REPORT

CAB402: Programming Paradigm

Le Minh Tuan Phan|n9590285| 23rd April 2018

Contents

Efficient and effectiveness

For the effectiveness of the 3 implementations, all approaches produce the same outputs for N =5, which are identical to the sample output image. However, there are differences between the execution time of the implementations.

The C# Implementation proved to be the most efficient paradigm to implement the algorithm in run-time, with an approximate time of 4 seconds for N = 5 and about 1 minute for N =6. This was significantly faster than the pure version with a recorded time of 5 minutes and 25 seconds for N =5 and 147 minutes for N=6. OOP treats everything as objects and most of the values are retrieved as references, rather than literal values, which operates much faster than the functional paradigm process. On the other hand, the pure implementation carries values constantly, which consequently consumes high amounts of memory, resulting in a slower result.

The execution time of the impure approach is about 4 minutes and 40 seconds for N =5, which is 10-15% faster compared to the pure version.

Approaches experience

Considering the three implementations, pure version was the easiest to program. However, at the beginning, the F# syntax and the usage of high-order functions are quite overwhelming for starters. The stddev function got many different steps and was one of the hardest function, especially for people who haven't done F# before. The functions in Segmentation module are really helpful for students to get the knowledge of high-order functions. Everything is straightforward in the pure implementation after gaining the knowledge about high-order functions and F# list, set.

The C# approach of the project is not really hard, but it requires the student to start from scratch. The C# version also doesn't have the discriminated union, which forces the user to use interface and inheritance to implement the Segment. Another inconvenience in C# is that 2 objects with the same value are not literally the same. Therefore, to compare Segments in C#, references must be used. The code for the C# is quite longer, but most of the code can be reused. In conclusion, it is not hard to implement C# version, but it is really time-consuming.

The impure version of F# is probably the hardest of the 3. Different approaches have been tried to reduce the run time, but only 1 of them working. Using Dictionary, HashSet or List<T> , which are mutable collections, increase the run time. The approach that works is the one which change the PixelMap from creating a new segment every time to compare with the segments from a mutable list and get the references. The implementation reduces the run time by 10-15% compared to the pure version. Different approaches can be found in the testing ConsoleApp (save different versions of Segmentation module, cannot be compiled.) The failed versions are also included in the submission.

Readability and maintainability

Overall, Both F# versions are easier to read compared to the C# version. F# deals with values instead of objects so it is more straightforward and the user can notice the errors immediately. The pipeline operators also help to make F# more readable, which let the users know the order of applied functions. When maintaining and adding new methods, C# seems to be more superior. It is easier to add more functions to the current implementation in C# compared to F#. However, C# version may face more logic problems compared to the F#, if the user doesn't notice the change of the data.

Result conciseness

Both F# versions are more concise compared to the C# version, because of the elements in F# versions are heavily value-relied. In the C# one, the developer has to create different classes and objects, but it is easier in the F# version. F# inbuilt function such as List.map , List. Fold are shorter and more concise than using loops in other versions. The F# implementation is significantly shorter compared to the C# version. However, in some cases, it is hard to keep track of the indentation in F# and might make some fatal errors. C# notify the user where is the end of the method by using return, but it is quite hard to determine in F#.

Code Efficient

All approaches have their own advantages. For F#, it is easier to compare two items of the same type because they relied heavily on the values. So, when the program compares two items, it is comparing their values. However, in C#, when the program compares a and b, it is checking if object a and b has the same reference. When working with F#, it is working with values so it might be more straightforward. On the other hand, C# code is more familiar and easier to be reused compared to the F# versions. It is also easier to add more functions to the object in C# compared to the F#. Therefore, although F# codes are shorter and more concise, they are not as efficient as the C#.

Appendix

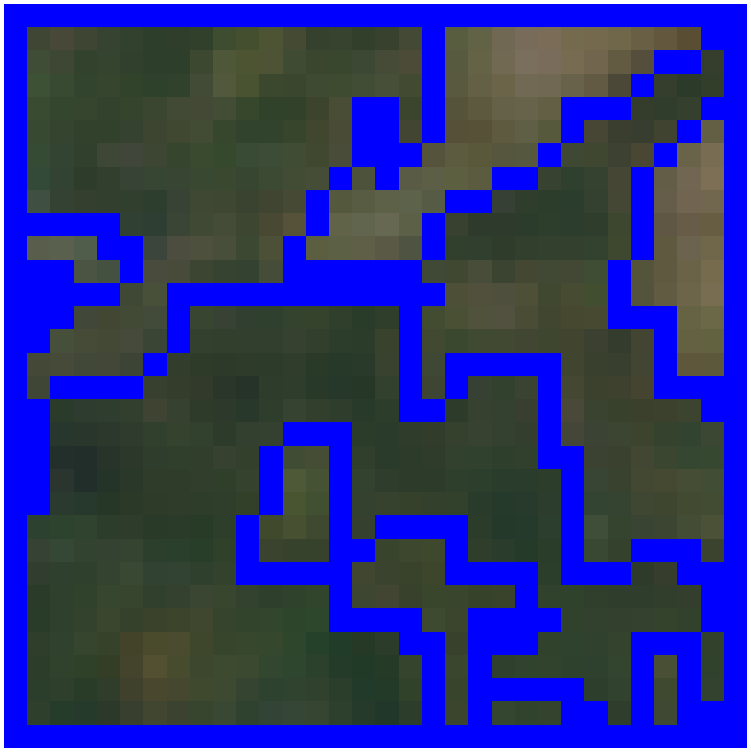

Figure 1: C# output of n=5, run time: 4 seconds

Figure 2: F# pure output of N=5, run time: 5mins 15 seconds

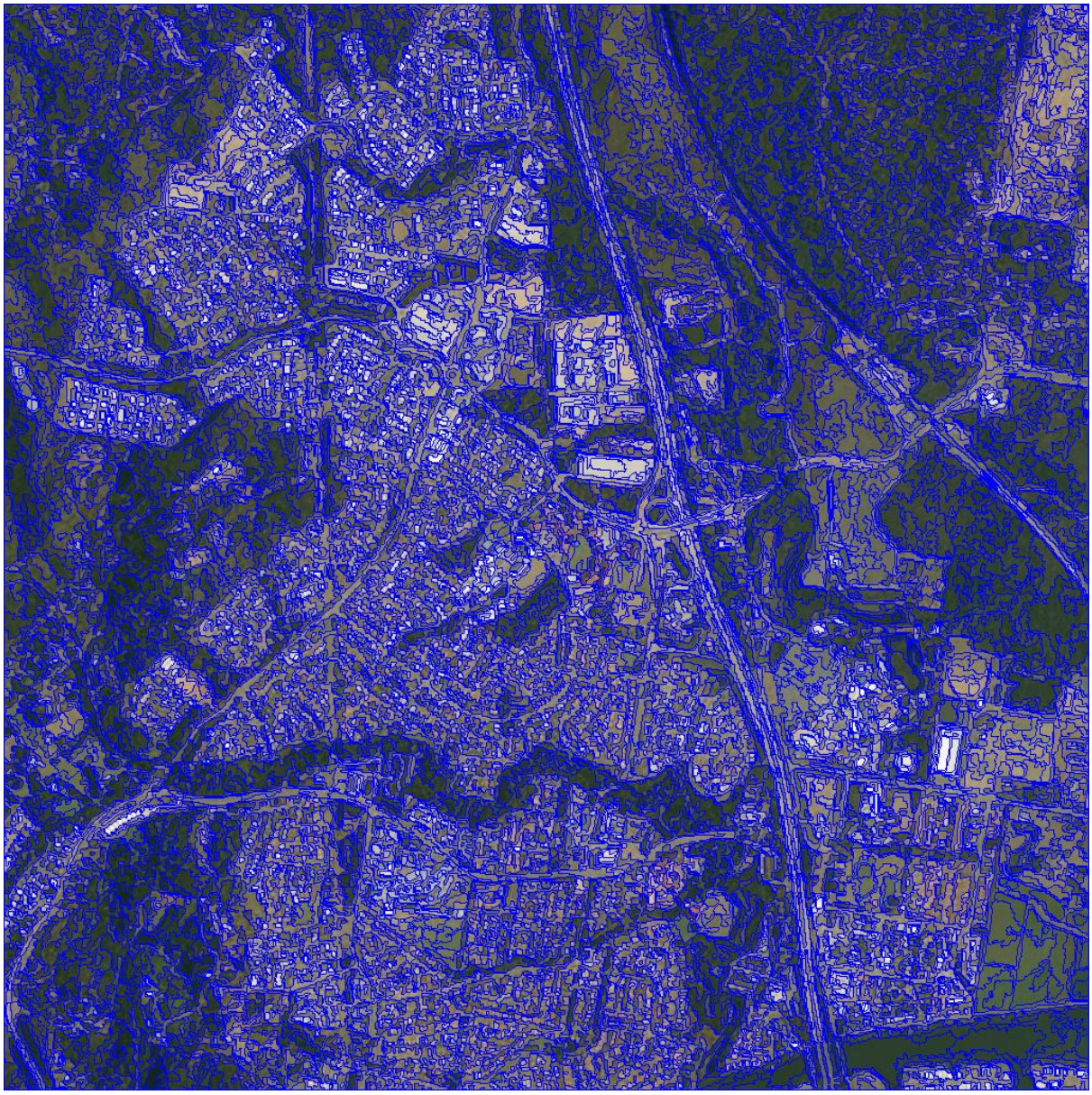

Figure 3: C# output of n=10, run time 8 hours

```
// Impure Implementation, i
let createPixelsList (image:TiffModule.Image) (size:int) =
    let imageSize = pown 2 size
    let listOfPixels = new List<Segment>()

    for x= 0 to  (imageSize-1)   do
        for y =0 to (imageSize-1) do
            listOfPixels.Add( Pixel( (x,y) , TiffModule.getColourBands image (x,y) )) |> ignore
    listOfPixels



// Find the largest/top level segment that the given segment is a part of (based on the current segmentation)
let rec findRoot (segmentation: Segmentation) segment : Segment =
    match segmentation.TryFind segment with
    | None -> segment
    | Some value -> findRoot segmentation value

// Initially, every pixel/coordinate in the image is a separate Segment
// Note: this is a higher order function which given an image,
// returns a function which maps each coordinate to its corresponding (initial) Segment (of kind Pixel)
let createPixelMap (N:int) (pixelsList : List<Segment>) : (Coordinate -> Segment) =
    let size = (pown 2 N)
    let pixelMap ((x,y) : Coordinate)  =
        let currentPosition = size * x + y
        pixelsList.[ currentPosition ]
    pixelMap
```

Figure 4: difference in impure implementation

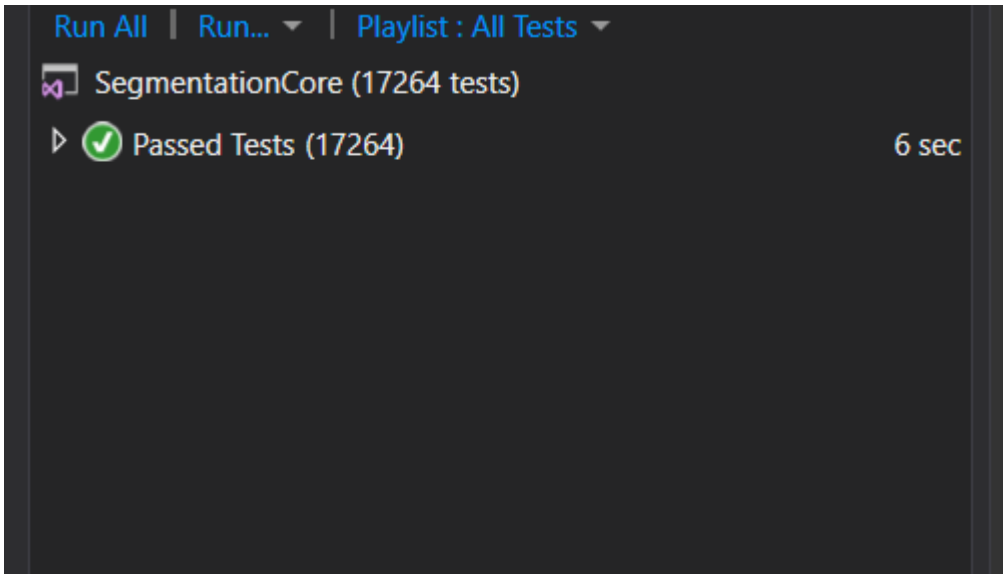

Figure 5: All the tests passed for pure version