TÖL212M Rökstudd Forritun - Einstaklingsverkefni 4

Andri Fannar Kristjánsson

9. febrúar 2025

Einstaklingsverkefni 4

1

Klárið að forrita Dafny skrána `E4-skeleton.dfy`

1.1 Svar:

Hér fyrir neðan má sjá leystu útgáfuna sem `Dafny` samþykkir. Hægt er einnig að sjá kóðann hér: https://tinyurl.com/mr3a5zr2. Ath. að ég fékk ekki `tio` til að keyra skránna vegna einhverra fullyrðinga í gefna hlutanum, líklega vegna þess að þar er verið að keyra eldri útgáfu af `Dafny`. Útgáfa 4.10 af `Dafny` samþykkir þetta.

```
// Author of question:  Snorri Agnarsson, snorri@hi.is
// Author of solution:     Andri Fannar Kristjánsson, afk6@hi.is
// Permalink of solution:  https://tinyurl.com/mr3a5zr2

//////////////////////////////////////////////////////////////
// This is the start of the part of the file that should not
// be changed. Following that part is the part you should
// change.
//////////////////////////////////////////////////////////////

// IsSorted(a) er satt þá og því aðeins að
// sannað sé að a sé raðað í minnkandi röð.
// IsSorted(a) is true if and only if it is
// proven that a is in descending order.
predicate IsSorted( a: seq<int> )
{
  forall p,q | 0 <= p < q < |a| :: a[p] >= a[q]
}

// Sannar að poki með einu staki samsvarar runu
// með einu staki.  Dafny þarf smávegis olnbogaskot
// til að fatta það. Þetta er gagnlegt til að sanna
// að útkoman úr Sort sé rétt í sértilvikinu þegar
// raðað er poka m með aðeins einu gildi x, sem
// gefur þá rununa s == [x].
// Proves that a multiset with one element corresponds
// to a sequence with one value. Dafny needs a little
// help to realize this. This is useful to prove that
// the result from Sort is correct in the special case
// where we are sorting a multiset m with only one value
// x which then gives the sequence s == [x].
lemma Singleton( m: multiset<int>, s: seq<int>, x: int )
  requires x in m
  requires x in s
  requires |s| == 1 == |m|
```

```
  ensures |m-multiset{x}| == 0
  ensures s == [x]
  ensures m == multiset{x}
  ensures m == multiset(s)
  ensures IsSorted(s)
{}

method RemoveOne( a: multiset<int> ) returns( b: multiset<int>, x: int )
  requires |a| >= 1
  ensures a == b+multiset{x}
{
  x :| x in a;
  b := a-multiset{x};
}

// Þessi hjálparsetning er gagnleg til að hjálpa
// Dafny að sanna að útkoman úr röðuninni sé rétt.
// This lemma is useful to help Dafny to prove
// that the result from sorting is correct.
lemma LomutoLemma    ( a: multiset<int>
                     , a': seq<int>
                     , x: int
                     , b: multiset<int>
                     , b': seq<int>
                     , c: seq<int>
)
  requires a == multiset(a')
  requires b == multiset(b')
  requires IsSorted(a')
  requires IsSorted(b')
  requires forall z | z in a :: z>=x
  requires forall z | z in b :: z<=x
  requires c == a'+[x]+b'
  ensures forall p | 0<=p<|a'| :: a'[p] in a
  ensures forall p | 0<=p<|b'| :: b'[p] in b
  ensures forall z | z in a' :: z in a && z>=x
  ensures forall z | z in b' :: z in b && z<=x
  ensures forall z | z in a' :: z in a && z>=x
  ensures forall z | z in b' :: z in b && z<=x
  ensures IsSorted(c)
  ensures multiset(c) == a+multiset{x}+b
{
  assert |c| == |a'|+1+|b'|;
  assert forall p,q | 0<=p<q<|c| :: q<|a'| ==> c[p]>=c[q];
  assert forall p,q | 0<=p<q<|c| :: q==|a'| ==> c[q]==x &&
    p<|a'| && c[p]==a'[p] && c[p] in a;
  assert forall p,q | 0<=p<q<|c| :: q==|a'| ==> c[q]==x &&
    p<|a'| && c[p]==a'[p] && c[p] in a && c[p]>=c[q];
  assert forall p,q | 0<=p<q<|c| :: p<|a'| && q>|a'| ==>
    c[p] in a && c[q] in b && c[p]>=c[q];
  assert forall p,q | 0<=p<q<|c| :: p==|a'| && q>|a'| ==>
    c[p]==x && c[q] in b && c[p]>=c[q];
  assert forall p,q | 0<=p<q<|c| :: p>|a'| && q>|a'| ==>
    c[p]>=c[q];
}

// Prófunarfall sem staðfestir að Partition og Sort
// séu áreiðanlega að virka sannanlega rétt.
```

```
// Alls ekki má breyta þessu falli.  Athugið að
// þetta fall skilgreinir í raun þá virkni sem
// Partition og Sort eiga að hafa, þ.e. forskilyrði
// og eftirskilyrði þeirra falla.
// A test function that validates that Partition and
// Sort are provably correct. This function must not
// be modified. Notice that this function does in
// fact define the functionality that Partition and
// Sort should have, i.e. the preconditions and
// postcoditions of those functions.
method Test( m: multiset<int> )
{
  var s := Sort(m);
  assert IsSorted(s);
  assert m == multiset(s);
  if |m| > 0
  {
    var a,p,b := Partition(m);
    assert m == a+multiset{p}+b;
    assert forall z | z in a :: z>=p;
    assert forall z | z in b :: z<=p;
  }
}

// Aðalforritið er óþarfi, en er sett hér til gamans
// svo hægt sé að keyra eitthvað.
// The Main function is not necessary but is put here
// for fun so we have something to run.
method Main()
{
  var x := Sort(multiset{0,9,1,8,2,7,3,6,4,5
                ,0,9,1,8,2,7,3,6,4,5
                }
  );
  print x;
}

//////////////////////////////////////////////////////////////
// This is the end of the unchangable part of the file.
// Following this is the part you should modify in order to
// implement a version of quicksort.
//////////////////////////////////////////////////////////////


method Partition( a: multiset<int> )
    returns ( b: multiset<int>, p: int, c: multiset<int> )
  decreases |a|
  requires |a| > 0
  ensures b + multiset{p} + c == a
  ensures forall z | z in b :: z >= p
  ensures forall z | z in c :: z <= p
  // | >= p | p | =< p |
{
  // Forritið stofn fallsins.
  // Þið megið nota lykkju eða endurkvæmni.
  // Hjálparfallið RemoveOne verður væntanlega gagnlegt.

  // Program the body of the function.
```

```
  // You may use a loop or recursion.
  // The helper function RemoveOne may be useful.

  // Remove one value from a.
  var a', x := RemoveOne(a);

  // If a' is then empty, we're done and return x,
  // and the empty sets on both sides (a')
  if (a' == multiset{}) { return a', x, a'; }

  // If a' is not empty, then we recursively partition a'.
  var b', p', c' := Partition(a');

  // If x is less than or equal to the pivot from that partition,
  // we add x to the left partition and put p' as the new pivot.
  if (x <= p') { return b', p', c'+multiset{x}; }

  // If x is greater than the pivot from the partition,
  // we add x to the right partition.
  else { return b'+multiset{x}, p', c'; }
}

method Sort( m: multiset<int> ) returns ( r: seq<int> )
  decreases m
  ensures multiset(r) == m
  ensures IsSorted(r)
  // Bætið við requires/ensures/decreases eftir þörfum
  // Add requires/ensures/decreases as needed.
{
  // Forritið stofn fallsins.
  // Þið munið vilja nota endurkvæmni.
  // Hjálparsetningin LomutoLemma
  // verður væntanlega gagnleg.
  // Hugsanlega viljið þið einnig
  // nota hjálparsetninguna Singleton.

  // Program the body of the function.
  // You will want to use recursion.
  // The lemma LomutoLemma will be
  // useful. Perhaps you will also
  // want to use the lemma Singleton.

  // If the multiset is empty, we return the empty sequence.
  if (m == multiset{}) { return []; }

  // If the multiset is not empty, we partition it.
  var a, p, b := Partition(m);

  // We then recursively sort the right and left partitions.
  var aSorted := Sort(a);
  var bSorted := Sort(b);

  // We finally return the sorted sequences,
  // and add the pivot in between them.
  // Dafny needs this LomutoLemma to prove that the sorting is correct.
  LomutoLemma(a, aSorted, p, b, bSorted, aSorted + [p] + bSorted);
  return aSorted + [p] + bSorted;
}
```