Algorytmy i Struktury Danych

10b: Algorytmy przeglądania grafów i drzew

©Marcin Sydow

Spis zagadnień

- rekurencyjne porządki przeglądania drzew binarnych: pre/in/post-order
- przeszukiwanie grafów (rola, schemat ogólny, zastosowania)
- przeszukiwanie wszerz BFS
- zastosowania BFS
- przeszukiwanie w głąb DFS (wersja stosowa i rekurencyjna)
- klasyfikacja krawędzi

Rekurencyjne przeszukiwanie drzew binarnych *

W wielu zastosowaniach ważne jest systematyczne przejrzenie wszystkich wierzchołków (i krawędzi) drzewa binarnego.

Oprócz porządku standardowego (dla dowolnych drzew uporządkowanych) rozróżnia się m.in. pewne 3 specyficzne porządki rekurencyjnego przeglądania drzewa binarnego:

- pre-order (bieżący, lewy, prawy)
- in-order (lewy, bieżący, prawy)
- post-order (lewy, prawy, bieżący)

Każdy z powyższych porządków zaczyna przeszukiwanie od korzenia i rekurencyjnie wywołuje przeszukiwanie w poddrzewach w odpowiedniej kolejności.
(lewy, prawy: oznaczają poddrzewa bieżącego wierzchołka)

Geometryczna interpretacja rekurencyjnych przeszukiwań drzew *

Gdy narysujemy drzewo binarne w standardowym porządku, i zakreślimy linię otaczającą drzewo zaczynając ponad korzeniem i obchodząc "naokoło" wszystkie gałęzie przeciwnie do ruchu wskazówek zegara, to mamy następujące interpretacje geometryczne:

- pre-order: wypisz wierzchołek pierwszy raz go napotykając
- post-order: wypisz wierzchołek ostatni raz go napotykając
- in-order: wypisz wierzchołek posiadający lewego syna drugi raz go napotykając, a każdy inny pierwszy raz go napotykając

Zastosowania rekurencyjnych porządków przeszukiwania *

Rekurencyjne podejście do przeszukiwania drzew binarnych upraszcza zapis wielu ważnych algorytmów na tych drzewach, np. obliczanie:

- liczby wierzchołków w drzewie
- wysokości drzewa
- głębokości każdego wierzchołka
- liczby liści, etc.

Przeszukiwanie grafu

Wiele zadań lub algorytmów na grafach sprowadza się do systematycznego odwiedzenia wszystkich elementów grafu (wierzchołków i/lub krawędzi).
Przez przeszukiwanie grafu rozumiemy taki systematyczny sposób, w którym:

- zaczynamy odwiedzanie z pewnego wierzchołka startowego
- dozwolonym ruchem jest przejście po krawędzi (od wierzchołka odwiedzonego)
- każdy wierzchołek i krawędź grafu odwiedzone są dokładnie raz

Przeszukiwanie można wykonywać zarówno na grafach nieskierowanych jak i skierowanych.
W wierzchołkach lub krawędziach wykonujemy pewne operacje, zależnie od zadania (np. wypisywanie na wyjście, ustawianie wartości pewnych atrybutów, etc.)

Ogólny schemat przeszukiwania grafu

(początkowo wszystkie wierzchołki są nieodwiedzone)

umieść startowy wierzchołek w strukturze danych X

dopóki struktura X jest niepusta:

1. wyjmij kolejny wierzchołek v ze struktury X i odwiedź go
2. włóż do X kolejno wszystkich nieodwiedzonych sąsiadów v

Konkretne warianty przeszukiwania zależą od:

- wyboru struktury danych X (np. kolejka lub stos)
- wyboru kolejności sąsiadów (np. etykiety alfabetycznie)

Las przeszukiwania i klasyfikacja krawędzi

Pojedyncze wykonanie całej procedury generuje tzw. **drzewo przeszukiwania** (jest to drzewo ukorzenione).

Procedura może być wykonywana wielokrotnie, aż nie ma nieodwiedzonych wierzchołków w grafie.

Rezultatem całego przeszukiwania jest więc zbiór (rozłącznych wierzchołkowo) drzew przeszukiwania zwany **lasem przeszukiwania**.

Umowne kolory wierzchołków

W przedstawianych pseudokodach będziemy nadawać wierzchołkom umowne kolory:

- biały (nieodwiedzony)
- szary (odwiedzony i przetwarzany; w strukturze danych)
- czarny (odwiedzony i zakończony)

Uwaga: do prawidłowej implementacji wystarcza rozróżnienie między nieodwiedzonym a odwiedzonym, szary został dodany dla rozjaśnienia prezentacji.

Klasyfikacja krawędzi w wyniku przeszukiwania grafu

Ze względu na strukturę lasu przeszukiwania, w wyniku przeszukiwania każda krawędź (u, v) jest zaklasyfikowana do dokładnie jednej z poniższych kategorii:

drzewowa (T): v jest odwiedzony z u w wyniku przejścia (u, v)

w przód (F): nie jest drzewowa i v jest potomkiem u w drzewie

w tył (B): nie jest drzewowa i v jest przodkiem u w drzewie

poprzeczna (C): w pozostałych przypadkach
(pomiędzy gałęziami lub drzewami)

Przeszukiwanie wszerz (BFS - breadth-first search)

Jest równoważne użyciu *kolejki* w ogólnym schemacie przeszukiwania. Efekt polega na odwiedzaniu wierzchołków równomiernie "we wszystkich kierunkach" zgodnie z rosnącą odległością od startowego.

Zastosowania przeszukiwania wszerz (przykłady):

- obliczanie składowych spójnych (każde drzewo ją stanowi)
- obliczanie odległości od wierzchołka startowego
- obliczanie domknięcia przechodniego relacji ($n \times$BFS)

Przykład implementacji BFS


```
for-each node in V:
  node.color = white; node.d = infinity; node.p = null

s.color = gray; s.d = 0; queue.in(s)

while(!queue.empty()){
  currNode = queue.out()

  process(currNode)

  for-each node in currNode.adjList:
     if (node.color == white):
        queue.in(node)
        node.color = gray
        node.d = currNode.d + 1
        node.p = currNode

  currNode.color = black
}
```

W powyższym pseudokodzie BFS:

- obliczamy odległości od startowego (atrybut d)
- zapamiętujemy strukturę drzewa (atrybut p)
- za `process(currNode)` można podstawić dowolną operację na wierzchołku

Obserwacje dotyczące przeszukiwania wszerz (BFS)

Złożoność czasowa BFS: $O(|V| + |E|)$ (liniowa)

Zauważmy, że w BFS dla grafów *nieskierowanych*:

- nie występują krawędzie w przód ani wstecz
- dla krawędzi drzewowej (u, v) zachodzi: $v.d = u.d + 1$
- dla krawędzi poprzecznej: $v.d = u.d$ lub $v.d = u.d + 1$

Natomiast w BFS dla grafów *skierowanych*:

- nie występują krawędzie w przód
- dla krawędzi drzewowej (u, v) zachodzi: $v.d = u.d + 1$
- dla krawędzi poprzecznej: $v.d \leq u.d + 1$
- dla krawędzi wstecznej: $0 \leq v.d \leq u.d$

Przeszukiwanie w głąb (DFS - depth-first seach)

Jest równoważne użyciu *stosu* w ogólnym schemacie przeszukiwania. Efekt polega na docieraniu stopniowo do możliwie najdalszego wierzchołka a następnie minimalnym cofnięciu się, aby kontynuować przeszukiwanie.

W przeszukiwaniu w głąb każdemu wierzchołkowi v przypisywane są *dwa ważne atrybuty*:

- **czas odwiedzenia**: $v.d$ (gdy staje się szary)
- **czas zakończenia**: $v.f$ (gdy staje się czarny)

Atrybuty te mają zastosowanie do rozwiązywania konkretnych problemów grafowych.

Przeszukiwanie w głąb (wersja rekurencyjna)

Uwaga: Implementacje DFS z użyciem stosu czy rekurencji są równoważne co do idei algorytmu, ale mogą dawać inny porządek odwiedzanych wierzchołków.

```
DFS(){
  time = 0
  for-each v in V:
     v.color = white; v.parent = null
  for-each v in V:
     if (v.color == white):
        recursiveDFS(v)
}
recursiveDFS(GraphNode v){
  v.d = time++
  v.color = gray
  process(v)
  for-each u in v.adjList:
     if (u.color == white):
        u.parent = v
        recursiveDFS(u)
  v.color = black
  v.f = time++
}
```

Własności przeszukiwania w głąb (DFS)

Złożoność czasowa: $O(|V| + |E|)$ (liniowa)

Dla dowolnych 2 wierzchołków u, v, ich przedziały czasów przetwarzania $[u.d, u.f]$ i $[v.d, v.f]$ są albo rozłączne albo jeden z nich zawiera się w drugim (tzw. "struktura nawiasowa")

Zachodzi następujący warunek (twierdzenie "o białej ścieżce"): W drzewie DFS v jest potomkiem $u \Leftrightarrow$ w momencie $u.d$, istnieje ścieżka z u do v składająca się tylko z białych wierzchołków.

Rodzaje krawędzi w grafach

W DFS dla grafów *nieskierowanych* nie występują krawędzie w przód ani poprzeczne.

W DFS dla grafów *skierowanych* mogą występować wszystkie 4 rodzaje krawędzi. Dokładniej, kiedy DFS przechodzi krawędź (u, v), krawędź ta jest:

- drzewowa, jeśli v jest biały
- wstecz, jeśli v jest szary
- w przód lub poprzeczna, jeśli v jest czarny

Kategoryzacja krawędzi za pomocą czasów d i f

Krawędź (v, w) w przeszukiwaniu DFS jest:

- drzewowa lub w przód $\Leftrightarrow v.d < w.d < w.f < v.f$
- w tył $\Leftrightarrow w.d < v.d < v.f < w.f$
- poprzeczna $\Leftrightarrow w.d < w.f < v.d < v.f$

Zastosowania przeszukiwania w głąb

Schemat przeszukiwania w głąb ma rozliczne zastosowania dla rozwiązywania konkretnych problemów grafowych, np:

- testowanie acykliczności (nieistnienie krawędzi wstecz)
- sortowanie topologiczne
- składowe silnie spójne
- znajdowanie punktów artykulacji
- znajdowanie mostów
- rozkład grafu na bloki

Podsumowanie

- rekurencyjne porządki przeglądania drzew binarnych
- przeszukiwanie BFS i własności
- przeszukiwanie DFS i własności

Przykładowe ćwiczenia

- wypisz kolejność odwiedzania wierzchołków dla podanego drzewa w omawianych porządkach przeszukiwania (pre/in/post-order)
- wyćwicz umiejętność pisania i analizowania prostych funkcji rekurencyjnych na drzewach binarnych (np. obliczanie liczby węzłów, wysokości, etc.)
- wypisz kolejność odwiedzanych wierzchołków, odległości(BFS), czasy odwiedzenia i zakończenia(DFS), las przeszukiwania danego grafu i klasyfikację krawędzi za pomocą BFS i DFS.
- zaproponuj algorytm wychodzenia z labiryntu oparty na jednym (którym?) ze sposobów przeglądania grafu: BFS lub DFS

Dziękuję za uwagę