Programmeringsprojekt: Fase 2

Hlynur Æ. Guðmundsson, Mathias B. Jensen & Valdemar B. Reib

Fasekoordinator - Mathias B. Jensen

Institut for Matematik og Datalogi, Syddansk Universitet

2024-11-29

Indholdsfortegnelse

1 Problemstilling

I denne rapport behandler vi fase 2 af eksamensprojektet i kursuset DM574: Introduktion til programmering. I fase 2 bygges der videre på den første inplementering af et komparatornetværk, hvor vi udvider det således at der kan konstrukteres et sortingsnetværk på n-kanaler, som kan benyttes ved hjælp af en CLI (*Command Line Interface*).

2 Introduktion

I dette projekt er problemstillingen tredelt i faser, som giver anlæg til systematisk problemløsning. I anden fase har vi som gruppe fået til opgave at udvikle de grundlæggende moduler, som varetager dannelsen af sorteringsnetværk på n antal kanaler. Der skal til hvert modul laves et antal af funktioner, således at det færdige produkt af fase 2, kan danne et sorteringsnetværk til et bestemt antal kanaler.

3 Fremgangsmåde og Strategi

Den største ændring i vores fremgangsmåde sammenlignet med fase 1, er at der i højere grad benyttes redskaberne fra funktionelt programming og i nogen grad rekursiv programming. Efter at have fået feedback fra fase 1, opdagede vi at der var en kritisk misforståelse af komparatornetværkets væremåde. Vi lavede funktioner og moduler der laver bivirkninger og reviderer inputværdier, som de ikke skal, da det er vigtigt for projektet som helhed, at vores program gemmer og husker de oprindelige inputværdier, så de blandt andet kan genbruges flere gange. Metoderne fra funktionelt og rekursiv programming har den fordel, at de som udgangspunkt ikke vil revidere inputværdier og skabe data, der ikke er den samme hukommelse som inputværdien. Til gengæld kan programmer lavet med funktionelt og især rekursivt programming være hukommelsemæssigt intense. Da målet i fase 2 er at implementere sorteringsnetværker og ikke optimere dem til perfektion, så er ulempen ikke lige så betydningsfuld som i andre sammenhænge. I de tilfælde hvor vi ikke har brugt funktionelt eller rekursiv programming, har vi sat fokus på, at vores program laver kopier af inputværdier og behandler dem i stedet for, at være sikre på, at vores inplementering ikke laver uønsket

bivirkninger.

Vores misforståelse af funktionerne fra fase 1 har medført, at vores første inplementering af komparatornetværker ikke stemmer overens med programmingskontrakten og kravene fra fase 2. Derfor har vi aktivt valgt at gøre brug af vores undervisers implementere af komparatornetværker. Det har den store fordel, at vi har muligheden for, at korrekt kunne implementere sortingsnetværker med *Command Line Interfaces*. Ulempen er dog at vi skal arbejde med en implementere, som vi ikke har lige så meget indsigt i og skal derfor tildele noget af vores arbejdstid for at forstå det.

Ligesom i fase 1, så har vi benyttet softwaren *Git* og hjemmesiden *GitHub* [Azu24] til, at kunne samarbejde, programmere og fordele arbejdsopgaverne i praksis. Vores forbrug af *Git* er ikke perfekt, da vi ikke har meget erfaring med det. Udover det har vi benyttet hjemmesiden og LLM *Large Language Model* ChatGPT [Ope24] til, at omformuelere projektoplægget og programkontrakten, så vi kunne få en tydeligere ide om, hvad vi skulle programmere. Det har til dels være nyttigt da gruppen er flersproget. Kun enkelte tilfælde har vi benyttet ChatGPT til at lave kodeforslag, men alle gangene har vi afvist forslagene, da de enten ikke virkede eller benyttede metoder vi ikke havde erfaring med og kunne bedømme kvaliteten af.

4 Programmering af *filter.py*

Modulet filter har til hovedeformål at definere en datastruktur ved navn *Filter*. Strukturen benyttes i høj grad af de senere moduler til, at opbevarer vigtig data således, at vi kan til sidst i fasen opstille et sorteringsnetværk. Datastrukturen skal som minimum indeholde et netværk, som beskrevet af *network.py*-modulet, og alle dens binære permutationer, som er opstillet som en liste af lige lange lister. I løbet fasen skal der kunne tilføjes flere komparatorer, som er defineret i *comparator.py*-modulet, til datastrukturen, hvor der skal kunne gøres forskel på nyttige og redundante komparatorer. Funktioner der beskriver væremåden af Filter datastrukturen er beskrevet i de følgende kapitler.

4.1 *make_empty_filter()*

Formålet med *Make_empty_filter()* er at returnerer et tomt filter, bestående af et tomt netværk og alle de binære permutationer af den givne inputværdi

n. I praksis er et Filter opstillet af en dataclass, der indeholder et netværk, en liste af binære permutationer og den mængde kanaler Filteret er lavet til at kunne behandle. Alternativt kunne vi havde brugt en liste, hvor vi reserverer pladserne i listen til de forskellige værdier, men det frarådes at bearbejde for mange forskellige datatyper i samme liste. Filteret kreeres ved at bruge funktionen fra *network.py*-modulet, ved navn *empty_network()* og *_all_binary_inputs()*, som tilsammen danner det tomme filter og den tilhørende liste af permutationer. Udover det tilføjes også størrelsen, som er lig emd inputværdien n, som i praksis er et heltal.

4.2 *Net() & Out()*

Net() og *Out()* funktionerne giver muligheden for henholdsvis at aflæse det pågældende netværk ud eller de pågældende binære permutationer. Dette opnås ved brug af det objekt orienterede paradigme, da Filteret er et objekt med attributter, kan vi tilgå det givne netværk eller de givne binære permutationer, ved at kalde *"f.n"* eller *"f.out"*.

4.3 *Is_redundant()*

Is_redundant() er en funktion der tjekker om en given komparator er redundant ved at tilføje den til et givent filters netværk, og dertil tjekke om de binære permutationer ændres. Hvis dette er tilfældet dømmes komparatoren ikke-redundant, og ellers redundant. Dette opnås ved at tage en kopi at det nuværende stadie af netværket og de binære permutationer. Derefter tilføjes der den givne komparator, og der forsøges at sortere de binære permutationer. Til sidst tjekkes der om de tidligere binære permutationer er ens til de nye. Hvis dette er tilfældet, er komparatoren som nævnt redundant, og ellers ikke-redundant. Dette tillader gruppen at frasortere de ikke-redundate komparatorer. Dette bruges i samspil med *add()*-funktionen i *generate.py* modulet til at udvide netværket.

4.4 *Add()*

Add()-funktionen tilføjer en komparator i enden af netværket på et filter. Dette gøres med et simpelt kald til Filter med opdateret argumenter, der indeholder den ønskede komparator. Funktionen bruges i samspil med *is_redundant()* i *generate.py*-modulet, til at udvide alle de pågælende filtre.

4.5 *Is_sorting()*

Is_sorting() funktionen tjekker om det givne netværk i et filter er et korrekt sorteringsnetværk, ved at tjekke om netværket kan sortere alle binære permutationer fra mindst til størst. I praksis gøres det ved at aktivere *network.py*-modulets *is_sorting()*-funktion, hvor vi giver et Filters netværk.

5 Opstilling af *generate.py* og *extend()*-funktionen

Efter at have defineret filterdatastrukturen skal det behandles i *generate*-modulet. Modulet består af en enkelt funktion som hedder *extend()*. Funktionen kræver en liste af filtre og et positivt heltal, som beskriver mængden af kanaler filtrerne er lavet til at behandle. Funktionens endemål er at udvide alle netværkerne i filtrene med en ikke-redundant standardkomparator på alle mulige måder. Matematisk er funktionen tilsvarende til, at opstille delmængden af det kartesiske produkt mellem alle netværkene i filtrene og alle standardkomparatorerne. Det er en delmængde fordi outputtet skal kun indeholde filtre udvidet med ikke-redundante komparatorer. For eksempel hvis man startede med en liste, der kun indeholdte det tomme filter, som er lavet til tre kanaler og aktiverede *extend()*-funktionen med listen som input, så ville man få en ny liste, der indeholdte tre filtre. Den første filter indeholder komparatoren mellem kanal 0 og 1, den anden har komparatoren for 1 og 2 og den sidste har komparatoren til kanal 0 og 2. Hvis man aktiverede *extend*-funktionen igen, men med listen lavet af den forrige *extend()*-funktionskald, så ville filtret der indeholdte komparatoren mellem kanal 0 og 1 ikke blive udvidet med samme komparator, da den nye komparator ville være redundant. Det ikke-udvidet filter ville ikke blive inkluderet i funktionsproduktet. Funktionens væremåde er afbildet på nedenstående flowchart:

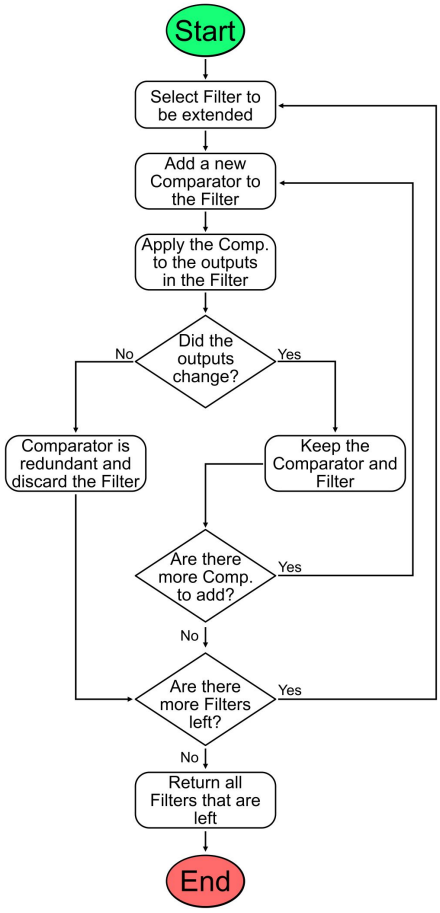

Fig. 1: Trinvis gennemgang af *extend()*-funktionen. Egen tilvirkning

I praksis valgte vi at implementere *extend()*-funktionen ved hjælp af funktionel programmering. Vi benyttede funktionerne *map()* til at danne de individuelle kartesiske produkter, *reduce()* til at sammensætte produkterne til en enkelt liste, og til sidst *filter()*-funktionen til at fjerne de redundante filtre. Desværre var der nogle udfordringer med at bruge funktionel programmering. Funktionerne *map()* og *filter()* understøtter brug af logiske operatorer, såsom *if*-sætninger og sammenligninger. I vores specifikke tilfælde vil vi kun konstruere og beholde filtre med ikke-redundante komparatorer, hvor vi forkaster resten. I vores implementering med *map()*-funktionen, så hvis man kun tog højde for de relevante komparatorer, så vil funktion-

sproduktet af filtrene og de redundante komparatorer være lig med Pythons *None*-datatype, da de ikke er defineret i den anonyme funktion. For at undgå at skulle behandle *None*-datatypen, så valgte vi at definere funktionsproduktet, filtrene og de redundante komparatorer til, at være lig med den tomme liste. Vi vælger at bruge den tomme liste, da det kan nemt af frasortere fra resten af funktionsproduktet ved hjælp af *filter()*-funktionen, og dertil forebygger at vi ikke blander for mange datatyper sammen. Hvis man var mere erfaren inden for funktionel programmering, kunne man højst sandsynligt opstille en pænere løsning, men vores første prioritet er at lave en funktion der virker og spiller sammen med resten af modulerne.

6 Opbygning af *network_finder.py*

Modulet *Network_finder.py*-modulet har til formål, at være en rugergrænsefladen, hvor en brugeren kan opstille et sorteringsnetværk for en mængde kanaler de selv bestemmer. Modulet skal kunne behandle inputværdier fra brugeren og til sidst fremvise et sorteringsnetværk i form af samling af Python-kode, der beskriver hvordan netværket fungerer. Brugergrænsefladen er opbygget som et CLI (*Command Line Interface*), hvor der gøres stor brug af Pythons *print*-funktion. I modulet har vi defineret en funktion ved navn *make_sorting_network()*, som opstiller selve sorteringsnetværket til brugeren. Den er inplementeret rekursivt, hvor den skal have en liste med et tomt Filter og mængden af kanaler som startværdi. Funktionen starter med at tjekke om nogen at Filtrene i listen indeholder et sorteringsnetværk ved hjælp af *is_sorting()* og *map()*, hvis det ikke er tilfældet så udvides alle Filtrene i listen med ikke redundante komparatorer ved hjælp af *extend()*-funktionen. Funktionsproduktet af *extend()*-funktion bliver derefter behandlet af *prune()*-funktionen fra *prune.py*, hvor til sidst produktet af *prune()* bliver sendt til at blive behandlet af *make_sorting_network()* igen. Hvis man finder et sorteringsnetværk i listen af Filtre, så returneres det første Filter der har et sorteringsnetværk. Efter man har fundet et sorteringsnetværk til den mængde kanaler som brugeren ønsker, bruges *to_program()* fra netværksmodulet til, at fremvise hvordan sorteringsnetværket virker.

6.1 Køretiden af *network_finder.py*

Køretiden for *make_sorting_network()* er meget svær at beregne. Først ville vi finde ud af, hvad køretiden er for en kørsel (en kanal med 1 filter). Der blev brugt Generativ AI [Ope24] til en estimering på $O(2^n \cdot n)$, som er meget langsom. Dog er køretiden med flere kanaler og flere filtre $O(2^n \cdot n \cdot size(w) \cdot log(size(w))$ hvor $size(w)$ repræsenterer antallet af Filtre. Denne køretid er ekstremt lang. Med $n = 3$ kanaler er antallet af filtre 4, med $n = 4$ filtre er der allerede 340, og med $n = 5$ er antallet af filtre kommet op på cirka 1,2 millioner efter 12 timer, og processen er stadig ikke færdig.

7 Test og evaluering

I denne sektion vil der gennemgås hvordan, vores program blev testet, og dertil også evalueret på baggrund af de forventet resultater. Ved at gøre brug af den kontraktbaserede struktur, som projektet gøre brug af, kan vi teste vores funktioner og moduler systematisk for, at kunne bedømme hvor vidt de er i overensstemmelse med kontrakten og den forventede udfald. Dette ses igennem brug af *DOCTEST*, *preconditions* og de opstille test. De omtalte *DOCTEST* og *preconditions* kan findes til hver defineret funktion i modulerne. De systematiske test findes i filen *test.py*. textitDOCTEST'ene, *preconditions* og testene tilsammen sikre os, at hver enkelt funktion opfører sig som forventet.

7.1 Test af kode

I testen af koden blev der valgt at undlade testen af modulet *network_finder.py*. Da modulet kun håndterer selve brugergrænsefladen og indeholder en enkelt funktion ved navn *make_sorting_network()*. Da opbygningen af modulet besværliggør test af modulet, da kald af funktion *make_sorting_network()* afventer et input fra brugeren, før resten af programmet udføres. I praksis gør det, at testfilen ikke viser alle de andre resultater, før der indgives et input. Derfor har gruppen fravalgt at teste funktionen og ser i stedet korrektheden igennem kørsel af modulet adskilt. I testen af *extend()*-funktionen i *generate.py*-modulet lavede gruppen en test, hvor der først bliver vist 3 filtre med en komparator tilføjet. Derefter kaldes *extend()*-funktion og resultat printes trinvist, så hvert enkelt udvidet filter vises. Dette tillader gruppen

at se forskellen, samt at kompleksiteten af *extend()*-funktionen tydeliggøres. Strukturen af hele testfilen er ens til den af første fase. Testene er inddelt efter modul og underopdelt i individuelle funktioner. Dette tillader at gruppen kan se eventuelle forskelle på det kontraktbaserede resultat og resultatet i praksis.

7.2 Evaluering af kode

Ved at gøre brug af systematiske test, kunne gruppen konkludere at dannelsen af et sorteringsnetværk på n-kanaler var muligt, med en praksis øvre grænse på 5 kanaler. Da køretiden vokser superekseponentielt med antallet af kanaler, var det ikke muligt for gruppen at få dannet et netværk på 5 kanaler. Der blev forsøgt med at lade programmet kører i 38 timer, og den sidste iteration var endnu ikke gennemført, efter en estimering af den asymptotiske køretid, indså gruppen at der ville gå meget lang tid før den kunne færdiggøres. Som tidligere nævnt ramte programmet 1,2 millioner i antallet af filtre, derfor tager programmet så lang tid, da den skal udvide 1,2 millioner filtre. Selvom gruppen ikke definitivt kunne sige at programmet virkede for 5 kanaler, blev der i projektbeskrivelsen for fase 2 også nævnt, at det var forventeligt at det kunne køres op til 4 eller 5 kanaler [Cru24]. Dermed kunne gruppen uden yderligere indsats sikre sig, at opgaven var løst. I fase 3 forventes der, at der implementeres et modul ved navn *prune.py* som ville varetage *prune*-delen af *generate-prune* algoritmen. Dette ville drastisk reducere køretiden, da der i denne fase kun kaldes på *extend()*-funktionen. Dette gør at programmet danner et sorteringsnetværk, og dertil udvider netværket, uden at optimere netværket yderlige, dette sænker køretiden voldsomt, da der skal testes med det flere kombinationer.

7.3 Valg af løsningsforslag

Overordnet havde gruppen succes med vores fremgangsmåde og strategi. Selvom gruppen som helhed ikke er lige så erfaren med rekursiv og funktionel programmering, opstillede vi et sorteringsnetværk op til 4 kanaler. Ud fra resultaterne af vores test, undgik vi at lave uønsket bivirkninger i vores forskellige funktioner. Dette skyldes både at vi er blevet mere erfaren og har en dybere forståelse af projekter, men også fordi vi gjorde brug rekursiv og funktionel programmering, der som udgangspunkt ikke laver bivirkninger. Her i fase 2 gjorde vi også mindre brug af ChatGPT [Ope24], da vi i højere

grad gjorde brug af de forskellige metoder gennemgået i kurset, i stedet for at få ChatGPT til at opstille kodeforslag. Der var dog tilfælde, hvor vores erfaring inden for funktionel programmering blev udfordret, hvor vi endte med at opstille fungerende men ikke pæne løsninger. Dog da vores test er i overensstemmelse med kontrakten forventer vi, at vores moduler vil være tilstrækkelige i den næste fase.

8 Konklusion

I denne rapport er den anden fase i det tredelt eksamensprojekt blevet gennemgået. Der er blevet programmeret tre moduler ved navn *filter.py*, *generate.py* og *network_finder.py*. Disse moduler kan sammen med første fase, danne et sorteringsnetværk på n antal kanaler. Funktionaliteten af de tre moduler er blevet vist igennem systematiske tests. Ydeligere så er gruppens fremgangsmåde, valg og tankegang blevet forklaret og diskuteret. Ved at evaluere fase 2 igennem de systematiske test, ses der at det færdige produkt, løser problemstillingen tilfredsstillende, og danner derved grundlag til fase 3.

9 Litteraturliste

[Azu24] AzureX. *SDU-Programmeringsprojek*. Accessed: 08-11-2024. 2024. URL: https://github.com/xXmemestarXx/SDU-Programmeringsprojekt/tree/AzureX/.

[Cru24] Luis Cruz-Filipe. *Gruppeprojekt - Efterår 2024 — Introduktion*. PDF document. 2024.

[Ope24] OpenAI. *ChatGPT*. Accessed: 08-11-2024. 2024. URL: https://openai.com/.

10 Bilag og kildekode

10.1 filter.py

```
"""
filter.py

Written by Hlynur, Mathias and Valdemar H8G03

DM574 Exam project
"""

"""
Importing 3rd party libaries
"""
from dataclasses import *

"""
Importing our lecturer's modules
"""
import network as Netw
import comparator as Comp

@dataclass
class Filter:
    """
    We define a singular Filter as a
    dataclass that contains a network,
    its binary outputs and the amount
    of channels the Filter works on.

    We use a dataclass since it is not
    recommend to use a list containing
    different datatypes
    """
    netw: Netw.Network
    outp: list[list[int]]
    size: int

```

```
36 def make_empty_filter(n: int) -> Filter:
37     """
38     Preconditions: n > 0
39 
40     Returns a filter with an empty network and all binary
41     permutations of length n.
42 
43     DOCTEST
44     >>> make_empty_filter(2)
45     [[], [[0, 0], [1, 0], [0, 1], [1, 1]]]
46     """
47     net = Netw.empty_network()
48     out = Netw._all_binary_inputs(n)
49     size = n
50     return Filter(net,out,size)
51 
52 def net(f: Filter) -> Netw.Network:
53     """
54     Returns the Network of a Filter
55 
56     DOCTEST
57     test_filter = make_empty_filter(2)
58     >>> net(test_filter)
59     []
60     """
61     copy = f.netw
62     return copy
63 
64 def out(f: Filter) -> list[list[int]]:
65     """
66     Returns the outputs of a Filter
67 
68     DOCTEST
69     test_filter = make_empty_filter(2)
70     >>> out(test_filter)
71     [[0, 0], [1, 0], [0, 1], [1, 1]]
72     """
73     copy = f.outp
74     return copy
75 
```

```
76  def size(f: Filter) -> int:
77      """
78      Returns the size of the filter
79
80      DOCTEST
81      test_filter = make_empty_filter(2)
82      >>> get_size(test_filter)
83      2
84      """
85      copy = f.size
86      return copy
87
88  def is_redundant(c: Comp.Comparator, f: Filter)-> bool:
89      """
90      Checks if the Comparator, c, would be redundant if it were
91      to be added to the Network in the Filter, f.
92
93      DOCTEST
94      n = 3
95      filt_test = filt.make_empty_filter(n)
96      filt_test = filt.add(2, filt_test)
97      Filter(n=[2], out=[[0, 0, 0], [0, 0, 1], [0, 1, 1], [1, 1, 1]])
98      >>> is_redundant(2, filt_test)
99      True
100     """
101     copy_net = net(f)
102     copy_per = out(f)
103
104     copy_net = copy_net + [c]
105
106     new_per = Netw.outputs(copy_net, copy_per)
107     return f.outp == new_per
108
109 def add(c: Comp.Comparator, f: Filter) -> Filter:
110     """
111     Appends a Comparator to the end of a Network in
112     a Filter
113
114     DOCTEST
115     test_comp = Comp.make_comparator(0, 2)
```

```
116     test_filter = make_empty_filter(3)
117     >>> test_filter = add(test_comp, test_filter)
118     [[5], [[0, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]]
119     """
120     new_net = Netw.append(c,net(f))
121     new_out = Netw.outputs(new_net, out(f))
122     same_size = size(f)
123     return Filter(new_net,new_out,same_size)
124
125
126 def is_sorting(f: Filter) -> bool:
127     """
128     Checks if the network in the filter is a sorting network.
129
130     DOCTEST
131     n = 3
132     filt_test = filt.make_empty_filter(n)
133     filt_test = filt.add(2, filt_test)
134     filt_test = filt.add(5, filt_test)
135     filt_test = filt.add(8, filt_test)
136     Filter(n=[2, 5, 8], out=[[0, 0, 0], [0, 0, 1], [0, 1, 1], [1, 1, 1]], size=3)
137     >>> is_sorting(filt_test)
138     False
139     """
140     return (size(f) < 2) or Netw.is_sorting(net(f),size(f))
```

10.2 generate.py

```
1  """
2  generate.py
3
4  Written by Hlynur, Mathias and Valdemar H8G03
5
6  DM574 Exam project
7  """
8
9  """
10 Importing 3rd party libaries
```

```
11 """
12 import functools as Func
13 from dataclasses import *
14
15 """
16 Importing our own and our lecturer's modules
17 """
18 import comparator as Comp
19 import network as Netw
20 import filter as Filt
21
22 """
23 Importing definitions of data structures
24 """
25 Comparator = int
26 Network = list[Comparator]
27 Filter = list[Netw.Network, list[list[int]]]
28
29 def extend(w: list[Filter], n: int) -> list[Filter]:
30     """
31     Preconditions: n > 0 and len(w) > 0
32
33     Adds all non-redundant standard comparators for
34     n amount of channels to the Filters in
35     the input list, w.
36
37     This is equal to calculating a subset of the
38     cartesian product of w and all standard
39     comparators for n amount of channels
40
41     DOCTEST
42     n = 2
43     filt_test = Filt.make_empty_filter(n)
44     w = [filt_test]
45     filt_test2 = Gene.extend(w, n)
46     >>> filt_test2 = extend(w,2)
47     [Filter(n=[2], out=[[0, 0], [0, 1], [1, 1]], size=2)]
48     """
49
50     """
```

```
51      Start by getting all the standard Comparators
52      """
53      stdComp = Comp.std_comparators(n)
54
55      """
56      The following map functions makes a subset
57      of the cartesian product of the input Filters
58      and all standard comparators for n amount of
59      channels. It is a subset since we only keep
60      the non-redundant comparators and the original
61      Filters.
62
63      We need to use the list() function multiple
64      times since Python's version of the map
65      function returns an object.
66      """
67      #carte_prod = list(map(lambda f: list(map(lambda c: _check_and_add(c,f),stdComp)),w))
68
69      carte_prod = list(map(lambda f: list(map(lambda c: Filt.add(c,f)
70                  if not Filt.is_redundant(c,f) else [],stdComp)),w))
71      """
72      Since the lambda function needs to always return some
73      some type value, the lambda function will return
74      the empty list when a Comparator is redundant. If we
75      removed the line then the auxillary function would
76      return None if a Comparator is redundant.
77      """
78
79      combined_filters = Func.reduce(lambda x,y: x+y,carte_prod)
80
81      """
82      Python always requires an else clause when using
83      if-statements in a map. There _add_and_check()
84      returns a empty list when a Comparator is redundant. We
85      remove all the 0's using the filter function
86      """
87      extended_filters = list(filter(lambda x: x != [],combined_filters))
88
89      return extended_filters
```

10.3 network_finder.py

```
"""
network_finder.py

Written by Hlynur, Mathias and Valdemar H8G03

DM574 Exam project
"""

"""
Importing 3rd party libaries
"""
from dataclasses import *

"""
Importing our own and our lecturer's modules
"""
import comparator as Comp
import network as Netw
import filter as Filt
import generate as Gene
import prune as Prun


"""
network_finder is essentially a small Command Line Interface program, or CLI.
Therefore the user interface, UI, and the user experience, UX,
needs to be taking into acoount. The simplest way to do this is by printing
guiding helpful messages, so the user knows when
and how they are using the program wrong.
"""

def make_sorting_network(f: list[Filt.Filter], n: int, i: int) -> Filt.Filter:
    """
    Preconditions: n > 0 and len(f) > 0

    Checks if there is one or more sorting networks in the filter list, and then returns t
    """
    if any(list(map(Filt.is_sorting, f))):
```

```
39          return list(filter(lambda x: Filt.is_sorting(x) == True, f))[0]
40
41      i = i + 1
42      extended_filters = Gene.extend(f, n)
43      clean_list = Prun.prune(extended_filters, n)
44      return make_sorting_network(clean_list, n, i)
45
46  done = False
47
48  while not done:
49      print(
50          """
51          .----------------------------------------.
52          |                                        |
53          |   Welcome to Network Finder program |
54          |     by Hlynur, Mathias & Valdemar  |
55          '----------------------------------------
56          """)
57
58      print("Tip: Time needed to calculate is proportional to amount of channels ")
59
60      channel_amount = int(input("Please enter how many channels you want to sort: "))
61
62      while 1 > channel_amount:
63          """
64          Checks wheter the input is valid number or not
65          """
66          if 0 >= channel_amount:
67              print("Please enter a number greater than zero")
68
69          else:
70              print("Invalid input")
71          channel_amount = int(input())
72
73
74      all_filters = [Filt.make_empty_filter(channel_amount)]
75
76      print(f"Finding a sorting network for {channel_amount} channels... \n")
77
78      sorting_network = make_sorting_network(all_filters, channel_amount, 0)
```

```
79
80      print(f"Found a sorting network for {channel_amount} channels with size {Netw.size(Fil
81
82      print(f"An implementation of the sorting network in Python would look like: \n")
83
84
85      program_string = Netw.to_program(Filt.net(sorting_network),'','')
86
87      for i in range(0,len(program_string)):
88          print(program_string[i])
89
90
91      done = True
```

10.4 test.py

```
1  import comparator as Comp
2  import network as Netw
3  import filter as Filt
4  import generate as Gene
5
6  print(f"")
7  print(f"------- test filter.py -------")
8  print(f"")
9  print(f"------- test 1 begin -------")
10 print(f"")
11 print(f"Testing make_empty_filter()")
12 print(f"")
13
14 filt_test = Filt.make_empty_filter(0)
15 filt_test2 = Filt.make_empty_filter(2)
16 print(f"{filt_test} => {filt_test}")
17 print(f"{filt_test2} => {filt_test2}")
18
19 print(f"")
20 print(f"------- test 1 end -------")
21 print(f"")
22
```

```
23 print(f"------- test 2 begin -------")
24 print(f"")
25 print(f"Testing net()")
26 print(f"")
27
28 comp_test = Comp.make_comparator(0, 1)
29 filt_test = Filt.make_empty_filter(0)
30 filt_test2 = Filt.make_empty_filter(2)
31 filt_test2 = Filt.add(comp_test, filt_test2)
32 print(f"{filt_test} => {Filt.net(filt_test)}")
33 print(f"{filt_test2} => {Filt.net(filt_test2)}")
34 print(f"")
35 print(f"------- test 2 end -------")
36 print(f"")
37
38 print(f"------- test 3 begin -------")
39 print(f"")
40 print(f"Testing out()")
41 print(f"")
42
43 filt_test = Filt.make_empty_filter(0)
44 filt_test2 = Filt.make_empty_filter(2)
45 print(f"{filt_test} => {Filt.out(filt_test)}")
46 print(f"{filt_test2} => {Filt.out(filt_test2)}")
47 print(f"")
48 print(f"------- test 3 end -------")
49 print(f"")
50
51 print(f"------- test 4 begin -------")
52 print(f"")
53 print(f"Testing size()")
54 print(f"")
55
56 filt_test = Filt.make_empty_filter(0)
57 filt_test2 = Filt.make_empty_filter(2)
58 print(f"{filt_test} => size(filter) => {Filt.size(filt_test)}")
59 print(f"{filt_test2} => size(filter) => {Filt.size(filt_test2)}")
60 print(f"")
61 print(f"------- test 4 end -------")
62 print(f"")
```

```
63
64 print(f"------- test 5 begin -------")
65 print(f"")
66 print(f"Testing is_redundant()")
67 print(f"")
68
69 filt_test = Filt.make_empty_filter(3)
70
71 filt_test2 = Filt.make_empty_filter(3)
72 filt_test2 = Filt.add(2, filt_test)
73
74 # Doing this to minmize length of the statement
75 print_help = Filt.is_redundant(2, filt_test)
76 print_help2 = Filt.is_redundant(2, filt_test2)
77
78 print(f"{filt_test} => Testing a single comparator => {print_help}")
79 print(f"{filt_test2} => Testing duplicated comparators => {print_help2}")
80 print(f"")
81 print(f"------- test 5 end -------")
82 print(f"")
83
84 print(f"------- test 5 begin -------")
85 print(f"")
86 print(f"Testing add()")
87 print(f"")
88
89 filt_test = Filt.make_empty_filter(2)
90 filt_test2 = Filt.make_empty_filter(2)
91 filt_test2 = Filt.add(2, filt_test2)
92
93 print(f"{filt_test} => add(2, filter) => {filt_test2}")
94
95 print(f"------- test 5 end -------")
96 print(f"")
97
98 print(f"------- test 6 begin -------")
99 print(f"")
100 print(f"Testing is_sorting()")
101 print(f"")
102
```

```
103 filt_test = Filt.make_empty_filter(4)
104 filt_test = Filt.add(2, filt_test)
105 filt_test = Filt.add(5, filt_test)
106 filt_test = Filt.add(8, filt_test)
107
108 filt_test2 = Filt.make_empty_filter(3)
109 filt_test2 = Filt.add(2, filt_test2)
110 filt_test2 = Filt.add(5, filt_test2)
111 filt_test2 = Filt.add(8, filt_test2)
112
113 print(f"{filt_test} => is_sorting(filter) => {Filt.is_sorting(filt_test)}")
114 print(f"{filt_test2} => is_sorting(filter) => {Filt.is_sorting(filt_test2)}")
115
116 print(f"------- test 6 end -------")
117 print(f"")
118
119 print(f"")
120 print(f"------- test generate.py -------")
121 print(f"")
122 print(f"------- test 1 begin -------")
123 print(f"")
124 print(f"Testing extend()")
125 print(f"")
126
127 F1 = Filt.add(2,Filt.make_empty_filter(3))
128 F2 = Filt.add(5,Filt.make_empty_filter(3))
129 F3 = Filt.add(8,Filt.make_empty_filter(3))
130
131 all_filters = [F1,F2,F3]
132
133 for i in range(0,len(all_filters)):
134     print(f"Filter {i+1} network:{Filt.net(all_filters[i])} => {Filt.out(all_filters[i])}\
135
136 new = Gene.extend(all_filters,3)
137
138 print("After Extending")
139
140 for i in range(0,len(new)):
141     print(f"Filter {i+1} network:{Filt.net(new[i])} => {Filt.out(new[i])}\n")
142
```

```
143 print(f"")
144 print(f"------- test 1 end -------")
145 print(f"")
```