

Aula 16

Estrutura de dados para Conjuntos Disjuntos

Prof. Marco Aurélio Stefanés

marco em dct.ufms.br

www.dct.ufms.br/~marco

Aula 16 – p. 1

Estrutura de Dados para Conjuntos Disjuntos

- Seja $S = \{s_1, s_2, \dots, s_k\}$ conjunto de conj. disjuntos
- Cada s_i tem um representante $x \in s_i$ que identifica s_i
- Qualquer método de escolha de x serve, desde que o representante não mude se s_i não muda

Operações

- **Make_Set(x):** Cria um novo conjunto cujo único elemento é x
- **Union(x, y):** Cria um conjunto que é a união dos conjunto contendo x e y , digamos s_x, s_y . Supomos que $s_x \cap s_y = \emptyset$. s_x e s_y são destruídos. Um elemento é escolhido como representante da união
- **Find_Set(x):** devolve um apontador para o representante do único conjunto contendo x .

Aula 16 – p. 2

Estrutura de Dados para Conjuntos

Complexidade será medida em função:

- n número de operações Make_Set
- m número total de operações Make_Set, Union e Find_Set
- Obs.1: $m \geq n$
- Obs.2: Número de Op. Union $\leq n - 1$

Exemplo: Determinar as componentes conexas de um grafo $G = (V, E)$

Def.: Componente conexa: Subgrafo conexo maximal de G

Aula 16 – p. 3

Componentes conexas

Componentes_Conexas(G)

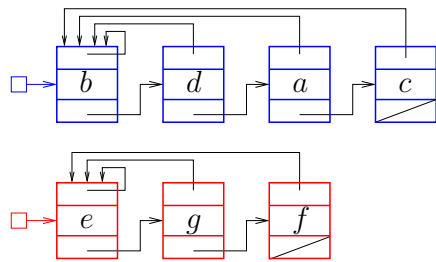
- 1: **for** Cada $v \in V$ **do**
- 2: Make_Set(v)
- 3: **for** cada aresta $(u, v) \in E$ **do**
- 4: **if** Find_Set(u) $\neq$ Find_Set(v) **then**
- 5: Union(u, v)

Mesma_Componente(u, v)

- 1: **if** Find_Set(u) = Find_Set(v) **then**
- 2: devolva **T**
- 3: **else**
- 4: devolva **F**

Aula 16 – p. 4

Implementação por listas ligadas



- Make_Set, Find_Set:
 b e e são os representantes

Aula 16 – p. 5

Implementação por listas ligadas

Seqüência de m operações requer $\Theta(m^2)$

Make_Set(x_1)

⋮

Make_Set(x_n)

Union(x_1, x_2)

Union(x_2, x_3)

⋮

Union(x_{n-1}, x_n)

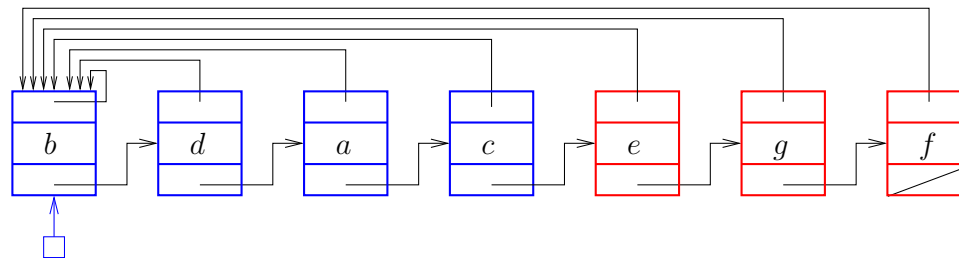
Obs.: $m = 2n - 1$

Melhoria: Na união concatena a lista menor no final.
É necessário manter contadores nos representantes

Aula 16 – p. 7

Implementação por listas ligadas

- Union(x, y): concatena a lista de x no final da lista de y .
É necessário atualizar os apontadores para os representantes da lista contendo x



Aula 16 – p. 6

Implementação por listas ligadas

Teorema Usando a melhoria anterior, uma seqüência de m operações, sendo n delas Make_Set, é feita em tempo $O(m + n \lg n)$

Prova: Considere um elemento x . Quantas vezes o apontador de x para seu representante pode ser atualizado?

Cada vez que houver uma atualização, o conj. contendo x pelo menos dobra de tamanho. Portanto o apontador pode ser atualizado no máximo $\lceil \lg n \rceil$ vezes. Assim, tempo total para op. Union é $O(n \lg n)$.

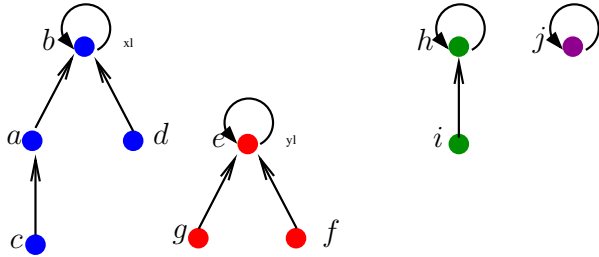
Como Make_Set e Find_Set são feitos em tempo $O(1)$ e existem $O(m)$ delas, o tempo do algoritmo é $O(m + n \lg n)$

Aula 16 – p. 8

Implementação por Florestas disjunta

A raiz da árvore é o representante

- Make_Set: árvore com um nó
- Find_Set: percorre a árvore até a raiz



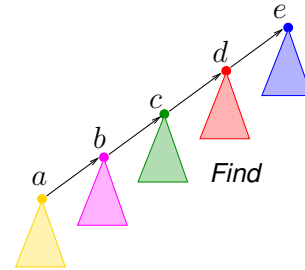
Aula 16 – p. 9

Melhorias

Melhoria 1 : União por rank

Melhoria 2 : Compressão de caminho

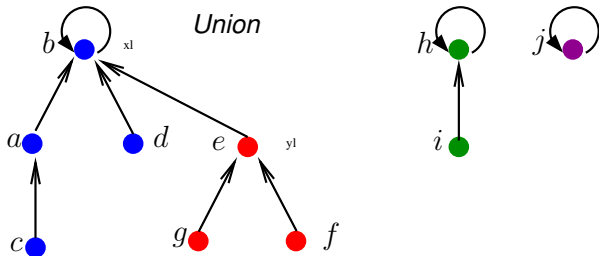
Rank: inteiro, limitante superior na altura de x



Aula 16 – p. 11

Implementação por Florestas disjunta

- Union: a raiz de uma árvore passa a apontar para a raiz da outra



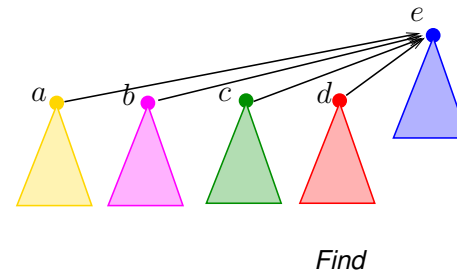
Aula 16 – p. 10

Melhorias

Melhoria 1 : União por rank

Melhoria 2 : Compressão de caminho

Rank: inteiro, limitante superior na altura de x



Aula 16 – p. 12

Algoritmos

Make_Set(x)

- 1: $p[x] = x$
- 2: $rank[x] = 0$

Find_Set(x)

- 1: **if** $p[x] \neq x$ **then**
- 2: $p[x] = \text{Find_Set}(p[x])$
- 3: **devolva** $p[x]$

Union(x, y)

- 1: Link(Find_Set(x), Find_Set(y))

Algoritmos

Link(x, y)

- 1: **if** $rank[x] > rank[y]$ **then**
- 2: $p[y] = x$
- 3: **else**
- 4: $p[x] = y$
- 5: **if** $rank[x] = rank[y]$ **then**
- 6: $rank[y] = rank[y] + 1$

Vamos mostrar que o tempo de processamento de m operações com as melhorias é $O(m \lg^* n)$

$$\lg^{(0)} n = n$$

$$\lg^{(k)} n = \lg(\lg^{(k-1)} n)$$

$$\lg^* n = \min\{k \geq 0 \mid \lg^{(k)} n \leq 1\}$$

Propriedades dos Ranks

Lema 22.2 Qualquer x , $rank[x] \leq rank[p[x]]$, com desigualdade estrita se $x \neq p[x]$. O valor de $rank[x]$ inicia com 0 e aumenta até que $x \neq p[x]$. A partir daí $rank[x]$ não muda.

Prova: Indução no no. de operações (**Exerc.**)

Seja $size[x]$ o no. de nós da árvore com raiz x .

Propriedades dos Ranks

Lema 22.3 Qualquer raiz x , $size[x] \geq 2^{rank[x]}$

Prova: Indução no no. de operações Link.

Base 0 Links: $rank[x] = 0$ e a árvore só contém x

P.I. Pela HI $size[x] \geq 2^{rank[x]}$ e $size[y] \geq 2^{rank[y]}$.

Caso $rank[x] \neq rank[y]$. SPG, supor $rank[x] < rank[y]$.

$$\begin{aligned} size'[y] &= size[y] + size[x] \\ &\geq 2^{rank[y]} + 2^{rank[x]} \\ &= 2^{rank'[y]} \end{aligned}$$

Caso $rank[x] = rank[y]$:

$$\begin{aligned} size'[y] &\geq 2^{rank[y]} + 2^{rank[x]} \\ &= 2^{rank[y]+1} \\ &= 2^{rank'[y]} \end{aligned}$$

Propriedades dos Ranks

Lema 22.4 Qualquer $r \geq 0$, inteiro há no máximo $\frac{n}{2^r}$ nós de rank r .

Prova: Fixe um valor de r . Cada x com $\text{rank}[x] = r$, associamos pelo menos 2^r nós: os nós que estavam na árvore de raiz x quando $\text{rank}[x]$ se tornou r . Note que cada elemento y fica associado a no máximo um nó x , qualquer novo ancestral de y terá rank pelo menos $r + 1$. Portanto, temos no máximo $\frac{n}{2^r}$ nós de rank r .

Corolário 22.5 Cada nó tem rank no máximo $\lfloor \lg n \rfloor$.

Prova: Caso algum nó tenha rank $> \lfloor \lg n \rfloor$, pelo Lema terá $\frac{n}{2^{\lfloor \lg n \rfloor + 1}} < 1$ nós. Absurdo.

Aula 16 – p. 17

Propriedades dos Ranks

Lema 22.6 Converta uma seqüência S' de m' operações Make_Set, Union e Find_Set em uma seqüência S de m operações Make_Set, Link e Find_Set, transformando cada Union em duas Find_Set e uma Link. Se S é executada em tempo $O(m \lg^* n)$, S' é executada em tempo $O(m' \lg^* n)$.

Prova: Como $m \leq 3m'$, tempo $O(m \lg^* n)$ para S implica em tempo $O(m' \lg^* n)$ para S' .

Teorema 22.7 Uma seqüência de m operações Make_Set, Link e Find_Set, n das quais são Make_Set, são executadas em tempo $O(m \lg^* n)$ numa floresta disjunta com união por rank e compressão de caminhos

Prova: Análise amortizada. Make_Set e Link tem custo amortizado $O(1)$.

Aula 16 – p. 18

Análise Amortizada do Find_Set

Particione os nós em blocos de acordo com o rank: elemento de rank r está no bloco $\lg^* r$.

$$B(j) = \begin{cases} -1 & \text{se } j = -1 \\ 1 & \text{se } j = 0 \\ 2 & \text{se } j = 1 \\ \left. \begin{matrix} 2^2 \cdots 2^2 \\ 2 \end{matrix} \right\} j & \text{se } j \geq 2 \end{cases}$$

Obs. último bloco é $\lg^* n - 1$, pois rank máximo é $\lfloor \lg n \rfloor$. Então, o j^o bloco, $j = 0, 1, \dots, \lg^* n - 1$, consiste do conj. de nós de rank $\{B(j-1) + 1, \dots, B(j)\}$. Para uma op. Find_Set vamos usar dois tipos de custo: tipo bloco e tipo caminho.

Aula 16 – p. 19

Análise Amortizada do Find_Set

Suponha que o caminho do Find_Set consiste de $x_0, x_1, \dots, x_l$, onde o nó $x_i = p[x_{i-1}]$ e x_l é a raiz.

Para $j = 0, 1, \dots, \lg^* n - 1$, atribuímos um custo tipo bloco ao último nó com rank no bloco j do caminho e também ao nó x_{l-1} . Para cada nó sem custo tipo bloco atribuímos um custo tipo caminho.

Total de custo tipo blocos: No máximo $\lg^* n + 1$ custo tipo bloco pro Find_Set. Como o total destas operações é limitado por m , o total de custo blocos é $O(m \lg^* n)$.

Total de custo tipo caminho: Uma vez que um nó x ($x \neq \text{raiz}$ e $x \neq \text{filho da raiz}$) recebe um custo tipo bloco ele nunca mais recebe um custo tipo caminho: $\text{rank}[x]$ permanece constante e a diferença $\text{rank}[p[x]] - \text{rank}[x]$ só pode aumentar.

Portanto, x sempre terá custos tipo blocos daí até o final das operações.

Aula 16 – p. 20

Análise Amortizada do Find_Set

Se um nó x recebe um custo tipo caminho então $p[x] \neq x$ antes da compressão. Portanto $p[x]$ mudará após a compressão. Ademais, o novo pai de x terá um rank maior do que o do seu velho pai. Quantas vezes isto pode ocorrer?

Este número é máximo quando x tem o menor rank do bloco, isto é, $B(j-1) + 1$. Seus pais teriam rank $\{B(j-1) + 2, \dots, B(j)\}$. Portanto cada nó pode ter custo caminho $B(j) - B(j-1) - 1$ vezes.

Seja $N(j)$ o no. de nós com rank no bloco j . Então

$$N(j) \leq \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r}$$

$$j = 0, N(0) \leq n$$

$$j \geq 1, N(j) \leq \frac{n}{2^{B(j-1)+1}} \sum_{i=0}^{B(j)-B(j-1)-1} \frac{1}{2^i} \leq \frac{n}{B(j)}$$

Portanto $N(j) \leq \frac{n}{B(j)}$ para todo $j \geq 0$

Aula 16 – p. 21

Análise Amortizada do Find_Set

Seja $P(n)$ o total de custo tipo caminho

$$\begin{aligned} P(n) &\leq \sum_{j=0}^{\lg^* n - 1} \frac{n}{B(j)} (B(j) - B(j-1) - 1) \\ &\leq \sum_{j=0}^{\lg^* n - 1} \frac{n}{B(j)} B(j) \\ &= n \lg^* n \end{aligned}$$

Portanto, o total de custo durante as operações Find_Set é $O(m \lg^* n)$ pois $n < m$. Segue o teorema.

Aula 16 – p. 22