AAA – Classical Search

Ian Sanders

Second Semester, 2024

Current plan

- Analyse various classical algorithms and problems
- Introduce problem
- Discuss naïve approach
- Look at possible improvements:
  - Algorithmic improvements
  - Assumptions about the data
  - Better data structures
- Consider correctness, complexity and optimality

Classical Search

- Given a list of numbers, find a particular key
- Assumptions
  - Key is in the list
  - Key appears exactly once
- Linear search
- Binary search
  Also called *bisection search*

Basic Linear Search

Algorithm linearSearch(*myList*, *n*, *key*)

    Input: $myList, n, key$ where *myList* is an array with *n* entries (indexed $0 \ldots (n-1)$), and *key* is the item sought (must be in *myList*).

    Output: *index* the location of *key* in *myList*.

    $index \leftarrow 0$

    While $key \neq myList[index]$

        $index \leftarrow index + 1$

    Return *index*

Basic linear search – Correctness

- If the key is in the list, will linear search find it?
  - Yes!
  - Why?
- Sketch proof:
  - Direct proof
  - See notes for inductive proof (especially when key may not be in list)

Complexity

Suppose we are searching a list as below:

10	21	42	29	92	61

- Measure run time in number of *basic operations*.
- In this case, *comparisons*.
- Best case
  - Key is 10 therefore 1 comparison
  - In general: When key is in the first position in the list
  - O(1)
- Worst case:
  - Key is 61 therefore length of list comparisons
  - In general: When key is in the last position in the list
  - O(n)

Complexity continued

- Let us now consider the *average* case
- Recall that $g_A(n) = \sum_{i \in D_n} p(i) \cdot t(i)$
  where i is each possible input instance,
  $p(i)$ is the probability of that instance occuring and
  $t(i)$ is the cost of that instance.
- Key could be any number in list
- All numbers equally likely:
  $P(\text{index } i \text{ is key}) = \frac{1}{n}$
- Number comparisons if key is at index i is $i + 1$
- So average work is $\frac{1}{n} \sum_{i=0}^{n-1} i + 1 = \frac{1}{n} \cdot \frac{n(n-1)}{2} = \frac{(n-1)}{2}$ $=\mathrm{O}(n)$

Optimality

- Is there a better algorithm out there?
- i.e. given the same assumptions, can some algorithm perform fewer comparisons?
- Answer: No, linear search is optimal
- For an unordered list, any correct algorithm must check $O(n)$ values.
- Adversarial argument: Algorithm must check elements, but could be in any order
- Adversary could construct a list so that the key is always the last value checked!

Binary search / Bisection Search

- Assume list is sorted
- Use this to divide the problem $\rightarrow$ limit search space

Algorithm $bisectionSearch(myList, n, key)$

  Input: array $myList$ and integers n and key
  where the values in $myList$ are such that
  $myList[0] \leq myList[1] \leq \ldots \leq myList[n-1]$
  Once again key must be in $myList$
  Output: mid the location of key in $myList$.

  $low \leftarrow 0$
  $high \leftarrow n - 1$
  $mid \leftarrow \lfloor (low + high)/2 \rfloor$

  While $key \neq myList[mid]$
    If $key < myList[mid]$
      Then $high \leftarrow mid - 1$
      Else $low \leftarrow mid + 1$
    $mid \leftarrow \lfloor (low + high)/2 \rfloor$
  Output mid

Correctness

- If the key is in the sorted list, will binary search find it?
- Yes, by the rules of arithmetic
- See notes for slightly longer discussion!

Complexity

look again at our list – except that now it is in ascending order.

10	21	29	42	61	92

- Measure in terms of comparisons
  - Best case
    Recall: $mid \leftarrow \lfloor (low + high)/2 \rfloor$
    $\lfloor (0 + 5)/2 \rfloor = 2$
    So if Key is 29 then it is found immediately
    Therefore 1 comparison
  - Worst case:
    Key is last number checked (in this case 21 or other options)
    3 Comparisons

Worst case complexity

- For list of length n, binary search will do at most
  $g(n) = 1 + g(\lfloor n/2 \rfloor)$ comparisons
  1 comparison plus the worst case cost of searching a list of length $n/2$
- We also have the boundary condition $g(1) = 1$
- Now need to solve this recurrence relation
- Solution: $g(n) = \lfloor lgn \rfloor + 1 \in \mathrm{O}(lgn)$.
- Note: that average case is also in $\mathrm{O}(lgn)$.
  It takes a bit of effort to prove this.

Optimality

- Is there a better algorithm out there?
- i.e. given the same assumptions, can some algorithm perform fewer comparisons?
- Answer: No, binary search on sorted list is optimal
  See discussion in notes

To-do: Read analysis of linear and binary search for when key may not be in list