How do you check if one array is a subsequence of another?

Asked 6 years, 4 months ago Active 7 months ago Viewed 3k times

2

I'm looking to explore different algorithms, both recursive and dynamic programming, that checks if one arrayA is a subsequence of arrayB. For example,

```
arrayA = [1, 2, 3]
arrayB = [5, 6, 1, 7, 2, 9, 3]

thus, arrayA is indeed a subsequence of arrayB.
```

I've tried a few different searches, but all I can seem to find is algorithms to compute the longest increasing subsequence.

algorithm recursion dynamic-programming subsequence

Share Follow

edited Oct 16, 2015 at 16:30

Sergey Kalinichenko
692k 74 1039 1452

asked Oct 16, 2015 at 16:10

Talen Kylon
1,818 7 30 53

do you need efficient solution? – sve Oct 16, 2015 at 17:29

@svs Yes efficiency is important here. – Talen Kylon Oct 16, 2015 at 17:39

So the elements to find in `arrayB` is always in the same order as `arrayA` ? – Sylwester Oct 16, 2015 at 20:30

5 Answers

Active | Oldest | Score

Since you must match *all* elements of `arrayA` to some elements of `arrayB`, you never need to *backtrack*. In other words, if there are two candidates in `arrayB` to match an element of `arrayA`, you can pick the earliest one, and never retract the choice.

Therefore, you do not need DP, because a straightforward linear greedy strategy will work:

```
bool isSubsequence(int[] arrayA, int[] arrayB) {
    int startIndexB = 0;
    foreach (int n in arrayA) {
        int next = indexOf(arrayB, startIndexB , n);
        if (next == NOT_FOUND) {
            return false;
        }
        startIndexB = next+1;
    }
    return true;
}
```

Share Follow

answered Oct 16, 2015 at 16:26

Sergey Kalinichenko
692k 74 1039 1452

This is a clever solution, but are there ways to do it even faster via DP or recursion? – Talen Kylon Oct 16, 2015 at 17:40

1 @TalenKylon In general, DP never helps unless you need to backtrack. This is an O(Na+Nb) solution, where Na and Nb are element counts of `arrayA` and `arrayB` . This algorithm can be implemented with forward-only iterators in O(1) memory. The same item is never looked at more than once, which means that the solution is asymptotically as fast as it gets. – Sergey Kalinichenko Oct 16, 2015 at 18:11

2

As dasblinkenlight has correctly said(and i could not have phrased it better than his answer!!) a greedy approach works absolutely fine. You could use the following pseudocode (with just a little more explanation but totally similar to what dasblinkenlight has written)which is similar to the merging of two sorted arrays.

```
A = {..}
B = {..}

j = 0, k = 0
/*j and k are variables we use to traverse the arrays A and B respectively*/

for(j=0;j<A.size();){

    /*We know that not all elements of A are present in B as we
      have reached end of B and not all elements of A have been covered*/
```

```
        if(k==B.size() && j<A.size()){
            return false;
        }

        /*we increment the counters j and k both because we have found a match*/
        else if(A[j]==B[k]){
            j++,k++;
        }

       /*we increment k in the hope that next element may prove to be an element match*/
        else if(A[j]!=B[k]){
            k++;
        }
    }

    return true; /*cause if we have reached this point of the code
                   we know that all elements of A are in B*/
```

Time Complexity is O(|A|+|B|) in the worst case, where |A| & |B| are the number of elements present in Arrays `A` and `B` respectively. Thus you get a linear complexity.

Share Follow

answered Oct 16, 2015 at 17:43

Rahul Nori
678 6 16

1

As @sergey mentioned earlier, there is no need to do backtracking in this case. Here just another Python version to the problem:
[Time complexity: `O(n)` - worst]

```
>>> A = [1, 2, 3]
>>> B = [5, 6, 1, 7, 8, 2, 4, 3]
>>> def is_subsequence(A, B):
        it = iter(B)
        return all(x in it for x in A)

>>> is_subsequence(A, B)
True
>>> is_subsequence([1, 3, 4], B)
False
>>>
```

Share Follow

edited Dec 31, 2020 at 0:59

answered Nov 24, 2020 at 22:01

Daniel Hao
2,333 2 6 18

0

Here is an example in Ruby:

```
def sub_seq?(a_, b_)
  arr_a = [a_,b_].max_by(&:length);
  arr_b = [a_,b_].min_by(&:length);
  arr_a.select.with_index do |a, index|
    arr_a.index(a) &&
    arr_b.index(a) &&
    arr_b.index(a) <= arr_a.index(a)
  end == arr_b
end

arrayA = [1, 2, 3]
arrayB = [5, 6, 1, 7, 2, 9, 3]

puts sub_seq?(arrayA, arrayB).inspect #=> true
```

Share Follow

edited Apr 28, 2018 at 9:14

answered Apr 28, 2018 at 8:59

lacostenycoder
9,518 4 27 43

0

Here is an example in **GOLANG**...

```
func subsequence(first, second []int) bool {
k := 0
for i := 0; i < len(first); i++ {
    j := k
    for ; j < len(second); j++ {
        if first[i] == second[j] {
            k = j + 1
            break
        }
    }
```

```
        if j == len(second) {
            return false
        }
    }
    return true
    }

    func main(){
          fmt.Println(subsequence([]int{1, 2, 3}, []int{5, 1, 3, 2, 4}))
    }
```

Share Follow

answered Jul 12, 2021 at 15:32

Utsav Singh

1