Sum to target

Serge Kruk

May 16, 2022

Moral (Never forget this!)

An algorithm, or at least its runtime, is intimately tied to the data structure used.

- Note that in some languages, the exact same code can be used for various data structures (array or linked list), but the runtimes may be vastly different.

A short exercise to review everything we have seen.

- A problem
- A solution
  - Its correctness
  - Its runtime
- Rinse and repeat

Discuss (Design, Correctness, Efficiency)

- Problem 1a: Given an array of integers, find two, if there are two in the array, that sum to zero.

A Brute force approach

- Always be ready to produce the brute-force approach.
- We have a choice in what to return. Lets return the positions.
- Here is a solution, jumbled up. Put lines in the right order.

```
def zero_sum(a):
for i in range(0, n-1):
for j in range(i+1,n):
if a[i]+a[j] == 0:
n = len(a)
return None
return a[i],a[j]
```

A Brute force approach

```
def zero_sum(a):
    n = len(a)
    for i in range(0, n-1):
        for j in range(i+1,n):
            if a[i]+a[j] == 0:
                return a[i],a[j]
    return None
```

A few tests

```
print(zero_sum([-1, 2, 3, 6, -2, 9]))
print(zero_sum([-1, 2, 3, 6, -4, 9]))
```

(1, 4) (-1, -1)

Correctness

The algo terminates since the for loops have finite bounds.
The loop invariant: If we enter the outer loop with i at value k, then no element in $a[0..k-1]$ has an inverse anywhere in the array. In the inner loop we add $a[k]$ to each element from $a[k+1]$ to $a[n-1]$. If one pair sums to zero, we return its indices. If we exit the inner loop at the end, then $a[k]$ has no inverse anywhere in the array.
If we consider all pairs without success, we return an indicator of failure.

Run time

The worst case is when there are no elements summing to zero. Let us consider that case. Let us count the number of times the `if` statement is executed. Let us assume $n = len(a)$

$$\begin{aligned}
\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 &= \sum_{i=0}^{n-2} n-1-(i+1)+1 \\
&= \sum_{i=0}^{n-2} n-i-1 \\
&= \sum_{i=0}^{n-2} n-1-\sum_{i=0}^{n-2} i \\
&= (n-2-0+1)(n-1)-\frac{(n-1)(n-2)}{2} \\
&\in \Theta(n^2)
\end{aligned}$$

Discuss (Design, Correctness, Efficiency)

- Problem 1b: Assume that you have access to a super-fast sorting function (i.e. faster than $\Theta(n^2)$). Can you improve your solution's runtime?

A scanning approach

First we sort, then we can scan from left and right. Jumbled up version. You put the lines in order.

```
a = sorted(a)
def zero_sum_s(a):
elif s<0:
elif s>0:
if s==0:
l = l+1
l,r = 0,len(a)-1
r = r-1
return None,None
return a[l],a[r]
s = a[l]+a[r]
while l < r:
```

A scanning approach

```
def zero_sum_s(a):
    a = sorted(a)
    l,r = 0,len(a)-1
    while l < r:
        s = a[l]+a[r]
        if s==0:
            return a[l],a[r]
        elif s<0:
            l = l+1
        elif s>0:
            r = r-1
    return None,None
```

Correctness

- We start with l and r at the first and last positions of the array. At each step we either increase the left or decrease the right until they cross. Therefore the algorithm terminates.
- Inside the loop we consider $a[l] + a[r]$.
  - If it sums to zero we return these elements, which is required.
  - If the sum is negative, we know that $a[l]$ has no inverse element in the range $[l..r]$ since it would be larger than $a[r]$ and therefore to the right. We can therefore increase l.
  - If the sum is negative, the inverse argument shows that $a[r]$ has no inverse in the range $[l..r]$.

Run time

- The runtime is the time to sort plus the time to scan the whole array in the worst case, hence $sort(n) + \Theta(n)$
- In the best case $sort(n)$.

Most optimistic runtime

Can it be done in $\Theta(n)$ time? (Assuming n is the length of the array.)
Maybe, but clearly not any better since we need to read the array.

Best possible algorithm?

Hash the elements, then read the array again looking for -a[i] in the hash table. (Minor variation to avoid reading twice.)

Discuss (Design, Correctness, Efficiency)

Problem 2: Given an array of integers and a number, T, find two elements in the array, if there are such two elements, that sum to T.

Discuss (Design, Correctness, Efficiency)

Problem 3: Given an array of integers, find a subset, if there is one, that sums to zero.

Homework/Test questions

```
def f(n,p):
  for i in range(n):
    for j in range(i,p):
      for k in range(n*p):
        dosomething(i,j,k) / dosomething(j,i,k)
```

- Write down the number of calls to `dosomething` $T(n, p)$ in summation notation.
- Evaluate $T(n, p)$

Homework/Test questions

Evaluate the following:

$$T(n) = \sum_{i=0}^{n} \left(2 + \sum_{j=i}^{n-1} 3 \right)$$

Homework/Test questions

```
def f(n):
  if n <= 0:
    return 42
  elif n == 1:
    return 43
  else:
    return f(n-1) + f(n-2)
```

Runtime?