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

23/06/2023

Apoio:
CNPq

Sumário

Introdução

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

Objetivo Geral

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

Definições - Grafo direcionado

- $G = \langle V, E, \delta_0, \delta_1 \rangle$;
  - 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

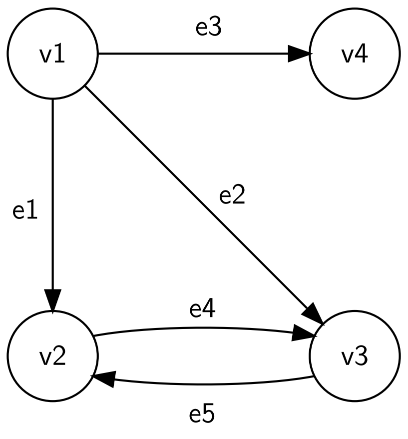

Figura: Grafo direcionado

Fonte: O autor

- 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

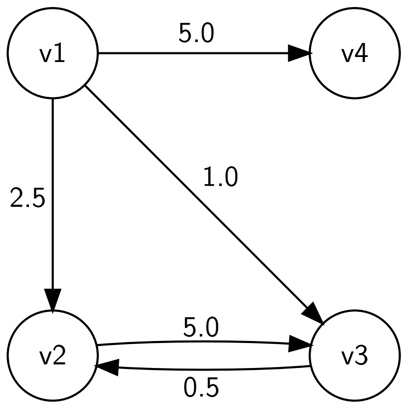

Figura: Grafo direcionado ponderado

Fonte: O autor

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

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

$$
\begin{aligned}
L_1 &= [(v_2, 2.5); (v_3, 1.0); (v_4, 5.0)] \\
L_2 &= [(v_3, 5.0)] \\
L_3 &= [(v_2, 0.5)] \\
L_4 &= []
\end{aligned}
$$

Algoritmos - DFS e BFS

- 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 as arestas partindo do vértice v
- Executar até que $V - S$ esteja vazio

Algoritmos - A*

- 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
- 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

Coq

- Assistente de provas que usa o Cálculo de Construções Indutivas (BERTOT; CASTÉRAN, 2013);
- Dividido em quatro componentes (SILVA, 2019):
  - 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 $\mathcal{L}$tac para implementar novas táticas e automatizar provas.

- Biblioteca de formalismo matemático para o Coq
- 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>
- 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.

```
From mathcomp Require Import all_ssreflect.
Set Implicit Arguments.
Unset Strict Implicit.
Unset Printing Implicit Defensive.

Section CaminhoPonderado.
    Variable T : finType.
    Variable φ : T -> T -> nat.
```

Figura: Definições de caminhos ponderados

```
Definition φ' (x y : T) :=
    if x == y then 0
    else φ x y.

Fixpoint φ_C (c : seq T) : nat :=
    match c with
    | [::] => 0
    | x :: c' => φ' x (head x c') + φ_C c'
    end.

Definition concat_caminho
    (x1 x2 : T)
    (s1 s2 C1 C2 : seq T) : seq T :=
    if (C1 == x1 :: s1) && (C2 == x2 :: s2)
        && (last x1 s1 == x2) then C1 ++ s2
    else [::].
```

Figura: Definições de caminhos ponderados

Implementação - Lemas Auxiliares

```
Lemma peso_mesmo_vertice (x : T) : φ' x x = 0.

Lemma peso_concat_seq (x1 x2 : T) (s1 s2 : seq T) :
    φ_C ((x1 :: s1) ++ (x2 :: s2)) =
    φ_C (x1 :: s1) +  φ_C (x2 :: s2) + φ' (last x1 s1) x2.
```

Figura: Declaração dos lemas auxiliares

```
Lemma peso_caminho_positivo (c : seq T) : 0 <= φ_C c.

Lemma peso_cons_caminho (x : T) (c : seq T) :
    φ_C (x :: c) = φ' x (head x c) + φ_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 ->
    φ_C (concat_caminho x1 x2 s1 s2 C1 C2) =
    φ_C C1 + φ_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.

Cronograma

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

BERTOT, Y.; CASTÉRAN, 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.