Paralelní grafové algoritmy

Použité značení

Definition

Buď $G = (V, E)$ graf. Pro libovolný uzel $u \in V$ budeme značit:

- $N_V(u)$ množinu všech sousedních uzlů k uzlu u, tj.

$$N_V(u) \equiv \{v \in V \mid \exists (u, v) \in E\},$$

- $N_E(u)$ množinu všech přilehlých hran k uzlu u, tj.

$$N_E(u) \equiv \{(u, v) \mid v \in V \wedge (u, v) \in E\}.$$

- graf je jednoznačně určen tzv. **adjacenční maticí** $\mathbb{A} \in \mathbb{R}^{n,n}$ pro $n = |V|$
  - $a_{ij} \neq 0 \Leftrightarrow$ z uzlu s indexy i vede hrana do uzlu s indexem j
  - ve váženém grafu odpovídá hodnota maticového prvku váze hrany mezi oběma vrcholy

Polookruhy

Definition

Polookruh (semiring) je algebraická struktura $(S, \oplus, \odot, 0, 1)$, kde

- S je množina prvků
- $\oplus$ a $\odot$ jsou binární operace na množině S
- 0 a 1 jsou prvky splňující
  - $(S, \oplus, 0)$ je komutativní monoid
  - $(S, \odot, 1)$ je monoid
  - platí $a \odot 0 = 0 \odot a = 0$

Procházení grafu do šířky

BFS, breadth first search

- tento algoritmus projde všechny vrcholy v dané komponentě souvislosti
- začíná s jedním (nebo několika) startovním vrcholem
- v prvním kroku navštíví všechny jeho sousedy
- v dalších krocích navštíví vždy všechny doposud nenavštívené vrcholy sousedící s některým již navštíveným

Procházení grafu do šířky

```
1:  procedure BFS( G(V, E), s))
2:      V_F ← {s}
3:      while V_F ≠ V do
4:          for all v ∈ V_F do
5:              for all u ∈ N_V(v) do
6:                  if u ∉ V_F then
7:                      visit(u)
8:                      V_F := V_F ∪ u
9:                  end if
10:             end for
11:         end for
12:     end while
13: end procedure
```

Procházení grafu do šířky

- i-tý řádek adjacenční matice říká, do kterých uzlů vede hrana z i-tého uzlu
- j-tý sloupec adjacenční matice říká, ze kterých uzlů vede hrana do j-tého uzlu
- buď $\vec{x}$ vektor obsahující 1 na pozici startovního uzlu a všude jinde 0
- vektor $\vec{y} = \mathbb{A}^T\vec{x}$ obsahuje právě i-tý řádek matice $\mathbb{A}$
- indexy nenulových prvků $\vec{y}$ tedy odpovídají indexům uzlů, do kterých se lze dostat z uzlu i

Procházení grafu do šířky

- definujme operaci $\mathbb{A}(\wedge.\vee)\vec{x}$ takto

$$\mathbb{A}(\wedge.\vee)\vec{x}\mid_i = x_1 \wedge a_{i1} \vee \ldots \vee x_n \wedge a_{in}$$

- potom $\mathbb{A}^T(\wedge.\vee)\vec{x} = \vec{y}$ je vektor obsahující 1 pro uzly, kdo kterých se lze dostat ze všech uzlů označených 1 ve vektoru $\vec{x}$
- příslušný polookruh $(\{0,1\}, \wedge, \vee, 0, 1)$ se nazývá *booleovský polookruh*

Procházení grafu do šířky

1: **procedure** MATRIXBFS($G(V,E)$, r))
2: $\vec{x} = \vec{y} = \vec{0}, \vec{x}[r] = 1, step := 0$
3: **while** $\vec{x} \neq \mathbb{A}^T(\wedge.\vee)\vec{x}$ **do**
4: $\vec{x} = \mathbb{A}^T(\wedge.\vee)\vec{x}$
5: $step++$
6: **for all** $\vec{y}[i] = 0 \wedge \vec{x}[i] \neq 0$ **do**
7: $visit(u[i])$
8: $\vec{y}[i] = step$
9: **end for**
10: **end while**
11: **end procedure**

Implementace v TNL

Nejkratší cesta z jednoho uzlu

SSSP, single-source shortest path

Definition

Mějme souvislý graf $G(V, E)$ a zvolme v něm jeden vrchol jako startovní. Chceme najít nejkratší cestu do všech ostatních uzlů.

Aplikace:

- navigace, jizdní řády
- zpracování obrazu
- řešení PDR - eikonální a Hamiltonovy-Jacobiho rovnice (fast-marching method)
- dynamické programování
- porovnávání řetězců v bioinformatice
- program `diff`

Dijkstrův algoritmus

1: **procedure** DIJKSTRA($G(V, E)$, w, s))
2: $V_F \leftarrow \{s\}\,; V_T \leftarrow \emptyset$
3: $\vec{l} = \vec{\infty}; l[s] = 0$
4: **while** $V_F \neq V$ **do**
5: $u = \arg\min_{v \in V_T} l[v]$
6: $V_F \leftarrow V_F \bigcup \{u\}$
7: **for all** $v \in N_V(u) \setminus V_F$ **do**
8: $l[v] = \min\{l[v], l[u] + w(u, v)\}$
9: $V_T \leftarrow V_T \bigcup \{v\}$
10: **end for**
11: **end while**
12: **return** l
13: **end procedure**

Bellmanův-Fordův algoritmus

- Dijsktrův algoritmus je výrazně sekvenční
- while-cyklus na řádcích 10-17 nelze paralelizovat
- jiný přístup volí Bellmanův-Fordův algoritmus
- má spíše iterativní povahu
- v jednotlivých iterací neustále přepočítává a hledá kratší cesty do jednotlivých vrcholů
- nakonec se výpočet ustálí ve stavu, kdy známe nejkratší cestu do každého vrcholu

Bellmanův-Fordův algoritmus

```
1:  procedure BellmanFord( G(V, E), 𝔸, s))
2:      d⃗ = ∞⃗; π = 0⃗
3:      d(s) = 0
4:      for all k = 1, ... |V| − 1 do
5:          for all u ∈ V do
6:              for all v ∈ N_E(v) do
7:                  a = min{d[u], d[v] + 𝔸_uv}
8:                  if a < d[u] then
9:                      d[u] = a; π[u] = v
10:                 end if
11:             end for
12:         end for
13:     end for
14:     return d⃗
15: end procedure
```

Algebraický Bellmanův-Fordův algoritmus

- definujme operaci $\mathbb{A}(\min .+)\vec{x}$

$$\mathbb{A}(\min .+)x\ |_{ij} = \min\{x_1 + a_{i1}, \ldots, x_n + a_{in}\}$$

- příslušný polookruh $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ se nazývá *tropický polookruh*
- platí, že $\pi(u) = v^* \Leftrightarrow \vec{d}(v^*) = \vec{d}(u) + \mathbb{A}_{uv^*}$
- pro $v \neq v^*$ je $\vec{d}(v) \geq \vec{d}(u) + \mathbb{A}_{uv}$
- proto můžeme psát $\pi(v^*) = \arg\min_{u \neq v}\{\vec{d}[u] + \mathbb{A}_{uv}\}$
- celkem tedy

$$\pi = (\mathbb{A} + diag(\infty))^T (\arg\min .+)\vec{d}$$

Bellmanův-Fordův algoritmus lze pak zapsat tatko ...

Algebraický Bellmanův-Fordův algoritmus

1: **procedure** BELLMANFORD($G(V,E)$, $\mathbb{A}$, s))
2: $\quad \vec{d}^{(0)} = \vec{\infty}; d^{(0)}(s) = 0; k = 0$
3: $\quad$ **repeat**
4: $\quad\quad k++$
5: $\quad\quad \vec{d}^{(k)} = \mathbb{A}^T(\min .+)\vec{d}^{(k-1)}$
6: $\quad$ **until** $\vec{d}^{(k)} \neq \vec{d}^{(k-1)}$
7: $\quad \pi = (\mathbb{A} + diag(\infty))^T(\arg\min .+)\vec{d}^{(|V|-1)}$
8: $\quad$ return $(\vec{d}^{(k)}, \pi)$
9: **end procedure**

- výhodou je, že jsme úlohu převedli na algoritmus velice podobný násobení řídké matice a vektoru, který lze implementovat velice efektivně

Implementace v TNL

Nejkratší cesta mezi všemi dvojicemi uzlů

Místo vektoru $\vec{d}$ udávající délku nejkratší cesty z určitého uzlu nyní budeme hledat matici $\mathbb{D}$, pro niž platí, že $D_{u,v}$ je délka nejkratší cesty z uzlu u do uzlu v.

Dijkstrův algoritmus

Úlohu lze řešit pomocí n-krát aplikovaného Dijkstrova algoritmu, tj. se složitostí $O(n^3)$.
Paralelizaci lze provést dvěma způsoby:

- **paralelizace přes zdrojové vrcholy** – pro každý vrchol grafu G spustíme paralelně sekvenční verzi Dijkstrova algoritmu
- **paralelizace dekompozicí grafu** – postupně (sekvenčně) řešíme úlohu pro jeden uzel za druhým a použijeme paralelní verzi Dijkstrova algoritmu
- **kombinujeme oba přístupy**

Floydův-Warshallův algoritmus

Floydův-Warshallův algoritmus je založen na následující myšlence:

- buď d_{uv} doposud nejkratší známá cesta mezi uzly u a v
- jinou kratší cestu můžeme najít tak, že objevíme uzel w, pro který platí $d_{u,w} + d_{w,v} < d_{u,v}$
- pak položíme $d_{uv} = d_{u,w} + d_{w,v}$
- začneme s adjacenční maticí $\mathbb{A}$ grafu G
  - ta obsahuje první odhad nejkratších cest mezi libovolnou dvojicí uzlů
  - uzly, mezi kterými nevede hrana, mají délku nejkratší cesty $+\infty$
- bereme jeden uzel po druhém a zkoušíme, které cesty v grafu lze s jeho pomocí zkrátit

Floydův-Warshallův algoritmus

1: **procedure** FLOYD($G(V, E)$))
2: $\quad \mathbb{D}^0 := \mathbb{A}_{G(V,E)}$
3: $\quad$ **for** $k := 1$ **to** n **do**
4: $\quad\quad$ **for** $i := 1$ **to** n **do**
5: $\quad\quad\quad$ **for** $j := 1$ **to** n **do**
6: $\quad\quad\quad\quad \mathbb{D}^k_{i,j} := \min\left\{\mathbb{D}^{k-1}_{i,j}, \mathbb{D}^{k-1}_{i,k} + \mathbb{D}^{k-1}_{k,j}\right\}$
7: $\quad\quad\quad$ **end for**
8: $\quad\quad$ **end for**
9: $\quad$ **end for**
10: $\quad$ **return** $\mathbb{D}^n$
11: **end procedure**

Floydův-Warshallův algoritmus

- také se na tuto úlohu mohu dívat tak, že do $Bellmanova - Fordova$ algoritmu zadám současně n vektorů, kde každý reprezentuje cestu z jednoho vrcholu

---

1: **procedure** FLOYDWARSHALL(V, E, $\mathbb{A}$, s))
2: **for all** $u, v \in V$ **do**
3: $\mathbb{D}_{uv}^{(0)} = \infty$
4: **end for**
5: **for all** $v \in V$ **do**
6: $\mathbb{D}_{vv}^{(0)} = 0$
7: **end for**
8: **for all** $k = 1, \ldots |V| - 1$ **do**
9: $\mathbb{D}^{(k)} = \mathbb{A}^T(\min .+)\mathbb{D}^{(k-1)}$
10: **end for**
11: $\Pi = (\mathbb{A} + diag(\infty))^T(\arg\min .+)\mathbb{D}^{(|V|-1)}$
12: **return** $\mathbb{D}^n; \Pi$
13: **end procedure**

Maximální nezávislá množina

Definition

Buď $I \subset V$. S je *nezávislá množina (IS, independent set)*, pokud $\forall u, v \in V$ platí $(u, v) \notin E$. Pokud $\forall w \in V \setminus I$ platí, že $w \cup I$ není nezávslá, pak se I nazývá *maximální nezávislá množina, (MIS, maximal independent set)*. Je-li I' nezávislá a platí, že $|I'| \geq |I|$ pro všechny nezávislé množiny I, pak řekneme, že I' je *největší nezávislá množina*.

Remark

Nalezení největší nezávislé množiny je NP-úplná úloha.

Maximální nezávislá množina

Sekvenční algoritmus pro MIS:

```
1: procedure GreedyMIS( G(V, E))
2:     I ← ∅, V' ← V
3:     while V' ≠ V do
4:         Vyber v ∈ V'
5:         Přidej v do I
6:         Odeber v a N_V(v) z V'
7:     end while
8:     return I
9: end procedure
```

Maximální nezávislá množina

Paralelní Lubyho algoritmus pro MIS:

1: **procedure** LUBY($G(V, E)$)
2: $I \leftarrow \emptyset$
3: $V' \leftarrow V$
4: **while** $V' \neq \emptyset$ **do**
5: Vyber náhodnou množinu $U \subset V'$, kde každý vrchol v je vybrán s pravděpodobností $P(v) = \frac{1}{2d(v)}$, kde $d(v)$ je stupeň vrcholu v.
6: $\forall e = (u, v) \in E$, pokud $u \in U \wedge v \in V$ odeber vrchol a nižším stupněm
7: Přidej U do I.
8: Odeber U z V' spolu se všemi sousedními vrcholy.
9: **end while**
10: return I
11: **end procedure**

Vrcholové obarvení

Definition

Vrcholové obarvení grafu (*vertex coloring*) $G(V, E)$ je zobrazení $b : V \to \{1, 2, \ldots k\}$ takové, že pro libovolnou hranu $(u, v) \in E$ platí, že $b(u) \neq b(v)$.

Vrcholové obarvení

Sekvenční algoritmus pro vrcholové obarvení:

1: **procedure** GREEDYCOLORING($G(V, E)$))
2: $V' \leftarrow V; \vec{b} = \vec{0}$
3: **while** $V' \neq \emptyset$ **do**
4: Vyber $u \in V'$
5: $B \leftarrow \{i \in \mathbb{N} \mid i = b[v]; v \in N_V(u)\}$
6: $b[u] = \min\{\mathbb{N} \setminus B\}$
7: Odeber v z V'
8: **end while**
9: return $\vec{b}$
10: **end procedure**

Vrcholové obarvení

Paralelní algoritmus pro vrcholové obarvení:

1: **procedure** PARALLELCOLORING($G(V, E)$))
2: $V' \leftarrow V; \vec{b} = \vec{0}; k = 1$
3: **while** $V' \neq \emptyset$ **do**
4: Vyber paralelně nezávislou množinu $I \subset V'$
5: Nastav $b[v] = k; \forall v \in I$.
6: $k++$
7: Odeber I z V'
8: **end while**
9: return $\vec{b}$
10: **end procedure**