Teoria dos Grafos

Algoritmos de Busca e Caminhos

Paulo Henrique Ribeiro Gabriel

Faculdade de Computação
Universidade Federal de Uberlândia

2019/1

Buscas em Grafos

Definição

Busca Busca em grafo é o processo de percorrer os vértices e arestas de um grafo de forma a resolver problemas que envolvam as informações contidas nos vértices e arestas, ou a detecção de estruturas nos grafos como articulações/pontes, conectividade, ciclos, cliques, e muitas outras

Algoritmos de Busca

- Algoritmos de buscas são os algoritmos que merecem mais destaque em algoritmos em grafos
- Em geral, podem ser adaptados para solucionar diversos problemas

Algoritmos de Busca

Problemas que podem ser resolvidos:

- Visita a todas as arestas e/ou vértices
- Conectividade
- Detecção de ciclos
- Determinação de componentes conexas
- Descoberta de elementos estruturais (pontes, articulações, blocos, etc.)
- Reconhecimento de grafos bipartido, árvores, etc.
- Caminho mínimo entre par de vértices
- Determinação de orientações acíclicas

Busca Genérica

Entrada: Grafo G e vértice $r \in V(G)$.

marcar r como visitado;
para $vw \in E(G)$ *tal que* v *é visitado e* vw *é não explorada* **faça**
    marcar vw como explorada;
    se w *é não visitado* **então**
        marcar w como visitado;
        marcar vw como descoberta;

Algoritmo 1: Procedimento $busca\_generica(G, r)$.

Aplicação: Decidir se o grafo é conexo

Entrada: Grafo G.
Saída: Valor booleano.

escolha um vértice de origem r;
$busca\_generica(G, r)$;
para cada $v \in V(G)$ **faça**
    se v *é não visitado* **então**
        retorna *falso*;
retorna *verdadeiro*;

Algoritmo 2: Função $grafo\_conexo(G)$.

Aplicação: Detecção de ciclos

Entrada: Grafo G.
Saída: Valor lógico.

escolha um vértice de origem r;
$busca\_generica(G, r)$;
para cada $vw \in V(G)$ **faça**
    se vw *é* *não descoberta* **então**
        retorna *verdadeiro*;
retorna *falso*;

Algoritmo 3: Função $tem\_ciclo(G)$.

Outras Aplicações

- Decisão se grafo é floresta
- Decisão se grafo é árvore
- Obter floresta geradora
- Obtenção de componentes

Questão em aberto

Como computar a condição?

$vw \in E(G)$ tal que v é visitado e vw é não explorada

Busca em Profundidade

- Seja o seguinte conjunto:

$$V' = \{v \in V(G) \mid \exists\, vw \in E(G), v \text{ é visitado}, vw \text{ é não explorado}\}$$

- Busca em profundidade: escolha $v \in V'$ tal que v é o vértice mais recentemente alcançado pela busca

Busca em Profundidade

Entrada: Grafo G e um vértice v.

marcar v como visitado;
para $w \in N(r)$ **faça**
se w *é visitado* **então**
se vw *é não explorada* **então**
marcar vw como explorada;
senão
marcar vw como explorada;
marcar vw como descoberta;
$busca\_profundidade(G, w)$;

Algoritmo 4: Procedimento $busca\_profundidade(G, v)$.

Busca em Profundidade

Teorema

Seja T uma árvore de profundidade do grafo G. Se $vw \in E(G)$, então apenas uma das duas afirmações a seguir é verdadeira:

1. *vw é uma aresta de árvore em T*
2. *v é descendente não-filho de w em T ou vice-versa (chamada de aresta de retorno).*

Busca em Largura

- Seja o seguinte conjunto:

$$V' = \{v \in V(G) \mid \exists\, vw \in E(G), v \text{ é visitado}, vw \text{ é não explorado}\}$$

- Busca em largura: escolha $v \in V'$ tal que v é o vértice menos recentemente alcançado pela busca

Busca em Largura

Entrada: Grafo G e um vértice v.

crie uma fila F;
marcar v como visitado, inserir v em F;
para $F \neq \varnothing$ **faça**
    $v \leftarrow desenfileira(F)$;
    para $w \in N(v)$ **faça**
        se w *é visitado* **então**
            se vw *é não explorada* **então**
                marcar vw como explorada;
        senão
            marcar vw como explorada, marcar vw como descoberta;
            marcar w como visitado, inserir w em F;

Algoritmo 5: Procedimento $busca\_largura(G, v)$.

Busca em Largura

Teorema

Seja T uma árvore de largura do grafo G. Se $vw \in E(G)$, então apenas uma das duas afirmações a seguir é verdadeira:

1. vw *é aresta de árvore de* T
2. *nem* v *é descendente de* w *em* T*, nem vice-versa (chamada de aresta de cruzamento).*

Problema de Caminhos Mínimos

Definição (Caminho mínimo)

Caminho mínimo (ou distância) entre par de vértices de um grafo G com peso nas arestas, onde $p(vw)$ é o peso de uma aresta vw, é um caminho P tal que

$$custo(P) = \sum\{p(vw) \in E(P)\}$$

Problema de Caminhos Mínimos

Aplicações:

- Menor distância entre cidades
- Menor custo de transporte entre cidades
- Menor tempo para disseminar uma informação de um nó de uma rede a todos os demais

Algoritmo de Dijkstra

- Computa o caminho mínimo de um vértice s para todos os demais
- Pressupõe pesos não-negativos nas arestas
- Baseia-se no fato que, se $P = v_1, \ldots, v_k$ é o menor caminho entre v_1 e v_k, então $v_1, \ldots, v_{k-1}$ é o menor caminho entre v_1 e v_{k-1}

Algoritmo de Dijkstra

Observação

Caso houvesse um caminho P' entre v_1 e v_{k-1} menor que $v_1, \ldots, v_{k-1}$, então:

1. Se $v_k \in P'$, o sub-caminho de P' que vai de v_1 até v_k seria um caminho menor que P
2. Se $v_k \notin P'$, o caminho P', v_k seria um caminho menor entre v_1 e v_k que P

Algoritmo de Dijkstra

Entrada: Grafo G e vértice $s \in V(G)$.

$dist[s] \leftarrow 0$;
para cada *vértice* $v \in V(G) - \{s\}$ **faça**
    $dist(v) \leftarrow \infty$;
$S \leftarrow \varnothing$;
$Q \leftarrow V(G)$;
enquanto $Q \leq \varnothing$ **faça**
    $v \leftarrow \arg\min\{dist[v] \mid v \in Q\}$;
    $S \leftarrow S \cup \{v\}$;
    para cada $w \in N(v)$ **faça**
        $dist[w] = \min\{dist[w], dist[v] + p(v, w)\}$;
retorna $dist$;

Algoritmo 6: Função $dijkstra(G, s)$.

Algoritmo de Floyd

- O algoritmo de Floyd é mais geral que o de Dijkstra
- Determina o caminho mínimo entre quaisquer pares de vértices
- Para um grafo de n vértices, o algoritmo produz uma matriz quadrada de $n \times n$
  - Cada posição $[i, j]$ da matriz representa a distância entre os vértices i e j
  - Se o valor da distância for infinito, não há ligação possível entre esses vértices
  - A distância de um vértice a ele mesmo é 0

Algoritmo de Floyd-Warshall

- Inicialmente, o algoritmo define uma matriz D de distâncias da seguinte maneira:

$$D[i,j] = \begin{cases} 0, & \text{se } i = j \\ d(i,j), & \text{se existe uma aresta ligando } i \text{ e } j \\ \infty, & \text{caso contrário} \end{cases}$$

- Uma vez inicializada a matriz, inicia-se a iteração principal do algoritmo, mostrada a seguir

Entrada: Grafo G.

inicialize a matriz de distâncias D;

para $k \leftarrow 1$ **até** n **faça**

  para $i \leftarrow 1$ **até** n **faça**

    para $j \leftarrow 1$ **até** n **faça**

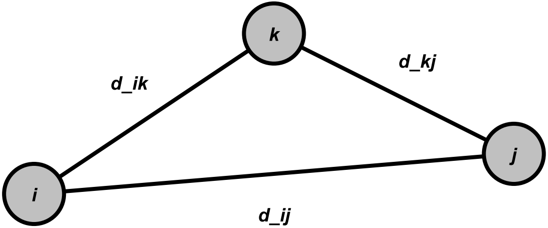
      $D[i,j] = \min\{D[i,j], D[i,k] + D[k,j]\}$;

retorna D;

Algoritmo 7: Função $floyd\_warshall(G)$.

Algoritmo de Floyd

- Esse algoritmo se baseia na chamada operação tripla
  - Seja três vértices i, j e k, conforme mostrados abaixo



  - Se $d(j,k) + d(k,j) < d(i,j)$, então o caminho i, k, j é considerado mais curto que i, j

Créditos

Parte deste material foi baseada nas notas de aula do Prof. Fabiano Oliveira.