Brute-force Algorithms

Chapter Four

Dedan Kimathi University
of Technology

Learning Outcomes

Describe the brute-force algorithm.

Justify the important of brute-force algorithm

Analyze the time complexity of selection and bubble sort

Brute-force Algorithms

Brute force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

The “force” implied by the strategy’s definition is that of a computer and not that of one’s intellect.

Brute-force strategy is the one that is easiest to apply.

We have already encountered at least two brute-force algorithms in the the consecutive integer checking algorithm for computing.

Example

- Compute for nonzero number and nonnegative integer .
- The problem is trivial but illustrates brute-force algorithm well.
- This suggests simply computing by multiplying by times.

Importance of brute-force

Brute-force is

- applicable to a very wide variety of problems.

- yields reasonable algorithms of at least some practical value with no limitation on instance size problems—e.g., sorting, searching, matrix multiplication, string matching.

- justified if the expense of designing a more efficient algorithm is only for a few instances of a problem needing to be solved.

Brute-force is

- can still be useful for solving small-size instances of a problem
- can serve an important theoretical or educational purpose as a yardstick with which to judge more efficient alternatives for solving a problem.

Examples: Sorting

- The application of the brute-force approach to the problem of sorting:
- given a list of orderable items (e.g., numbers, characters from some alphabet, character strings), rearrange them in nondecreasing order.
- “What would be the most straightforward method for solving the sorting problem?”
- There are two example
- Selection sort
- Bubble sort

Selection sort

Selection sort by scanning the entire given list to find its smallest element and exchange it with the first element, putting the smallest element in its final position in the sorted list. Then we scan the list, starting with the second element, to find the smallest among the last $n - 1$ elements and exchange it with the second element, putting the second smallest element in its final position.

Selection Sort Pseudocode

```
selectionSort(array)
    for i from 0 to length(array) - 1
        minIndex = i
        for j from i + 1 to length(array)
            if array[j] < array[minIndex]
                minIndex = j
        swap(array[i], array[minIndex])
```

Example

- Applying selection sort to the numbers (89, 45, 68, 90, 29, 34, 17)

- Example of sorting with selection sort. Each line corresponds to one iteration of the algorithm, i.e., a pass through the list’s tail to the right of the vertical bar; an element in **bold** indicates the smallest element found.

- Elements to the left of the vertical bar are in their final positions and are not considered in this and subsequent iterations.

Analysis of Selection Sort

Outer Loop (for i from 0 to length(array) - 1):
- This loop runs from the first element to the second-to-last element of the array.
- The number of iterations is $n - 1$, where n is the length of the array.
- Therefore, the outer loop runs $O(n)$ times.

Inner Loop (for j from i + 1 to length(array)):
- For each iteration of the outer loop, the inner loop runs from the element right after i to the last element of the array.
- In the first iteration of the outer loop (i.e., when $i = 0$), the inner loop runs $n - 1$ times.
- In the second iteration of the outer loop (i.e., $when\ i = 1$), the inner loop runs $n - 2$ times, and so on.
- Generally, for each $i - th$ iteration of the outer loop, the inner loop runs $n - 1 - i$ times.

Analysis…

Comparison Operation (if array[j] < array[minIndex]):

•The comparison inside the inner loop is executed every time the inner loop runs.

•Summing up the number of comparisons over all iterations of the inner loop gives:

$$(n-1)+(n-2)+(n-3)\ \ldots\ldots+1+0=\sum_{i=1}^{n-1} i=\frac{(n-1)n}{2}$$

•This is $O(n^2)$

Swap Operation (swap(array[i], array[minIndex])):

•The swap operation occurs once per iteration of the outer loop.

•There are $n-1$ swaps in total, which gives us $O(n)$ for the swap operations.

•However, this does not affect the overall time complexity as it is dominated by the comparisons.

The Best, Average and worst. Case are all $O(n^2)$

Bubble Sort

Uses brute-force by comparing adjacent elements of the list and exchanging them if they are out of order.

It end up “bubbling up” the largest element to the last position on the list.

The next pass bubbles up the second largest element, and so on, until after passes the list is sorted.

Pass of bubble sort can be represented by the following diagram.

Bubble sort pseudocode

https://www.youtube.com/watch?v=JP5KkzdUEYI&list=PPSV

ALGORITHM

//Sorts a given array by bubble sort

//Input: An array of orderable elements

//Output: Array sorted in nondecreasing order

Example

First Pass

Numbers: 89, 45, 68, 90, 29, 34, 17

First Pass

Numbers: 45, 89, 68, 90, 29, 34, 17

First Pass

Numbers: 45, 68, 89, 90, 29, 34, 17

First Pass

Numbers: 45, 68, 89, 90, 29, 34, 17

First Pass

Numbers: 45, 68, 89, 29, 90, 34, 17

First Pass

Numbers: 45, 68, 89, 29, 34, 90, 17

First Pass

Numbers: 45, 68, 89, 29, 34, 17, 90 |

Second Pass

Numbers: 45, 68, 89, 29, 34, 17, 90 |

Second Pass

Numbers: 45, 29, 68, 89, 34, 17, 90 |

Second Pass

Numbers: 45, 29, 68, 34, 89, 17, 90 |

Second Pass

Numbers: 45, 29, 68, 34, 17, 89, 90 |

Analysis of bubble sort

- In analyzing bubble sort
- The number of key swaps, however, depends on the input. In the worst case of decreasing arrays, it is the same as the number of key comparisons:

Conclusion

- Brute-force strategy, the first version of an algorithm obtained can often be improved upon with a modest amount of effort.
- Example: We can improve the crude version of bubble sort given above by exploiting the following observation; if a pass through the list makes no exchanges, the list has been sorted and we can stop the algorithm.
- Even among elementary sorting methods, bubble sort is an inferior choice,
- A first application of the brute-force approach often results in an algorithm that can be improved with a modest amount of effort.

Exercises

Sort the list _E, X, A, M, P, L, E_ in alphabetical order by selection sort and bubble sort.

Alternating disks You have a row of 2n disks of two colors, n dark and n light. They alternate dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighbouring disks.

Design an algorithm for solving this puzzle and determine the number of moves it takes.

THANK YOU

Dedan Kimathi University of Technology

Dr Jane Kuria

Private Bag- Dedan kimathi, Nyeri-Mweiga road

Telephone: ***+254-721709511***

Email: ***jane.kuria@dkut.ac.ke***

Website: ***www.dkut.ac.ke***