

Chapter 21 Solusion

github.com/frc123/CLRS

1/3/2022

21.1

21.1-1

Edge processed	Collection of disjoint sets										
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}	{k}
(d, i)	{a}	{b}	{c}	{d, i}	{e}	{f}	{g}	{h}		{j}	{k}
(f, k)	{a}	{b}	{c}	{d, i}	{e}	{f, k}	{g}	{h}		{j}	
(g, i)	{a}	{b}	{c}	{d, g, i}	{e}	{f, k}		{h}		{j}	
(b, g)	{a}	{b, d, g, i}	{c}		{e}	{f, k}		{h}		{j}	
(a, h)	{a, h}	{b, d, g, i}	{c}		{e}	{f, k}				{j}	
(i, j)	{a, h}	{b, d, g, i, j}	{c}		{e}	{f, k}					
(d, k)	{a, h}	{b, d, f, g, i, j, k}	{c}		{e}						
(b, j)	{a, h}	{b, d, f, g, i, j, k}	{c}		{e}						
(d, f)	{a, h}	{b, d, f, g, i, j, k}	{c}		{e}						
(g, j)	{a, h}	{b, d, f, g, i, j, k}	{c}		{e}						
(a, e)	{a, e, h}	{b, d, f, g, i, j, k}	{c}								

21.1-2

Proof. By contents in B.4, we know that the connected components of a graph are the equivalence classes of vertices under the “is reachable from” relation. The collection of the disjoint sets is exactly the quotient set of $G.V$ by the “is reachable from” relation. It is not hard to find out that CONNECTED-COMPONENTS construct such the quotient set since the procedure unions vertices based on all edges, and edges connect two reachable vertices with the smallest length of the path (recall that a equivalence relation must be transitive). Two vertices are in the same connected component if and only if they are reachable from each other. \square

21.1-3

FIND-SET: $2 \cdot |E|$

UNION: $|V| - k$

21.2

21.2-1

```
1  struct Set
2  {
3      Node *head;
4      Node *tail;
5      int size;
6  };
7
8  struct Node
9  {
10     int key;
11     Set *set;
12     Node *next;
13 };
14
15 void MakeSet(Node *x)
16 {
17     x->next = nullptr;
18     x->set = new Set; // need to be freed
19     x->set->head = x;
20     x->set->tail = x;
21     x->set->size = 1;
22 }
23
24 Node* FindSet(Node *x)
25 {
26     return x->set->head;
27 }
28
29 void Union(Node *x, Node *y)
30 {
31     Node *node;
32     if (x->set->size < y->set->size)
33     {
34         Union(y, x);
35     }
36     else
```

```
37     {
38         node = y->set->head;
39         x->set->size += y->set->size;
40         x->set->tail->next = node;
41         x->set->tail = y->set->tail;
42         delete y->set;
43         while (node)
44         {
45             node->set = x->set;
46             node = node->next;
47         }
48     }
49 }
```

21.2-2

collection before line 3:

$$\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}, \{x_6\}, \{x_7\}, \{x_8\}, \{x_9\}, \{x_{10}\}, \{x_{11}\}, \{x_{12}\}, \{x_{13}\}, \{x_{14}\}, \{x_{15}\}, \{x_{16}\}\}$$

collection before line 5:

$$\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_5, x_6\}, \{x_7, x_8\}, \{x_9, x_{10}\}, \{x_{11}, x_{12}\}, \{x_{13}, x_{14}\}, \{x_{15}, x_{16}\}\}$$

collection before line 7:

$$\{\{x_1, x_2, x_3, x_4\}, \{x_5, x_6, x_7, x_8\}, \{x_9, x_{10}, x_{11}, x_{12}\}, \{x_{13}, x_{14}, x_{15}, x_{16}\}\}$$

collection before line 8:

$$\{\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}, \{x_9, x_{10}, x_{11}, x_{12}\}, \{x_{13}, x_{14}, x_{15}, x_{16}\}\}$$

collection before line 9:

$$\{\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}, \{x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}\}\}$$

collection before line 10:

$$\{\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}\}\}$$

Hence $\text{FIND-SET}(x_2)$ and $\text{FIND-SET}(x_{11})$ return a pointer points to x_1 .

21.2-3

Lemma 1. Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of h UNION operations on a disjoint set that has never been operated UNION takes $O(h \lg h)$ time.

Proof. We claim that, after h UNION operations, the largest set has at most $h + 1$ members. Notice that the number of sets decreases by one each time UNION is called. Suppose that we have n sets in which each set contains one member in the beginning. After the h UNION operations, we have $(n - h)$ sets. Note that each set must contain one member. In order to maximize the number of members in the the largest set, we let $(n - h - 1)$ sets contains one member, and let the remaining set contains all remaining members. Then the remaining set contains

$$n - (n - h - 1) = h + 1$$

members.

We claim that each object's pointer back to its set object is updated at most $\lceil \lg h \rceil$ times over all the UNION operations. Let x be an arbitrary object. By the similar approach in the proof of Theorem 21.1, we know that for any $k \leq h + 1$, after x 's pointers has been updated $\lceil \lg k \rceil$ times, the resulting set must have at least k members. Since the largest set has at most $h + 1$ members, each object's pointer is updated at most $\lg(h + 1)$ times over all the UNION operations.

We claim that there are h elements have been updated their pointers back to their set objects at least once. Consider a set contains k members. Then within this set, there are $(k - 1)$ members have been updated their pointers back to their set objects at least once since there must exists exactly $(k - 1)$ members updated their pointers from the initial pointer to the current one. Let \mathcal{S} be our collection of sets. Then after the h UNION operations, the number of elements have been updated their pointers is

$$\sum_{A \in \mathcal{S}} (|A| - 1) = \sum_{A \in \mathcal{S}} |A| - |\mathcal{S}| = n - (n - h) = h$$

Since each object's pointer is updated at most $\lceil \lg h \rceil$ times and there are h elements have been updated their pointers, we conclude h UNION operations on a disjoint set that has never been operated UNION takes $O(h \lg h)$ time. \square

Claim 2. The amortized time of MAKE-SET and FIND-SET is $O(1)$, and the amortized time of UNION is $O(\lg n)$.

Proof. Suppose that we performed h UNION operations. Since n MAKE-SET operations are performed, we know $(m - n - h)$ FIND-SET operations are performed. By the lemma, we know that the total actual cost of UNION is $O(h \lg h)$. Hence the total actual cost of the sequence is

$$O(\underbrace{n}_{\text{MAKE-SET}} + \underbrace{(m - n - h)}_{\text{FIND-SET}} + \underbrace{h \lg h}_{\text{UNION}}) = O(m - h + h \lg h)$$

The total amortized cost of the sequence is

$$O(\underbrace{n}_{\text{MAKE-SET}} + \underbrace{(m - n - h)}_{\text{FIND-SET}} + \underbrace{h \lg n}_{\text{UNION}}) = O(m - h + h \lg n)$$

Since $h < n$, we have showed the claim successfully. \square

21.2-4

In the i th UNION operation, we call $\text{UNION}(x_{i+1}, x_i)$. At this time, the size of set contains x_i contains i members, and the size of set contains x_{i+1} contains 1 members. Then we notice, for all $i \geq 2$, we append the list contains x_{i+1} onto the list contains x_i with the weighted-union heuristic, and this only takes $\Theta(1)$ time for each operation. We operate n times MAKE-SET and $(n - 1)$ times UNION, so the sequence takes $\Theta(n + (n - 1)) = \Theta(n)$ time.

21.2-5

```
1  struct Node
2  {
3      int key;
4      Node *next;
5      // let the tail element be the set's representative
6      union
7      {
8          Node *tail; // for non-tail elements
9          Node *head; // for the tail element
10     } representative;
11     int size; // only for the tail element
12 };
13
14 void MakeSet(Node *x)
15 {
16     x->next = nullptr;
17     x->representative.head = x;
18     x->size = 1;
19 }
20
21 Node* FindSet(Node *x)
22 {
23     return x->next ? x->representative.tail : x;
24 }
25
26 void Union(Node *x, Node *y)
27 {
28     Node **node, *x_head, *y_head, *x_representative, *y_representative;
29     if (x->representative.tail->size < y->representative.tail->size)
30     {
31         Union(y, x);
```

```
32     }
33     else
34     {
35         x_representative = FindSet(x);
36         y_representative = FindSet(y);
37         x_head = x_representative->representative.head;
38         y_head = y_representative->representative.head;
39         x_representative->size += y_representative->size;
40         node = &y_head;
41         while (*node)
42         {
43             (*node)->representative.tail = x_representative;
44             node = &((*node)->next);
45         }
46         *node = x_head;
47         x_representative->representative.head = y_head;
48     }
49 }
```

21.2-6

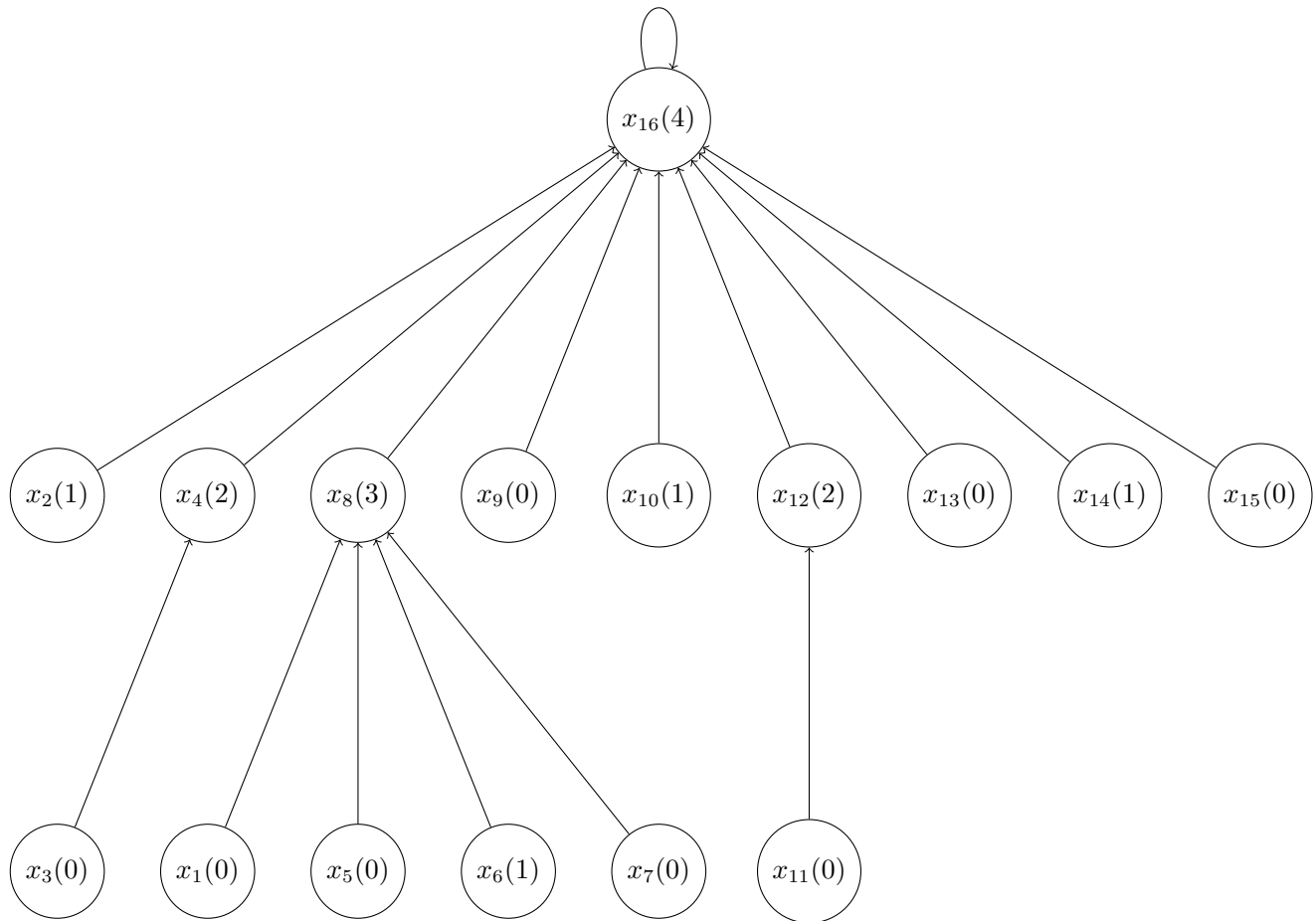
```
1  struct Set
2  {
3      Node *head;
4      int size;
5  };
6
7  struct Node
8  {
9      int key;
10     Set *set;
11     Node *next;
12 };
13
14 void MakeSet(Node *x)
15 {
16     x->next = nullptr;
17     x->set = new Set; // need to be freed
18     x->set->head = x;
19     x->set->size = 1;
```

```
20 }
21
22 Node* FindSet(Node *x)
23 {
24     return x->set->head;
25 }
26
27 void Union(Node *x, Node *y)
28 {
29     Node **node, *x_second;
30     if (x->set->size < y->set->size)
31     {
32         Union(y, x);
33     }
34     else
35     {
36         x->set->size += y->set->size;
37         x_second = x->next;
38         x->next = y->set->head;
39         node = &(x->next);
40         delete y->set;
41         while (*node)
42         {
43             (*node)->set = x->set;
44             node = &((*node)->next);
45         }
46         *node = x_second;
47     }
48 }
```

21.3

21.3-1

The data structure in the end (rank of each node is in the parentheses):



Hence $\text{FIND-SET}(x_2)$ and $\text{FIND-SET}(x_{11})$ return a pointer points to x_{16} .

21.3-2

```
1 Node* FindSet(Node *x)
2 {
3     Node *representative, *tmp;
4     representative = x;
5     while (representative != representative->p)
6     {
7         representative = representative->p;
8     }
9     while (x->p != representative)
10    {
11        tmp = x->p;
12        x->p = representative;
13        x = tmp;
14    }
```



```
15     return representative;
16 }
```

21.3-3

Note that we are proving that the upper bound $O(m \lg n)$ is tight (least upper bound), instead of prove it is a tight bound $\Theta(m \lg n)$. In order to prove it, we just need to find an example that takes $\Omega(m \lg n)$ time, which is what the question is asking for. We want our sequence to take as much as possible time. WLOG, assume that $n = 2^k$ for some $k \in \mathbb{N}$. Consider the following sequence:

$$\begin{aligned} &\langle \text{MAKE-SET}(x_1), \text{MAKE-SET}(x_2), \dots, \text{MAKE-SET}(x_n), \\ &\quad \text{UNION}(x_1, x_2), \text{UNION}(x_3, x_4), \dots, \text{UNION}(x_{n-1}, x_n), \\ &\quad \text{UNION}(x_1, x_3), \text{UNION}(x_5, x_7), \dots, \text{UNION}(x_{n-3}, x_{n-1}), \\ &\quad \text{UNION}(x_1, x_5), \text{UNION}(x_9, x_{13}), \dots, \text{UNION}(x_{n-7}, x_{n-3}), \\ &\quad \vdots \\ &\quad \text{UNION}(x_1, x_{n/2+1}), \\ &\quad \text{FIND-SET}(x_1) \dots \quad (\text{until all } m \text{ operations are performed}) \rangle \end{aligned}$$

We performed n MAKE-SET, $(n-1)$ UNION, and $(m-2n+1)$ FIND-SET. We observed we performed $n/2$ UNION(x_i, x_{i+1}), $n/4$ UNION(x_i, x_{i+2}), $n/8$ UNION(x_i, x_{i+4}), \dots . We conclude we performed $n/2^j$ UNION($x_i, x_{i+2^{j-1}}$) for all $j = \{1, 2, \dots, \lg n\}$. For each j , the height of each tree increases by 1. Hence after all $(n-1)$ UNION operations, the height of the tree (for the only set) is $\lg n$. Note that x_1 is the deepest element in the tree. Then, each of FIND-SET(x_1) operation takes $\Theta(\lg n)$ time, and we perform $(m-2n+1)$ times FIND-SET(x_1) operation. Hence all of FIND-SET(x_1) take $(m-2n+1)\Theta(\lg n) = \Theta(m \lg n)$ time. We successfully find a sequence that takes $\Omega(m \lg n)$ time.

21.3-4

We just need to modify LINK procedure to maintain the data structure.

```
1 void Link(Node *x, Node *y)
2 {
3     Node *y_next;
4     if (x->rank > y->rank)
5     {
6         y->p = x;
7     }
8     else
9     {
10        x->p = y;
11        if (x->rank == y->rank)
```

```
12         ++y->rank;
13     }
14     // maintain the circular list
15     y_next = y->next;
16     y->next = x->next;
17     x->next = y_next;
18 }
19
20 std::list<Node*> PrintSet(Node *x)
21 {
22     Node *node;
23     std::list<Node*> result;
24     result.push_back(x);
25     for (node = x->next; node != x; node = node->next)
26     {
27         result.push_back(node);
28     }
29     return result;
30 }
```

21.3-5

Let a_i be the number of nodes with depth greater than 0 (i.e. non-root) in the forest after the i th operation. Let b_i be the number of nodes with depth greater than 1 (i.e. non-root and non-child-of-root) in the forest after the i th operation. Suppose that we start to perform FIND-SET operations in the k th operation. Let the potential function be

$$\Phi(D_i) \begin{cases} a_i & \text{if } i < k, \\ b_i & \text{if } i \geq k \end{cases}$$

where D_i is the disjoint forest after the i th operation. Let $\Phi(D_0) = 0$. Observed each of MAKE-SET and LINK takes $O(1)$ time. Denote $\text{depth}_i(x)$ as the depth of the node x in the tree after the i th operation. Then FIND-SET(x) moves $\max(0, \text{depth}_{i-1}(x) - 1)$ nodes to be the children of the root node. Denote c_i as the cost of the i th operation. Then we assume

$$c_i = \begin{cases} 1 & \text{if MAKE-SET is performed in the } i\text{th operation,} \\ 1 & \text{if LINK is performed in the } i\text{th operation,} \\ \max(1, \text{depth}_{i-1}(x)) & \text{if FIND-SET}(x) \text{ is performed in the } i\text{th operation} \end{cases}$$

Case 1. MAKE-SET is performed in the i th operation.

Then $a_i = a_{i-1}$, so

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 0 = 1$$

Case 2. $\text{LINK}(x, y)$ is performed in the i th operation.

Note that x and y must be root nodes of different tree. This operation make either x or y be a non-root node. Then $a_i = a_{i-1} + 1$, so

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

Case 3. $\text{FIND-SET}(x)$ is performed in the i th operation.

Note that $b_i \leq a_i$ for all i . If $i = k$, then

$$\Phi(D_i) - \Phi(D_{i-1}) = b_i - a_{i-1} \leq b_i - b_{i-1}$$

If $i > k$, then

$$\Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1}$$

Note that

$$b_i - b_{i-1} = -(\max(1, \text{depth}_{i-1}(x)) - 1) = 1 - \max(1, \text{depth}_{i-1}(x))$$

Hence

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \leq c_i + b_i - b_{i-1} = \max(1, \text{depth}_{i-1}(x)) + (1 - \max(1, \text{depth}_{i-1}(x))) = 1$$

We have shown that amortized of each operation is $O(1)$ with the path-compression heuristic, no matter whether we use union by rank or not. Hence the sequence takes $O(m)$ time with the path-compression heuristic, no matter whether we use union by rank or not.

Updating...