

Formalização e Prova de algoritmos de menor caminho usando Coq

João Vitor Fröhlich

Universidade do Estado de Santa Catarina
joaovitorfrohlich@gmail.com

Orientadora: Dra Karina Girardi Roggia

20/06/2023

Apoio:



Sumário

- 1 Introdução
- 2 Objetivos
- 3 Teoria de Grafos
- 4 Algoritmos
- 5 Coq
- 6 Implementação
- 7 Conclusões Parciais
- 8 Referências



- Formalização e Prova
 - Especificação Formal
- Algoritmos de menor caminho
 - Teoria de Grafos
- Coq
 - Assistentes de Provas



O objetivo geral deste trabalho é formalizar e provar em Coq algoritmos de busca do menor caminho entre dois pontos em grafos.



Objetivos Específicos

- ① Estudar os principais algoritmos determinísticos de busca do menor caminho de grafos
- ② Estudar os principais algoritmos heurísticos de busca do menor caminho em grafos
- ③ Implementar alguns algoritmos de busca do menor caminho em assistente de provas
- ④ Provar a corretude da implementação dos algoritmos definidos



- $G = \langle V, E, \delta_0, \delta_1 \rangle$;
 - Restrição de laço: $\forall e \in E, \delta_0(e) \neq \delta_1(e)$
 - Restrição de aresta paralela:
 $\forall e_1, e_2 \in E, \delta_0(e_1) = \delta_0(e_2) \wedge \delta_1(e_1) = \delta_1(e_2) \implies e_1 = e_2$
- Vizinhança:
 $\exists e \in E, (\delta_0(e) = u \wedge \delta_1(e) = v) \vee (\delta_0(e) = v \wedge \delta_1(e) = u)$
- Diretamente alcançável: $\delta_0(e) = u \wedge \delta_1(e) = v$



Exemplo - Grafo direcionado

- $G = \langle V, E, \delta_0, \delta_1 \rangle$;
- $V = \{v_1, v_2, v_3, v_4\}$;
- $E = \{e_1, e_2, e_3, e_4, e_5\}$;

	δ_0	δ_1
e1	v1	v2
e2	v1	v3
e3	v1	v4
e4	v2	v3
e5	v3	v2

Tabela: Definição de δ_0 e δ_1 no grafo de exemplo



Exemplo - Grafo direcionado

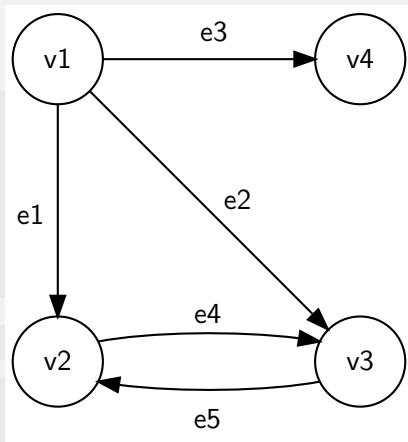


Figura: Grafo direcionado

Fonte: O autor



Definições e Exemplo - Grafo direcionado ponderado

- Adição de uma função $\varphi : E \rightarrow \mathbb{R}^+$
- $G = \langle V, E, \delta_0, \delta_1, \varphi \rangle$;

	φ
e1	2.5
e2	1.0
e3	5.0
e4	5.0
e5	0.5

Tabela: Definição de φ no exemplo 2



Exemplo - Grafo direcionado ponderado

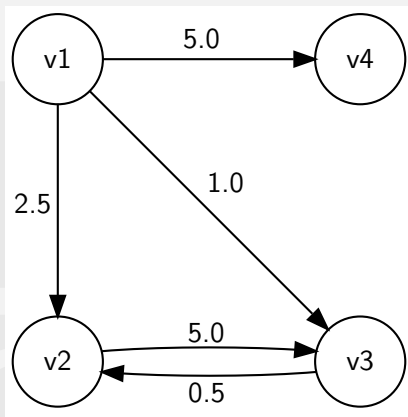


Figura: Grafo direcionado ponderado

Fonte: O autor



- Uma lista não vazia de arestas: $C = [e_1, e_2, \dots, e_n]$;
 - $\forall i \in \{1, \dots, n-1\}, \delta_1(e_i) = \delta_0(e_{i+1})$
- As funções δ_0 , δ_1 e φ (em grafos ponderados) podem ser definidas para cada caminho finito C no grafo, onde
 - $\delta_0(C) = \delta_0(e_1)$
 - $\delta_1(C) = \delta_1(e_n)$
 - $\varphi(C) = \sum_{i=1}^{|C|} \varphi(e_i)$
- Ciclos: $\delta_0(C) = \delta_1(C)$
- Menor caminho de u para v : C' tal que
 $\forall C (\delta_0(C) = u \wedge \delta_1(C) = v), \varphi(C') = \min(\varphi(C))$



Representação - Matriz de Adjacência

- Matriz $n \times n$, onde n é o número de vértices do grafo
- O valor de cada célula $M_{i,j}$ é calculado como

$$M_{i,j} = \begin{cases} \varphi(e) & \forall e \mid \delta_0(e) = i \wedge \delta_1(e) = j \\ 0 & \text{se } i = j \\ -1 & \text{caso contrário} \end{cases} \quad (1)$$

	v_1	v_2	v_3	v_4
v_1	0	2.5	1.0	5.0
v_2	-1	0	5.0	-1
v_3	-1	0.5	0	-1
v_4	-1	-1	-1	0

Tabela: Matriz de adjacência



Representação - Listas de Adjacência

- Lista de pares
- Definida para cada vértice do conjunto V de um grafo ponderado
- $L_i = [(\delta_1(e), \varphi(e))], \forall e \mid \delta_0(e) = i$

$$L_1 = [(v_2, 2.5); (v_3, 1.0); (v_4, 5.0)]$$

$$L_2 = [(v_3, 5.0)]$$

$$L_3 = [(v_2, 0.5)]$$

$$L_4 = []$$



- Algoritmos usados para buscar todos os vértices de um grafo
- Um vértice é dito descoberto quando ele é visitado pela busca
- Os vértices são representados como $v = \langle L_v, \pi_v, C_v \rangle$
- DFS
 - Sendo v o vértice descoberto mais recentemente, uma busca é feita nas arestas partindo de v por vértices não descobertos
 - Quando um novo vértice é descoberto, a busca passa a ser realizada nesse vértice
 - Quando um vértice sendo explorado não alcança diretamente nenhum vértice inexplorado, a busca é encerrada nesse vértice
- BFS
 - Armazena os vértices recém descobertos em uma fila
 - Para cada vértice u diretamente alcançável a partir de v , u é inserido no final da fila
 - Após explorar cada aresta partindo de v , v é removido da fila



Algoritmo para resolver o problema do caminho mínimo de um vértice de um grafo direcionado ponderado para qualquer outro vértice (CORMEN et al., 2022)

- Seja G um grafo, V o conjunto de vértices de G , $o' \in V$ um vértice de origem
- Seja S um conjunto de vértices que já possuem o menor caminho determinado de o' até si.
- Selecionar um vértice $v \in V - S$ de acordo com $\varphi_V(v)$
- Adicionar v a S e relaxar o vértice v
- Executar até que $V - S$ esteja vazio



- Utiliza uma função heurística h que calcula uma aproximação de cada vértice até o vértice destino
- $\forall v, h(v) = 0$ é equivalente ao algoritmo de Dijkstra
- Uma função heurística muito comum é a distância euleriana



- Programas que auxiliam no desenvolvimento de provas formais (SILVA, 2019)
- No começo não eram muito aceitos, mas começaram a se mostrar confiáveis com o passar do tempo (GEUVERS, 2009)
- Teorema das quatro cores
- Conceito de De Bruijn (BARENDREGT; GEUVERS, 2001)
- Vantagens:
 - A verificação mecânica é rápida e confiável
 - As provas são processadas interativamente
 - Motor de busca para teoremas e lemas já provados
 - Automatização de provas com métodos não deterministas
 - Permite a extração de programas para outras linguagens



- Assistente de provas que usa o Cálculo de Construções Indutivas (BERTOT; CASTÉLAN, 2013);
- Dividido em quatro componentes:
 - A linguagem de programação e especificação *Gallina*, que implementa o Cálculo de Construções Indutivas.
 - A linguagem de comandos *vernacular*, que faz a interação com o assistente.
 - Um conjunto de táticas para realização das provas, que são traduzidas para termos em *Gallina*.
 - A linguagem *Ltac* para implementar novas táticas e automatizar provas.



- Começou a ser desenvolvida a partir da prova do teorema das quatro cores
- Formaliza teorias da matemática, desde estruturas básicas até tópicos avançados da álgebra (MAHBOUBI; TASSI, 2022)
- Implementa algumas mudanças sintáticas ao Coq



- Foi implementada uma formalização de caminhos ponderados
 - Os pesos foram restritos aos números naturais
 - Os códigos estão disponíveis em [<https://github.com/joao-frohlich/dijkstra-coq>](https://github.com/joao-frohlich/dijkstra-coq)
- As seguintes propriedades foram provadas
 - ① Todo caminho é positivo
 - ② A adição de um vértice ao início de um caminho qualquer C faz com que o peso deste novo caminho seja igual ao peso de C somado ao peso definido para a aresta que liga este novo vértice ao primeiro vértice de C ;
 - ③ O peso da concatenação de dois caminhos C_1 e C_2 , desde que obedecendo às restrições impostas pela função de concatenação de caminhos, será igual à soma do peso de C_1 com o peso de C_2 .



```
1 From mathcomp Require Import all_ssreflect.
2 Set Implicit Arguments.
3 Unset Strict Implicit.
4 Unset Printing Implicit Defensive.
5
6 Section CaminhoPonderado.
7   Variable T : finType.
8   Variable  $\varphi$  : T -> T -> nat.
```

Figura: Definições de caminhos ponderados



```
10 Definition  $\varphi'$  (x y : T) :=
11   if x == y then 0
12   else  $\varphi$  x y.
13
14 Fixpoint  $\varphi\_C$  (c : seq T) : nat :=
15   match c with
16   | [::] => 0
17   | x :: c' =>  $\varphi'$  x (head x c') +  $\varphi\_C$  c'
18   end.
19
20 Definition concat_caminho
21   (x1 x2 : T)
22   (s1 s2 C1 C2 : seq T) : seq T :=
23   if (C1 == x1 :: s1) && (C2 == x2 :: s2)
24   | && (last x1 s1 == x2) then C1 ++ s2
25   else [::].
```

Figura: Definições de caminhos ponderados



```
Lemma peso_mesmo_vertice (x : T) :  $\varphi'$  x x = 0.  
  
Lemma peso_concat_seq (x1 x2 : T) (s1 s2 : seq T) :  
   $\varphi_C ((x1 :: s1) ++ (x2 :: s2)) =$   
   $\varphi_C (x1 :: s1) + \varphi_C (x2 :: s2) + \varphi' (\text{last } x1 \ s1) \ x2.$ 
```

Figura: Declaração dos lemas auxiliares



```
Lemma peso_caminho_positivo (c : seq T) : 0 <=  $\varphi_C$  c.
```

```
Lemma peso_cons_caminho (x : T) (c : seq T) :  
   $\varphi_C$  (x :: c) =  $\varphi'$  x (head x c) +  $\varphi_C$  (c).
```

```
Lemma peso_concat_caminho (x1 x2 : T) (s1 s2 C1 C2 : seq T) :  
  C1 = x1 :: s1 -> C2 = x2 :: s2 -> last x1 s1 == x2 ->  
   $\varphi_C$  (concat_caminho x1 x2 s1 s2 C1 C2) =  
   $\varphi_C$  C1 +  $\varphi_C$  C2.
```

Figura: Declaração dos lemas



- A modelagem de grafos no Coq ainda está sendo estudada
- Resultados obtidos da implementação são positivos e indicam que é possível uma modelagem de grafos ponderados em Coq.



- ① Modelar grafos ponderados no Coq;
- ② Implementar e formalizar o Algoritmo de Dijkstra;
- ③ Estudar heurísticas que geram o menor caminho na Busca A*;
- ④ Implementar e formalizar o Algoritmo de Busca A*.





Etapas	2023/1	2023/2				
	Jul	Ago	Set	Out	Nov	Dez
1						
2						
3						
4						


Tabela: Cronograma Proposto para o TCC2




 BARENDREGT, H.; GEUVERS, H. Proof-assistants using dependent type systems. In: *Handbook of automated reasoning*. [S.l.: s.n.], 2001. p. 1149–1238.


 BERTOT, Y.; CASTÉLAN, P. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. [S.l.]: Springer Science & Business Media, 2013.

 CORMEN, T. H. et al. *Introduction to algorithms*. [S.l.]: MIT press, 2022.

 GEUVERS, H. Proof assistants: History, ideas and future. *Sadhana*, Springer, v. 34, p. 3–25, 2009.

 MAHBOUBI, A.; TASSI, E. *Mathematical Components*. Zenodo, 2022. Disponível em: <<https://doi.org/10.5281/zenodo.7118596>>.



 SILVA, R. C. G. *Uma certificação em COQ do algoritmo W monádico*. 2019. 78 p. Dissertação (Mestrado) — Universidade do Estado de Santa Catarina, Programa de Pós Graduação em Computação Aplicada, 2019.

