BFS, DFS, TOPSORT, SCCS

Petr Ryšavý

10. dubna 2025

Katedra počítačů, FEL, ČVUT

GRAFY

Co je graf?

Definice *Graf je trojice $G = (V, E, \epsilon)$, kde V je množina vrcholů, E je množina hran a $\varepsilon : E \mapsto V \times V$ je zobrazení incidence.*

- Grafy rozlišujeme na orientované (hrany jsou dvojice vrcholů, kde záleží na pořadí) a neorientované (hrany jsou množiny vrcholů bez pořadí).
- Často zkracujeme $m = |E|$ a $n = |V|$.

Jak uložit graf v paměti?

- V listu sousednosti si pamatujeme pro každý vrchol množinu jeho sousedů.
- V matici sousednosti si pamatujeme pro každou dvojici vrcholů příznak, zda jsou propojeny hranou.

Příklad

PROHLEDÁVÁNÍ GRAFŮ

Proč prohledávání grafů

- Zkontrolovat, zda je síť spojitá.
- Hledání nejkratší cesty, plánování cest.
- Prohledávání stavového prostoru, formulování plánu (např. jak vyřešit sudoku).
- Spočíst komponenty grafu.

Požadavky na algoritmus prohledávající graf

- Algoritmus musí umět nalézt cestu do všech vrcholů, které jsou dosažitelné.
- Nechceme procházet žádné části grafu vícekrát.
  - Požadujeme lineární čas běhu ($\mathcal{O}(m + n)$).

Obecný algoritmus

function GENERIC-GRAPH-SEARCH(*graph*, s)
    Označ s jako navštívené
    while lze nalézt hranu (u, v), že u bylo navštíveno a v ne **do**
        $(u, v) \leftarrow$ libovolná hrana, kde u bylo navštíveno a v ne
        Označ v jako navštívené
    end while
end function

Prohledávání je úplné

Tvrzení *Na konci je vrchol v navštívený právě tehdy, když G obsahuje cestu z s do v.*

Důkaz

Rozdíly mezi BFS a DFS

Algoritmy se odlišují podle toho v jakém pořadí vrcholy prohledávají

Rozdíly mezi BFS a DFS

Algoritmy se odlišují podle toho v jakém pořadí vrcholy prohledávají

- BFS (Breadth-First Search, prohledávání do šířky)
  - Vrcholy jsou prohledávány po úrovních.
  - Sousedi jedné úrovně tvoří další úroveň.
  - Může počítat nejkratší cesty a komponenty souvislosti.
  - $\mathcal{O}(m+n)$ s frontou.

Rozdíly mezi BFS a DFS

Algoritmy se odlišují podle toho v jakém pořadí vrcholy prohledávají

- BFS (Breadth-First Search, prohledávání do šířky)
  - Vrcholy jsou prohledávány po úrovních.
  - Sousedi jedné úrovně tvoří další úroveň.
  - Může počítat nejkratší cesty a komponenty souvislosti.
  - $\mathcal{O}(m+n)$ s frontou.
- DFS (Depth-First Search, prohledávání do hloubky)
  - Prohledáváme stále hlouběji, backtrackujeme jen když musíme.
  - Počítá topologické očíslování a silné komponenty souvislosti.
  - $\mathcal{O}(m+n)$ se zásobníkem.

BFS

Pseudokód

```
function BREADTH-FIRST-SEARCH(graph, s)
    Q ← new FIFO queue
    ADD(Q, s)
    MARK-VISITED(s)
    while SIZE(Q) ≠ 0 do
        v ← REMOVE(Q)
        for all edges (v, w) do
            if UNVISITED(w) then
                MARK-VISITED(w)
                ADD(Q, w)
            end if
        end for
    end while
end function
```

Příklad

BFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci BFS navštíví* v $\quad\Leftrightarrow\quad$ *v* G *existuje cesta z* s *do* v.

BFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci BFS navštíví* v $\Leftrightarrow$ *v* G *existuje cesta z* s *do* v.

Důkaz

BFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci BFS navštíví* v $\quad\Leftrightarrow\quad$ *v* G *existuje cesta z* s *do* v.

Důkaz

Tvrzení *Čas běhu BFS je* $\mathcal{O}(m_s + n_s)$.

BFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci BFS navštíví* v $\quad\Leftrightarrow\quad$ *v* G *existuje cesta z* s *do* v.

Důkaz

Tvrzení *Čas běhu BFS je* $\mathcal{O}(m_s + n_s)$.

Důkaz

Aplikace na hledání nejkratších cest

Cíl: spočíst $\mathrm{dist}(v)$, což je nejmenší počet hran na cestě z s do v.

Aplikace na hledání nejkratších cest

Cíl: spočíst $\mathrm{dist}(v)$, což je nejmenší počet hran na cestě z s do v.

function BREADTH-FIRST-SEARCH(*graph, s*)
    $Q \leftarrow$ new FIFO queue
    ADD(Q, *s*)
    MARK-VISITED(*s*)
    $\mathrm{dist}(v) = \begin{cases} 0, & \text{pro } v = s, \\ \infty & \text{jinak.} \end{cases}$
    while SIZE(Q) $\neq 0$ **do**
        $v \leftarrow$ REMOVE(Q)
        for all edges (v, w) **do**
            if UNVISITED(w) **then**
                MARK-VISITED(w)
                ADD(Q, w)
                $\mathrm{dist}(w) = \mathrm{dist}(v) + 1$
            end if
        end for
    end while

Příklad

BFS hledá nejkratší cesty

Tvrzení *Na konci BFS platí* $\mathrm{dist}(v) = i \quad \Leftrightarrow \quad v$ *leží v* i*-té vrstvě.*

BFS hledá nejkratší cesty

Tvrzení *Na konci BFS platí* $\mathrm{dist}(v) = i \quad \Leftrightarrow \quad v$ *leží v* i*-té vrstvě.*

Důkaz

DFS

Příklad

DFS prohledává graf více agresivně a vrací se co nejméně.

Pseudokód

Nahradíme frontu zásobníkem

```
function DEPTH-FIRST-SEARCH(graph, s)
    Q ← new LIFO stack
    ADD(Q, s)
    MARK-VISITED(s)
    while SIZE(Q) ≠ 0 do
        v ← REMOVE(Q)
        for all edges (v, w) do
            if UNVISITED(w) then
                MARK-VISITED(w)
                ADD(Q, w)
            end if
        end for
    end while
end function
```

Pseudokód, rekurzivně

```
function DEPTH-FIRST-SEARCH(graph, s)
    MARK-VISITED(s)
    for all edges (s, v) do
        if UNVISITED(v) then
            DEPTH-FIRST-SEARCH(graph, v)
        end if
    end for
end function
```

DFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci DFS navštíví* v $\quad\Leftrightarrow\quad$ *v* G *existuje cesta z* s *do* v.

DFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci DFS navštíví* v $\Leftrightarrow$ *v* G *existuje cesta z* s *do* v.

Důkaz

DFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci DFS navštíví* v $\Leftrightarrow$ *v* G *existuje cesta z* s *do* v.

Důkaz

Tvrzení *Čas běhu DFS je* $\mathcal{O}(m_s + n_s)$.

DFS splňuje požadavky na prohledávací algoritmus

Tvrzení *Na konci DFS navštíví* v $\Leftrightarrow$ *v* G *existuje cesta z* s *do* v.

Důkaz

Tvrzení *Čas běhu DFS je* $\mathcal{O}(m_s + n_s)$.

Důkaz

TOPOLOGICKÉ OČÍSLOVÁNÍ

Topologické očíslování

Definice (Topologické očíslování) ***Topologické očíslování*** *orientovaného grafu* $G = (V, E)$ *je bijekce* $f : V \mapsto 1, 2, \ldots, n$ *taková, že*

$$(u, v) \in E \quad \Rightarrow \quad f(u) < f(v).$$

Proč nás zajímá topologické očíslování?

- V jakém pořadí je třeba vystudovat předměty, abychom měli splněné prerekvizity.
- V jakém pořadí skládat výrobek.
- Aplikace v mnoha algoritmech.

Existence topologického očíslování

Věta *Graf má topologické očíslování právě tehdy, když neobsahuje orientovaný cyklus.*

Grafy bez orientovaných cyklů se nazývají acyklické, angličtině *directed acyclic* (DAG).

Naivní přístup

- Každý DAG musí obsahovat vrchol, ze kterého nevedou hrany
- Tento vrchol odstraníme (s odpovídajícími hranami) a opakujeme
- Všechny dosažitelné vrcholy musí být v číslování, abychom mohli vrchol odstranit

function DUMMY-TOPSORT(*graph*) **returns** topological sort f

$v \leftarrow$ sink vertex

$f(v) = n$

DUMMY-TOPSORT(*graph* $\setminus \{v\}$)

end function

Prohledáváním do hloubky

```
function TOPOLOGICAL-ORDERING(graph) returns topological ordering f of the graph
    f ← empty ordering
    current-label = VERTICES-COUNT(graph)
    for all node ∈ graph do
        if UNVISITED(node) then
            DEPTH-FIRST-SEARCH(graph, node)
        end if
    end for
end function

function DEPTH-FIRST-SEARCH(graph, s)
    MARK-VISITED(s)
    for all edges (s, v) do
        if UNVISITED(v) then
            DEPTH-FIRST-SEARCH(graph, v)
        end if
    end for
    f(s) = current-label
    current-label = current-label − 1
end function
```

Příklad

Čas běhu a korektnost

- Běží v $\mathcal{O}(m+n)$.

PROBLÉM SILNĚ SOUVISLÝCH KOMPONENT

Silně souvislá komponenta

Definice (Silně souvislá komponenta) *Nechť $G = (V, E)$ je orientovaný graf.* ***Silně souvislé komponenty*** *grafu G jsou množiny ekvivalence relace $u \sim v$, která je definovaná jako „existuje orientovaná cesta z u do v a z v do u“.*

- Silně souvislé komponenty někdy zkracujeme SCCs (z anglického *strongly connected components*)
- Je-li graf tvořený jednou komponentou souvislosti, pak mluvíme o silně souvislém grafu

Příklad

Jak najít silně souvislé komponenty?

- DFS nalezne všechny vrcholy dostupné z nějakého vrcholu
- V komponentě, která funguje podobně jako *sink vertex* toto funguje
- Potřebujeme tedy pustit DFS ze „správného“ vrcholu

Kosarajiho dvouprůchodový algoritmus

- Algoritmus spustí dvakrát DFS
- V prvním průchodu si seřadí topologicky representanty silně souvislých komponent
- Tomuto metagrafu tvořenému komponentami se říká *kondenzace grafu*
- V druhém průchodu jsou komponenty jedna po druhé odhalovány pomocí DFS
- Čas běhu v $\mathcal{O}(|V| + |E|)$

```
function STRONGLY-CONNECTED-COMPONENTS(graph) returns strongly connected components
of the graph
    reversed ← graph with all arcs reversed
    DFS-LOOP(reversed)
    DFS-LOOP(graph)
    return components represented by leader vertex
end function
Require: all nodes of the graph are labelled from 1 to n
function DFS-LOOP(graph)
    time = 0
    start = null
    for i = n to 1 do
        if UNVISITED(i) then
            start ← i
            DEPTH-FIRST-SEARCH(graph, i)
        end if
    end for
end function
function DEPTH-FIRST-SEARCH(graph, i)
    MARK-VISITED(i)
    SET-LEADER(i, start)
    for all edges (i, j) do
        if UNVISITED(j) then
            DEPTH-FIRST-SEARCH(graph, j)
        end if
    end for
    time ← time + 1
    f(i) = time
end function
```

Příklad

Proč je algoritmus korektní

Důkaz

Lze najít komponenty na jeden průchod?

- Označujeme komponenty podle času DFS návštěvy
- **low-link** je nejmenší index nějakého vrcholu, kam se dá dostat pomocí DFS
- Vrcholy se stejným low-linkem patří do stejné komponenty silné souvislosti

Stack invariant

- Low-link nesmíme počítat do topologicky následujících komponent
- Z tohoto důvodu držíme zásobník otevřených uzlů, které využíváme k updatu lowlinku
- Z platných uzlů na zásobníku se pak odebere komponenta (uzly tím uzavřeme)
- Poté, co navštívíme všechny sousedy, je-li low-link stejný jak index vrcholu, pak vrchol začíná na zásobníku komponentu

Aktualizace low-linku

- Při DFS návštěvě nějakého již navštíveného souseda
- Při návratu z DFS rekurze - jako minimum hodnoty a hodnoty potomka

Příklad

Tarjanův algoritmus (zdroj: Wikipedia)

```
algorithm tarjan(input: graph G = (V, E)):
    index := 0; S := empty stack
    for each v in V do: if v.index is undefined then:  strongconnect(v)

    function strongconnect(v)
        v.index := index; v.lowlink := index; index := index + 1
        S.push(v); v.onStack := true

        for each (v, w) in E do
            if w.index is undefined then
                // Successor w has not yet been visited; recurse on it
                strongconnect(w)
                v.lowlink := min(v.lowlink, w.lowlink)
            else if w.onStack then
                // Successor w is in stack S and hence in the current SCC
                // If w is not on stack, then (v, w) points to an SCC already found and mus
                // Note: The next line is correct; w.lowlink is correct too
                v.lowlink := min(v.lowlink, w.index)

        // If v is a root node, pop the stack and generate an SCC
        if v.lowlink = v.index then
            start a new strongly connected component
            repeat
                w := S.pop(); w.onStack := false
                add w to current strongly connected component
            while w != v output the current strongly connected component
```

References

- heavily inspired by Tim Roughgarden's online courses, `http://theory.stanford.edu/~tim/videos.html`
- Robert Sedgewick and Kevin Wayne, Algorithms, `http://algs4.cs.princeton.edu/home/`, namely `http://algs4.cs.princeton.edu/44sp/`
- Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). Competitive Programming 3. Lulu Independent Publish.

Děkuji za pozornost.

Čas na otázky!