utad

Planificação de horários e salas de exames em Haskell

Programação Funcional

David Gomes Fidalgo – al79881
Carolina Machado – al79359
João Fraga Silva – al78400

Índice

1. Introdução

Dando continuidade à primeira versão do trabalho, vamos explicar a solução apresentada que foi adicionada (2 parte), que teve como intuito, implementar as novas funcionalidades requeridas no protocolo. Sendo assim através de todos os conceitos abordados pretendemos à reformulação do código, aproveitando algumas funções já existentes e outras transformando-as para a manipulação dos dados de acordo com as necessidades de cada etapa na construção do código em Haskell.

2. Estruturação do código

Segundo o protocolo, vamos seguir respetivamente o pedido, criando as funções para a execução do código:

1. Passo: Criar uma função **"impruc"** que lê o ficheiro "**ucs.txt**" e retorna uma ação IO que, quando executada, produzirá uma string com o seu conteúdo.

```
impruc :: IO String
impruc = readFile "ucs.txt"
```

Figura1. Função "impruc".

2. Passo: Criar três funções: **"apreuc"**, **"toDisciplinas"** e **"toDisciplina"**. A função **"apreuc"** realiza a leitura do conteúdo do ficheiro ao invocar **"impruc"**. De seguida a função separa o texto em linhas individuais usando a condição 'lines'. Após este passo, a função **"apreuc"** chama a segunda função, **"toDisciplinas"**, que, por sua vez, chama a terceira função, **"toDisciplina"** através da condição '**map**'. A função **"toDisciplina"** recebe uma string que representa uma linha do arquivo, usando a condição '**words**' para dividi-la em palavras individuais. Neste caso, a linha contém dois números e uma palavra. A função **"toDisciplina"** retorna uma tupla com o identificador da unidade curricular, o valor numérico que representa a unidade e o nome da unidade, respectivamente.

```
apreuc :: IO [(Int, Int, String)]
apreuc = do
    texto <- impruc
    let list = lines texto
        disciplinas = toDisciplinas list
    return disciplinas
```

Figura2. Função "aprec".

```
toDisciplinas :: [String] -> [(Int, Int, String)]
toDisciplinas = map toDisciplina
```

Figura3. Função "toDisciplinas".

```haskell
toDisciplina :: String -> (Int, Int, String)
toDisciplina str = case words str of
    (n1:n2:rest) -> (read n1, read n2, unwords rest)
    _ -> error "Formato de linha inválido"
```

Figura4. Função "toDisciplina".

Após estes dois passos, é necessário criar mais oito funções semelhantes para os outros dois ficheiros pretendidos, isto é quatro para o ficheiro **"incrições.txt"** e outras quatro para **"listaalunos.txt".**

Funções para **"incrições.txt":**

```haskell
impins :: IO String
impins = readFile "inscrições.txt"
```

Figura5. Função "impins".

```haskell
apreins :: IO [(String, Int)]
apreins = do
    texto <- impins
    let list = lines texto
        inscricao = toinscri list
    return inscricao
```

Figura6. Função "apreins".

```haskell
toinscri :: [String] -> [(String, Int)]
toinscri = mapMaybe toinscr
```

Figura7. Função "toinscri".

```haskell
toinscr :: String -> Maybe (String, Int)
toinscr str = case words str of
    (n1:n2:rest) -> Just (n1, read [head n2])
    _ -> Nothing
```

Figura8. Função "toinscr".

***Nota*:** Neste caso, a tupla é retornada com o identificador do aluno e o idenificador numérico da unidade curricular, respetivamente. Também utilizamos a condição '**Maybe**' que nos dá possibilidade de o valor estar presente ('**Just**') ou ausente ('**Nothing**') que faz exatamente o mesmo que o '**error**', sendo assim um método mais rápido e menos tradicional.

Funções para **“listaalunos.txt”**:

```
imprlisal :: IO String
imprlisal = readFile "listaalunos.txt"
```

Figura9. Função “imprlisal”.

```
apreal :: IO [(String, Int, String)]
apreal = do
    texto <- imprlisal
    let list = lines texto
        alunos = toalunos list
    return alunos
```

Figura10. Função “apreal”.

```
toalunos :: [String] -> [(String, Int, String)]
toalunos = mapMaybe toaluno
```

Figura11. Função “toalunos”.

```
toaluno :: String -> Maybe (String, Int, String)
toaluno str = case words str of
                (n1:n2:rest) -> Just (n1, read n2, unwords rest)
                _ -> Nothing
```

Figura12. Função “toaluno”.

***Nota*:** Desta vez a tupla é retornada com referência ao identificador do aluno, valor numérico que representa o ano curricular em que o aluno está inscrito e por fim, o nome do aluno, respetivamente.

3. Passo: **Criar as funções relacionadas às alíneas 1) e 2).** Primeiro temos de declarar um tipo de dados, neste caso definido por **“data Exame”**, que define cinco campos, sendo estes a **‘uc’** (do tipo String), **‘salas’** (uma lista de Strings), dia (String), **‘ano’** (Int), e ‘alunos’ (uma lista de Strings), que estão relacionados com os exames. A condição **‘deriving’** (Show, Eq, Ord) indica que existem três classes de que permitem exibir **“data Exame”** como uma string (Show), igualdade (Eq), ou ordenação (Ord).

```
data Exame = Exame { uc :: String, salas :: [String], dia :: String, ano :: Int, alunos :: [String] } deriving (Show, Eq, Ord)
```

Figura13. Tipo de dados “data Exame”.

4. Passo: Criar mais duas funções similares, sendo estas **"enumerarsalas"** e **"enumerardias".** Estas duas funções recebem um inteiro **'n'** como argumento e retornam uma lista de strings. Estas funções utilizam a condição **'map'** que percorre toda a lista de numeros de **'1'** até **'n'**, sendo que a condição em frente ao **'map'**, designa-se pela condição lambeda que toma o número **'i'** como entrada, ou seja, ambas as funções vão printar o que está dentro de aspas (sala ou dia) com o respetivo **'i'**, que retornará como uma string através do **'show'**.

```haskell
enumerarsalas :: Int -> [String]
enumerarsalas n = map (\i -> "Sala" ++ show i) [1..n]

enumerardias :: Int -> [String]
enumerardias n = map (\i -> "Dia" ++ show i) [1..n]
```

Figura14. Funções "enumerarsalas" e "enumerardias".

5. Passo: Criar uma função para fazer o escalonamento (**"gerar escalonamento"**). Esta função recebe três listas de tuplas '(String, Int)' que representam as unidades curriculares e seus respectivos anos, '(String, String)' que simbolizam salas e dias, uma lista de tuplas '(String, [String])' que se referem às unidades curriculares e aos alunos inscritos nelas, e um número inteiro que traduz a capacidade máxima de alunos por sala. Passando agora para a parte principal da função, esta utiliza a condição **'zipWith'** é usada para combinar as listas para cada par de tuplas e respetivamente, é criado um novo Exame. A condição **'lookup'** é usada para encontrar a lista de alunos para a respectiva uc na 'alunoLista'. Se a uc não for encontrada na alunoLista, **'fromMaybe []'** garante que uma lista vazia seja usada. A condição **'map'** atualiza a lista de salas para cada Exame, com base na quantidade de alunos. De seguida é necessário, criar uma função secundária **"splitStudents"** para fazer essa atualização. Esta divide os alunos entre as salas, com base na capacidade máxima. Se o número de alunos for maior que a capacidade máxima, é criada uma nova sala que se chama a si mesma (recursividade), sendo que esta é adicionada à lista de salas, e insere os alunos restantes, após remover a quantidade que foi alocada na sala atual. Este processo continua até que todos os alunos tenham sido alocados corretamente nas salas. O número da sala é atualizado para o seguinte a partir de uma terceira função **"nextSala"**.

```haskell
gerarEscalonamento :: [(String, Int)] -> [(String, String)] -> [(String, [String])] -> Int -> [Exame]
gerarEscalonamento ucs slots alunoLista max_capacidade =
  let exames = zipWith (\(uc, ano) (sala, dia) -> Exame uc [sala] dia ano (fromMaybe [] (lookup uc alunoLista))) ucs slots
  in map (\exame -> exame { salas = splitStudents (salas exame) (alunos exame) }) exames
  where
    splitStudents :: [String] -> [String] -> [String]
    splitStudents salas alunos
      | length alunos > max_capacidade = splitStudents (salas ++ [nextSala (last salas)]) (drop max_capacidade alunos)
      | otherwise = salas
    nextSala :: String -> String
    nextSala sala = "Sala" ++ show (read (drop 4 sala) + 1)
```

Figura15. Função "gerarEscalonamento".

6. Passo: Criar mais 3 funções associadas ao escalonamento e incopatibilidade dos exames sendo estas **“igualdia”, “verificarConflitos”, “getAlunosInscritos”.** Na primeira, esta recebe dois exames como entrada e retorna **‘True’** se tiverem o mesmo ano e dia, e se as suas listas de salas não intersetarem (isto é, não compartilharem nenhuma sala em comum). Usa as condições ano, dia e salas para acessar os campos dos exames, e a condição ‘intersect’ para encontrar a interseção das listas de salas.
   Na segunda função recebida uma lista de exames como entrada e é retornado **‘True’** se houver qualquer exame na lista que tenha o mesmo ano e dia e compartilhe pelo menos uma sala com outro exame na lista. É usada a condição **‘any’** para verificar se qualquer elemento da lista satisfaz a condição, e a função **‘filter’** para obter todos os exames na lista que correspondem à condição especificada pela primeira função (**“igualdia”**).
   A terceira função recebe um exame e uma lista de exames como entrada, e retorna a lista de alunos inscritos no exame. Usa a função **‘filter’** para obter todos os exames na lista que têm a mesma unidade curricular (uc) que o exame dado, e então seleciona o primeiro desses exames (supondo que só haja um) e retorna a lista de alunos desse exame.

```
igualdia :: Exame -> Exame -> Bool
igualdia exame1 exame2 = ano exame1 == ano exame2 && dia exame1 == dia exame2 && null (intersect (salas exame1) (salas exame2))

verificarConflitos :: [Exame] -> Bool
verificarConflitos exames = any (\exame -> length (filter (igualdia exame) exames) > 1) exames

getAlunosInscritos :: Exame -> [Exame] -> [String]
getAlunosInscritos exame exames = alunos $ head $ filter ((== uc exame) . uc) exames
```

Figura16. Função “igualdia”, “verificalConflitos”, “getAlunosInscritos”.

7. Passo: Criar a função que processa a incompatibilidade, sendo esta a **“calcIncompatibilidadesponto1”.** Esta função recebe uma lista de exames e retorna uma lista de triplas, onde cada elemento é uma dupla de ucs (as strings) e a quantidade de incompatibilidades entre elas (o Int).
   A função mapeia **‘map uc’** a lista de exames para obter uma lista de todas as ucs. De seguida, é gerado uma lista de todas as possíveis duplas de ucs, onde cada uc é diferente da outra. A lista **‘paresUCs’** mapeia (map) sobre a lista de duplas de ucs para calcular a quantidade de incompatibilidades para cada dupla. Para cada dupla (uc1, uc2), a função obtém os exames para uc1 e uc2 (assumindo que existe exatamente um exame para cada uc), calcula a interseção das listas de alunos para os dois exames (ou seja, os alunos que estão inscritos em ambos os exames), e conta o número de alunos na interseção. O resultado é uma lista de triplas, e a quantidade de incompatibilidades entre elas. Finalmente, a função filtra **‘filter’** a lista de triplas para remover todas as triplas onde a quantidade de incompatibilidades é 0. O resultado é uma lista das duplas de ucs que têm pelo menos um aluno em comum (ou seja, pelo menos uma incompatibilidade), juntamente com a quantidade de incompatibilidades para cada dupla.

```haskell
calcIncompatibilidadesponto1 :: [Exame] -> [(String, String, Int)]
calcIncompatibilidadesponto1 exames =
  let ucs = map uc exames
      paresUCs = [(uc1, uc2) | uc1 <- ucs, uc2 <- ucs, uc1 /= uc2]
      incompatibilidades = map (\(uc1, uc2) ->
        let exame1 = head $ filter ((== uc1) . uc) exames
            exame2 = head $ filter ((== uc2) . uc) exames
            inc = length $ intersect (alunos exame1) (alunos exame2)
        in (uc1, uc2, inc)) paresUCs
  in filter (\(_,_,inc) -> inc > 0) incompatibilidades
```

Figura17. Função "calcIncopatibilidadesponto1".

8. Passo: Criar uma função (**"WriteIncompatibilidades"**) para escrever as informações sobre as incompatibilidades entre exames no arquivo de texto chamado "escalonamento.txt".

```haskell
writeIncompatibilidades :: [Exame] -> IO ()
writeIncompatibilidades exames = do
  let incompatibilidades = calcIncompatibilidadesponto1 exames
  appendFile "escalonamento.txt" "\nIncompatibilidades entre pares de exames:\n"
  mapM_ (\(uc1, uc2, incompat) -> appendFile "escalonamento.txt" $ "UCs " ++ uc1 ++ " e " ++ uc2 ++ ": " ++ show incompat ++ " alunos inscritos em ambas\n") incompatibilidades
```

Figura18. Função "WriteIncompatibilidades".

9. Passo: Criar a função (**"ponto 1"**) que gera o escalonamento para os exames e escreve-o escalonamento, bem como quaisquer incompatibilidades encontradas no arquivo de texto (escalonamento.txt). Primeiro, a função extraí as ucs e os anos correspondentes usando a função **'extrairUCSeAno'**. De seguida, enumera as salas e os dias disponíveis usando as funções **'enumerarsalas'** e **'enumerardias'**. A função então establece todos os possíveis slots para os exames combinando as salas e os dias disponíveis. Posto isto a função verifica se o número de slots é suficiente para acomodar todos os exames necessários. Se não for, +e impressa uma mensagem de erro e termina. Se houver slots suficientes, a função gera o escalonamento dos exames usando a função **'gerarEscalonamento'**. A função então verifica se há conflitos no escalonamento usando a função **'verificarConflitos'**. Se houver conflitos a função declara erro. Se não houver conflitos, a função escreve o escalonamento no arquivo **"escalonamento.txt"** e usa a função **'writeIncompatibilidades'** para escrever as incompatibilidades entre os exames no mesmo arquivo.
   Finalmente, a função imprime uma mensagem que informa o utilizador referindo que o arquivo **"escalonamento.txt"** foi atualizado.

```haskell
ponto1 :: [(Int, Int, String)] -> [(String, [String])] -> Int -> Int -> Int-> IO ()
ponto1 a alinsUC b c max_capacidade= do
  let ucseano = extrairUCSeAno a
  let salas = enumerarsalas b
  let dias = enumerardias c
  let slots = [(sala, dia) | sala <- salas, dia <- dias]

  if length slots < length ucseano
    then putStrLn "Número insuficiente de salas e/ou dias disponíveis para acomodar todos os exames existentes."
    else do
        let escalonamento = gerarEscalonamento ucseano slots alinsUC max_capacidade
        if verificarConflitos escalonamento
         then putStrLn "Há conflitos no escalonamento dos exames."
         else do
           writeFile "escalonamento.txt" (unlines $ map show escalonamento)
           writeIncompatibilidades escalonamento
           putStrLn "Visialize o ficheiro escalonamento.txt para visualizar as alteraçoes feitas!"
```

Figura19. Função "ponto1".

10. Passo: **Criar as funções relacionadas às alíneas 2) e 4).** A primeira a criar é uma função para buscar os alunos da uc sendo esta denominada por **"getAlunosInscritosponto2"**. A função recebe uma uc e uma lista de pares (UC, [Aluno]), onde cada par representa uma uc e a lista de alunos inscritos nessa uc. Após isso é usada a condição **'lookup'** para encontrar o par na lista que corresponde à uc dada. A função 'lookup' retorna um valor **'Maybe [Aluno]'**, que é **'Just [Aluno]'** se a uc foi encontrada na lista e 'Nothing' se a uc não foi encontrada. Finalmente, a função usa **'fromMaybe []'** para converter o valor **'Maybe [Aluno]'** em um valor **'[Aluno]'**. Se a uc foi encontrada na lista, **'fromMaybe []'** retorna a lista de alunos inscritos na uc. Se a uc não foi encontrada, **'fromMaybe []'** retorna uma lista vazia.

```haskell
getAlunosInscritosponto2 :: UC -> [(UC, [Aluno])] -> [Aluno]
getAlunosInscritosponto2 uc lst = fromMaybe [] (lookup uc lst)
```

Figura20. Função "getAlunosInscritosponto2".

11. Passo: Criar a função **"calcIncompatibilidades".** Esta função para cada par de ucs (uc1, uc2), utiliza a lista de alunos inscritos em cada uc usando a função **"getAlunosInscritosponto2"**, e calcula a interseção dessas duas listas (ou seja, os alunos que estão inscritos em ambas as ucs), posto isto conta o número de alunos na interseção. A função retorna uma lista de todas as incompatibilidades.

```haskell
calcIncompatibilidades :: [(UC, [Aluno])] -> [(UC, UC, Int)]
calcIncompatibilidades inscricoes =
  let ucs = map fst inscricoes
      paresUCs = [(uc1, uc2) | uc1 <- ucs, uc2 <- ucs, uc1 /= uc2]
      incompatibilidades = map (\(uc1, uc2) -> (uc1, uc2, length (intersect (getAlunosInscritosponto2 uc1 inscricoes) (getAlunosInscritosponto2 uc2 inscricoes)))) paresUCs
  in incompatibilidades
```

Figura21. Função "calcIncompatibilidades".

12. Passo: Criar uma função **“ponto2”** responsável por calcular e exibir as incompatibilidades entre diferentes unidades curriculares (ucs) baseado nas inscrições dos alunos.
    Primeiramente esta usa a função **‘calcIncompatibilidades’** para calcular as incompatibilidades entre todas as ucs na lista de entrada. Cada incompatibilidade é representada por uma tupla contendo duas ucs e o número de alunos inscritos em ambas. De seguida, ela imprime uma string para indicar que as incompatibilidades estão prestes a serem impressas. Finalmente, usa **‘mapM_’** para aplicar uma função a cada elemento na lista de incompatibilidades. Por fim, imprime uma string que descreve as incompatibilidades, incluindo as duas ucs e o número de alunos inscritos em ambas.

```
ponto2 :: [(String, [String])] -> IO ()
ponto2 a = do
  let incompatibilidades = calcIncompatibilidades a
  putStrLn "Incompatibilidades entre pares de UCs:"
  mapM_ (\(uc1, uc2, incompat) -> putStrLn $ "UCs " ++ uc1 ++ " e " ++ uc2 ++ ": " ++ show incompat ++ " alunos inscritos em ambas") incompatibilidades
```

Figura22. Função “ponto2”.

3. Resultados

Vamos exibir todos os resultados através do terminal da consola:

1. Introdução de número de dias, de salas e lugares disponíveis para os exames escolhidos pelo utilizador:

```
**********MENU**********

->INTRODUZA O NUMERO DE DIAS EM QUE VAO OCORRER OS EXAMES

10

->INTRODUZA O NUMERO DE SALAS EXISTENTES

20

->INTRODUZA O NUMERO DE LUGARES DISPONIVEIS EM CADA SALA

30
```

Figura23. Menu.

2. Indicar a opção que prefere:

```
Indique a opção que prefere
        1->Criar ficheiro para Escalonamento
        2->Apresentação de incompatibilidades
```

Figura24. Opções para criar ficheiro para escalonamento e apresentar incopatibilidades.

2.1 Opção 1→

```
1
Visialize o ficheiro escalonamento.txt para visualizar as alteraçoes feitas!
```

Figura25. Opção 1 escolhida.

toprog.hs ● | escalonamento.txt M ×

lonamento.txt

```
Exame {uc = "Algebra linear", salas = ["Sala1"], dia = "Dia1", ano = 1, alunos = ["al001","al002","al003","al006","a
Exame {uc = "Analise matematica I", salas = ["Sala1"], dia = "Dia2", ano = 1, alunos = ["al004","al007"]}
Exame {uc = "Laboratorio de informatica e computadores", salas = ["Sala1"], dia = "Dia3", ano = 1, alunos = ["al001"
Exame {uc = "Topicos de matematica discreta", salas = ["Sala1"], dia = "Dia4", ano = 1, alunos = ["al001","al002","a
Exame {uc = "Programacao procedimental", salas = ["Sala1"], dia = "Dia5", ano = 1, alunos = ["al003","al005","al008"
Exame {uc = "Seminario de informatica", salas = ["Sala1"], dia = "Dia6", ano = 1, alunos = ["al003","al005","al008",
Exame {uc = "Analise matematica II", salas = ["Sala1"], dia = "Dia7", ano = 1, alunos = ["al003","al005","al008","al
Exame {uc = "Elementos de fisica geral", salas = ["Sala1"], dia = "Dia8", ano = 1, alunos = ["al004","al007"]}
Exame {uc = "Engenharia nas organizacoes", salas = ["Sala1"], dia = "Dia9", ano = 1, alunos = ["al004","al007"]}
Exame {uc = "Laboratorio de programacao", salas = ["Sala1"], dia = "Dia10", ano = 1, alunos = []}
Exame {uc = "Programacao funcional", salas = ["Sala2"], dia = "Dia1", ano = 1, alunos = []}
Exame {uc = "Sistemas computacionais", salas = ["Sala2"], dia = "Dia2", ano = 1, alunos = []}
Exame {uc = "Compiladores", salas = ["Sala2"], dia = "Dia3", ano = 2, alunos = []}
Exame {uc = "Engenharia de software", salas = ["Sala2"], dia = "Dia4", ano = 2, alunos = []}
Exame {uc = "M\9500\174todos estat\9500\161sticos", salas = ["Sala2"], dia = "Dia5", ano = 2, alunos = []}
Exame {uc = "Programacao orientada a objetos", salas = ["Sala2"], dia = "Dia6", ano = 2, alunos = []}
Exame {uc = "Sistemas operativos", salas = ["Sala2"], dia = "Dia7", ano = 2, alunos = []}
Exame {uc = "Algoritmia e estruturas de dados", salas = ["Sala2"], dia = "Dia8", ano = 2, alunos = []}
Exame {uc = "Bases de dados", salas = ["Sala2"], dia = "Dia9", ano = 2, alunos = []}
Exame {uc = "Comunicacao de dados", salas = ["Sala2"], dia = "Dia10", ano = 2, alunos = []}
Exame {uc = "Laboratorio de planeamento e desenvolvimento de software", salas = ["Sala3"], dia = "Dia1", ano = 2, al
Exame {uc = "Metodos computacionais em engenharia", salas = ["Sala3"], dia = "Dia2", ano = 2, alunos = []}
Exame {uc = "Programacao multiplataforma", salas = ["Sala3"], dia = "Dia3", ano = 2, alunos = []}
Exame {uc = "Engenharia web", salas = ["Sala3"], dia = "Dia4", ano = 3, alunos = []}
Exame {uc = "Gestao de projetos em engenharia", salas = ["Sala3"], dia = "Dia5", ano = 3, alunos = []}
Exame {uc = "Inteligencia artificial", salas = ["Sala3"], dia = "Dia6", ano = 3, alunos = []}
```

Figura26. Ficheiro escalonamento.txt após as alterações realizadas.

2.2 Opção 2→

projetoprog.hs ● | escalonamento.txt M

projetoprog.hs > ...

```
201
202   -- Função auxiliar para buscar os alunos inscritos em uma UC
203   getAlunosInscritosponto2 :: UC -> [(UC, [Aluno])] -> [Aluno]
204   getAlunosInscritosponto2 uc lst = fromMaybe [] (lookup uc lst)
205
206   -- Função para calcular as incompatibilidades entre cada par de UCs
207   calcIncompatibilidades :: [(UC, [Aluno])] -> [(UC, UC, Int)]
208   calcIncompatibilidades inscricoes =
209     let ucs = map fst inscricoes
210         paresUCs = [(uc1, uc2) | uc1 <- ucs, uc2 <- ucs, uc1 /= uc2]
211         incompatibilidades = map (\(uc1, uc2) -> (uc1, uc2, length (intersect (getAlunosInscr
212     in incompatibilidades
213
214   -- Exemplo de uso
215   ponto2 :: [(String, [String])] -> IO ()
```

PROBLEMS 8 | OUTPUT | DEBUG CONSOLE | TERMINAL

```
UCs Topicos de matematica discreta e Laboratorio de programacao: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Programacao funcional: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Sistemas computacionais: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Compiladores: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Engenharia de software: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e M├®todos estat├¡sticos: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Programacao orientada a objetos: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Sistemas operativos: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Algoritmia e estruturas de dados: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Bases de dados: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Comunicacao de dados: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Laboratorio de planeamento e desenvolvimento de software: 0 alunos
UCs Topicos de matematica discreta e Metodos computacionais em engenharia: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Programacao multiplataforma: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Engenharia web: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Gestao de projetos em engenharia: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Inteligencia artificial: 0 alunos inscritos em ambas
UCs Topicos de matematica discreta e Laboratorio de aplicacoes web e bases de dados: 0 alunos inscrito
UCs Topicos de matematica discreta e Redes de computadores: 0 alunos inscritos em ambas
```

Figura27. Terminal a apresentar as incompatibilidades a partir da opção 2 selecionada.

4. Conclusão

Em suma, o programa desenvolvido apresenta um menu intuitivo e funcional que permite a construção de calendários de exames e a identificação de conflitos entre cadeiras. No entanto, é importante realçar que houve uma limitação na conclusão da parte II deste projeto, conforme solicitado no protocolo, principalmente devido à falta de tempo disponível. É possível que algumas etapas ou funcionalidades não tenham sido implementadas ou finalizadas como planejado. Apesar dessa lacuna, o programa desenvolvido ainda oferece funcionalidades úteis para a calendarização de exames.

5. Web-bibliografia

Fontes URL:

http://learnyouahaskell.com/chapters - (acedido sucessivamente de 31 de maio de 2023 a 4 de junho de 2023)

https://www.fpcomplete.com/haskell/library/containers/-(acedido dia 31 de maio de 2023 e 2 de junho de 2023)

https://stackoverflow.com/questions/5726445/how-would-you-define-map-and-filter-using-foldr-in-haskell - -(acedido de 30 de maio de 2023 e 1 de junho de 2023)

http://haskell.tailorfontela.com.br/making-our-own-types-and-typeclasses - - (acedido 3 junho de 2023)