C são armazenados em v.

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor PAI indexados por $[1..n]$.
2. os elementos de C são armazenados em v.
3. para cada $v[i] \in C$, o valor de $\text{PAI}[i]$ é o índice em v de um elemento de C da mesma classe que $v[i]$.

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor $\textsc{pai}$ indexados por $[1..n]$.
2. os elementos de C são armazenados em v.
3. para cada $v[i] \in C$, o valor de $\textsc{pai}[i]$ é o índice em v de um elemento de C da mesma classe que $v[i]$.
4. Dizemos daí que o elemento $v[\textsc{pai}[i]]$ é o *pai* de $c = v[i]$ e que c é *filho* de $v[\textsc{pai}[c]]$.

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor PAI indexados por $[1..n]$.
2. os elementos de C são armazenados em v.
3. para cada $v[i] \in C$, o valor de $\text{PAI}[i]$ é o índice em v de um elemento de C da mesma classe que $v[i]$.
4. Dizemos daí que o elemento $v[\text{PAI}[i]]$ é o *pai* de $c = v[i]$ e que c é *filho* de $v[\text{PAI}[c]]$.

- Observe que como c é armazenado em um vetor, quando estamos falando do elemento, o que é realmente relevante é o índice i de v que guarda c

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor $\textsc{pai}$ indexados por $[1..n]$.
2. os elementos de C são armazenados em v.
3. para cada $v[i] \in C$, o valor de $\textsc{pai}[i]$ é o índice em v de um elemento de C da mesma classe que $v[i]$.
4. Dizemos daí que o elemento $v[\textsc{pai}[i]]$ é o *pai* de $c = v[i]$ e que c é *filho* de $v[\textsc{pai}[c]]$.

- Observe que como c é armazenado em um vetor, quando estamos falando do elemento, o que é realmente relevante é o índice i de v que guarda c

Convenção: diremos algumas vezes "o elemento $c \in C$" para referir-nos ao "índice do elemento $c \in C$ em v".

Análise de Hopcroft-Ullman para Union-find

Uma solução simples:

1. usamos um vetor v e um vetor PAI indexados por $[1..n]$.
2. os elementos de C são armazenados em v.
3. para cada $v[i] \in C$, o valor de $\text{PAI}[i]$ é o índice em v de um elemento de C da mesma classe que $v[i]$.
4. Dizemos daí que o elemento $v[\text{PAI}[i]]$ é o *pai* de $c = v[i]$ e que c é *filho* de $v[\text{PAI}[c]]$.

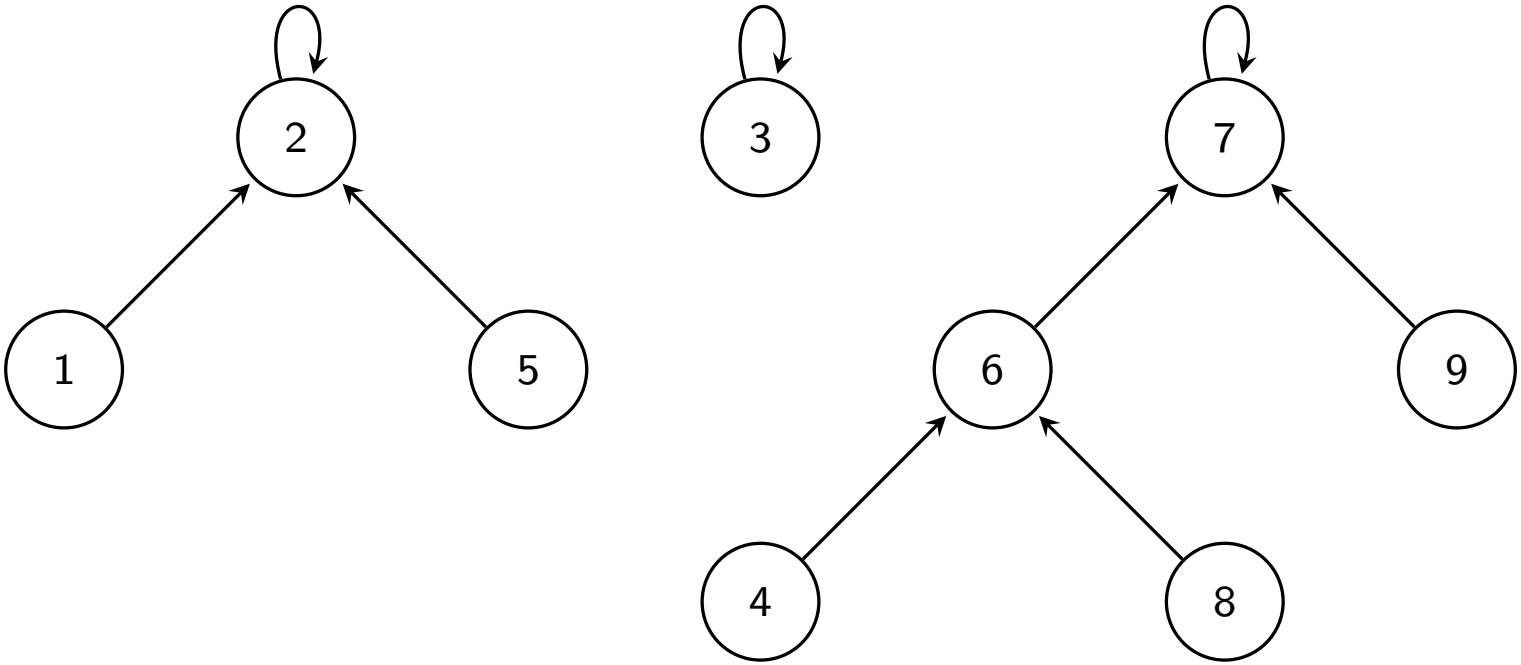
- Observe que como c é armazenado em um vetor, quando estamos falando do elemento, o que é realmente relevante é o índice i de v que guarda c

Convenção: diremos algumas vezes "o elemento $c \in C$" para referir-nos ao "índice do elemento $c \in C$ em v".

Definimos: $\text{PAI}(i) = v[\text{PAI}[i]]$.

Análise de Hopcroft-Ullman para Union-find

Figura: Classes vistas como árvores



2	2	3	6	2	7	7	6	7
1	2	3	4	5	6	7	8	9

Obs: o vetor v está omitido aqui. Temos apenas o vetor de pais. Como mencionado anteriormente, os 9 elementos do conjunto são referenciados pelos índices de v.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\textsc{pai}(c) = c$.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\textsc{pai}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

1. inicialmente a estrutura tem n classes unitárias.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

1. inicialmente a estrutura tem n classes unitárias.
   (i.e., cada elemento é raiz de sua classe)

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

1. inicialmente a estrutura tem n classes unitárias.
   (i.e., cada elemento é raiz de sua classe)
2. não há circularidade, isto é, se c não é raiz de sua classe, então $\text{PAI}^k(c) \neq c$ para todo $k > 0$.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

1. inicialmente a estrutura tem n classes unitárias.
   (i.e., cada elemento é raiz de sua classe)
2. não há circularidade, isto é, se c não é raiz de sua classe, então $\text{PAI}^k(c) \neq c$ para todo $k > 0$.
3. $\text{PAI}^n(c) = R(c)$, para todo $c \in C$.

Análise de Hopcroft-Ullman para Union-find

Raiz: c é *raiz* ou *representante* de sua classe se $\text{PAI}(c) = c$.

- A raiz da classe de c será denotada por $R(c)$.

Requerimentos para os algoritmos

1. inicialmente a estrutura tem n classes unitárias.
   (i.e., cada elemento é raiz de sua classe)
2. não há circularidade, isto é, se c não é raiz de sua classe, então $\text{PAI}^k(c) \neq c$ para todo $k > 0$.
3. $\text{PAI}^n(c) = R(c)$, para todo $c \in C$.
4. $R(c) = R(c')$ se e somente se c e c' estão na mesma classe.

Análise de Hopcroft-Ullman para Union-find

Algoritmos:

Análise de Hopcroft-Ullman para Union-find

Algoritmos:

---

union(c_1, c_2)

Entrada : (os índices em v de) dois elementos de classes distintas.
$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
$\text{PAI}[R_1] \leftarrow R_2$

---

find(c)

Entrada : (o índice em v de) um elemento de C.
Saída : (o índice em v da) raiz da classe de c.
$R \leftarrow c$
Enquanto $\text{PAI}[R] \neq R$
    $R \leftarrow \text{PAI}[R]$
Devolva R

Análise de Hopcroft-Ullman para Union-find

Algoritmos:

---

union(c_1, c_2)

Entrada : (os índices em v de) dois elementos de classes distintas.
$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
$\text{PAI}[R_1] \leftarrow R_2$

---

find(c)

Entrada : (o índice em v de) um elemento de C.
Saída : (o índice em v da) raiz da classe de c.
$R \leftarrow c$
Enquanto $\text{PAI}[R] \neq R$
    $R \leftarrow \text{PAI}[R]$
Devolva R

- Está correto?

Análise de Hopcroft-Ullman para Union-find

Algoritmos:

---

union(c_1, c_2)

Entrada : (os índices em v de) dois elementos de classes distintas.
$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
$\text{PAI}[R_1] \leftarrow R_2$

---

find(c)

Entrada : (o índice em v de) um elemento de C.
Saída : (o índice em v da) raiz da classe de c.
$R \leftarrow c$
Enquanto $\text{PAI}[R] \neq R$
    $R \leftarrow \text{PAI}[R]$
Devolva R

- Está correto? Prove! (Exercício bônus)

Análise de Hopcroft-Ullman para Union-find

Algoritmos:

---

union(c_1, c_2)

Entrada : (os índices em v de) dois elementos de classes distintas.
$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
$\text{PAI}[R_1] \leftarrow R_2$

---

find(c)

Entrada : (o índice em v de) um elemento de C.
Saída : (o índice em v da) raiz da classe de c.
$R \leftarrow c$
Enquanto $\text{PAI}[R] \neq R$
    $R \leftarrow \text{PAI}[R]$
Devolva R

- Está correto? Prove! (Exercício bônus)
- Quanto custa?

Análise de Hopcroft-Ullman para Union-find

Rank de um elemento

A *rank de* $c \in C$ é

$$r(c) = \begin{cases} 0, & \text{se } c \text{ é folha,} \\ 1 + \max\{r(f) \mid f \text{ é filho de } c\}, & \text{se } c \text{ não é folha.} \end{cases}$$

Análise de Hopcroft-Ullman para Union-find

Rank de um elemento

A *rank de* $c \in C$ é

$$r(c) = \begin{cases} 0, & \text{se } c \text{ é folha,} \\ 1 + \max\{r(f) \mid f \text{ é filho de } c\}, & \text{se } c \text{ não é folha.} \end{cases}$$

Observe que $r(c) < r(pai(c))$, para todo $c \in C$.

Análise de Hopcroft-Ullman para Union-find

Rank de um elemento

A *rank de* $c \in C$ é

$$r(c) = \begin{cases} 0, & \text{se } c \text{ é folha,} \\ 1 + \max\{r(f) \mid f \text{ é filho de } c\}, & \text{se } c \text{ não é folha.} \end{cases}$$

Observe que $r(c) < r(pai(c))$, para todo $c \in C$.

Nível de um elemento

O *nível* de c é

$$l(c) = \min\left\{k \in \mathbb{N} \mid \text{PAI}^k(c) = R(c)\right\}$$

Análise de Hopcroft-Ullman para Union-find

Rank de um elemento

A *rank de* $c \in C$ é

$$r(c) = \begin{cases} 0, & \text{se } c \text{ é folha,} \\ 1 + \max\{r(f) \mid f \text{ é filho de } c\}, & \text{se } c \text{ não é folha.} \end{cases}$$

Observe que $r(c) < r(pai(c))$, para todo $c \in C$.

Nível de um elemento

O *nível* de c é

$$l(c) = \min\left\{k \in \mathbb{N} \mid \text{PAI}^k(c) = R(c)\right\}$$

Observe que $r(R(c)) = \max\{l(d) \mid d \text{ é descendente de } c\}$, para todo $c \in C$.

Análise de Hopcroft-Ullman para Union-find

Rank de um elemento

A *rank de* $c \in C$ é

$$r(c) = \begin{cases} 0, & \text{se } c \text{ é folha,} \\ 1 + \max\{r(f) \mid f \text{ é filho de } c\}, & \text{se } c \text{ não é folha.} \end{cases}$$

Observe que $r(c) < r(pai(c))$, para todo $c \in C$.

Nível de um elemento

O *nível* de c é

$$l(c) = \min\left\{k \in \mathbb{N} \mid \text{PAI}^k(c) = R(c)\right\}$$

Observe que $r(R(c)) = \max\{l(d) \mid d \text{ é descendente de } c\}$, para todo $c \in C$.

Caminho até a raiz

O *caminho* de c até sua raíz é a sequência $(c, \text{PAI}(c), \text{PAI}^2(c), \ldots, \text{PAI}^{l(c)}(c) = R(c))$.

Análise de Hopcroft-Ullman para Union-find

Lema

O tempo de execução de find(c) é $\Theta(l(c))$.

Análise de Hopcroft-Ullman para Union-find

Lema

O tempo de execução de find(c) é $\Theta(l(c))$.

Corolario

O tempo de execução de union(c_1, c_2) é $\Theta(\max\{l(c_1), l(c_2)\})$.

Análise de Hopcroft-Ullman para Union-find

Lema

O tempo de execução de find(c) é $\Theta(l(c))$.

Corolario

O tempo de execução de union(c_1, c_2) é $\Theta(\max\{l(c_1), l(c_2)\})$.

Corolario

O tempo de execução de pior caso de find(c) e de union(c_1, c_2) é $\Theta(\max\{r(c) \mid c \in C\})$.

Análise de Hopcroft-Ullman para Union-find

Lema

O tempo de execução de find(c) é $\Theta(l(c))$.

Corolario

O tempo de execução de union(c_1, c_2) é $\Theta(\max\{l(c_1), l(c_2)\})$.

Corolario

O tempo de execução de pior caso de find(c) e de union(c_1, c_2) é $\Theta(\max\{r(c) \mid c \in C\})$.

Corolario

O tempo de execução de pior caso de find(c) e de union(c_1, c_2) é $\Theta(n)$.

Análise de Hopcroft-Ullman para Union-find

Lema

O tempo de execução de find(c) é $\Theta(l(c))$.

Corolario

O tempo de execução de union(c_1, c_2) é $\Theta(\max\{l(c_1), l(c_2)\})$.

Corolario

O tempo de execução de pior caso de find(c) e de union(c_1, c_2) é $\Theta(\max\{r(c) \mid c \in C\})$.

Corolario

O tempo de execução de pior caso de find(c) e de union(c_1, c_2) é $\Theta(n)$.

Prova: Exercício.

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

$\text{union}_B(c_1, c_2)$

$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
Se $R_1.rank < R_2.rank$
    $\text{PAI}[R_1] \leftarrow R_2$
Senão
    $\text{PAI}[R_2] \leftarrow R_1$
Se $R_1.rank = R_2.rank$
    $R_1.\text{rank} \leftarrow R_1.\text{rank} + 1$

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

$\text{union}_B(c_1, c_2)$

$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
Se $R_1.rank < R_2.rank$
    $\text{PAI}[R_1] \leftarrow R_2$
Senão
    $\text{PAI}[R_2] \leftarrow R_1$
Se $R_1.rank = R_2.rank$
    $R_1.\text{rank} \leftarrow R_1.\text{rank} + 1$

Observe que

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

---

union$_B(c_1, c_2)$

$R_1 \leftarrow$ find(c_1)
$R_2 \leftarrow$ find(c_2)
Se $R_1.rank < R_2.rank$
    PAI$[R_1] \leftarrow R_2$
Senão
    PAI$[R_2] \leftarrow R_1$
Se $R_1.rank = R_2.rank$
    R_1.rank $\leftarrow R_1$.rank $+ 1$

Observe que

1. a execução de find(c) não altera o rank de nenhum elemento de C.

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

---

union$_B(c_1, c_2)$

$R_1 \leftarrow$ find(c_1)
$R_2 \leftarrow$ find(c_2)
Se $R_1.rank < R_2.rank$
    PAI$[R_1] \leftarrow R_2$
Senão
    PAI$[R_2] \leftarrow R_1$
Se $R_1.rank = R_2.rank$
    R_1.rank $\leftarrow R_1$.rank $+ 1$

Observe que

1. a execução de find(c) não altera o rank de nenhum elemento de C.
2. a execução de union$_B(c_1, c_2)$ altera somente o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank. Neste caso o novo rank de $R(c_1)$ é acrescido de exatamente 1.

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Sim, usando a estratégia das classes baixas ("união por rank")

---

union$_B(c_1, c_2)$

$R_1 \leftarrow \text{find}(c_1)$
$R_2 \leftarrow \text{find}(c_2)$
Se $R_1.rank < R_2.rank$
    $\text{PAI}[R_1] \leftarrow R_2$
Senão
    $\text{PAI}[R_2] \leftarrow R_1$
Se $R_1.rank = R_2.rank$
    $R_1.\text{rank} \leftarrow R_1.\text{rank} + 1$

Observe que

1. a execução de find(c) não altera o rank de nenhum elemento de C.
2. a execução de union$_B(c_1, c_2)$ altera somente o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank. Neste caso o novo rank de $R(c_1)$ é acrescido de exatamente 1.

- Quanto custa?

Análise de Hopcroft-Ullman para Union-find

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

- Seja $(d = x_1, \ldots, x_k)$ o caminho de d à sua raiz.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

- Seja $(d = x_1, \ldots, x_k)$ o caminho de d à sua raiz.

Como d é descendente de c_1, então $c_1 = x_i$ para algum $i \in [1..k]$.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

- Seja $(d = x_1, \ldots, x_k)$ o caminho de d à sua raiz.

Como d é descendente de c_1, então $c_1 = x_i$ para algum $i \in [1..k]$.

Pelo mesmo argumento $c_2 = x_j$ para algum $j \in [1..k]$.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

- Seja $(d = x_1, \ldots, x_k)$ o caminho de d à sua raiz.

Como d é descendente de c_1, então $c_1 = x_i$ para algum $i \in [1..k]$.

Pelo mesmo argumento $c_2 = x_j$ para algum $j \in [1..k]$.

Então c_1 é descendente de c_2 ou vice-versa.

Análise de Hopcroft-Ullman para Union-find

- $d \in C$ é *descendente* de $a \in C$, se a está no caminho de d à sua raiz
- $a \in C$ é *ancestral* de $d \in C$ se d é descendente de a

Observe que se a é ancestral de d e $a \neq d$, então a e d tem ranks diferentes.

Lema

Conjuntos de descendentes de elementos de mesmo rank são disjuntos.

Prova:

Sejam c_1 e c_2 elementos distintos de mesmo rank.
Suponha que tem um descendente comum d.

- Seja $(d = x_1, \ldots, x_k)$ o caminho de d à sua raiz.

Como d é descendente de c_1, então $c_1 = x_i$ para algum $i \in [1..k]$.

Pelo mesmo argumento $c_2 = x_j$ para algum $j \in [1..k]$.

Então c_1 é descendente de c_2 ou vice-versa.

Isso contradiz a hipótese de que são distintos com mesmo rank.

Análise de Hopcroft-Ullman para Union-find

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Prova

Como somente a união de classes pode alterar os conjuntos de descendentes dos elementos de C, vamos provar por indução no número u de uniões de classes que

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Prova

Como somente a união de classes pode alterar os conjuntos de descendentes dos elementos de C, vamos provar por indução no número u de uniões de classes que

- Após u uniões, c contém pelo menos $2^{r(c)}$ descendentes, $\forall c \in C$ e $\forall u \in \mathbb{N}$

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Prova

Como somente a união de classes pode alterar os conjuntos de descendentes dos elementos de C, vamos provar por indução no número u de uniões de classes que

- Após u uniões, c contém pelo menos $2^{r(c)}$ descendentes, $\forall c \in C$ e $\forall u \in \mathbb{N}$
- A prova será por indução em u

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Prova

Como somente a união de classes pode alterar os conjuntos de descendentes dos elementos de C, vamos provar por indução no número u de uniões de classes que

- Após u uniões, c contém pelo menos $2^{r(c)}$ descendentes, $\forall c \in C$ e $\forall u \in \mathbb{N}$
- A prova será por indução em u

Base: $u = 0$.

Análise de Hopcroft-Ullman para Union-find

Lema

O conjunto dos descendentes de c tem pelo menos $2^{r(c)}$ elementos.

Prova

Como somente a união de classes pode alterar os conjuntos de descendentes dos elementos de C, vamos provar por indução no número u de uniões de classes que

- Após u uniões, c contém pelo menos $2^{r(c)}$ descendentes, $\forall c \in C$ e $\forall u \in \mathbb{N}$
- A prova será por indução em u

Base: $u = 0$.

Neste caso, $r(c) = 0$ para todo $c \in C$ e a classe contendo c tem tamanho $1 = 2^{r(c)}$.

Análise de Hopcroft-Ullman para Union-find

Prova (cont.)

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Após a execução de $\text{union}(c_1, c_2)$ o conjunto de descendentes de $R(c_1)$ é $D_1 \cup D_2$, onde D_1 e D_2 são os conjunto de descendentes de $R(c_1)$ e $R(c_2)$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Após a execução de $\text{union}(c_1, c_2)$ o conjunto de descendentes de $R(c_1)$ é $D_1 \cup D_2$, onde D_1 e D_2 são os conjunto de descendentes de $R(c_1)$ e $R(c_2)$.

Pela H.I.,

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Após a execução de $\text{union}(c_1, c_2)$ o conjunto de descendentes de $R(c_1)$ é $D_1 \cup D_2$, onde D_1 e D_2 são os conjunto de descendentes de $R(c_1)$ e $R(c_2)$.

Pela H.I., $|D_1| \geq 2^k$ e $|D_2| \geq 2^k$.

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Após a execução de $\text{union}(c_1, c_2)$ o conjunto de descendentes de $R(c_1)$ é $D_1 \cup D_2$, onde D_1 e D_2 são os conjunto de descendentes de $R(c_1)$ e $R(c_2)$.

Pela H.I., $|D_1| \geq 2^k$ e $|D_2| \geq 2^k$.

Como D_1 e D_2 ambos são disjuntos (Corolário), temos

Prova (cont.)

H.I.: Seja $a \in \mathbb{N}$ tal que após k uniões de classes, o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$ e todo $k \in [1..a]$.

Passo: Vamos provar que após $a+1$ uniões o conjunto dos descendentes de c contém pelo menos $2^{r(c)}$ elementos, para todo $c \in C$.

Digamos que a $a+1$-ésima união seja efetuada por $\text{union}_B(c_1, c_2)$.

- Já vimos que $\text{union}(c_1, c_2)$ só altera o rank de $R(c_1)$ e isso somente no caso em que $R(c_1)$ e $R(c_2)$ tem mesmo rank antes da execução.
- Basta então provar que, se $r(R(c_1)) = r(R(c_2)) = k$ antes da execução de $\text{union}(c_1, c_2)$, o conjunto de descendentes de $R(c_1)$ após a execução de tem pelo menos 2^{k+1} elementos, já que o rank de $R(c_1)$ após a execução será $k+1$.

Após a execução de $\text{union}(c_1, c_2)$ o conjunto de descendentes de $R(c_1)$ é $D_1 \cup D_2$, onde D_1 e D_2 são os conjunto de descendentes de $R(c_1)$ e $R(c_2)$.

Pela H.I., $|D_1| \geq 2^k$ e $|D_2| \geq 2^k$.

Como D_1 e D_2 ambos são disjuntos (Corolário), temos

$$|D_1 \cup D_2| = |D_1| + |D_2| \geq 2^k + 2^k = 2^{k+1}.$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right|$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)|$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)| \overset{\text{Lema}}{\geq} \sum_{a \in A} 2^r$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)| \overset{\text{Lema}}{\geq} \sum_{a \in A} 2^r = |A| 2^r,$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)| \overset{\text{Lema}}{\geq} \sum_{a \in A} 2^r = |A| 2^r,$$

Portanto,

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)| \overset{\text{Lema}}{\geq} \sum_{a \in A} 2^r = |A| 2^r,$$

Portanto,

$$|A| \leq \frac{|C|}{2^r}$$

Análise de Hopcroft-Ullman para Union-find

Corolário (Lema do Rank Baixo)

O número de elementos de rank r é no máximo $n/2^r$, para todo $r \in \mathbb{N}$.

Prova:

Seja $A \subseteq C$ o conjunto dos elementos de rank r.

Para cada $a \in A$ seja $D(a)$ o conjunto de seus descendentes.

Com isso,

$$|C| \geq \left| \bigcup_{a \in A} D(a) \right| \overset{\text{Lema}}{=} \sum_{a \in A} |D(a)| \overset{\text{Lema}}{\geq} \sum_{a \in A} 2^r = |A| 2^r,$$

Portanto,

$$|A| \leq \frac{|C|}{2^r} = \frac{n}{2^r}.$$

Análise de Hopcroft-Ullman para Union-find

Corolario

Nenhum elemento de C tem rank maior que $\lg n$.

Análise de Hopcroft-Ullman para Union-find

Corolario

Nenhum elemento de C tem rank maior que $\lg n$.

Prova

$$\frac{n}{2^r} < 1, \text{ para todo } r > \lg n.$$

Análise de Hopcroft-Ullman para Union-find

Corolario

Nenhum elemento de C tem rank maior que $\lg n$.

Prova

$$\frac{n}{2^r} < 1, \text{ para todo } r > \lg n.$$

Corolario

O tempo de execução de pior caso de find(c) e de union$_B(c_1, c_2)$ é $\Theta(\log n)$.

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Amortizadamente sim.

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Amortizadamente sim.

Compressão de Caminhos: transformar todos os elementos do *caminho* de c até sua raiz em filhos da raiz de c a cada execução de find(c).

Análise de Hopcroft-Ullman para Union-find

- É possível fazer melhor?
- Amortizadamente sim.

Compressão de Caminhos: transformar todos os elementos do *caminho* de c até sua raiz em filhos da raiz de c a cada execução de find(c).

---

find$_C(c)$

$l \leftarrow$ lista vazia
$R \leftarrow c$
Enquanto $\text{PAI}[R] \neq R$
    acrescente R em l
    $R \leftarrow \text{PAI}[R]$
Enquanto l *não é vazia*
    remova um elemento c de l
    $\text{PAI}[c] \leftarrow R$
Devolva R

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

1. pode não ser mais verdade que

$$c.\text{rank} = r(c) \text{ para todo } c \in C,$$

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

1. pode não ser mais verdade que

$$c.\text{rank} = r(c) \text{ para todo } c \in C,$$

- A operação $\text{find}_C(c)$ não altera $x.\text{rank}$, $\forall x \in C$

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

1. pode não ser mais verdade que

$$c.\text{rank} = r(c) \text{ para todo } c \in C,$$

- A operação $\text{find}_C(c)$ não altera $x.\text{rank}$, $\forall x \in C$
- Mas diminui $r(x)$ para x no caminho de c a $R(c)$, $x \notin c, R(c)$

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

1. pode não ser mais verdade que

$$c.\text{rank} = r(c) \text{ para todo } c \in C,$$

   - A operação $\text{find}_C(c)$ não altera $x.\text{rank}$, $\forall x \in C$
   - Mas diminui $r(x)$ para x no caminho de c a $R(c)$, $x \notin c, R(c)$

2. mas é verdade que

$$r(c) \leq c.\text{rank} \text{ para todo } c \in C,$$

Análise de Hopcroft-Ullman para Union-find

Após uma execução de $\text{find}_C(c)$,

1. pode não ser mais verdade que

$$c.\text{rank} = r(c) \text{ para todo } c \in C,$$

   - A operação $\text{find}_C(c)$ não altera $x.\text{rank}$, $\forall x \in C$
   - Mas diminui $r(x)$ para x no caminho de c a $R(c)$, $x \notin c, R(c)$

2. mas é verdade que

$$r(c) \leq c.\text{rank para todo } c \in C,$$

3. e consequentemente, que

$$c.\text{rank} < pai(c).\text{rank}, \text{ para todo } c \in C$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$,

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\lg 2^{65536} \quad = \quad 65536,$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \end{aligned}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \\ \lg 16 &= 4, \end{aligned}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \\ \lg 16 &= 4, \\ \lg 4 &= 2, \end{aligned}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \\ \lg 16 &= 4, \\ \lg 4 &= 2, \\ \lg 2 &= 1, \end{aligned}$$

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \\ \lg 16 &= 4, \\ \lg 4 &= 2, \\ \lg 2 &= 1, \end{aligned}$$

ou seja

Análise de Hopcroft-Ullman para Union-find

O *logaritmo iterado* de $x \in \mathbb{R}$ é

$$\lg^* x = \min \left\{ k \in \mathbb{N} \mid \log^k x \leq 1 \right\}.$$

i.e., o número de vezes que temos que aplicar lg em n para chegar a 1. Ou seja,

$$\lg^*(x) = \begin{cases} 0, & \text{se } n \leq 1, \\ 1 + \lg^* \lg x, & \text{se } n > 1. \end{cases}$$

O número estimado de particulas elementares no Universo é

$$10^{86} < 2^{286}$$

e para todo $n \leq 2^{65536}$ temos $\lg^* n \leq 5$, pois

$$\begin{aligned} \lg 2^{65536} &= 65536, \\ \lg 65536 &= 16, \\ \lg 16 &= 4, \\ \lg 4 &= 2, \\ \lg 2 &= 1, \end{aligned}$$

ou seja

$$\lg^5 2^{65536} = \lg \lg \lg \lg \lg 2^{65536} = 1.$$

Análise de Hopcroft-Ullman para Union-find

Para cada $-1 \leq k \leq \lg^* n$ definimos o *bloco* B_k por

Análise de Hopcroft-Ullman para Union-find

Para cada $-1 \leq k \leq \lg^* n$ definimos o *bloco* B_k por

$$
\begin{aligned}
B_{-1} &= [0..0] = \{0\}, \\
B_0 &= [0+1..2^0] = [1..1] = \{1\}, \\
B_1 &= [2^0+1..2^1] = [2..2] = \{2\}, \\
B_2 &= [2^1+1..2^2] = [3..4], \\
B_3 &= [2^2+1..2^4] = [5..16], \\
B_4 &= [2^4+1..2^{16}] = [17..65536], \\
B_5 &= [2^{16}+1..2^{65526}] = [65537..2^{65526}], \\
&\ldots,
\end{aligned}
$$

Análise de Hopcroft-Ullman para Union-find

Para cada $-1 \leq k \leq \lg^* n$ definimos o *bloco* B_k por

$$
\begin{aligned}
B_{-1} &= [0..0] = \{0\}, \\
B_0 &= [0+1..2^0] = [1..1] = \{1\}, \\
B_1 &= [2^0+1..2^1] = [2..2] = \{2\}, \\
B_2 &= [2^1+1..2^2] = [3..4], \\
B_3 &= [2^2+1..2^4] = [5..16], \\
B_4 &= [2^4+1..2^{16}] = [17..65536], \\
B_5 &= [2^{16}+1..2^{65526}] = [65537..2^{65526}], \\
&\ldots, \\
B_k &= [2^{[k-1]^2}+1..2^{[k]^2}] \\
&\ldots,
\end{aligned}
$$

Análise de Hopcroft-Ullman para Union-find

Para cada $-1 \leq k \leq \lg^* n$ definimos o *bloco* B_k por

$$
\begin{aligned}
B_{-1} &= [0..0] = \{0\}, \\
B_0 &= [0+1..2^0] = [1..1] = \{1\}, \\
B_1 &= [2^0+1..2^1] = [2..2] = \{2\}, \\
B_2 &= [2^1+1..2^2] = [3..4], \\
B_3 &= [2^2+1..2^4] = [5..16], \\
B_4 &= [2^4+1..2^{16}] = [17..65536], \\
B_5 &= [2^{16}+1..2^{65526}] = [65537..2^{65526}], \\
&\ldots, \\
B_k &= [2^{[k-1]^2}+1..2^{[k]^2}] \\
&\ldots, \\
B_{\lg^* n} &= [2^{[\lg^* n-1]^2}+1..2^{[\lg^* n]^2}].
\end{aligned}
$$

Análise de Hopcroft-Ullman para Union-find

Observe que

Análise de Hopcroft-Ullman para Union-find

Observe que

1. se um bloco termina em x o bloco seguinte termina em 2^x.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. se um bloco termina em x o bloco seguinte termina em 2^x.
2. se um bloco termina em 2^x, ele começa em $x + 1$.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. se um bloco termina em x o bloco seguinte termina em 2^x.
2. se um bloco termina em 2^x, ele começa em $x+1$.

Elementos bons

Diremos que $c \in C$ é *bom* se é raiz ou filho da raiz de sua classe ou se c.rank e $pai(c)$.rank estão em blocos diferentes.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. se um bloco termina em x o bloco seguinte termina em 2^x.
2. se um bloco termina em 2^x, ele começa em $x+1$.

Elementos bons

Diremos que $c \in C$ é *bom* se é raiz ou filho da raiz de sua classe ou se c.rank e $pai(c)$.rank estão em blocos diferentes.

Elementos ruins

Diremos que c é *ruim* se c não é bom.

Análise de Hopcroft-Ullman para Union-find

Observe que

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.
4. Se c não é raiz, c.rank nunca mais muda.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.
4. Se c não é raiz, c.rank nunca mais muda.
5. Se c é ruim, então c não é raiz e c.rank nunca mais muda.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.
4. Se c não é raiz, c.rank nunca mais muda.
5. Se c é ruim, então c não é raiz e c.rank nunca mais muda.
6. Os únicos jeitos de um elemento ruim c ficar bom são

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.
4. Se c não é raiz, c.rank nunca mais muda.
5. Se c é ruim, então c não é raiz e c.rank nunca mais muda.
6. Os únicos jeitos de um elemento ruim c ficar bom são
    1. virar filho da raiz por ter sido visitado num find; neste caso ele pode voltar a ser ruim se seu pai deixar de ser raiz num union subsequente.

Análise de Hopcroft-Ullman para Union-find

Observe que

1. c.rank só pode aumentar (de 1 unidade) a cada union
2. c.rank só pode mudar se c é raiz.
3. Se c não é raiz, nunca mais será.
4. Se c não é raiz, c.rank nunca mais muda.
5. Se c é ruim, então c não é raiz e c.rank nunca mais muda.
6. Os únicos jeitos de um elemento ruim c ficar bom são
   1. virar filho da raiz por ter sido visitado num find; neste caso ele pode voltar a ser ruim se seu pai deixar de ser raiz num union subsequente.
   2. $pai(c)$.rank aumentar e mudar para o bloco seguinte ao de c.rank; nesse caso ele nunca mais deixa de ser bom porque c.rank não vai mais mudar e $pai(c)$.rank se mudar só vai aumentar.

Análise de Hopcroft-Ullman para Union-find

Lema

Em uma execução de $\text{find}_C(c)$ a quantidade de elementos bons percorridos é no máximo $\lg^* n + 4$.

Análise de Hopcroft-Ullman para Union-find

Lema

Em uma execução de $\text{find}_C(c)$ a quantidade de elementos bons percorridos é no máximo $\lg^* n + 4$.

Prova:

Considere uma execução de $\text{find}_C(c)$.

Análise de Hopcroft-Ullman para Union-find

Lema

Em uma execução de $\text{find}_C(c)$ a quantidade de elementos bons percorridos é no máximo $\lg^* n + 4$.

Prova:

Considere uma execução de $\text{find}_C(c)$.

Todos os elementos bons que não são a raiz nem o filho da raiz no caminho de c até sua raiz tem que estar em blocos distintos, caso contrário não seriam elementos bons.

Análise de Hopcroft-Ullman para Union-find

Lema

Em uma execução de $\text{find}_C(c)$ a quantidade de elementos bons percorridos é no máximo $\lg^* n + 4$.

Prova:

Considere uma execução de $\text{find}_C(c)$.

Todos os elementos bons que não são a raiz nem o filho da raiz no caminho de c até sua raiz tem que estar em blocos distintos, caso contrário não seriam elementos bons.

Como só existem $\lg^* + 2$ blocos distintos, o máximo de elementos bons no caminho de c até $R(c)$ é $\lg^* n + 4$.

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

$$R(d).\text{rank}, > p.\text{rank}.$$

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

$$R(d).\text{rank}, > p.\text{rank}.$$

pois, antes da execução a distância de $R(d)$ até seu descendente mais distante é maior que a distância de p até seu descendente mais distante.

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

$$R(d).\text{rank}, > p.\text{rank}.$$

pois, antes da execução a distância de $R(d)$ até seu descendente mais distante é maior que a distância de p até seu descendente mais distante.

Ao final da execução, $\text{PAI}(c) = R(d)$

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

$$R(d).\text{rank}, > p.\text{rank}.$$

pois, antes da execução a distância de $R(d)$ até seu descendente mais distante é maior que a distância de p até seu descendente mais distante.

Ao final da execução, $\text{PAI}(c) = R(d)$ e, portanto,

Análise de Hopcroft-Ullman para Union-find

Lema

Toda vez que um elemento ruim c é visitado, aumenta a o valor de $pai(c)$.rank.

Prova:

Seja c um elemento ruim visitado durante a execução de find(d)

- seja p seu pai antes da execução.

Como c é ruim, $c \neq R(d)$ e $p \neq R(d)$.

Necessariamente,

$$R(d).\text{rank}, > p.\text{rank}.$$

pois, antes da execução a distância de $R(d)$ até seu descendente mais distante é maior que a distância de p até seu descendente mais distante.

Ao final da execução, $\text{PAI}(c) = R(d)$ e, portanto,

$$\text{PAI}(c).\text{rank} = R(d).\text{rank} > p.\text{rank}.$$

Análise de Hopcroft-Ullman para Union-find

Corolario

Um elemento ruim c pode ser visitado sendo ruim no máximo $|B|$ vezes onde B é o bloco ao qual c.rank pertence.

Análise de Hopcroft-Ullman para Union-find

Corolario

Um elemento ruim c pode ser visitado sendo ruim no máximo $|B|$ vezes onde B é o bloco ao qual c.rank pertence.

Prova:

Se c é ruim, c.rank e $\text{PAI}(c)$.rank estão no mesmo bloco. A cada visita, o valor de $\text{PAI}(c)$.rank aumenta. Após $|B|$ visitas, o valor de $\text{PAI}(c)$.rank não estará mais em $|B|$.

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

- Seja c um elemento ruim tal que $c.\text{rank} \in B = [k + 1..2^k]$.

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

- Seja c um elemento ruim tal que $c.\text{rank} \in B = [k+1..2^k]$.

Pelo Corolário anterior c pode ser visitado no máximo $|B|$ vezes.

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

- Seja c um elemento ruim tal que $c.\text{rank} \in B = [k+1..2^k]$.

Pelo Corolário anterior c pode ser visitado no máximo $|B|$ vezes.

Pelo Corolário (rank baixo) o número de elementos com rank r é no máximo $n/2^r$.

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

- Seja c um elemento ruim tal que $c.\text{rank} \in B = [k+1..2^k]$.

Pelo Corolário anterior c pode ser visitado no máximo $|B|$ vezes.

Pelo Corolário (rank baixo) o número de elementos com rank r é no máximo $n/2^r$.

Então o número visitas a elementos ruins com rank em B é no máximo

Análise de Hopcroft-Ullman para Union-find

Corolario

O número de vezes que elementos ruins são percorridos numa sequência de execuções de find_C é menor que $n(\lg^* n + 2)$.

Prova:

Considere uma sequência de execuções de find_C

- Seja c um elemento ruim tal que $c.\text{rank} \in B = [k+1..2^k]$.

Pelo Corolário anterior c pode ser visitado no máximo $|B|$ vezes.

Pelo Corolário (rank baixo) o número de elementos com rank r é no máximo $n/2^r$.

Então o número visitas a elementos ruins com rank em B é no máximo

$$\sum_{r \in B} \frac{n}{2^r} = n \sum_{r=k+1}^{2^k} \frac{1}{2^r}$$

$$= \frac{n}{2^{k+1}} \sum_{r=0}^{2^k-(k+1)} \frac{1}{2^r} \leq \frac{n}{2^{k+1}} \sum_{r=0}^{\infty} \frac{1}{2^r} = \frac{n}{2^{k+1}} 2 = \frac{n}{2^k}$$

$$< \frac{n}{|B|}$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Daí, o número total de visitas a elementos ruins na sequência execuções de find$_C$ é $\leq$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Daí, o número total de visitas a elementos ruins na sequência execuções de find$_C$ é $\leq$

$$\sum_{k=-1}^{\lg^* n} \frac{n}{|B_k|} =$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Daí, o número total de visitas a elementos ruins na sequência execuções de find_C é $\leq$

$$\sum_{k=-1}^{\lg^* n} \frac{n}{|B_k|} = n \sum_{k=-1}^{\lg^* n} \frac{1}{|B_k|} \leq$$

Prova (cont.):

Daí, o número total de visitas a elementos ruins na sequência execuções de find_C é $\leq$

$$\sum_{k=-1}^{\lg^* n} \frac{n}{|B_k|} = n \sum_{k=-1}^{\lg^* n} \frac{1}{|B_k|} \leq n \sum_{k=-1}^{\lg^* n} 1 =$$

Prova (cont.):

Daí, o número total de visitas a elementos ruins na sequência execuções de find_C é $\leq$

$$\sum_{k=-1}^{\lg^* n} \frac{n}{|B_k|} = n \sum_{k=-1}^{\lg^* n} \frac{1}{|B_k|} \leq n \sum_{k=-1}^{\lg^* n} 1 = n(\lg^* n + 2).$$

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Considere uma sequência de m execuções de $\text{find}_C(c)$ e seja T o tempo de execução dessa sequência.

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Considere uma sequência de m execuções de $\text{find}_C(c)$ e seja T o tempo de execução dessa sequência.

- Seja ainda B o número de vezes que um elemento bom foi percorrido

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Considere uma sequência de m execuções de $\text{find}_C(c)$ e seja T o tempo de execução dessa sequência.

- Seja ainda B o número de vezes que um elemento bom foi percorrido
- Seja R o número de vezes que um elemento ruim foi percorrido.

Análise de Hopcroft-Ullman para Union-find

Teorema

O tempo de pior caso de uma sequência de $\Omega(n)$ execuções de union_B e/ou find_C é $\mathcal{O}(m \lg^* n)$.

Prova:

A execução de $\text{union}_B(r_1, r_2)$ tem o tempo de duas execuções de $\text{find}_C(c)$ mais $\Theta(1)$.

Basta, então, provar que o tempo de qualquer sequência de m execuções de $\text{find}_C(c)$ é $\mathcal{O}(m \lg^* n)$.

- Note que $\text{union}_C(c)$ faz modificações na estrutura de dados
- Entretanto, relembre as 6 observações sobre isso (alguns slides atrás)

Considere uma sequência de m execuções de $\text{find}_C(c)$ e seja T o tempo de execução dessa sequência.

- Seja ainda B o número de vezes que um elemento bom foi percorrido
- Seja R o número de vezes que um elemento ruim foi percorrido.

É imediato que

$$T = \Theta(B + R).$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Lema (limintante de visita a vértices bons) temos

$$B \leq m(\lg^* n + 4),$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Lema (limintante de visita a vértices bons) temos

$$B \leq m(\lg^* n + 4),$$

e, portanto,

$$B = \mathcal{O}(m \lg^* n)$$

Prova (cont.):

Do Lema (limintante de visita a vértices bons) temos

$$B \leq m(\lg^* n + 4),$$

e, portanto,

$$B = \mathcal{O}(m \lg^* n)$$

e daí, se $m = \Omega(n)$,

$$B = \mathcal{O}(n \lg^* n).$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Corolário anterior temos

$$R \leq n(\lg^* n + 2)$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Corolário anterior temos

$$R \leq n(\lg^* n + 2)$$

e, portanto,

$$R = \mathcal{O}(n \lg^* n)$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Corolário anterior temos

$$R \leq n(\lg^* n + 2)$$

e, portanto,

$$R = \mathcal{O}(n \lg^* n)$$

Como

$$T = \Theta(B + R).$$

Prova (cont.):

Do Corolário anterior temos

$$R \leq n(\lg^* n + 2)$$

e, portanto,

$$R = \mathcal{O}(n \lg^* n)$$

Como

$$T = \Theta(B + R).$$

concluimos que

$$T = \mathcal{O}(m \lg^* n).$$

Análise de Hopcroft-Ullman para Union-find

Prova (cont.):

Do Corolário anterior temos

$$R \leq n(\lg^* n + 2)$$

e, portanto,

$$R = \mathcal{O}(n \lg^* n)$$

Como

$$T = \Theta(B + R).$$

concluimos que

$$T = \mathcal{O}(m \lg^* n).$$

Corolario

O tempo de pior caso amortizado de cada execução de union_B e/ou find_C em uma sequência de $\Omega(n)$ execuções é $\mathcal{O}(\lg^* n)$.