VE477
Introduction to Algorithms

Lab 7

Manuel — UM-JI (Fall 2021)

Goals of the lab

- Search algorithms
- Complexity vs. running time
- Python Programming

Unless specified otherwise, all the programs are expected to be completed in Python or O'caml.

1 Programming

In this lab we want to compare three search algorithms. Let A be an array of size n and k be a value to find in A.

1. The first algorithm, `RandomSearch`, consists in randomly searching A for k. One simply selects a random index i and test if $A[i] = k$. If true, then the algorithm returns i and otherwise randomly chooses a new i until either all the indices have been explored or k has been found. In this algorithm a same index can be generated more than once.
   (a) Implement `RandomSearch`.
   (b) Determine the average number of indices that are picked if:
      i. There is no index i such that $A[i] = k$.
      ii. There is exactly one index i such that $A[i] = k$.
      iii. There is more than one index i such that $A[i] = k$.
2. The second algorithm, `LinearSearch`, consists in performing a linear search on A, i.e. testing if k is in $A[1], \cdots, A[n]$, and either return i, the index where k is stored in A, or exit if k is not found.
   (a) Implement `LinearSearch`.
   (b) Determine the average-case and worst-case running times if:
      i. There is no index i such that $A[i] = k$.
      ii. There is exactly one index i such that $A[i] = k$.
      iii. There is more than one index i such that $A[i] = k$.
3. The third algorithm, `ScrambleSearch`, consists in randomly permuting A into A' and then run the `LinearSearch` algorithm on A'.
   (a) Implement `ScrambleSearch`.
   (b) Determine average-case and worst-case running times if:
      i. There is no index i such that $A[i] = k$.
      ii. There is exactly one index i such that $A[i] = k$.
      iii. There is more than one index i such that $A[i] = k$.
4. From the previous complexities which algorithm seems to be the best.
5. Generate A, a random array of one million elements, and time each of the previous algorithms searching a random k on it. Repeat the process one thousand times, keeping track of the previous runs.

(a) Which algorithm performs best in practice?

(b) Discuss the result by comparing to your expected result in item 4.

2 Interview Problems

- Following 11.11 you have received a new pair of binoculars and absolutely want to test them. Unfortunately through your window you can only see other dorms. Without even realising it, you end up counting the number of clothes hanging on each balcony. With your binoculars you can only see k balconies at once. Then you have to slightly move to observe a new one, and as a result a balcony that you could previously see disappears. Write an algorithm which return the max number of clothes you can count on a balcony with your binoculars.

  Example: on input $k = 3$ and $[1, 2, 2, 1, 4]$, return $[2, 2, 4]$. It means that you can observe three balconies at once, for a building featuring five balconies. When you look at the three first ones then the max is 2, then from the second to the fourth max is 2, and finally from the third to the fifth the max is 4.

- With global warming you are afraid of the water level increasing and flooding cities. Therefore you decide to investigate what would be the safest city to go to by looking at its skyline. You assume that water could fill up the space between two buildings and as such decide to travel to the city which would retain as little water as possible. Given a array representing the skyline, find the amount of water a city can retain.

  Example: for a city with skyline in black you add water wherever it can get stuck, e.g. Based on the skyline, you construct the array $[1, 0, 2, 3, 1, 1.6, 2, 1]$ corresponding the height of each building in black, and assuming a constant width of one, you want to determine the amount of water in blue.