Avancerade algoritmer DD2440

Homework D

Jim Holmström - 890503-7571

December 15, 2011

Exercise 1. Analyze the amortized complexity of a series of inserts and deletes assuming.

- **The table starts empty (num=0) and with (size = 0)**
- **If the table is of size zero and an element is added it grows to size 1.**
- **If a tabl of size $s \leq 1$ is full and an element is added, the table is grown to size $2n$, elements are relocated and the new element is added.**
- **If, after removal, a table of size $s \leq 1$ has less than $s/3$ elements is shrunk to size $\lceil 2s/3 \rceil$, and the elements are relocated.**
- **The cost of adding or deleting an element is 1 (except if the table grows or shrinks)**
- **The cost of growing or shrinking the table is equal to the number of elements moved as a result of relocation.**

Use the potential function $\Phi = |2\mathbf{num} - \mathbf{size}|$

Solution. Define: $\alpha_i = \frac{num_i}{size_i}$

We always know that $1/3 < \alpha_{i-1} < 1$ else the size would have already increased or decreased. $\Phi_i = |2num_i - size_i| = |(2\alpha_i - 1)size_i| = |c_i|$

Lemma 0.1. Using the fact that $b > a, |c-a| - |c-b|$ is $(b-a)$ when $c \leq a$, $(a-b)$ when $c \geq b$ and takes every value (in the continuous case) in between when $a < c < b$ by continuity reasons, more precisely $2c - (a+b)$

Insert:

Without expand we have $size_i = size_{i-1} \wedge num_i = num_{i-1} + 1$

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + |2num_i - size_i| - |2num_{i-1} - size_{i-1}| \\
&= 1 + |c_i| - |c_i - 2| \\
&= 1 + \{c_i \geq 2 : c_i - c_i - 2 = -2, c_i \leq 0 : -c_i + c_i + 2 = 2, c_i = 1 : 0\}
\end{aligned} \tag{1}$$

With expand we have $size_i = 2 \cdot size_{i-1} \wedge num_i = num_{i-1} + 1$

$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$
$= 1 + |2num_i - size_i| - |2num_{i-1} - size_{i-1}|$
$= 1 + |2(num_{i-1} + 1) - size_{i-1}/2| - |2(num_{i-1} - size_{i-1}|$
$= 1 + |2(\alpha_{i-1} - 1/2)size_{i-1} + 2| - |(2\alpha_{i-1} - 1)size_{i-1}|$
$= 1 + (2\alpha_{i-1} - 1/2 - |2\alpha_{i-1} - 1|)size_{i-1} + 2$
{Can for all ranges find a α that eliminates $size_{i-1}$ and bounds the equation}
$< 0 * size_{i-1} + 3 = 3$

Delete:

Without shrinking we have $size_i = size_{i-1} \wedge num_i = num_{i-1} - 1$

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + |2num_i - size_i| - |2num_{i-1} - size_{i-1}| \\ &= 1 + |c_i| - |c_i + 1| \\ &= \{\text{In the same way as the first equation}\} = 1 + \{-1, 1\} \end{aligned} \tag{2}$$

With shrinking we have $size_i = \lceil (2/3) \cdot size_i - 1 \rceil \wedge num_i = num_{i-1} - 1$
Ignoring the ceiling in the bound-analysis since it can only contribute with 1.

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + |2num_i - size_i| - |2num_{i-1} - size_{i-1}| \\ &= 1 + |2(\alpha_{i-1} - 2/3)s_{i-1} - |(2\alpha i - 1 - 1) \cdot size_{i-1}| \\ &= \{\alpha_{i-1} > 1/3 \text{and ignoring a constant number of border-cases}\} \\ &= 1 - 2 + (2\alpha_{i-1} - 2/3 - |2\alpha_{i-1} - 1|)size_{i-1} \\ &= \{\text{In the same way as before}\} < 1 - 2 + 0 * size_{i-1} = -1 \end{aligned} \tag{3}$$

And since all is bounded by a constant the amortized cost for insert/delete is $O(1)$ That is any sequence of n operations is $O(n)$

Exercise 2. Let `makeSet`,`union`, and `findSet` be the three operations of the union-find data structore implmented using a disjoint-set forest with union by rank and path compression. Draw a picture of the data structure that results from running the following code.

```
for (i = 1; i <= 16; i++)
    makeSet(i)
for (i = 1; i <= 15; i += 2)
    union(i, i+1)
for (i = 5; i <= 13; i += 4)
    union(i, i+2)
```

```
#draw the datastructure at this point in time
union(3, 15)
union(10, 8)
union(16, 2)
union(13, 9)
#draw the datastructure at this point in time
```

Solution. I'm sorry for the bad representation but couldn't get the graphics to work properly.

```
(value,rank,p)
```

And splitting the different trees by spaces when it's possible without reordering the nodes.

After first for-loop:

```
( 1, 0, 2)
( 2, 1, 2)

( 3, 0, 4)
( 4, 1, 4)

( 5, 0, 6)
( 6, 1, 6)

( 7, 0, 8)
( 8, 1, 8)

( 9, 0,10)
(10, 1,10)

(11, 0,12)
(12, 1,12)

(13, 0,14)
(14, 1,14)

(15, 0,16)
(16, 1,16)
```

After second for-loop:

```
( 1, 0, 2)
( 2, 1, 2)
```

```
( 3, 0, 4)
( 4, 1, 4)

( 5, 0, 6)
( 6, 1, 8)
( 7, 0, 8)
( 8, 2, 8)

( 9, 0,10)
(10, 1,12)
(11, 0,12)
(12, 2,12)

(13, 0,14)
(14, 1,16)
(15, 0,16)
(16, 2,16)
```

After the three first unions:

```
( 1, 0, 2)
( 2, 2, 2) //union(16,2)

( 3, 0, 4)
( 4, 1,16) //union(3,15)

( 5, 0, 6)
( 6, 1, 8)
( 7, 0, 8)
( 8, 3, 8) //union(10,8)

( 9, 0,10)
(10, 1,12)
(11, 0,12)
(12, 2, 8) //union(10,8)

(13, 0,14)
(14, 1,16)
(15, 0,16)
(16, 2,16)
```

After the last union:

```
( 1, 0, 2)
```

```
( 2, 2, 2)

( 3, 0, 4)
( 4, 1,16)

( 5, 0, 6)
( 6, 1, 8)
( 7, 0, 8)
( 8, 3, 8)

( 9, 0, 8) //2nd path compression
(10, 1, 8) //3rd path compression
(11, 0,12)
(12, 2, 8)

(13, 0,16) //1st path compression
(14, 1,16)
(15, 0,16)
(16, 2, 8) //union(13,9)
```

Exercise 3. In class we analyzed disjoint-set forest with union by rank with path compression. In the analsysis we used without proof that for any element, its rank never exceeds $log_2 n$, prove this. More precisely. Show that in a sequence of m operation where n operations are `makeSet` no element gets rank more than $log_2 n$.

Solution. Having multiple trees will just shrink n for the "deepest" (highest rank) tree, so it won't even be considered. Firstly the trivial case is easy to so that it holds:
$log_2(1) = 0$
$log_2(2) = 1$
since there only exists one way to put them together up to permutation. The rank only increases when 2 trees with the same root rank is linked. And when linked the top root will have at least 2 siblings, the original one and the one that it's linked to's root. Applying this recursivly all the nodes except for a constant number of ones near the leaves will have at least 2 siblings. Counting the number of elements in the root we have at least 2^k number of elements and

Exercise 4. There are n professors and $2n$ classrooms. The input is a table `t[1..n,1..2*n]` where `t[p, c]` is the distance from the office of professor p to a classroom c. Design an efficient algorithm that finds a way to assign each professor to a classroom so that each professor gets a classroom and no classroom is used by more than one professor, and maximize the sum of the distance the professors need to walk if they all walk from their offices to their classrooms. Your algorithm should run in time polynomial in n, and you may analyze it using unit cost. For full credit, its complexity should be noticeably better then $O(n^4)$

Solution. Firstly create $2n - n = n$ "empty-classroom"-professors with all their distances to the classrooms set to zero (n^2), this is to make a perfect matching. Then you want to make an complete assignment of professors to classrooms, including the "empty-classroom"-professors to classrooms, so that each professor gets an unique classroom.

Lemma 0.2. Adding or subtracting a distance c to all distances such that no distance is negative, won't change the assignment.

Lemma 0.3. If all distances are positive $max(distances)$ has the same solution as $min(-distances)$ because mutual distances is still the same and ordering is just shifted.

Create a new set of edges $distance'_{ij} = max(distance_{ij}) - distance_{ij}$ ensuring that all are still positive and has the same solution as the original problem by the lemmas above. $O(n^2)$

The problem can now be easily solved as a minimum weighted matching problem in a bipartite graph by for example the Hungarian algorihm running in $O(n^3)$

Apply this solution to the original problem preserving indeces and this will be a correct solution by the lemmas. $O(n^2)$

This results in a solution in the running time $O(n^3)$ $\square$

Exercise 5. This problem is the same as Problem 4, except that we have a different target function. This time you should maximize the distance that the professor with the shortest path need to take.

Solution. Sort all edges according to their weights in a new list edgelist. $O(|E| + |E|log(|E|)) = O(n^2log(n))$

Binarysearch on indeces i, w_i in the sorted edgelist: $//O(log(|E|) = log(n))$
    Create a unweighted set of edges uw_i from w_i and higher from edgelist. $O(n^2)$
    if(bipartite-matching(uw_i)=n): //it exists a "perfect" match
        try higher value in the binarysearch
    else:
        try lower value in the binarysearch

Where the bipartite-matching for an unweighted graph can be found in "Notes for the course advanced algorithms" by Johan Håstad in chapter 13.0 and has a complexity of $O(n^2\sqrt{n})$

So the total running time in the binarysearch is $O(n^2\sqrt{n}log(n))$