Téc. Avançadas de Programação

Pseudocódigos

Prof. Leandro M. Zatesko

1º semestre de 2014

Algoritmos em Grafos

1 Fluxo em redes

Uma *rede* é uma estrutura $N = (G, s, t, c)$ em que G é um grafo dirigido, $s \in V(G)$ é um vértice com $d_G^-(s) = 0$, chamado de *origem*, $t \in V(G)$ é um vértice com $d_G^+(t) = 0$, chamado de *destino*, e c é uma função $c\colon E(G) \to \mathbb{R}^+$ que atribui a cada aresta uma *capacidade*. Um *fluxo* numa rede $N = (G, s, t, c)$ é uma função $f\colon E(G) \to \mathbb{R}$ tal que:

1. $0 \leqslant f(e) \leqslant c(e)$ para toda aresta $e \in E(G)$;
2. para todo vértice $u \in V(G)$ distinto de s e de t, $\sum_{e \in E_G^-(u)} f(e) = \sum_{e \in E_G^+(u)} f(e)$.

O *valor* de um fluxo f numa rede $N = (G, s, t, c)$ é a soma

$$\sum_{e \in E_G^+(s)} f(e) = \sum_{e \in E_G^-(t)} f(e).$$

O problema é o seguinte: *Dada uma rede, qual é o valor máximo de um fluxo na rede?* O seguinte algoritmo deve suas iniciais a Ford–Fulkerson–Edmonds–Karp e tem complexidade de tempo $O(|V(G)||E(G)|^2)$. No algoritmo, G_f denota o grafo residual de G por f: grafo dirigido munido de uma função de pesos nas arestas c_f tal que $V(G_f) = V(G)$ e:

(i) se $c(u,v) - f(u,v) > 0$, tanto (u,v), chamada de *aresta direta*, quanto (v,u), chamada de *aresta reversa*, são arestas de G_f, e $c_f(u,v) = c(u,v) - f(u,v)$ e $c_f(v,u) = f(u,v)$;

(ii) se $c(u,v) - f(u,v) = 0$, apenas a aresta reversa (v,u) é aresta de G_f, com $c_f(v,u) = f(u,v)$.

Um caminho P de s a t em G_f é chamado de *caminho f-aumentante*. Encontrar um caminho f-aumentante com número de arestas mínimo, como pede o algoritmo, pode ser feito por uma simples busca em largura em G_f.

FFEK(G,s,t,c):

para todo $e \in E(G)$, **faça**: $f(e) \leftarrow 0$;

enquanto existe P caminho f-aumentante em G_f com número de arestas mínimo, **faça**:

  encontre a capacidade δ do gargalo de P;

  para toda aresta (u_i, u_{i+1}) de P, **faça**:

    se $(u_i, u_{i+1}) \in E(G)$, **então**, $f(u_i, u_{i+1}) \leftarrow f(u_i, u_{i+1}) + \delta$;

    senão, $f(u_{i+1}, u_i) \leftarrow f(u_{i+1}, u_i) - \delta$;

devolva f.

Algoritmo 1

2 Emparelhamento em grafos bipartidos

Um *emparelhamento* num grafo G é um subconjunto M de $E(G)$ formado por arestas não-adjacentes. O *Problema do Emparelhamento Máximo em Grafos Bipartidos* é o seguinte: *Dado um grafo bipartido $G = (A \cup B, E)$, encontre um emparelhamento em G com o maior número possível de arestas.* Este problema pode ser resolvido com o *Algoritmo Húngaro*. No Algoritmo, um caminho $P = x_1 x_2 \cdots x_k$, $k \geqslant 1$, em G é M-*aumentante* se:

(i) x_1 e x_k não são cobertos por M;

(ii) $\{x_i, x_{i+1}\} \notin M$ para todo i par;

(iii) $\{x_i, x_{i+1}\} \in M$ para todo i ímpar;

HÚNGARO(A,B,E):

1 $M \leftarrow \emptyset$;

2 **se** existe $u \in A$ não coberto por M, **então**, $(S,T) \leftarrow (\{u\}, \emptyset)$;

3 **senão**, **devolva** M; // M cobre A

4 **se** $N(S) = T$, **devolva** M; // M não cobre A, mas é máximo

5 tome $b \in N(S) \setminus T$;

6 **se** existe $a \in V \setminus S$ tal que $\{a,b\} \in M$, **então**:

7   insira a em S;

8   insira b em T;

9   **retorne à linha** 4;

10 **senão**:

11   encontre um caminho P M-aumentante de u a b;

12   $M \leftarrow (M \cup E(P)) \setminus (M \cap E(P))$;

13   **retorne à linha** 2.

Algoritmo 2

Este algoritmo também é conhecido como Algoritmo de Edmonds, já que Edmonds melhorou a performance do algoritmo e o generalizou para grafos com pesos nas arestas em 1967. No entanto, o algoritmo original é de 1955 e não é de Edmonds, mas de Harold Kuhn, que se baseou em trabalhos ainda mais anteriores de matemáticos húngaros.

A lógica do algoritmo começar com um emparelhamento M vazio e, a cada iteração, tentar encontrar um caminho M-aumentante para substituir M por um emparelhamento com uma aresta a mais. Para procurar o caminho M-aumentante numa iteração para um vértice não-coberto $u \in A$, o algoritmo vai armazenando em $S \subseteq A$ e $T \subseteq B$ vértices alcançáveis a partir de u até encontrar algum $b \notin T$ tal que todo $a \in N_G(b)$ coberto por M já está em S. O algoritmo segue do *Teorema de Hall*: *Num grafo bipartido $G = (A \cup B, E)$ existe emparelhamento M que cobre A se e somente se $|N_G(S)| \geqslant |S|$ para todo $S \subseteq A$.*

O Problema do Emparelhamento Máximo em Grafos Bipartidos também pode ser resolvido com o Algoritmo de Ford–Fulkerson–Edmonds–Karp. Dado um grafo bipartido $(A \cup B, E)$ em que A e B são conjuntos independentes, podemos construir uma rede do seguinte modo:

1. Adicione dois vértices artificiais s (superorigem, com aresta *para* todo vértice de A) e t (superdestino, com aresta *de* todo vértice de B) ao grafo.

2. Torne todas as arestas entre A e B dirigidas *de A para B*.

3. Estabeleça que o peso de toda aresta é 1.

Assim, é fácil perceber que um fluxo máximo de s a t define um emparelhamento máximo entre A e B e vice-versa.

3 Corte mínimo em grafos

Sendo S um subconjunto do conjunto de vértices de um grafo G, o *corte* definido por S é o conjunto de todas as arestas com um extremo em S e o outro extremo fora de S. Se as arestas possuem direção, consideramos como arestas do corte apenas aquelas que vão *de* dentro de S *para* fora de S. Se as arestas de G possuem pesos, a *capacidade* de um corte é a soma dos pesos de todas as arestas do corte. Se fixamos uma origem s e um destino t, um s–t-corte é um corte definido por um conjunto S tal que $s \in S$ e $t \notin S$. Há várias versões do *Problema do Corte Mínimo em Grafos*. Consideremos uma primeira versão, que chamaremos de STMINCUT: *Dada uma rede (G, s, t, c), qual é a menor capacidade de um s–t-corte em G?*

A resposta para o STMINCUT é o famoso *Min-Cut Max-Flow Theorem*: *O valor máximo de um fluxo de s a t numa rede é igual à capacidade mínima de um s–t-corte*. Então, basta aplicarmos FFEK e está feito! Se queremos *um s–t*-corte mínimo, podemos proceder da seguinte forma, uma vez obtido o fluxo máximo f:

1. inicialize S com s;

2. realize uma busca (em largura ou profundidade, tanto faz), visitando apenas vértices que podem ser alcançados por arestas com *f positivo*, adicionando a S todos os vértices visitados.

Ao final, o conjunto das arestas que partem *de* vértices de S *para* vértices fora de S é um s–t-corte mínimo.

Se um grafo dirigido possui muitas origens e muitos sorvedouros e estamos interessados em encontrar a menor capacidade de um s–t-corte *para toda* origem s e *todo* sorvedouro t, podemos apenas adicionar uma *superorigem* s_0 e um *superdestino* t_0 e adicionar arestas de s_0 para toda origem s e arestas de todo sorvedouro t para t_0, atribuindo ∞ para as capacidades dessas arestas novas.

Se o grafo não é dirigido mas possui pesos nas arestas (mas é conexo, do contrário a capacidade do corte mínimo seria obviamente nula), procedemos da seguinte forma:

1. Para toda aresta $\{u, v\}$ crie duas arestas dirigidas (u, v) e (v, u), ambas com o mesmo peso que a aresta $\{u, v\}$ original.

2. Escolha qualquer vértice s para ser a origem, removendo as arestas que chegam em s.

Pronto! Agora, tudo o que você tem de fazer é avaliar o que ocorre para todo par (s, t) possível. Lembre-se de, a cada t escolhido, remover todas as arestas que saem de t (lembre-se também de recolocá-las no grafo para a próxima escolhe de t).

Se o grafo não é dirigido nem possui pesos nas arestas, o que se quer é o corte com o menor número de arestas. Neste caso, basta atribuir peso 1 para toda aresta e proceder como no caso anterior.

4 Problema do Caminho Mínimo e do Caminho Máximo em DAGs

Estratégia: Programação Dinâmica sobre uma ordenação topológica do DAG.

<u>SSSP_DAG(G, s)</u>:

encontre uma ordenação topológica $\pi(1) = s, \pi(2), \dots, \pi(|V(G)|)$ sobre G;
$\boldsymbol{dist}[s] \leftarrow 0$;
para j **de** 2 **até** $|V(G)|$, **faça**:
    $\boldsymbol{dist}[\pi(j)] \leftarrow \min_{u \in N^-_G(\pi(j))} \big(\boldsymbol{dist}[u] + \rho(u, \pi(j))\big)$.

Algoritmo 3

Complexidade: linear! (melhor do que se fosse usado Dijkstra). Se a origem s não é dada, basta iterar o algoritmo sobre todas as origens do grafo.

Curiosidade: O Problema do Caminho *Máximo* é $\mathcal{NP}$-completo. No entanto, em grafos acíclicos (sejam dirigidos ou não), pode ser resolvido em tempo *linear*! Basta multiplicar o peso de cada aresta por -1 e resolver o Problema do Caminho *Mínimo* obtido.

5 Grafos Eulerianos

Um grafo (conexo) é euleriano se e só se todos os seus vértices têm grau par. Um grafo (conexo) possui trilha euleriana se e só se no máximo dois de seus vértices têm grau ímpar.

Algoritmo de Hierholzer para grafos já sabidos eulerianos: encontre um ciclo no grafo; remova as arestas desse ciclo; continue encontrando ciclos e removendo arestas até o grafo ficar sem arestas; costure os ciclos encontrados.

Se o grafo possui dois vértices com grau ímpar e o que se quer é encontrar uma trilha euleriana (aberta), basta adicionar uma aresta entre os vértices de grau ímpar, encontrar um circuito euleriano (também chamado de trilha euleriana fechada) e remover a aresta adicionada.