Análise da complexidade do algoritmo de Kruskal (1)

Linhas

3 Construção da partição

MAKE-SETS

4 Construção da fila com prioridade (*heap*)

$$\Theta(E)$$

6–11 Ciclo executado entre $|V| - 1$ e $|E|$ vezes

7 Remoção do menor elemento da fila (*heap*)

$$O(\log E) = O(\log V)$$

$$(\,|E| < |V|^2 \rightarrow \log|E| < log|V|^2 = 2\log|V| = O(\log V)\,)$$

8 $2 \times$ FIND-SET

10 Executada $|V| - 1$ vezes

UNION

Restantes operações com complexidade temporal constante

Análise da complexidade do algoritmo de Kruskal (2)

Juntando tudo, obtém-se

$$\text{MAKE-SETS} + \Theta(E) + |E| \times O(\log V) +$$
$$|E| \times 2 \times \text{FIND-SET} + (|V| - 1) \times \text{UNION}$$

ou

$$\Theta(E) + |E| \times O(\log V) + f(V, E)$$

com

$$f(V, E) = \text{MAKE-SETS} + 2 \times |E| \times \text{FIND-SET} + (|V| - 1) \times \text{UNION}$$

Partição

Conjuntos disjuntos (*Disjoint sets*)

Abstracção da implementação de conjuntos disjuntos com os elementos do conjunto $\{1, 2, \ldots, n\}$

Operações suportadas

MAKE-SETS(n)
: Cria conjuntos singulares com os elementos $\{1, 2, \ldots, n\}$

FIND-SET(i)
: Devolve o representante do conjunto que contém o elemento i

UNION(i, j)
: Reúne os conjuntos a que pertencem os elementos i e j

Também é conhecido como *Union-Find*

Partição

Implementação em vector

MAKE-SETS(n)

```
let P[1..n] be a new array
for i <- 1 to n do
  P[i] <- -1      // i is the representative for set {i}
return P
```

FIND-SET(P, i)

```
while P[i] > 0 do
     i <- P[i]
return i
```

UNION(P, i, j)

```
P[FIND-SET(P,j)] <- FIND-SET(P, i)
```

Partição

Implementação em vector

Reunião por tamanho

Se $P[i] = -k$, o conjunto de que i é o representante contém k elementos

UNION-BY-SIZE(P, i, j)

```
ri <- FIND-SET(P, i)         // get i's set representative
rj <- FIND-SET(P, j)         // get j's set representative
if (P[rj] < P[ri])           // j's set is larger than i's
    P[rj] <- P[rj] + P[ri]
    P[ri] <- rj
else              // i's is larger or both have the same size
    P[ri] <- P[ri] + P[rj]
    P[rj] <- ri
```

Partição

Implementação em vector

Reunião por altura

Se $P[i] = -h$, a árvore do conjunto de que i é o representante tem altura h ou inferior

UNION-BY-RANK(P, i, j)

```
ri <- FIND-SET(P, i)
rj <- FIND-SET(P, j)
if (P[rj] < P[ri])          // j's set tree taller than i's
    P[ri] <- rj
else
    if (P[rj] == P[ri])     // both heights are the same
        P[ri] <- P[ri] - 1
    P[rj] <- ri
```

Partição

Implementação em vector

Compressão de caminho

FIND-SET-WITH-PATH-COMPRESSION(P, i)

```
if P[i] < 0 then
    return i
P[i] <- FIND-SET-WITH-PATH-COMPRESSION(P, P[i])
return P[i]
```

Análise da complexidade do algoritmo de Kruskal (3)

$$\Theta(E) + |E| \times O(\log V) + f(V, E)$$

$$f(V, E) = \text{MAKE-SETS} + 2 \times |E| \times \text{FIND-SET} + (|V| - 1) \times \text{UNION}$$

Implementação da Partição	Básica	União por tam./altura	+ Compressão de caminho
MAKE-SETS	$O(V)$	$O(V)$	$O((V + E)\,\alpha(V))$ [Tarjan 1975]
$2 \times \|E\| \times$ FIND-SET	$O(EV)$	$O(E \log V)$	
$(\|V\| - 1) \times$ UNION	$O(V^2)$	$O(V \log V)$	
$f(V, E)$	$O(EV)$	$O(E \log V)$	$O(E\,\alpha(V))$
Algoritmo de Kruskal	$O(EV)$	$O(E \log V)$	$O(E \log V)$

$\alpha(n) \leq 4$ para $n < 10^{80}$

Análise da complexidade do algoritmo de Kruskal (4)

$\alpha(n) = \min\{k \mid A_k(1) \geq n\}$

onde

$$A_k(j) = \begin{cases} j+1 & \text{se } k = 0 \\ A_{k-1}^{(j+1)}(j) & \text{se } k \geq 1 \end{cases}$$

$$\begin{aligned} A_0(1) &= 2 \\ A_1(1) &= A_0(A_0(1)) = 3 \\ A_2(1) &= A_1(A_1(1)) = 7 \\ A_3(1) &= 2047 \\ A_4(1) &\gg 2^{2048} \gg 10^{80} \end{aligned}$$

Iteração de uma função

$A_{k-1}^{(0)}(j) = j$ e $A_{k-1}^{(i)}(j) = A_{k-1}(A_{k-1}^{(i-1)}(j))$, para $i \geq 1$