CMPT307: Disjoint Sets

Week 10-1

Xian Qiu

Simon Fraser University

xianq@sfu.ca

Outline

- dynamic tables
  (another example on amortized analysis)
- disjoint sets

Dynamic Tables

- ▷ to allocate space for a table T using consecutive memories
- ▷ number of items to be stored in T is dynamic
- ▷ load factor $\alpha(T) = \frac{T.\text{num}}{T.\text{size}}$

if T is full, *i.e.* $\alpha(T) = 1$ (or close to one), we expand it:

- ▷ allocate a larger table T'
- ▷ copy T to T' and free T

if $\alpha(T)$ is small, we contract it analogously

Table Expansion

TABLE-INSERT(T, x)

```
1  if T.size == 0 then
2      allocate T.table with 1 slot;
3      T.size = 1;
4  if T.num == T.size then
5      allocate new-table with 2 · T.size slots;
6      insert all items in T.table into new-table;
7      free T.table;
8      T.table = new-table;
9      T.size = 2 · T.size;
10 insert x into T.table;
11 T.num = T.num + 1;
```

worst-case running time $O(n)$

Aggregate Analysis

c_i = cost of the ith TABLE-INSERT = ?

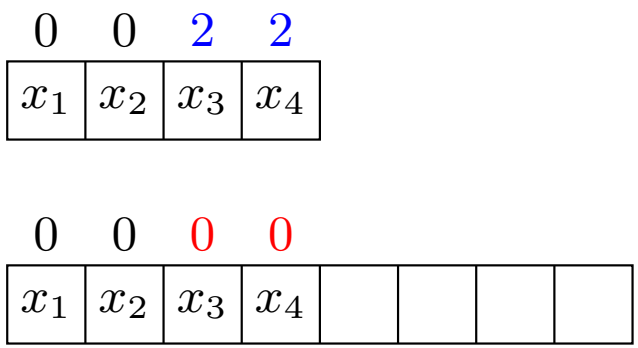
$$c_i = \begin{cases} i, & \text{if } i-1 \text{ is an exact power of 2} \\ 1, & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n = 3n \quad \Rightarrow \hat{c}_i = 3 = O(1)$$

Accounting Method

charge 3 dollars for each insertion

- direct insertion of an element costs 1 dollar
- 1 credit for moving itself to new table
- 1 credit for moving another item that has already been moved once when the table expands



Potential Method

$\Phi(T) = 2 \cdot T.\text{num} - T.\text{size}$ $\in \{0, 1, \ldots, T.\text{num}\}$

if the ith TABLE-INSERT does not trigger an expansion

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_{i-1} - \text{size}_{i-1}) \\ &= 1 + (2 \cdot \text{num}_i - \text{size}_i) - [2(\text{num}_i - 1) - \text{size}_i] = 3\end{aligned}$$

else, $\text{size}_i = 2 \cdot \text{size}_{i-1}$ and $\text{size}_{i-1} = \text{num}_{i-1} = \text{num}_i - 1$

$$\begin{aligned}\hat{c}_i &= \text{num}_i + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_{i-1} - \text{size}_{i-1}) \\ &= 1 - \text{size}_{i-1} + (2 \cdot \text{num}_i - 2 \cdot \text{size}_{i-1}) - [2(\text{num}_i - 1) - \text{size}_{i-1}] \\ &= 3\end{aligned}$$

Table Expansion and Contraction

TABLE-DELETE: halve the table size when $\alpha(T) < \frac{1}{4}$ why not $\frac{1}{2}$?

consider a sequence of n TABLE-INSERT, TABLE-DELETE operations

intuition: $\hat{c}_i = O(1)$ for insertion and deletion, so is their mixture

potential method

$$\Phi(T) = \begin{cases} 2 \cdot T.\text{num} - T.\text{size}, & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{1}{2} \cdot T.\text{size} - T.\text{num}, & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

≥ 0?

Disjoint Set Operations

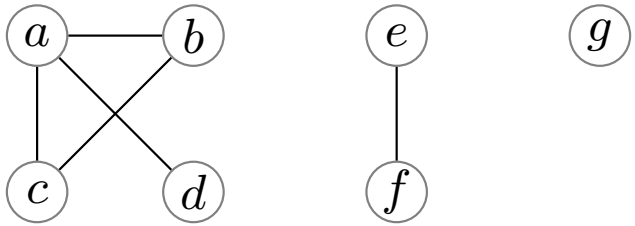
to maintain a collection of disjoint dynamic sets

- ▷ MAKE-SET(x) creates a new set $\{x\}$
- ▷ UNION(x, y) unites the dynamic sets that contain x, y
- ▷ FIND-SET(x) returns a pointer to the representative of the (unique) set containing x

straightforward idea: linked-list representation for dynamic set

applications: connected components, minimum spanning trees etc.

Connected Components



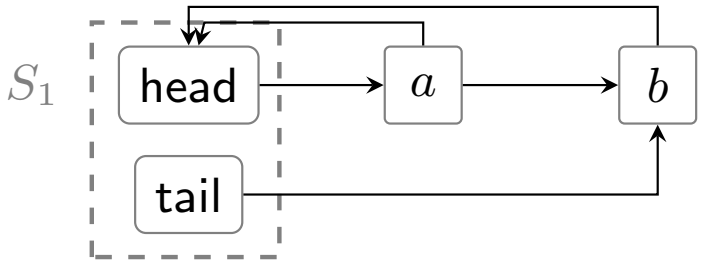
CONNECTED-COMPONENTS(G)

```
1 for v ∈ V(G) do
2     MAKE-SET(v);
3 for each edge (u, v) ∈ E(G) do
4     if FIND-SET(u) ≠ FIND-SET(v) then
5         UNION(u, v);
```

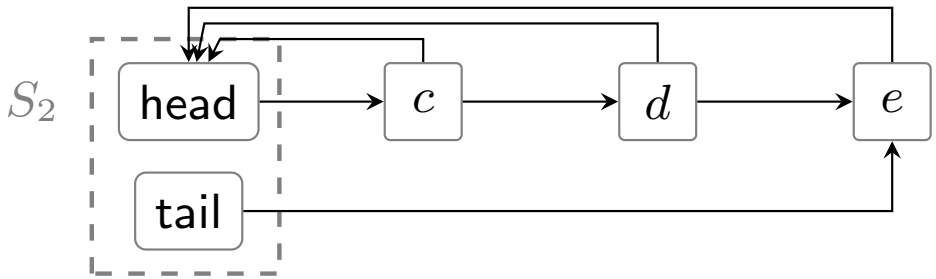
running time

- n times of MAKE-SET
- $O(m)$ times of UNION, FIND-SET

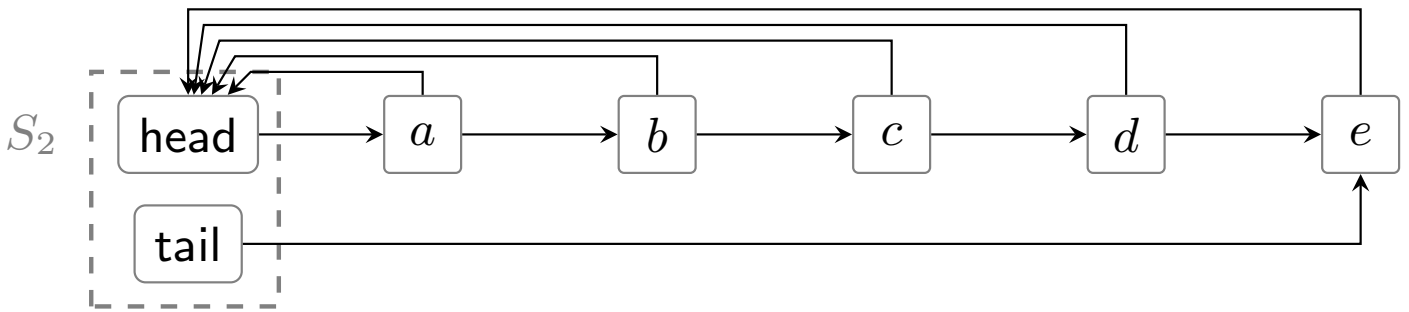
Linked-list Representation



FIND-SET(b)?



UNION(b, d)?



Analysis

- given n objects $x_1, \ldots, x_n$
- execute n times MAKE-SET , followed by $n-1$ times UNION
- # operations $m = 2n-1$, in which n operations are MAKE-SET
- average running time for each operation?

operation	# objects updated
MAKE-SET(x_1)	1
$\ldots$	$\ldots$
MAKE-SET(x_n)	1
UNION(x_2, x_1)	1
UNION(x_3, x_2)	2
$\ldots$	$\ldots$
UNION(x_n, x_{n-1})	$n-1$

- total running time $= n + \sum_{i=1}^{n-1} i = \Theta(n^2)$
- amortize running time per operation is $\Theta(n)$

Weight-union Heuristic

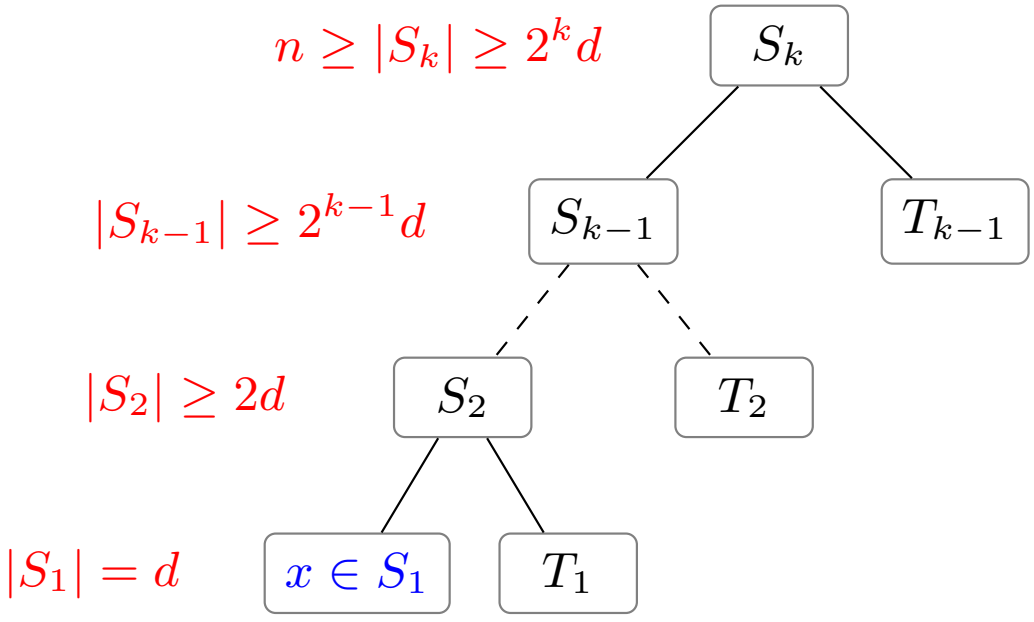
- always update the shorter list for UNION
- set a length attribute of each set and maintain it
- consider m operations (MAKE-SET, UNION, FIND-SET), in which n operations are MAKE-SET

Theorem

The total running time of the m operations is $O(m + n \log n)$.

- each operation costs $O(\log n)$
- $O(m)$ times MAKE-SET, FIND-SET take $O(m)$
- at most $n - 1$ UNION operations, the time for updating *length* is $O(n)$
- to show: at most $O(\log n)$ updates for each element

Proof of the Theorem



$$|T_i| \geq |S_i|,\ 1 \leq i \leq k-1$$

$$k \leq \log \tfrac{n}{d} \leq \log n$$