TÖL212M Rökstudd Forritun - Hópverkefni 4

Andri Fannar Kristjánsson

9. febrúar 2025

Hópverkefni 4

1

Klárið að forrita Dafny skrána `H4-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/d86x2hwz. Ath. að ég fékk ekki `tio` til að keyra forritið vegna þess að það notar eldri útgáfu af `Dafny` sem styður ekki aðferðina sem gefur lengdina á `multiset`, |a|. 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/d86x2hwz

////////////////////////////////////////////////////////////
// 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 vaxandi röð.
// IsSorted(a) is true if and only if it is
// proven that a is in ascending order.
predicate IsSorted( a: seq<int> )
{
  forall p,q | 0 <= p < q < |a| :: a[p] <= a[q]
}

// IsSegmented(a,b) er satt þá og því aðeins að
// sannað sé að öll gildi í a séu <= öll gildi í b.
// IsSegmented(a,b) is true if and only if it is
// proven that all values in a are <= all values in b.
predicate IsSegmented( a: seq<int> , b: seq<int> )
{
  (forall z,w | z in a && w in b :: z <= w) &&
  (forall p,q | 0 <= p < |a| && 0 <= q < |b| :: a[p] <= b[q])
}

// SortedEquals(a,b) sannar, fyrir raðaðar runur
// a og b, sem innihalda sama poka gilda, að runurnar
// eru jafnar.
// SortedEquals(a,b) proves that for sorted squences
// a and b that contain the same multiset of values,
```

```
// that the sequences are equal.
lemma SortedEquals( a: seq<int>, b: seq<int> )
  requires multiset(a) == multiset(b)
  requires IsSorted(a)
  requires IsSorted(b)
  ensures a == b
{
  if a == []
  {
    assert |b| == 0 || b[0] in multiset(a);
    return;
  }
  if b == []
  {
    assert |a| == 0 || a[0] in multiset(b);
    return;
  }
  assert a[0] in multiset(b);
  assert b[0] in multiset(a);
  assert a == a[0..1]+a[1..];
  assert b == b[0..1]+b[1..];
  assert a[0] == b[0];
  assert multiset(a[1..]) == multiset(a)-multiset{a[0]};
  assert multiset(b[1..]) == multiset(b)-multiset{b[0]};
  SortedEquals(a[1..],b[1..]);
}

// Samröðunarfall sem nota má í röksemdafærslu
// en ekki í raunverulegum útreikningum nema við
// séum að nota nýlega útgáfu af Dafny.
// A merge function that can be used in reasoning
// but not for real computations unless we are
// using a recent version of Dafny.
function MergeFun( a: seq<int>, b: seq<int> ): seq<int>
  decreases |a|+|b|
{
  if a == [] then
    b
  else if b == [] then
    a
  else if a[0] <= b[0] then
    [a[0]]+MergeFun(a[1..],b)
  else
    [b[0]]+MergeFun(a,b[1..])
}

// Sannar að MergeFun(a,b) skilar réttu gildi.
// Fyrir mannlega lesendur er það augljóst,
// en Dafny þarf smá hjálp til að sanna það
// með þrepasönnun. Þið munuð vilja kalla á
// þessa hjálparsetningu ef þið byggið ykkar
// samröðun á endurkvæmni.
// Proves that MergeFun(a,b) returns the
// correct value. For human readers this is
// obvious but Dafny needs a little help to
// prove this using induction. You will wish
// to call this lemma if you base your merge
// on recursion.
```

```
lemma MergeFunWorks( a: seq<int>, b: seq<int>, c: seq<int> )
  decreases |a|+|b|
  requires IsSorted(a)
  requires IsSorted(b)
  requires c == MergeFun(a,b)
  ensures multiset(c) == multiset(a)+multiset(b)
  ensures IsSorted(c)
  ensures a!=[] && b!=[] && a[0]<=b[0] ==> c==a[0..1]+MergeFun(a[1..],b)
  ensures a!=[] && b!=[] && a[0]>=b[0] ==> c==b[0..1]+MergeFun(a,b[1..])
{
  if a == [] || b == [] { return; }
  if a[0] <= b[0]
  {
    MergeFunWorks(a[1..],b,c[1..]);
    calc ==
    {
      multiset(c);
      multiset(c[0..1]+c[1..]);
      multiset(c[0..1])+multiset(c[1..]);
      multiset(c[0..1])+multiset(a[1..])+multiset(b);
      multiset(a[0..1])+multiset(a[1..])+multiset(b);
      multiset(a[0..1]+a[1..])+multiset(b);
      assert a[0..1]+a[1..] == a;
      multiset(a)+multiset(b);
    }
  }
  else
  {
    MergeFunWorks(a,b[1..],c[1..]);
    calc ==
    {
      multiset(c);
      multiset(c[0..1]+c[1..]);
      multiset(c[0..1])+multiset(c[1..]);
      multiset(c[0..1])+multiset(a)+multiset(b[1..]);
      multiset(b[0..1])+multiset(a)+multiset(b[1..]);
      multiset(a)+multiset(b[0..1])+multiset(b[1..]);
      multiset(a)+multiset(b[0..1]+b[1..]);
      assert b[0..1]+b[1..] == b;
      multiset(a)+multiset(b);
    }
  }
}

// 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
// nudge 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)
{}

// Fjarlægir tvö gildi úr poka.  Getur verið gagnlegt
// í Split fallinu.
// Removes two values from a multiset. Can be useful
// in the Split function.
method RemoveTwo( a: multiset<int> )
    returns( b: multiset<int>, x: int, y: int )
  requires |a| >= 2
  ensures a == b+multiset{x,y}
{
  b := a;
  x :| x in b;
  b := b-multiset{x};
  assert |b| >= 1;
  y :| y in b;
  b := b-multiset{y};
}

// Prófunarfall sem staðfestir að Split og Sort
// séu áreiðanlega að virka sannanlega rétt.
// Alls ekki má breyta þessu falli.  Athugið að
// þetta fall skilgreinir í raun þá virkni sem
// Split og Sort eiga að hafa, þ.e. forskilyrði
// og eftirskilyrði þeirra falla.
// A test function that validates that Split and
// Sort are provably correct. This function must not
// be modified. Notice that this function does in
// fact define the functionality that Split and
// Sort should have, i.e. the preconditions and
// postcoditions of those functions.
method Test( x: multiset<int> )
{
  var a,b := Split(x);
  assert a+b == x;
  assert (|a|==|b|) || (|a|==|b|+1);
  a,b := Split(x);
  assert a+b == x;
  assert (|a|==|b|) || (|a|==|b|+1);
  var c := Sort(x);
  assert multiset(c) == x;
  assert IsSorted(c);
}

// 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
// implements a version of merge sort.
//////////////////////////////////////////////////////////////

// Þið munuð kannski vilja nota þetta samröðunarfall í Sort fallinu.
// Annars megið þið eyða þessu því í nýrri útgáfum Dafny má kalla
// á MergeFun í keyrsluhæfum forritstexta.
// You will maybe want to use this merge method in the Sort method.
// Otherwise you may delete this because in recent versions of
// Dafny you may call MergeFun in executable source code.
method Merge( a: seq<int>, b: seq<int> ) returns( c: seq<int> )
  decreases |a|+|b|
  requires IsSorted(a)
  requires IsSorted(b)
  ensures IsSorted(c)
  ensures multiset(a)+multiset(b) == multiset(c)
  ensures c == MergeFun(a,b)
{
  // Forritið stofn fyrir þetta fall.
  // Þið getið notað lykkju eða endurkvæmni.
  // Sé endurkvæmni notuð þarf e.t.v. að bæta
  // við 'decreases' klausu í haus fallsins.
  //
  // Athugið að þið munuð næstum áreiðanlega
  // þurfa að kalla á hjálparsetninguna
  // MergeFunWorks á viðeigandi stöðum í
  // stofni fallsins.
  //
  // Ef þið notið lykkju þá er hugsanlegt að
  // hjálparsetningin SortedEquals verði
  // gagnleg.
  //
  // Einfaldara er að nota endurkvæmni en að
  // nota lykkju.  Munið að kalla á hjálpar-
  // setningar á viðeigandi stöðum til að
  // Dafny geti sannreynt það ástand sem
  // búið er að skapa.

  // Program a body for this method.
  // You can use a loop or recursion.
  // If recursion is used you may need
  // to add a 'decreases' clause to the
  // header of the method.
  //
  // Note that you will almost certainly
  // need to call the lemma MergeFunWorks
  // at approriate places in the body.
  //
  // If you use a loop then the lemma
  // SortedEquals may be useful.
```

```
  //
  // It is simpler to use recursion than
  // a loop. Remember to call lemmas at
  // appropriate places to help Dafny
  // verify the state that has been
  // achieved.

  // Check if either array is empty, and return the other if so.
  if ( a == [] ) { return b; }
  if ( b == [] ) { return a; }

  if ( a[0] <= b[0] )
  {
    // If a[0] is less than or equal to b[0],
    // then we add a[0] to the front and recursively call Merge
    // with the rest of a, and b.
    // We also need to remind Dafny that the result of this call
    // is correct by calling MergeFunWorks.
    var c' := Merge(a[1..],b);
    MergeFunWorks(a,b,MergeFun(a,b));
    return [a[0]]+c';
  }
  else
  {
    // If a[0] is greater than b[0],
    // then we add b[0] to the front and recursively call Merge
    // with a, and the rest of b.
    // We also need to remind Dafny that the result of this call
    // is correct by calling MergeFunWorks.
    var c' := Merge(a,b[1..]);
    MergeFunWorks(a,b,MergeFun(a,b));
    return [b[0]]+c';
  }

  // Ráðlegt er að láta þessi tvö köll á
  // hjálparsetningar vera það síðasta sem gerist
  // í fallinu, sérstaklega ef þið notið lykkju.
  // Ef þið notið lykkju er einfaldara að ferðast
  // gegnum a og b frá vinstri til hægri.

  // It is advisable to let the two following
  // calls on lemmas be the last thing that
  // happens in the body, especially if you
  // use a loop. If you use a loop it is
  // simpler to traverse a and b from left to
  // right.
  MergeFunWorks(a,b,MergeFun(a,b));
  SortedEquals(c,MergeFun(a,b));
}

// Skiptir innihaldi poka í tvennt þannig að pokarnir
// sem koma út eru nokkurn veginn jafn stórir.
// Split the contents of a multiset in two wuch that
// the resulting multisets are approximately of equal
// size.
method Split( a: multiset<int> )
  returns ( b: multiset<int>
          , c: multiset<int>
```

```
  )
  // Bætið við requires/ensures/decreases eftir þörfum
  // Add requires/ensures as needed.
  decreases |a|
  ensures b + c == a
  ensures ( |b| == |c| ) || ( |b| == |c|+1 )
{
  // Forritið stofn fyrir þetta fall.
  // Þið getið notað lykkju eða endurkvæmni.
  // Sé endurkvæmni notuð þarf e.t.v. að bæta
  // við 'decreases' klausu í haus fallsins.
  //
  // Fallið RemoveTwo er gagnlegt hér.

  // Program a body for this method.
  // You can use a loop or recursion.
  // If recursion is used you may need
  // to add a 'decreases' clause to the
  // header of the method.
  //
  // The method RemoveTwo is useful here.

  // If the size of a is less than or equal to 1,
  // then we return a (which might be empty), and the empty multiset.
  if ( |a| <= 1 ) { return a, multiset{}; }

  // If the size of a is greater than 1, then we remove two values from a.
  var a', x, y := RemoveTwo(a);

  // We then recursively split a'.
  b, c := Split(a');

  // We then return the two values we removed from a,
  // and the two multisets we got from the recursive call.
  return b + multiset{x}, c + multiset{y};
}

// Raðar innihaldi poka yfir í runu með mergesort.
method Sort( a: multiset<int> ) returns ( b: seq<int> )
  // Bætið við requires/ensures/decreases eftir þörfum
  // Add requires/ensures/decreases as needed.
  decreases a
  ensures multiset(b) == a
  ensures IsSorted(b)
{
  // Forritið stofn fyrir þetta fall.
  // Eðlilegt er að nota endurkvæmni hér.
  // Program  a body for this method.
  // It is natural to use recursion here.

  // We have to check the two base cases, as a won't
  // decrease if there is one element remaining.
  if ( |a| == 0 ) { return []; }
  if ( |a| == 1 ) {
    // Take the last remaining element from a, and remind Dafny that an
    // array of the singleton is the same as a multiset of the singleton.
    var x :| x in a;
    Singleton(a,[x],x);
```

```
    return [x];
  }

  // Split the multiset a into two multisets of equal (or near equal) size.
  var c, d := Split(a);

  // Sort the two multisets recursively.
  var c' := Sort(c);
  var d' := Sort(d);

  // Merge the two sorted multisets.
  b := Merge(c',d');
  return;
}
```