CS 624 - HW 6

Max Weiss

23 March 2020

1. Let's say that a self-organizing list L is *lazy* if never makes any transpositions. And, suppose that L is lazy. It follows that the expected cost of a sequence S of accesses is simply $|S|$ times the expected cost of a single access, where the probability of access of any given value of L is simply the frequency of its occurrence in S. Then the expected cost of a single access to L of a value chosen uniformly from the sequence S is given by

$$E[C^L(S)] = \sum_{x \in L} p(x) \operatorname{rank}_L(x) \tag{1}$$

where $p(x)$ is the frequency of occurrence of x in S. I will argue that this expectation is minimized whenever L is sorted in decreasing order of frequency of occurrence of its values.

Certainly, the set of all possible expected costs must have a minimum, since there are only finitely many reorderings of L, and every finite set of real numbers has a least element.

Now suppose that L does not arrange the values of L in decreasing order of their probability p of access. In that case, there are some values u, v such that $\operatorname{rank}_L u < \operatorname{rank}_L v$ while $p(u) < p(v)$. Let L' be the result of transposing u and v in L. By (1), there is some r such that

$$\begin{aligned} E[C^L(S)] &= r + p(u) \operatorname{rank}_L(u) + p(v) \operatorname{rank}_L(v) \\ E[C^{L'}(S)] &= r + p(v) \operatorname{rank}_{L'}(u) + p(u) \operatorname{rank}_{L'}(v) \end{aligned}$$

It therefore suffices to prove the more general statement that if $a < b$ and $c < d$ with $a, b > 0$, then $ad + bc < ac + bd$. Let $x = d - c$. Then $ax < bx$, so that

$$\begin{aligned} ad + bc &= ax + ac + bc \\ &< bx + ac + bc \\ &= ac + bd. \end{aligned}$$

It follows that the minimum expectation must be conferred by some list which arranges its values in decreasing order of their probability. Of course, there may

be more than one such list. So, we need finally to verify that any two such lists L and L' confer the same expectation. If we partition the set of values of L according to their probabilities, then each cell corresponds to a contiguous subsequence of L, and also to some such subsequence of L'. The lists L and L' differ only as to the internal order of these corresponding subsequences. By appeal to (1), we can conclude that they do not differ as to expected cost.

2. Let S be the sequence given by $S_i = \text{parity}(\lfloor \lg(i) \rfloor)$ for all i, with $i \geq 1$. And let $\hat{S}$ be the sequence of proportions of 0s in consecutively larger initial segments of S. I will argue that $\hat{S}$ does not converge.

First suppose that n is even. Then $S[..2^n - 1]$ is a concatenation of sequences of the form

$$\underbrace{0 \cdots 0}_{k} \underbrace{1 \cdots 1}_{2k}$$

so it contains twice as many 1s as 0s. It follows that $\hat{S}$ contains infinitely many occurrences of the ratio $\frac{1}{3}$.

However if n is odd, then $S[2..2^n - 1]$ is a concatenation of sequences of the form

$$\underbrace{1 \cdots 1}_{k} \underbrace{0 \cdots 0}_{2k}.$$

Thus, $S[2..2^n - 1]$ contains twice as many 0s as 1s for all odd n. Therefore, $\hat{S}$ asymptotically approaches $\frac{2}{3}$.

Since a convergent sequence cannot contain infinitely many occurrences of one value while also asymptotically approaching another, it follows that $\hat{S}$ does not converge.

3a. I used the Python script

```
def mtf(L, S):
  print(f"\nstep\tL\t\t\tsought\tcost")
  cost = 0
  for i, x in enumerate(S):
      # seek x in L
      loc = L.index(x)
      cost += loc + 1
      # move to front
      while loc != 0:
          L[loc], L[loc-1] = L[loc-1], L[loc]
          loc -= 1
          cost += 1
      print(f"{i+1}\t{L}\t{x}\t{cost}")
  return L, cost
```

which generated the following output:

step	L	sought	cost
1	['B', 'A', 'C', 'D']	B	3
2	['C', 'B', 'A', 'D']	C	8
3	['A', 'C', 'B', 'D']	A	13
4	['C', 'A', 'B', 'D']	C	16
5	['C', 'A', 'B', 'D']	C	17
6	['A', 'C', 'B', 'D']	A	20
7	['D', 'A', 'C', 'B']	D	27
8	['D', 'A', 'C', 'B']	D	28
9	['A', 'D', 'C', 'B']	A	31
10	['B', 'A', 'D', 'C']	B	38
11	['A', 'B', 'D', 'C']	A	41
12	['C', 'A', 'B', 'D']	C	48
13	['B', 'C', 'A', 'D']	B	53
14	['C', 'B', 'A', 'D']	C	56

3b. The proof of the Sleator-Tarjan theorem actually shows that if A is the MOVE-TO-FRONT algorithm, then $C_A(S) \leq \alpha \cdot (S) + k$ for $\alpha = 4$ and $k = 0$. So the optimal offline algorithm could achieve at best a cost of $56/4 = 14$.

3c. Even an offline algorithm could not achieve a cost of just 14 for the above list of fourteen queries. To see this, note that each query cost includes an access cost. The minimum access cost, namely 1, arises just for access of the first entry of the list. So, a sequence of fourteen queries could achieve a cost of 1 only by accessing just the list's first entry each time, while making no transpositions. So, that sequence of queries would have to return a constant sequence of return values. It follows that the above query could not be supported correctly at a total cost of only 14.

4a. Here is one way to implement a queue using two stacks. Call one stack In, and the other Out. To enqueue an object, simply push it onto In. To dequeue an object, first try to return the result of popping from Out. If that fails, then successively pop each element from In and push it onto Out. Try again to return the result of returning a pop from Out, raising an empty queue exception if this fails.

The enqueue operation is clearly $O(1)$, since it uses a single push. The cost of dequeuing depends on the state of the implementing structure. If the Out stack is not empty, then the cost is simply the constant cost of a single pop. So suppose that the Out stack is empty. If furthermore the In stack is empty, then the cost is the constant cost of raising an exception. So suppose finally that the In stack contains some $n > 0$ elements. Then the implementation requires n pops interleaved with n pushes, followed by another pop. So, the cost in the worst case is proportional to n. Of course, in this case, n is simply the stack size. So the dequeue operation is in the worst case linear in the stack size.

4b. Consider a sequence of n enqueue and dequeue operations, as implemented above. And suppose, for simplicity, that no element is enqueued more than once. Certainly, there can be no more than n enqueued elements. Furthermore, each enqueued element can be pushed at most twice, and popped at most twice. So, there are at most $4n$ stack operations. It follows that the amortized cost of both the enqueue and dequeue operations is constant.