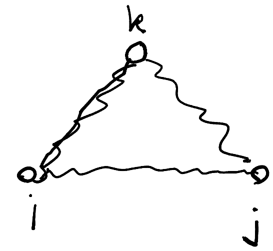
Problem 1: number of all-pairs shortest paths

2017년 11월 13일 월요일

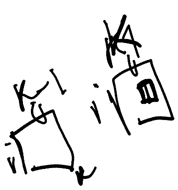
오후 1:11

$\cancel{d_{ij}^{0} = 0 \quad n_{ij}^{0} =}$

$$\left.\begin{aligned} d_{ij}^{0} &= l(v_i, v_j) \\ n_{ij}^{0} &= \boxed{1} \end{aligned}\right] \text{ if } \underline{(v_i, v_j) \in E}$$



$$\left.\begin{aligned} d_{ij}^{0} &= \infty \\ n_{ij}^{0} &= \boxed{0} \end{aligned}\right] \text{ if } \underline{(v_i, v_j) \notin E}$$



$$d_{ij}^{k} = \min\{\underline{d_{ij}^{k-1}}, d_{ik}^{k-1} + d_{kj}^{k-1}\} \quad \text{if } k > 0$$

$$n_{ij}^{k} = \begin{cases} n_{ij}^{k-1} & \text{if } \underline{d_{ij}^{k-1} < d_{ik}^{k-1} + d_{kj}^{k-1}} \\ n_{ik}^{k-1} \times n_{kj}^{k-1} & \text{if } > \\ n_{ij}^{k-1} + n_{ik}^{k-1} \times n_{kj}^{k-1} & \text{if } = \end{cases}$$

$n_{ij}^{k-1} + n_{ik}^{k-1} \times n_{kj}^{k-1}$

Algo

for $i \leftarrow 1$ to n **do**

  for $j \leftarrow 1$ to n **do**

    → **if** $i = j$ **then** $d_{ij}^0 = 0$; $n_{ij}^0 \leftarrow 1$

    elseif $(v_i, v_j) \in E$ **then**

      $d_{ij}^0 \leftarrow \ell(v_i, v_j)$, $n_{ij}^0 \leftarrow 1$

$O(n^2)$ **else** $d_{ij}^0 \leftarrow \infty$, $n_{ij}^0 \leftarrow 0$

    endif

  endfor

endfor

for $k \leftarrow 1$ to n do

  for $i \leftarrow 1$ to n **do**

    for $j \leftarrow 1$ to n **do**

      $\cancel{d_{ij}^k \leftarrow d_{ij}^{k-1}\ ;\ n_{ij} \leftarrow n_{ij}}$

      if $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$ **then**

$$n_{ij}^{k} = n_{ij}^{k-1} + n_{ik}^{k-1} \times n_{kj}^{k-1}$$

<u>elseif</u> $d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1}$

$$d_{ij}^{k} \leftarrow d_{ik}^{k-1} + d_{kj}^{k-1}$$

$$n_{ij}^{k} \leftarrow n_{ik}^{k-1} \times n_{kj}^{k-1}$$

$O(n^3)$

space $O(n^2)$

<u>endif</u>

<u>end for</u>

<u>end for</u>

<u>end for</u>

Problem 2; Dynamic Tables

2017년 11월 13일 월요일

오후 1:13

Scenario

- Have a table—maybe a hash table.
- Don't know in advance how many objects will be stored in it.
- When it fills, must reallocate with a larger size, copying all objects into the new, larger table.
- When it gets sufficiently small, *might* want to reallocate with a smaller size.

Details of table organization not important.

Goals

1. $O(1)$ amortized time per operation.
2. Unused space always $\leq$ constant fraction of allocated space.

Load factor $\alpha = num/size$, where num = # items stored, $size$ = allocated size.

If $size = 0$, then $num = 0$. Call $\alpha = 1$.

Never allow $\alpha > 1$.

Keep $\alpha >$ a constant fraction $\Rightarrow$ goal (2).

Table expansion

Consider only insertion.

- When the table becomes full, double its size and reinsert all existing items.
- Guarantees that $\alpha \geq 1/2$.
- Each time we actually insert an item into the table, it's an ***elementary insertion***.

```
TABLE-INSERT(T, x)
  if T.size == 0
      allocate T.table with 1 slot
      T.size = 1
  if T.num == T.size                                   // expand?
      allocate new-table with 2 · T.size slots
      insert all items in T.table into new-table       // T.num elem insertions
      free T.table
      T.table = new-table
      T.size = 2 · T.size
  insert x into T.table                                // 1 elem insertion
  T.num = T.num + 1
```

Initially, $T.num = T.size = 0$.

Running time

Charge 1 per elementary insertion. Count only elementary insertions, since all other costs together are constant per call.

c_i = actual cost of ith operation

- If not full, $c_i = 1$.
- If full, have $i - 1$ items in the table at the start of the ith operation. Have to copy all $i - 1$ existing items, then insert ith item $\Rightarrow c_i = i$.

n operations $\Rightarrow c_i = O(n) \Rightarrow O(n^2)$ time for n operations.

Of course, we don't always expand:

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

$\{1, 2, 4, \ldots, \lg n\}$

n

$\log_2 n$

$$\begin{aligned} \text{Total cost} &= \sum_{i=1}^{n} c_i \\ &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &= n + \frac{2^{\lfloor \lg n \rfloor + 1} - 1}{2 - 1} \\ &< n + 2n \\ &= 3n \end{aligned}$$

Therefore, **aggregate analysis** says amortized cost per operation = 3.

$\frac{3n}{n} = O(1)$

Accounting method

Charge \$3 per insertion of x.

- \$1 pays for x's insertion.
- \$1 pays for x to be moved in the future.
- \$1 pays for some other item to be moved.

Suppose we've just expanded, $size = m$ before next expansion, $size = 2m$ after next expansion.

m

- Assume that the expansion used up all the credit, so that there's no credit stored after the expansion.
- Will expand again after another m insertions.

Assume that the expansion used up all the credit, so that there's no credit stored after the expansion.

- Will expand again after another m insertions.
- Each insertion will put \$1 on one of the m items that were in the table just after expansion and will put \$1 on the item inserted.
- Have \$$2m$ of credit by next expansion, when there are $2m$ items to move. Just enough to pay for the expansion, with no credit left over!

Potential method

Dynamic Table

$\Phi(T) = 2 \cdot T.num - T.size$

4 $\quad n = 3$, $s = 4$ $\quad 6 - 4 = 2$

- Initially, $num = size = 0 \Rightarrow \Phi = 0$.
- Just after expansion, $size = 2 \cdot num \Rightarrow \Phi = 0$.
- Just before expansion, $size = num \Rightarrow \Phi = num \Rightarrow$ have enough potential to pay for moving all items.
- Need $\Phi \geq 0$, always.

$\Phi = num$

Always have

$$\begin{aligned} size \geq num &\geq size/2 \Rightarrow \\ 2 \cdot num &\geq size \Rightarrow \\ \Phi &\geq 0\,. \end{aligned}$$

$\Phi \geq 0$

Amortized cost of i th operation

num_i = num after ith operation ,

$size_i$ = $size$ after ith operation ,

Φ_i = Φ after ith operation .

- If no expansion:

$$size_i = size_{i-1}\,,$$
$$num_i = num_{i-1} + 1\,,$$
$$c_i = 1\,.$$

Then we have

expansion

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\ &= 1 + 2 \\ &= 3\,. \end{aligned}$$

3

$c_i + \Phi(T_i) - \Phi(T_{i-1}) =$

$1 \quad (2num_i - size_i) - (2num_{i-1} - size_{i-1})$

$c_i \quad + 2$

$1 \quad 2num_{i-1} \quad 2$

$size_i = 2size_{i-1}$

$num_i = num_{i-1} + 1$

$c_i = size_{i-1}$

- If expansion:

$$\begin{aligned} size_i &= 2 \cdot size_{i-1} \ , \\ size_{i-1} &= num_{i-1} = num_i - 1 \ , \\ c_i &= num_{i-1} + 1 = num_i \ . \end{aligned}$$

  Then we have

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i + \Phi_{i-1} \\ &= num_i + \ (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= num_i + \ (2 \cdot num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\ &= num_i + \ 2 - (num_i - 1) \\ &= 3 \ . \end{aligned}$$

Expansion and contraction

When α drops too low, contract the table.

- Allocate a new, smaller one.
- Copy all items.

Still want

- α bounded from below by a constant,
- amortized cost per operation $= O(1)$.

Measure cost in terms of elementary insertions and deletions.

"Obvious strategy"

- Double size when inserting into a full table (when $\alpha = 1$, so that after insertion α would become > 1).
- Halve size when deletion would make table less than half full (when $\alpha = 1/2$, so that after deletion α would become $< 1/2$).
- Then always have $1/2 \le \alpha \le 1$.
- Suppose we fill table.

$$
\begin{aligned}
\text{Then insert} &\Rightarrow \text{double} \\
\text{2 deletes} &\Rightarrow \text{halve} \\
\text{2 inserts} &\Rightarrow \text{double} \\
\text{2 deletes} &\Rightarrow \text{halve} \\
\dots &
\end{aligned}
$$

Not performing enough operations after expansion or contraction to pay for the next one.

Simple solution

- Double as before: when inserting with $\alpha = 1 \Rightarrow$ after doubling, $\alpha = 1/2$.
- Halve size when deleting with $\alpha = 1/4 \Rightarrow$ after halving, $\alpha = 1/2$.
- Thus, immediately after either expansion or contraction, have $\alpha = 1/2$.
- Always have $1/4 \le \alpha \le 1$.

Intuition

- Want to make sure that we perform enough operations between consecutive expansions/contractions to pay for the change in table size.
- Need to delete half the items before contraction.
- Need to double number of items before expansion.
- Either way, number of operations between expansions/contractions is at least a constant fraction of number of items copied.

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

T empty $\Rightarrow \Phi = 0$.

$\alpha \ge 1/2 \Rightarrow num \ge size/2 \Rightarrow 2 \cdot num \ge size \Rightarrow \Phi \ge 0$.

$\alpha < 1/2 \Rightarrow num < size/2 \Rightarrow \Phi \ge 0$.

Problem 3

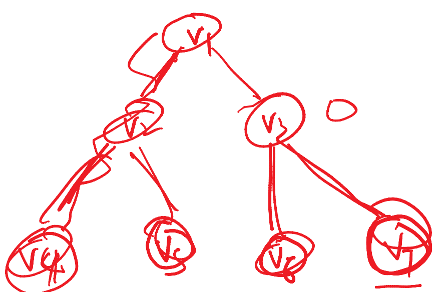
2017년 11월 13일 월요일
오후 1:14

3. CLRS 21.3-4: Suppose that we wish to add the UNION-FIND operation PRINT-SET.x/, which is given a node x and prints all the members of x's set, in any order. Show how we can add just a single attribute to each node in a disjoint-set forest so that PRINT-SET.x/ takes time linear in the number of members of x's set and the asymptotic running times of the other operations are unchanged. Assume that we can print each member of the set in $O(1)$ time.

Maintain a circular, singly linked list of the nodes of each set.
To print, just follow the list until you get back to node x, printing each member of the list.
The only other operations that change are FIND, which sets x.next = x, and LINK, which exchanges the pointers x.next and y.next.

Problem 4

2017년 11월 13일 월요일
오후 1:15



← intern info

↑ (rate, base-salary)

D(v) ≡ product of D(u) from root to leaf ℓ = salary of

~~D(v) × ~ × base-sal~~

search(numb)

v ← root

while v is not a leaf do

for each child u of v do

D(u) ← D(u) × D(v)

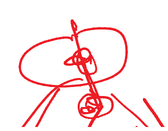
endfor

D(v) ← 1

endwhile

return(v)

end proc



hd proc

* Find Sibling