HW3 Solution. Yue Li

No. 1

Date

Ex 15.4-5.

① Use some sort algorithm to sort the numbers into an increasing array, say, quick sort. which takes $O(n \lg n)$ time.

② Use the LCS algorithm to find these two arrays (c sorted and unsorted) longes common string. which asks for $O(n^2)$ time.

is Total $O(n^2)$ time,

Prob. 15-4. Planning a company party

Let CR denotes the sum of the conviviality ratings of the guests, and the subtree's root is i, let C(i) denotes the conviviality rating of the guest i, consider the following situation:

1) If the guest i was included, then his child can not be included.

2) Else, the guest i wasn't included, then his child can be included.

Then we have the following reacrrence:

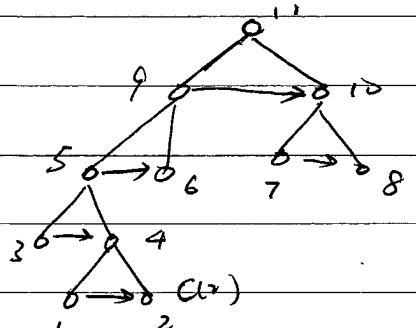
$$CR(i) = MAX\{\sum_{t \in Children(i)} CR(t),\ C(i) + \sum_{t \in Children(Children(i))} CR(t)\}$$

where, children(i) ~~is the~~ are the children of guest i node representing.

Date

The Algorithm:

first, index each node from leaves to root, see fig. below.



An example for indexing.

Then; for $i = 1$ to n

$$\text{if } \sum_{t \in \text{children}(i)} CR(t) > C(i) + \sum_{k \in \text{children}(\text{children}(i))} CR(k)$$

$$\text{then } CR(i) = \sum_{t \in \text{children}(i)} CR(t)$$

$$\text{else } CR(i) = C(i) + \sum_{k \in \text{children}(\text{children}(i))} CR(k)$$

Return $CR(n)$.

~~X~~.

Prob 15-5 $S = \langle \sigma_1 . \sigma_2 \dots \sigma_k \rangle$

a. Let $0 \le i \le k$ and $1 \le v \le n$ which index the letters of S and the vertices. Then define $A(i,v) = 1$ if there is a path from v_0 to v labeled with the prefix of length i of S, and $A(i,v) = 0$ otherwise. The final answer is "Yes, there's a way" if there is v such that $A(k,v) = 1$.

The recurrence becomes:

$$A(i,v) = \begin{cases} \text{MAX}\{ A(i-1,u) \mid (u,v) \in E, \sigma(u,v) = \sigma_i \} & \text{else} \\ 0 & \text{if } i \neq 0, \ v \neq v_0 \\ 1 & \text{if } v = v_0, \ i = 0 \end{cases}$$

The total time for the algorithm is $O(nk)$.

Algorithm:

Starting from $A(0, v_0)$, calculate each entry of the Array below,

$i \backslash v$	v_0 v_1 ... v_n
0	
1	
2	
⋮	
k	

If there's "1" in the last row, return "Yes" else return "no way"

Store the ~~path~~ vertex v_k with "1" in $A(i, v_j)$, if $\sigma(v_k, v_j) = \sigma_j$

To find the path, just trace back from the last row, ~~each~~ (the entry of each row should be "1")

The running time is $O(nk)$.

b. Similar with case (a)

$$P(i,v) = \begin{cases} \text{MAX}\{ P_{uv} \cdot P(i-1,u) \mid (u,v) \in E,\ \sigma(u,v) = \sigma_i \} \\ 1 \quad \text{if } V = V_0,\ i = 0 \\ 0 \quad \text{if } V \neq V_0,\ i \neq 0 \end{cases}$$

where P_{uv} is the probability for edge (u,v).

the running time is $O(nk)$

Ex 17.1-3

1 2 1 4 1 1 1 8 ....

Totally, there are $\lfloor \lg n \rfloor + 1$ terms which are exact power of 2. Therefore, we have

$$\text{Cost} = [n - (\lfloor \lg n \rfloor + 1)] \times 1 + \frac{1 \times (1 - 2^{\lfloor \lg n \rfloor + 1})}{1 - 2}$$

$$= n - \lfloor \lg n \rfloor - 1 + 2^{\lfloor \lg n \rfloor + 1} - 1$$

$$= 3n - \lfloor \lg n \rfloor - 2 \leq 3n$$

$\therefore$ Running time is $O(n)$

Amortized cost per operation is $O(n)/n = O(1)$

Ex 17.2-2.

Assign 3 to each operation as amortized cost.

We first prove such assignment will never make the analysis go into red;

Consider operations $2^i+1, 2^i+2, \ldots, 2^{i+1}-1, 2^{i+1}$, ($2^{i+1}-1-2^i-1+1 = 2^i-1$ terms)

We are going to prove the first 2^i terms above can pay the

2^i th term's cost:

The total credites for the first $2^i - 1$ terms is:

$$TC = (3-1) \times (2^i - 1)$$

$$= 2^{i+1} - 2$$

For the 2^i th term which is 2^{i+1}, the actual cost is 2^{i+1} and the payment is 3, together with credites TC, We have:

$$3 + TC = 3 + 2^{i+1} - 2 = 2^{i+1} + 1 > 2^{i+1}$$

∴ Our assignment is sound.

∴ total time is $O(n)$.

17.3-2:

Let $\phi(D_i) = 2i - 2^{\lfloor \lg i \rfloor + 1}$

$\phi(D_0) = 0$

Base case $i = 1$

$$\hat{c}_1 = c_1 + \phi(D_1) - \phi(D_0)$$

$$= 1 + 2 - 2 - 0 = 1$$

For op i, if i is the power of 2, we have, $i = 2^j$

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

$$= i + 2i - 2^{1+j} - [2(i-1) - 2^{1+j-1}]$$

$$= 2$$

if i is not power of 2

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

$$= 1 + 2i - 2^{1+\lfloor \lg i \rfloor} - [2(i-1) - 2^{\lfloor \lg(i-1) \rfloor + 1}]$$

$$= 3.$$

∴ Total time is $O(n)$

for each operation is $O(1)$.

≥x,

17.4-3

(a) If the TABLE-DELETE does not trigger the contraction,

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$= 1 + |2num_i - size_i| - |2(num_i + 1) - size_i|$$
$$\leq 3$$

(b) If the TABLE-DELETE triggers contraction

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$= num_i + 1 + |2 \cdot num_i - size_i| - |2 \cdot num_{i-1} - size_{i-1}|$$
$$\leq num_i + 1 - |2 \cdot num_i - 3 num_i| \leq 1$$

∴ Bounded by a constant.

Ex.

HW3. Yue Li

No. 7

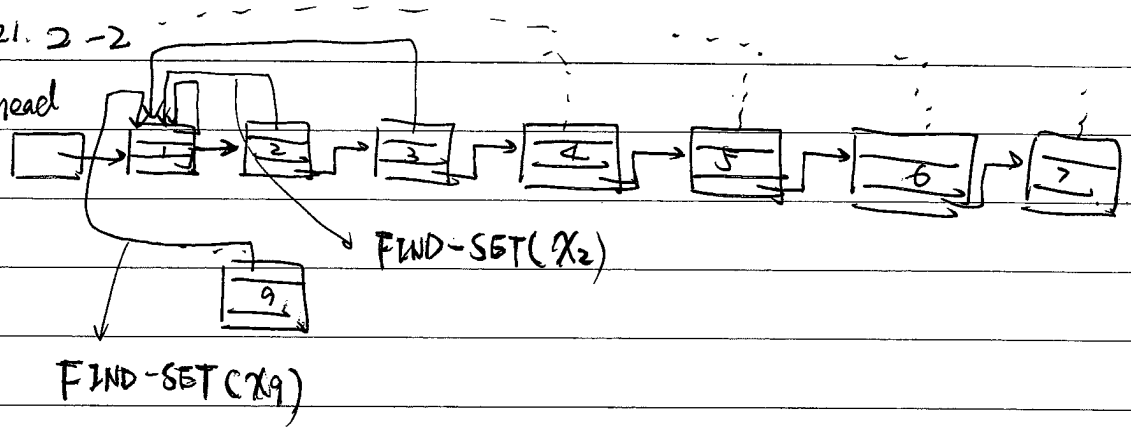
Date

21.1-3

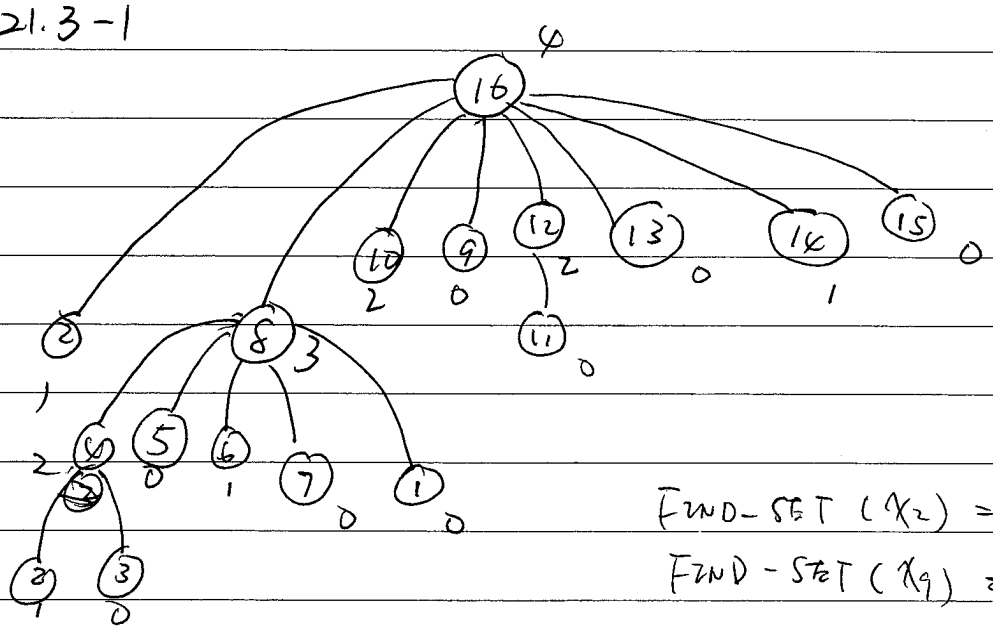
1. FIND-SET: $2|E|$ times

2. UNION: $|V|-k$ 棵 times.

21.2-2



21.3-1



FIND-SET(x_2) = x_{16}

FIND-SET(x_9) = x_{11}

Date

Ex. 21.4-4

If the path compression is not used, with the conclusion of 21.4-2, each FIND-SET operation takes at most $\lfloor \lg n \rfloor$ time. Thus, m Union operation takes $O(m \lg n)$ time.