Agenda

ANALYSIS OF ALGORITHMS:
ONLINE PROBLEMS AND AMORTIZED ANALYSIS
- COMPETITIVE ANALYSIS
  - REVIEW: PAGING PROBLEMS
- REVIEW: DICTIONARY DATA STRUCTURE
  - INPUT DISTRIBUTIONS

Sundar B. CS&IS, BITS, Pilani @Pilani

Amortized Analysis

- Usually, algorithms are analyzed for

  (i) worst case behavior and (ii) average case behavior
- Average case behavior is aggregated (to compute the average) over a sequence of inputs:
  - But often a specific input distribution is assumed
    - typically, it is the *uniform distribution*
  - Real workloads often behave differently:
    - *uniform distribution is not (necessarily) the common case!*

Amortized Analysis

- Recall the competitive analysis of page replacement algorithms:
  - Worst case behavior was measured but it was averaged over a sequence of inputs!
  - This is referred to as ***amortized analysis***.

Dictionary Data Structure

- Consider the dictionary data structure with its typical operations:
  - find, insert, and delete
- Usually analysis is done on an individual operation:
  - e.g. what is the worst case time complexity of a find operation in a list?
- Or it is averaged over a sequence of operations:
  - e.g. what is the average case time complexity of a find operation in a list?
    - The answer to this depends on input distribution.
- In fact the way the list can be best arranged will depend on the input distribution.

Dictionary Data Structure

- The way the list can be best arranged will depend on the input distribution:
  - i.e. in an offline problem scenario in which the designer knows the input sequence ahead of time
    - she can decide a data structure that is best for the sequence.
  - e.g. a Binary Search Tree where frequency of access of each element is known
    - and therefore more frequently accessed items can be placed near the root
    - *Recall the exercise on* *<u>Dynamic Programming algorithm for Optimal BST</u>*

Dictionary Data Structure

- But when operations are online, one needs adaptive data structures:
  - these are referred to as self-organizing lists:
    - e.g. can you rearrange the BST if you know the frequency of one or more input items?

Self-Organizing Lists: Abstract Model

- Assume the following dictionary model:
  - A dictionary stores its elements as an unsorted list
    - find scans the list sequentially, i.e. to locate the i^{th} item, the cost is i.
    - Similarly, insert would cost i+1 for the i^{th} item.

- Suppose accesses are independent of each other and suppose the probability of accessing item i is given, say, p_i
  - An optimum algorithm will arrange items in non-increasing order by probability :
    - let us refer to this algorithm as **DP** (for *decreasing probability*).

Self-Organizing Lists

- Self-Organizing Strategies:
  - Move-to-Front (MF):
    - On access/insertion, move the item to the front, without changing the relative order of other items.
  - Transpose (T):
    - On access/insertion, exchange it with the preceding item
  - Frequency Count (FC):
    - Maintain the list in non-increasing order by frequency count. Increase count on access/insertion.

S-O Lists: Performance

- Suppose
  - the list size is fixed,
  - accesses are independent of each other, and
  - the probability of accessing item i is given, say, p_i
    - [**Note:** *The last assumption is required in DP, but in other cases we use it only for analysis*. **End of Note**.]

- What would be the competitive performance of the online algorithms?

S-O Lists: Performance of FC

- How competitive is FC w.r.t. DP ?
  - $E_{FC} \;/\; E_{DP} \cong 1$
  - Intuitive argument (based on the Law of Large Numbers):
    - Consider a long sequence of operations i.e.
      - #operations >> size of list
    - FC would have put the most frequent items in the beginning of the list
      - *according to the frequency (at this point)*
    - Now if you run DP on this sequence of operations on this list, how would they (FC and DP) compare?

S-O Lists: Performance of MF

- Under assumptions similar to those of the last slide:
  - $E_{MF}(p) / E_{DP}(p) <= 2$.
- Proof:
  - Given a sequence S,
    - let **b(i,k)** be the (asymptotic) probability that the S_i appears in the list before S_k
    - S_i appears before S_k if the most recent access of S_i happened after the most recent access of S_k
      - Let **m** denote the number of intervening requests between accesses to S_i and S_k
      - Then

$$b(i,k) = p_i * \Sigma_{m=0}^{\infty} (1 - p_i - p_k)^m$$
$$= p_i / (p_i + p_k)$$

S-O Lists: Performance of MF [*contd.*]

- $E_{MF}(p) / E_{DP}(p) <= 2$.
- Proof [*contd.*]:
  - **$b(i,k) = p_i / (p_i + p_k)$**
  - The average search time (i.e. E_{MF}) is given by:
    - $\Sigma_{1<=k<=n} (p_k * (1 + \Sigma_{1<=i<=n,i<>k} b(i,k)))$
      $1 + 2 * \Sigma_{1<=i<k<=n} p_i*p_k / (p_i + p)$
  - The optimal offline time i.e. E_{DP} is
    - $\Sigma_{1<=k<=n} p_k * k$
  - The ratio of average time to optimal time turns out to be bounded by
    - $2 * (1 – 1/(n+1))$

CS&IS, BITS, Pilani @Pilani Sundar B.

S-O Lists: Performance of MF and T - Results

- $E_{MF}(p) / E_{DP}(p) <= 2$.
- $E_{T}(p) <= E_{MF}(p)$
  - But MF performs much better in practice:
    - because it soon converges to its asymptotic behavior given a random initial list
    - and it behaves close to a static decreasing frequency algorithm.
  - When tested on real data:
    - MF beats T consistently
    - MF is competitive with FC and sometimes better
      - *because MF is tuned for data with high locality*

Amortized Analysis

- Consider a hashtable with separate chaining on collision:
  - What is the cost of find operation?
    - What does it depend on?
  - How do you adapt when collisions increase?
    - What is the cost of adaptation?
  - Exercise:
    - Derive the amortized cost with rehashing:
      - Clearly state the assumptions i.e. *the values chosen for design parameters*:
        - *threshold load factor* for rehashing and *resize factor*.
  - Question:
    - What if you have to consider deletions as well as insertions?