Nome: Jorge Messa Junior

R.A.: 11069411

Tema: Implementação do Algoritmo para encontrar o Emparelhamento máximo em um Grafo não-dirigido bipartido.

INTRODUÇÃO

O algoritmo para calcular o emparelho máximo em grafos não-dirigidos bipartidos é utilizado para solucionar problemas comuns como determinar se um conjunto M por ser “casado” com um conjunto N, de modo que nenhum elemento de um conjunto M fique “solteiro”, como segue na imagem abaixo por exemplo, em que o conjunto M é representado pelos nós em azul, e o conjunto N, pelos nós em laranja. O conceito de emparelhamento é atrelado a inexistência de arestas sem pontas em comum. Ou seja, dois nós não podem compartilhar o mesmo lado de uma aresta.

Um emparelhamento M é máximo, se não existe outro emparelhamento M’ maior tal que (|M’|>|M|).

Neste exemplo da figura 1 abaixo, o emparelhamento máximo é 2, já que apenas 2 nós podem ser “casados”, já tanto o nó 0 e 1 estão ligados ao 4.

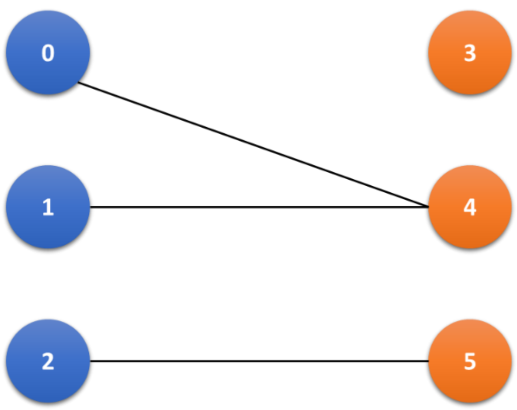


***Figura 1**: um exemplo, onde os nós do conjunto M não estão todos “casados”.*

Na figura 2 abaixo, é mostrado um emparelhamento perfeito, em que todos os nós estão emparelhados.

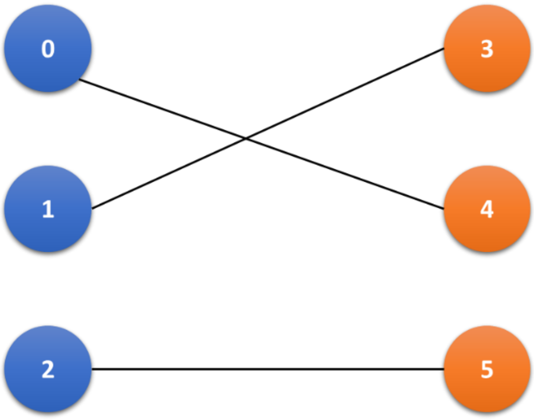


Figura 2: exemplo de emparelhamento máximo perfeito, em que todos nós estão emparelhados.

SOLUÇÃO

Para encontrar um emparelhamento que seja máximo, é necessário, utilizar de dois conceitos: caminho alternante e caminho aumentador.

Caminho alternante

Se v e w são vértices consecutivos numa sequência, então a aresta v-w pertence ao grafo. Vértices ‘a’, ‘b’, ‘c’, ‘d’, e ‘e’ podem ser representados pela aresta a-b-c-d-e.

Um caminho é dito simples, se os vértices não se repetem durante todo caminho.

É considerado um caminho alternante em relação a um emparelhamento M, se for simples (não possui vértices repetidos) e se suas arestas estiverem alternadamente em M e fora de M.

Caminho aumentador

Um caminho aumentador é um caminho alternante que começa em um vértice solteiro e termina num vértice solteiro, com um comprimento maior que 0.

Dado M um emparelhamento e, P um caminho aumentador, temos que $M \oplus P$ é um emparelhamento maior que M.

Algoritmo

O algoritmo faz uma busca largura em cada elemento de um dos dois conjuntos do grafo bipartido, utilizando-se de um Fila.

A cada vértice que está na fila, são analisados seus vizinhos e um emparelhamento é tentado.

Isso é feito até que nenhum caminho aumentador é encontrado, encerrando o algoritmo.

O trecho de código que calcula M⊕P, pode ser visto abaixo:

```
novoEmparelhamento::(Num a, Eq a, Ord a)  => [a]->[a]->a->[a]
novoEmparelhamento pa xs t = if t_1 == x then emp'' else
novoEmparelhamento pa emp'' t_1
  where
    x = retornaElemento t pa
    emp' = atualizaLista t x xs
    emp'' = atualizaLista x t emp'
    t_1 = retornaElemento x pa
```

Em cada iteração, uma aresta t-x entra no emparelhamento e a aresta x-pa[x] sai do emparelhamento

```
 v   x        v   x
\ / \        \ / \
w   t        w   t
```

O restante do código pode ser visto em:

https://github.com/jorgemessajr/paradigmas_prog/blob/master/Projeto-Final/src/Main.hs

CASOS DE EXEMPLO:

Um caso de exemplo que pode ser analisado é o caso abaixo:

M trabalhadores (em azul) querem aplicar para N empregos (em laranja).

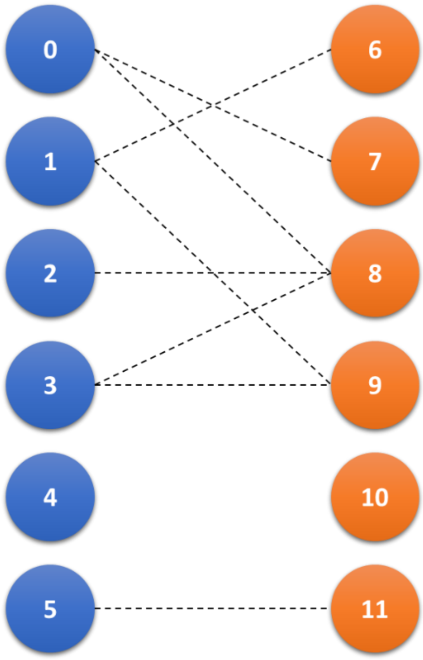


Figura 3: representação do grafo de candidatos x emprego.

Executando o trecho abaixo:

Neste caso

Branco = Azul e **Laranja = Preto**

```
let elem1 = Elemento ((0,Branco),[7,8])
let elem2 = Elemento  ((1,Branco),[6,9])
let elem3 = Elemento  ((2,Branco),[8])
let elem4 = Elemento  ((3,Branco),[8,9])
let elem5 = Elemento  ((4,Branco),[])
let elem6 = Elemento  ((5,Branco),[11])
let elem7 = Elemento  ((6,Preto),[])
let elem8 = Elemento  ((7,Preto),[])
let elem9 = Elemento  ((8,Preto),[])
let elem10 = Elemento  ((9,Preto),[])
let elem11 = Elemento ((10,Preto),[])
let elem12 = Elemento ((11,Preto),[])

let teste = [elem1, elem2, elem3, elem4, elem5, elem6, elem7, elem8, elem9,
elem10, elem11, elem12]
let grafo = Grafo (teste)

let match = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]

emparelhamentoMaximo grafo match Branco 0
```

```
Resposta:
```

```
(5,[7,6,8,9,-1,11,1,0,2,3,-1,5])
```

A primeira posição do resultado se refere ao tamanho do emparelhamento, e a segunda é a representação do emparelhamento como um vetor - **match**, tal que, se um vértice **v** está casado com um vértice **w**, temos **match[v] = w** e **match[w] =v**.

No máximo 5 trabalhadores vão conseguir aplicar para uma posição de emprego, resultado no grafo abaixo:

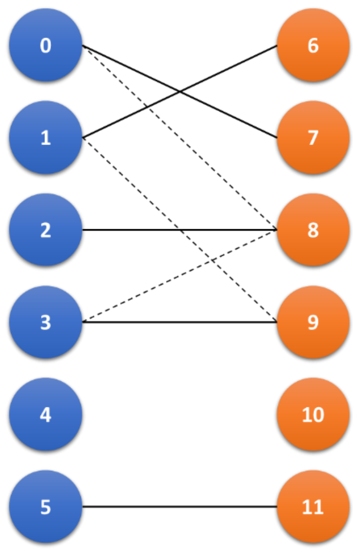


Figura 4: emparelhamento máximo de candidatos x emprego

Outro exemplo que temos é no caso em que pessoas querem usar camisetas que lhe servem, como segue abaixo:

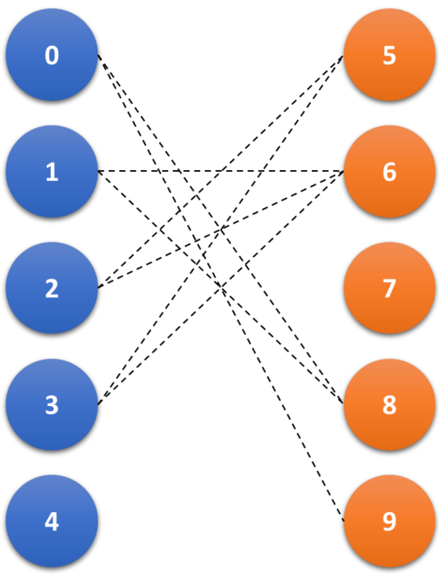


Figura 5: representação de pessoas querem vestir camisetas

Inserindo o trecho abaixo, obtemos:

Neste caso

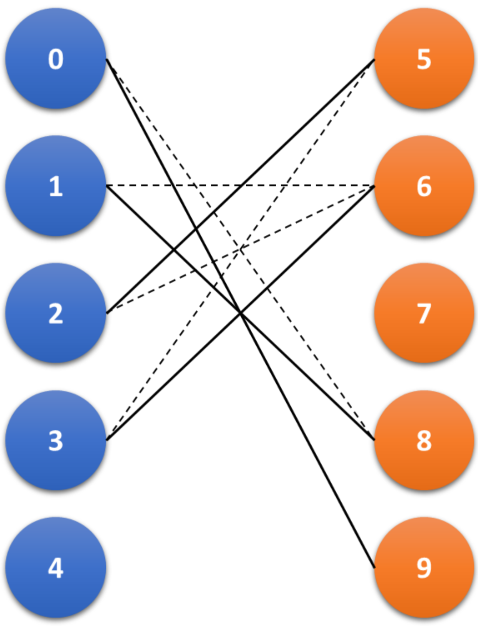
Branco = Azul e **Laranja = Preto**

```
let elem1 = Elemento((0,Branco),[8,9])
let elem2 = Elemento((1,Branco),[6,8])
let elem3 = Elemento((2,Branco),[6,5])
let elem4 = Elemento((3,Branco),[6,5])
let elem5 = Elemento((4,Preto),[])
let elem6 = Elemento((5,Preto),[])
let elem7 = Elemento((6,Preto),[])
let elem8 = Elemento((7,Preto),[])
let elem9 = Elemento((8,Preto),[])
let elem10 = Elemento((9,Preto),[])

let teste = [elem1, elem2, elem3, elem4, elem5, elem6, elem7, elem8, elem9,
elem10]

let grafo = Grafo (teste)

let match = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]


emparelhamentoMaximo grafo match Branco 0
```

Obtendo

```
(4,[9,8,5,6,-1,2,3,-1,1,0])
```

Que representa o grafo abaixo:



REFERÊNCIAS

IME-USP. **Emparelhamentos em grafos não-dirigidos bipartidos.** Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/matching-bipartite.html. Último acesso em 14/08/2018.

FRANÇA, Olivetti Fabrício. Notas de aula. Disponível em: https://folivetti.github.io/courses/ParadigmasProgramacao/. Último acesso em: 14/08/2018