UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Incremental Algorithm for large Netwotks

Final Report
Mid Term

Pol Forner Gomez

Thesis supervisor: Gerard Escudero Bakx (Department of Computer Science)
Thesis co-supervisor: Edelmira Pasarella Sanchez (Department of Computer Science)
GEP tutor: Joan Sardà Ferrer
Degree: Bachelor's Degree in Informatics Engineering (Computing)
February, 2024

Contents

Updates

1.1 Updates respect initial planning

1.1.1 About tasks and time planification

Changes

Regarding the initial planning of the project, there have not been very big changes. The main change has been that I have needed more investment of time in understanding the library and in coding the word counting problem. This is because I have found the need to make several adaptations to the library to allow more options and be able to implement my initial problem. Everything and so I have only needed 2 extra week in the overall time of the **RL** and **FA** sections.

In addition to this, I have been able to specify with Edelmira what improvements I can make to the library. This process was also planned for later, so it also changes the planning a bit.

Consequences

All these changes in planning mean that there is less time left to perform the following tasks (SA, D3, D4). But at the same time I have invested more time getting to know the library and programming, so I foresee that it will take me less time to improvements to the library that Edelmira proposes.

New planning

With this updates, the new planning is as follows:

Description	**TAG**	**Hours**	**Previous Tasks**	**Requirements**	**Human Resource**
Project definition and planning	**G**	**75**	**-**	**-**	**-**
Context and Scope	G1	20	Laptop	None	None
Planning	G2	12	G1	Laptop	GEP Tutor
Budget and Sustainability	G3	18	G2	Laptop	GEP Tutor
Final Document	G4	25	G3	Laptop	GEP Tutor
Research and Learning	**RL**	~~75~~ 95	**G**	**-**	**-**
Haskell Refresh	RL1	20	None	Laptop, Books	Supervisor
TFM assimilation	RL2	~~25~~ 15	None	Laptop	Supervisors
Dynamic Pipeline Framework	RL3	~~30~~ 60	RL1, RL2	Laptop	Juan Pablo
First Algorithm Development	**FA**	~~80~~ **100**	**RL**	**-**	**-**
Algortim Scaffold	FA1	20	None	None	Supervisors
Algorith Implementation	FA2	~~50~~ 70	FA1	Laptop	Supervisor
Algorithm Testing	FA3	10	FA2	Laptop	None
Second Algorithm Development	**SA**	~~130~~ **90**	**FA**	**-**	**-**
Finding Weak Points	SA1	~~15~~ 10	None	Laptop	Co-supervisor
Improvement Scaffolds	SA2	~~15~~ 10	SA1	None	None
Algorith Implementation	SA3	~~80~~ 50	SA2	Laptop	Supervisor
Algorithm Testing	SA4	20	SA3	Laptop	None
Documentation	**D**	**100**	**RL,FA,SA,G**	**-**	**-**
Documentation of RL section	D1	10	RL	Laptop	None
Documentation of FA section	D2	20	FA	Laptop	None
Documentation of SA section	D3	20	SA	Laptop	None
Final Documentation	D4	50	D1, D2, D3	Laptop	Supervisors
Total (G + RL + FA + SA + D): 460 hours					

Table 1.1: Summary of Project Planning, self elaborated

1.1.2 About scope

Changes

Since initially planning the scope of my work, I have had to modify it as I have progressed. The reason is that when working with the Haskell library, I have encountered some lack of functionality at some points. I have found that Juan Pablo made a very complete library but a little oriented to his work. That's why when trying to implement the word counting problem, I have had to expand and create new functionalities.

Consequences

No several consequences have been derived from this change. Just that I have had to invest more time in understanding the library, as reflected in the previous section.

1.1.3 About budget

Changes

Because the scope and time planification changes, budged has also changed. The human resource budged needs to be recalculated tacking into account the new time planification. The other sections of the budget like hardware, software and unexpected const remain the same.

Consequences

As we can apreciete in the table below, the new human resource budget is less that 0,5% higher than the initial budget, so the changes are not significant.

New Budget

Role	Avg. Salary (€/h)	Hours	Total (€)	Total with Social Security (€)
Researcher	15	~~85~~ 75 hours	~~1275~~ 1125	~~1657.5~~ 1462.5
Haskell Developer	18	~~200~~ 210 hours	~~3600~~ 3780	~~4680~~ 4914
Project Manager	20	175 hours	3500	4550
Total	~~18.2~~ 18.3	460	~~8375~~ 8405	~~10887.5~~ 10926.5

Table 1.2: New Human Resources Cost, self elaborated

Source	Cost (€)
Hardware	55,29
Software	0
Human Resources	~~10887.5~~ 10926.5
Unexpected	2037
Total	~~12979.79~~ **13018.79**

Table 1.3: New total budget, self elaborated

1.2 Laws and regulations

Consulting the BOE, law 14/2011, updated on 01/11/2023, article 15 (page 28) tells us about various obligations of the researcher and here I will comment on those that involve me **empty citation**

Avoid plagiarism and misappropriation of authorship of scientific works or third-party technologies

On this point, I have at all times cited other people's resources and whenever I have used a third-party technological resource I have made sure that it is open source and that its necessary citations are made.

Inform the entities for which it provides services of all the findings, discoveries and results susceptible to legal protection, and collaborate in the processes of protection and transfer of the results of their investigations

At all times I have been in contact with my director and have been reporting on my work.

Disseminate the results of your investigations, if applicable, as indicated in this law, so that the results are used through communication and transfer to other research, social or technological contexts, and if applicable, for their marketing and valuation. In particular, the research staff must ensure and take the initiative so that its results generate social value

This work will be public once finished and everyone will be able to take advantage of the discoveries.

Ensure that your work is relevant to society

This work can help expand resources on dynamic pipeline and Haskell code.

Use the name of the entities for which you provide services in the realization of its scientific activity, in accordance with the internal regulations of said entities and the agreements, pacts and conventions that they subscribe

On the cover of this work you will find at all times the name of the entity (UPC) and the faculty (FIB) as well as their logos.

Adopt the necessary measures to comply with the applicable regulations in matters of data protection and confidentiality

At all times, we are ensuring compliance with data protection, always using data that is free to use and that complies with the laws.

How to use the Dynamic Pipeline Haskell Library

2.1 Objective

In this section it will be explined the steps to given a dynamic pipeline algorithm, how to implement it using the Haskell library. First of all it is necessary to have in mind the structure of the pipeline and the logic of the algorithm. For better explanation, I will be using a toy problem to illustrate the steps. I will be using, as said previously, the word counting problem: given a list of words (like a text file or a book), count the number of times each word appears in the list.

2.2 Preliminaries

First of all, it is necessary to design the pipeline structure and our algorithm logic. For all dynamic pipeline, we need to define the following concepts:

- **Source:** Here it is defined how the data will be read and how it will be sent to the pipeline.
- **Generator:** Here it is defined when a filter is generated and which data processes.
- **Sink:** Here it is defined how the data will be received from the pipeline and how it will be written.

Now, let's try to define the structure of the pipeline for the word counting problem. Our input is a chain of caracters that form words and we want to output all unique words folowed by the number of times they appear in the input.

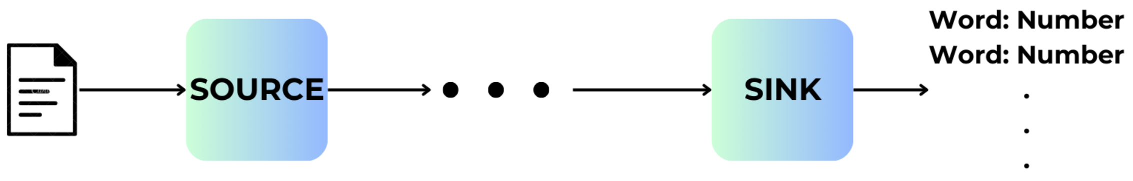

Figure 2.1: First Dynamic Pipeline scheme

Now we need to define whichs data types the pipeline will use. As we are reading caracters and we are procecing words we are going to use the standard Haskell types: Char and [Char] for characters and words, respectively. Also we need to keep track of the number of times each word appears, so we will be using a tuple ([Char], Int) to represent the word and the number of times it appears. So the source needs to read the data, get the words and send them to the pipeline as a list of tuples.

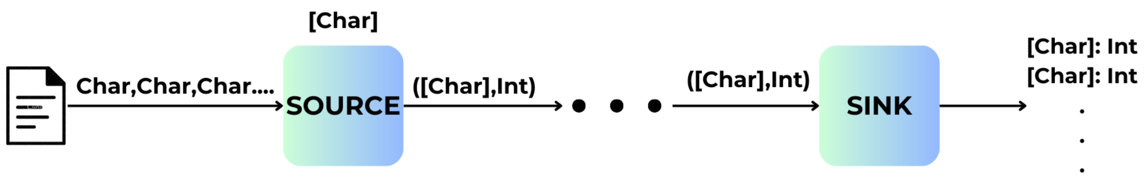

Figure 2.2: Dynamic Pipeline scheme with data types

At this point we can start to design the logic of the algorithm. In our initial state we have just the source, the sink and the generator, so the first thing that we need to decide is, for every data that the generator reads, when it will let the data pass to the sink. This is because in front of the generator there will be a series of filters processing the data. Perhaps sometimes it will receive data directly from the input, either modified data or data emitted by filters. Looking our problem, we see that we do not want to let pass all the data, but we do want to let it pass sometimes. As we explained at the beginning of this work, the data can be infinite and we want to have a mechanism to be able to print responses throughout the execution. That is why we have to be able to differentiate between when it is a ”normal” data, so we should not let it pass and when it is a response, in order to pass it. The solution that I have decided to use is based on the fact that for each word we have its word counter, therefore, when a data comes that is a response, the counter will be at least 1. This allows us make the source to initialize the tuple with ([Char],0), so when the generator receives a tuple with this shape it knows that it is input data.

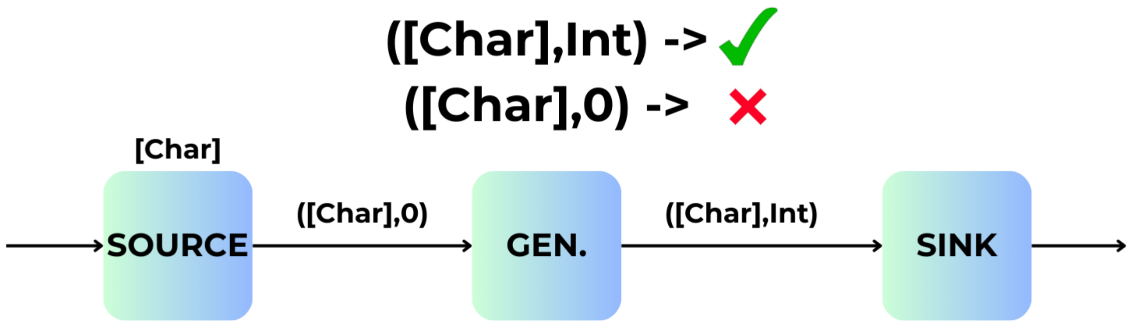

Figure 2.3: Dynamic Pipeline scheme with generator logic for letting pass the words

The next step is decide when the generator will emit a filter. But before that, we need to decide more or less how the filter will be. Filters acts similar to the others stages: it receives data, it can process it and it can emit data. A characteristic of each filter is that they have a interal parameter which can be used for the process. So, with all this in mind, we can make each filter responsible for counting a single word, so that if it receives a word that is not its own, it simply lets it pass. On the other hand, if it receives a word that matches its parameter, it will add it to its count and will not let it continue passing. With this filter scheme we can get each one to be in charge of a different word, so we would need to generate a filter for each unique word.

Now we can define when the generator will emit a filter. The generator will emit a filter each time it recives a new word, and because once a filter is generated for a word, it will not allow more of that word to pass. All the data (other than results) that the generator reads will be new words and a filter must be generated. In other words, if the generator recives a tuple with ([Char],0) it will generate a filter.

We have all set except for for few details. We need to define how the generator will initialize the filter parameter. Since the parameter needs to know which word it controls and how many times it appears, it seems logical to use the ([Char],Int) tuple again. So the generator will initialize the filter parameter with the tuple that it received from the source, but changing the counter to 1 (because it is the first time that the word appears). Last but not least, we have to define how to indicate to the pipeline that we want to receive a response in the output (for the incremental behavior). To do this, what we can do is try to introduce data into the stream that indicates that we should return the response. We can use a piece of data which does not represent any word, for example the character ".". In this way, every time a filter receives a tuple (".",Int), it must pass through the pipeline its parameter (which will contain the count) and also the point, so that the rest of the filters repeat the process.

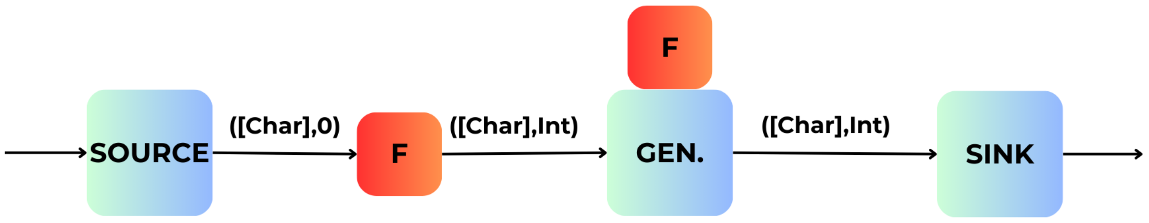

Figure 2.4: Example of a Dynamic Pipeline with one filter generated

2.3 Example

Lets make a quick example to illustrate all the concepts explained above. We will use the following input: ["cat","dog","cat","bird",".","dog","."] This is the initial state of the pipeline:

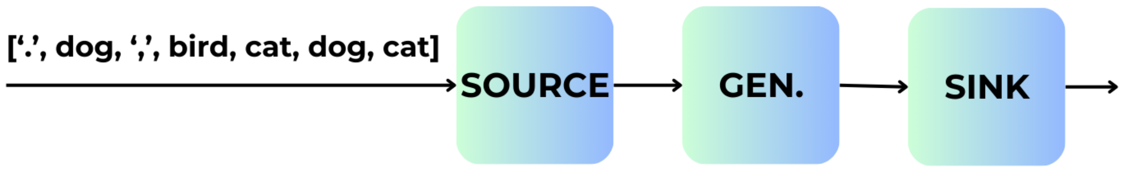

Figure 2.5: Initial state of the Dynamic Pipeline

The first word that will be processed is "cat". Here is the state of the pipeline after the first word is processed:

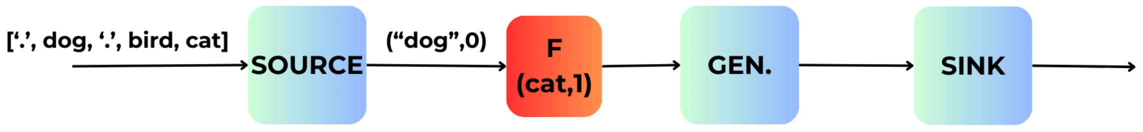

Figure 2.6: Dynamic Pipeline state after the process of the first word

If we keep processing the input before we reach the first ".", we will have the following state:

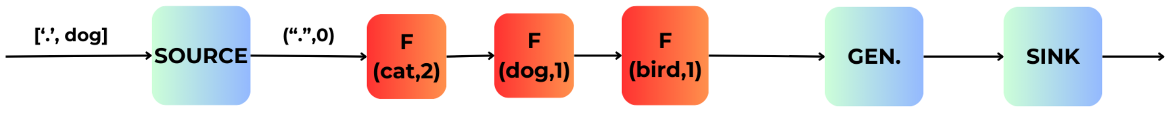

Figure 2.7: Dynamic Pipeline state after reading ["cat","dog","cat","bird"]

After reading the ".", the generator will let pass the tuples that the counter is not 0 leaving us with:

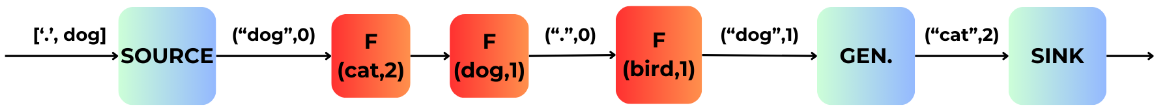

Figure 2.8: Dynamic Pipeline state after reading the first "."

The generator readed the tuple ("cat",2) and let it pass to the sink (without generating a filter, because the counter is not 0). If we finish all the input we will have the following final state:

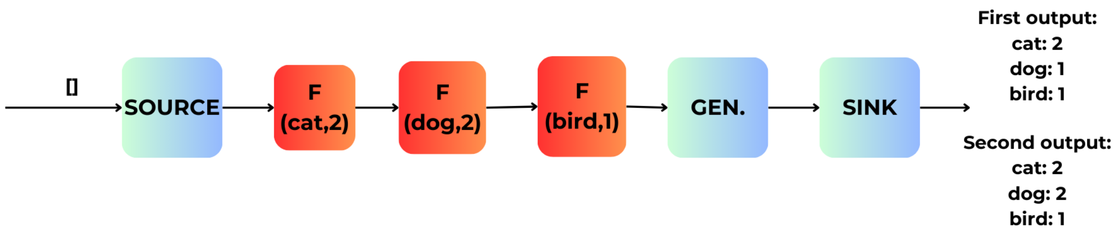

Figure 2.9: Final state of the Dynamic Pipeline

2.4 Implementation

Now we designed the pipeline structure and the logic of the algorithm, we can start to implement it using the Haskell library.

2.4.1 Defining the functions

Dynamic pipeline data type

If we check the library, almost all functions make use of a dpDefinition, which is a type that we define representing the structure of the pipeline. The tools for defining it provided by the library are the following:

```
data Sink
data Generator (a :: Type), a ~ Channel
data Source (a :: Type), a ~ Channel
data Channel (a :: Type), a ~ (Type :<+> Type :<+> ... :<+> Eof)
data DynamicPipeline dpDefinition  filterState  filterParam  st,
    dpDefinition  ~ Source (Channel ..)  :=> Generator (Channel ..) :=> Sink
data a :=> b, a & b ~ Stage
```

Figure 2.10: Data types provided by the library

We can see that it provides the data type for all the stages, so lets apply the structure that we designed to the library.

```
type DP = Source (Channel (([Char],Int) :<+> Eof)) :=> Generator (Channel (([Char],Int):<+> Eof)) :=> Sink
```

Figure 2.11: Dynamic pipeline type definition

As said before, we define the data types that will flow through the pipeline and asign them to the stages.

Building the DP

The next step is building the dynamic pipeline:

```
mkDP :: forall dpDefinition filterState st . . .
=> Stage (WithSource dpDefinition (DP st))
-> GeneratorStage dpDefinition filterState filterParam st
-> Stage (WithSink dpDefinition (DP st))
-> DP st ()
```

Figure 2.12: mkDP function definition

This functions is smart constructor for the dynamic pipeline. If we translate this definition into words, we can see that it receives the definition and the stages of the pipeline and returns the dynamic pipeline. So we need to define each stage of the pipeline and then use this function to build the dynamic pipeline.

```
dp' :: DP s ()
dp' = mkDP @DPExample source' generator' sink'
```

Figure 2.13: Definition of custom function dp'

Building the Source

The first stage that we are going to define is the source. The library also provide us with a combinator to build a source stage:

```
withSource :: forall (dpDefinition :: Type) st. WithSource dpDefinition (DP st)
-> Stage (WithSource dpDefinition (DP st))
```

Figure 2.14: Source stage combinator

Explined into words, we just need to provide the pipeline definition and a function reads some data and populates de pipeline

```
source' :: Stage (WriteChannel (ByteString,Int) -> DP s ())
source' = withSource @DPExample populateSource'

dp' :: DP s ()
dp' = mkDP @DPExample source' generator' sink'
```

Figure 2.15: source' function added to definitions

Building the Sink

Next stage is the sink, as is more simple. The library provides us with a combinator to build a sink stage:

```
withSinkSource :: forall (dpDefinition :: Type) st. WithSink dpDefinition (DP st)
-> Stage (WithSink dpDefinition (DP st))
```

Figure 2.16: Sink stage combinator

Similar to the source, we just need to provide the pipeline definition and a function that reads data form the pipeline and outputs it.

```
source' :: Stage (WriteChannel (ByteString,Int) -> DP s ())
source' = withSource @DP populateSource'

sink' :: Stage (ReadChannel (ByteString,Int) -> DP s ())
sink' = withSink @DP outputSink'

dp' :: DP s ()
dp' = mkDP @DPExample source' generator' sink'
```

Figure 2.17: sink' function added to definitions

Building the Generator

The generator is the most complex stage to build, because we need to define the filters and define the logic of the generator. The library provides us with both a smart constructor and a combinator to build the generator stage:

```
mkGeneratorSource :: Stage (WithGenerator dpDefinition ( Filter dpDefinition filterState filterParam st) (DP st))
-> Filter dpDefinition filterState filterParam st
-> GeneratorStage dpDefinition filterState filterParam st

withGeneratorSource :: forall (dpDefinition :: Type) (filter :: Type) st. WithGenerator dpDefinition filter (DP st)
-> Stage (WithGenerator dpDefinition filter (DP st))
```

Figure 2.18: Generator smart constructor and combinator

Seems dificult to understand, but combining this 2 functions with the pipeline definition, the generator logic and a filter template we can build the generator stage. Here is the structure to follow to build the generator stage:

```
source' :: Stage (WriteChannel (ByteString,Int) -> DP s ())
source' = withSource @DP populateSource'

generator' :: GeneratorStage DPExample (ByteString,Int) (ByteString,Int) s
generator' = mkGenerator withGenerator @DP genLogic'

sink' :: Stage (ReadChannel (ByteString,Int) -> DP s ())
sink' = withSink @DP outputSink'

dp' :: DP s ()
dp' = mkDP @DPExample source' generator' sink'
```

Figure 2.19: generator' function added to definitions

Summary

We have defined all stages of the pipeline and all the functions needed to build the dynamic pipeline.

2.4.2 Implementinc functions

We have all defined so we need to implement the diferent functions acording to what we defined previously.

Source stage

For the source we need to define the populateSource' function. This function will take a WriteChannel and will write the data to the pipeline. The library provides us with some functions to do this:

```
unfoldMSource :: forall m a b. MonadIO m
=> m a
-> (a -> b)
-> m Bool
-> WriteChannel b
-> m ()

unfoldFileSource :: MonadIO m
=> FilePath
-> WriteChannel b
-> (ByteString -> b)
-> m ()

unfoldT :: (MonadIO m, Foldable t)
=> t a
-> WriteChannel b
-> (a -> b)
-> m ()
```

Figure 2.20: Functions to populate a WriteChannel

This 3 functions populates a WriteChannel taking data from diferent inputs. For this example lets take the unfoldT function, because allow us to use Foldable (like lists or sets) as input.

```
populateSource' :: WriteChannel b -> m ()
populateSource' c = unfoldT input c convert
```

Figure 2.21: populateSource' definition

We just need to provide de input (that can be whatever Foldable) and the convert function, that is a function that convert the data type from the input to the WriteChannel. Remembering the data types that we defined for the pipeline, we need to convert the input [Char] to a ([Char],0) tuple. So all defined, the populateSource' function will be:

input:: [Char

```
input = [ . . .]

convert :: Char -> ([Char],Int)
convert w = (w,0)

populateSource' :: WriteChannel b -> m ()
populateSource' channel = unfoldT input channel convert
```

Figure 2.22: populateSource' definition with input and convert functions

Sink stage

For the sink we need to define the outputSink' function. This function will take a ReadChannel and will output the data. Similar to source stage, the library provides us with some functions to do this:

```
foldM_ :: MonadIO m
=> ReadChannel a
-> (a -> m ())
-> m ()
```

Figure 2.23: Functions to read a ReadChannel

Now we just have one function, so lets use it:

```haskell
outputSink' :: ReadChannel b -> m ()
outputSink' c = foldM_ c transform
```

Figure 2.24: outputSink' definition

Similiar to the source stage, we need to define the transform function that will convert the data from the ReadChannel to the output. For this example, we will just print out the data to the console.

```haskell
transform :: ([Char],Int) -> m ()
transform (w,c) = putStrLn $ w ++ ": " ++ show c

outputSink' :: ReadChannel b -> m ()
outputSink' c = foldM_ c transform
```

Figure 2.25: outputSink' definition with transform function

Generator stage

For this stage we do a

Bibliography

[1] Juan Pablo Royo Sales, “Incremental algorithm for large netwotks,” 2021. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/361615/163090.pdf?sequence=1.

[2] A. M. Sharp, *Incremental algorithms: solving problems in a changing world*. Citeseer, 2007.

[3] “What is scrum? — scrum.org.” (), [Online]. Available: https://www.scrum.org/resources/what-scrum-module (visited on 02/27/2024).

[4] “Git.” (), [Online]. Available: https://git-scm.com/ (visited on 02/27/2024).

[5] “Haskell.” (), [Online]. Available: https://www.cs.upc.edu/~jpetit/Haskell/#1 (visited on 03/04/2024).

[6] J. P. R. Sales, *Jproyo/dynamic-pipeline*, original-date: 2021-03-27T11:12:27Z, Jul. 21, 2021. [Online]. Available: https://github.com/jproyo/dynamic-pipeline (visited on 03/04/2024).

[7] Pol, *Polforner/TFG*, original-date: 2024-03-04T09:51:17Z, Mar. 4, 2024. [Online]. Available: https://github.com/polforner/TFG (visited on 03/04/2024).

[8] A. Herrero, “FACULTAT INFORMATICA DE BARCELONA (FIB) UNIVERSITAT,”

[9] “What is the average lifespan of a computer?” (), [Online]. Available: https://www.hp.com/in-en/shop/tech-takes/post/average-computer-lifespan (visited on 03/11/2024).

[10] “Salary: Researcher in spain 2024,” Glassdoor. (Mar. 10, 2024), [Online]. Available: https://www.glassdoor.com/Salaries/spain-researcher-salary-SRCH_IL.0,5_IN219_KO6,16.htm (visited on 03/11/2024).

[11] “Salary: Project manager in madrid, spain 2024,” Glassdoor. (Mar. 11, 2024), [Online]. Available: https://www.glassdoor.com/Salaries/madrid-project-manager-salary-SRCH_IL.0,6_IM1030_KO7,22.htm (visited on 03/11/2024).

[12] “Salary: Software developer in spain 2024,” Glassdoor. (Mar. 1, 2024), [Online]. Available: https://www.glassdoor.com/Salaries/spain-software-developer-salary-SRCH_IL.0,5_IN219_KO6,24.htm (visited on 03/11/2024).