CS580 Algorithm Design, Analysis, and Implementation

Lecture notes, Feb 10, 2022
Wufei Ma

1 Amortized Analysis

Binary counting. We have $n = \sum_{i=0}^{k} A[i] \cdot 2^i$. We increment the number stored in A with the following algorithm.

```
1    Procedure Increment:
2        i = 0
3        while A[i] = 1 do:
4            A[i] = 0
5            i = i + 1
6        endwhile
7        A[i] = 1
```

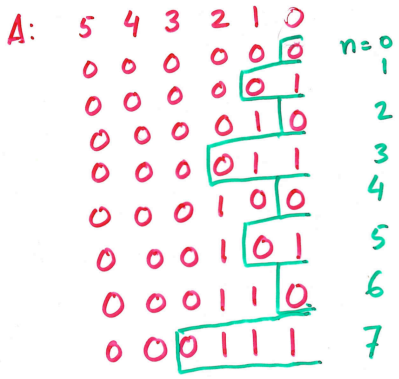

Figure 1: Binary counting problem.

Since A has $1 + \lfloor \log n \rfloor$ bits, the total time/number of steps it takes to increment from 0 to $n-1$ is $O(n \log n)$.

Aggregate method. Let b_i be the number of 1's in the binary representation of i. Let t_i be the trailing 1's in the binary representation of i. The total time is proportional to the number of bit changes

$$\sum_{i=0}^{n-2} 1 + t_i \leq n + \frac{n}{2} + \cdots + 1 \leq 2n$$

Therefore the total time is $T(n) = O(n)$ and the amortized cost per operation is $\frac{T(n)}{n} = O(1)$.

Accounting method. This analysis charges each operation an "amortized cost". If the amortized cost exceeds the actual cost, the excess remains in a data structure as credit. If the amortized cost is small enough so that the actual cost can not be covered, it is paid by credit.

Let the amortized cost of changing 0 to 1 be \$2 and the cost of changing 1 to 0 be \$0. When 0 is changed to 1, we pay \$1 and save \$1 in the credit, which is later used to change from 1 to 0. Each time we invoke `Increment`, the amortized cost is 2. Therefore, there are at most $2n$ costs.

Potential method. Very similar to the accounting method. The only difference is that no explicit credit is saved. Instead, the credits are expressed by a "potential" of the data structure involved.

Let c_i be the cost of the i-th operation. Let D_i be the data structure after the i-th operation. Let $\Phi(D_i)$ be the potential of D_i. We have

$$a_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

It follows that

$$\begin{aligned}\sum_{i=1}^{n} a_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_1)\end{aligned}$$

If we choose $\Phi(D_0) = 0$ and $\Phi(D_n) \geq 0$, then $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} a_i$, which means the total amortized cost is an upper bound of the acutal cost.

In the binary counting problem, we define $\Phi(D_i) = b_i$. It follows that

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1} = (b_{i-1} - t_{i-1} + 1) - b_{i-1} = 1 - t_{i-1} \\ c_i = t_{i-1} + 1 \\ a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 2\end{aligned}$$

Since $\Phi(D_0) = 0$ and $\Phi(D_n) \geq 0$, $\sum_{i=1}^{n} a_i = 2n$ is an upper bound of the total cost.

2 Fibonacci Heaps

Fibonacci heap supports the following operation efficiently in amortized analysis.

Fibonacci heap is a collection of heap-ordered trees. The parent has a lower key than its children.

1. Siblings are doubly-linked in a cycle.

operation	amortized time
Make heap	$O(1)$
Insert	$O(1)$
Min	$O(1)$
Meld	$O(1)$
Delete min	$O(\log n)$
Delete	$O(\log n)$
Decrease key	$O(1)$

Table 1: Amortized time of Fibonacci heap.

2. Parent pointer.
3. Child pointer (one child).

The roots of the heap-ordered trees are linked in a doubly-linked cycle plus a pointer to the minimum key node. Each node consists of

1. 4 pointers (parent `p(x)`, child `c(x)`, left sibling `l(x)`, and right sibling `r(x)`)
2. 1 real number as the key `k(x)`
3. 1 integer as the item `i(x)`
4. 1 bit as the marking `mark(x)`
5. 1 integer as the degree (number of children) `degree(x)`

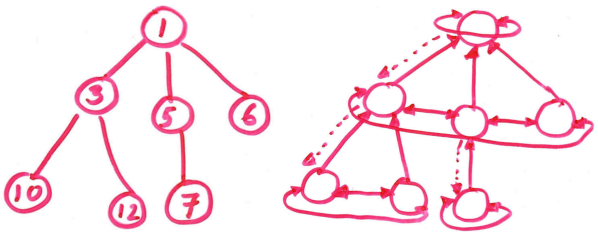

Figure 2: Fibonacci heap.

Linking. Given two heap-ordered trees, make root with bigger key the child of the other root. $O(1)$ time.

Unlinking. $O(1)$ time.

Merge cycle. $O(1)$ time.

Potential function. Let $z_1, \ldots, z_n$ be the sequence of operations. Let $t_i(H)$ be the number of roots in the root list of H after operation z_i. Let $m_i(H)$ be the number of

marked nodes in H after operation z_i. The potential is given by

$$P_i(H) = t_i(H) + 2m_i(H)$$

with $P_0(H) = 0$. Let $T(z_i)$ be the actual time for operation z_i. We have

$$\begin{aligned}
A(z_i) &= T(z_i) + (P_i(H) - P_{i-1}(H)) \\
\sum_{i=1}^{n} T(z_i) &= \sum_{i=1}^{n} (A(z_i) + P_{i-1} - P_i) \\
&= P_0 - P_n + \sum_{i=1}^{n} A(z_i) \\
&= (t_0 - t_n) + 2(m_0 - m_n) + \sum_{i=1}^{n} A(z_i) \\
&= -(t_n + 2m_n) + \sum_{i=1}^{n} A(z_i) \\
&\leq \sum_{i=1}^{n} A(z_i)
\end{aligned}$$

Make heap. Create a null pointer. $O(1)$ time.

Min. Return the minimum key in H. We have $P_i - P_{i-1} = 0$ and the amortized time is the actual time $O(1)$.

Meld. Merge the root cycles of H_1 and H_2. Adjust the min pointer to the new min. The potential change is

$$\begin{aligned}
P_i - P_{i-1} &= (t_i + 2m_i) - (t_{i-1}(H_1) - 2m_{i-1}(H-1)) - (t_{i-1}(H_2) - 2m_{i-1}(H-2)) \\
&= 0
\end{aligned}$$

The amortized cost is the actual cost $O(1)$.

Insert. Create a F-heap H_1 with one node x. Meld H and H_1. Let H' be the new heap. We have

$$P_i(H') = t_{i-1}(H) + 1 + 2m_{i-1}(H)$$

so

$$P_i - P_{i-1} = 1$$

The amortized time is one plus the actual time, which is $O(1)$.

Delete min. Remove the node with the minimal key from the root cycle. Merge the root cycle with the cycle of children of this node. While two roots r_1 and r_2 have same degrees, link r_1 and r_2. Adjust the minimum key.

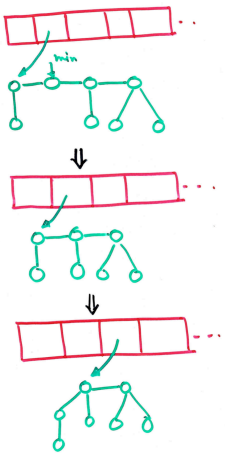

Figure 3: Step 3 of delete min.

Cost analysis. Let $D(n)$ be the max degree of any node in a n-node F-heap. Actual costs:

1. Step 1: $O(1)$
2. Step 2: $O(1)$
3. Step 3: $O(t_{i-1}(H) + D(n))$
4. Step 4: $O(t_{i-1}(H) + D(n))$

Potential change. $D(n)+1+2m_{i-1}(H)-t_{i-1}(H)+2m_{i-1}(H) = D(n)+1-t_{i-1}(H)$. The amortized cost is

$$\begin{aligned}\sum_{i=1}^{n} a_i &= O(t_{i-1}(H) + D(n)) + D(n) + 1 - t_{i-1}(H) \\ &= O(D(n))\end{aligned}$$

since we can scale up the units of the potential to dominate the constant hidden in $O(t_{i-1}(H))$. $D(n) = O(\log n)$ for heaps used in F-heap.