BFS, DFS, fronta, zásobník

Petr Ryšavý

25. září 2018

Katedra počítačů, FEL, ČVUT

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.

Zásobník (anglicky *stack*)

- Datová struktura pro uchovávání dat
- Data přicházejí a odcházejí v pořadí *last in first out*
- Data přidáváme i odebíráme z konce
- Implementace obvykle pomocí pole

Fronta (anglicky *queue*)

- Datová struktura pro uchovávání dat
- Data přicházejí a odcházejí v pořadí *first in first out*
- Data přidáváme na konec, odebíráme ze začátku
- Implementace obvykle pomocí kruhového bufferu nebo spojového seznamu

Čas na programování

- Proveďte úkoly 1.-5.
- Popis i implementaci v Javě naleznete na `http://introcs.cs.princeton.edu/java/43stack/`. Zde je implementace pomocí spojového seznamu.
- Implementace fronty v C++ naleznete na `http://www.programming-techniques.com/2011/11/queue-is-order-collection-of-items-from.html` a zásobníku na `http://www.algolist.net/Data_structures/Stack/Array-based_implementation`.
- Každopádně obě struktury zkuste naimplementovat sami a neinspirujte se referencí.

Přestávka

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 $\Leftrightarrow$ *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 $\Leftrightarrow$ *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

Programování

- Proveďte úkoly 6.-8.
- Vzorovou implementaci prohledávání do šířky naleznete na `http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/`. Opět se ale snažte vše naimplementovat a odladit sami.

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 $\Leftrightarrow$ *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

Programování

- Proveďte úkoly 9.-11. Tentokrát ale použijte prohledávání do hloubky místo do šířky.
- Vzorovou implementaci prohledávání do hloubky naleznete na `http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/`. Opět se ale snažte vše naimplementovat a odladit sami.

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!