Grafy i Zastosowania

4: Przeszukiwanie Grafów (BFS, DFS i zastosowania)

©Marcin Sydow

Spis zagadnień

- przeszukiwanie grafów (rola, schemat ogólny, zastosowania)
- realizacje (kolejka, stos, rekurencja)
- przeszukiwanie wszerz BFS
- zastosowania BFS
- przeszukiwanie w głąb DFS (wersja stosowa i rekurencyjna)
- klasyfikacja krawędzi
- zastosowania:
  - składowe spójne
  - testowanie acykliczności
  - znajdowanie mostów
  - znajdowanie punktów artykulacji
  - sortowanie topologiczne (podejście naiwne i DFS)
  - składowe silnie spójne

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.

przykład

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.

przykład

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

Stopień a cykliczność

Ścieżka DFS: fragment drzewa przeszukiwania DFS zbudowany do momentu, gdy jakiś wierzchołek staje się czarny.

Fakt:
Jeśli P jest ścieżką DFS kończącą się na wierzchołku t, to wszyscy sąsiedzi t leżą na ścieżce P. Ponadto, jeśli $deg(t) > 1$ to wszyscy sąsiedzi t leżą na pewnym cyklu

Wniosek:
Jeśli w grafie prostym G minimalny stopień wierzchołka to $\delta > 1$, to graf zawiera cykl o długości conajmniej $\delta + 1$

Znajdowanie mostów

Fakty:
Krawędź jest mostem $\Leftrightarrow$ nie leży na żadnym cyklu

Dowolny graf spójny, po odjęciu mostów składa się ze składowych 2-spójnych (krawędziowo) (tzw. bloków)

Wszystkie wierzchołki leżące na pewnym cyklu są w tym samym bloku.

Kontrakcja podgrafu H: uczynienie z H pojedynczego wierzchołka

przykład

Fakt:
Przez kontrakcję każdej składowej spójnej grafu G otrzymujemy drzewo H, którego krawędzie odpowiadają mostom grafu G.

Algorytm znajdowania mostów

Wejście: spójny graf nieskierowany G
Wyjście: graf H składający się jedynie z mostów grafu G

```
H = G
while |V(H)|>1
  utwórz ścieżkę DFS do wierzchołka t, który staje się czarny
  if deg(t) = 1
    zaznacz krawędź incydentną z t jako most
    H = H - t
  else // t i jego sąsiedzi leżą na cyklu C
    H jest wynikiem kontrakcji cyklu C
```

(algorytm stopniowo dokonuje kontrakcji wszystkich wykrytych cykli aż zostaną same mosty)

Korzeń jako punkt artykulacji

(punkt artykulacji: po jego usunięciu jest więcej składowych)

Fakt:
wierzchołek v jest punktem artykulacji $\Leftrightarrow$ istnieją wierzchołki u,w (różne od v), takie że każda droga z u do w przechodzi przez v.

Twierdzenie:
Jeśli T jest drzewem DFS dla pewnego grafu, to jego korzeń r jest punktem artykulacji $\Leftrightarrow$ r ma więcej niż jednego syna w tym drzewie.

Nie-korzeń jako punkt artykulacji

Twierdzenie A:
Jeśli T jest drzewem DFS, to jego wierzchołek v (nie będacy korzeniem) jest punktem artykulacji $\Leftrightarrow$ v ma syna w, takiego że żaden potomek wierzchołka w ani on sam nie jest połączony krawędzią nie-drzewową z właściwym przodkiem wierzchołka v.

przykład

Niech **low(w)** oznacza mniejszą z liczb: $w.d$ lub najmniejsza z liczb $u.d$ pośród wszystkich wierzchołków u połączonych nie-drzewową krawędzią z w lub pewnym jego potomkiem.

przykład

Twierdzenie B:
Jeśli T jest drzewem DFS, to wierzchołek v (nie będący korzeniem T) jest punktem artykulacji $\Leftrightarrow$ v ma syna w, dla którego $low(w) \geq v.d$

Schemat znajdowania punktów artykulacji

Wejście: spójny graf G
Wyjście: zbiór K punktów artykulacji grafu G

1. K = pusty
2. T = drzewo DFS z dowolnego wierzchołka r
3. jeśli r ma więcej niż 1 syna to $K = K + r$
4. oblicz wartości $low(w)$ dla wszystkich wierzchołków
5. dla każdego wierzchołka v nie będącego korzeniem, dodaj v do zbioru K, jeśli ma syna w, takiego że $low(w) \geq v.d$

Uwaga: wartości $low(w)$ jak i cały powyższy algorytm mogą być obliczone podczas wykonania pojedynczego przeszukiwania DFS na grafie G: $low(w) = min(v.d, z.d, low(y))$, gdzie z to dowolny wierzchołek połączony krawędzią niedrzewową z v, a y to dowolny syn wierzchołka v.

Przykład znajdowania punktów artykulacji

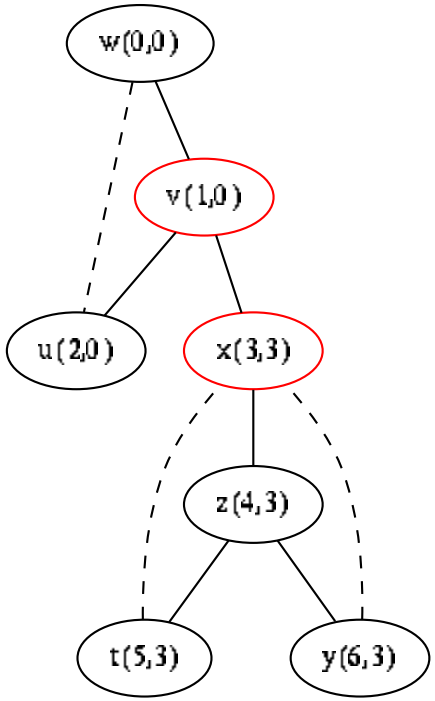

Figure : Drzewo DFS (krawędzie drzewowe zaznaczono linią ciągłą). W każdym wierzchołku a umieszczono parę liczb $(a.d, low(a))$. Zaznaczono punkty artykulacji, czyli wierzchołki, które mają syna s takiego, że $low(s)$ jest niemniejszy od ich czasu odkrycia.

Sortowanie topologiczne (tylko grafy skierowane)

Polega na takim ustawieniu wierzchołków grafu w ciąg, że jeśli w grafie jest krawędź (u, v), to u musi być przed v (ustawienie "zgodne" z krawędziami).

Graf da się posortować topologicznie $\Leftrightarrow$ nie zawiera cykli (skierowanych).

Rozwiązanie:

- zastosować DFS
- ustawić wierzchołki od największego czasu zakończenia do najmniejszego

przykład[1]

[1] istnieje też rozwiązanie przez iteracyjne usuwanie wierzchołków o stopniu wejściowym 0, też można zaimplementować w czasie liniowym

Znajdowanie składowych silnie spójnych

1. oblicz za pomocą DFS czasy zakończenia ($v.f$)
2. odwróć kierunki krawędzi (tzw. transpozycja grafu)
3. wykonaj DFS na transponowanym grafie, ale wywołuj główną procedurę zgodnie z malejącym czasem zakończenia z kroku 1
4. wynik: poszczególne drzewa otrzymane w kroku 3 to poszczególne składowe silnie spójne

przykład

Podsumowanie

- przeszukiwanie BFS i własności
- przeszukiwanie DFS i własności
- sprawdzanie acykliczności
- znajdowanie punktów artykulacji
- znajdowanie mostów
- sortowanie topologiczne
- obliczanie składowych silnie spójnych

Przykładowe ćwiczenia

- wypisz kolejność odwiedzanych wierzchołków, odległości(BFS), czasy odwiedzenia i zakończenia(DFS), las przeszukiwania danego grafu i klasyfikację krawędzi za pomocą BFS i DFS.
- wykonaj algorytm sprawdzania czy dany graf jest acykliczny
- wykonaj algorytm znajdowania mostów
- wykonaj algorytm znajdowania punktów artykulacji
- wykonaj sortowanie topologiczne danego grafu (2 metody)
- wykonaj algorytm obliczania składowych silnie spójnych danego grafu

Przykładowe zadania

- zaproponuj liniowy algorytm znajdowania w nieskierowanym grafie ścieżki, która przechodzi przez każdą krawędź dokładnie raz w każdą stronę.
- zaproponuj algorytm (i oszacuj złożoność czasową) znajdowania wyjścia z labiryntu gdy masz do dyspozycji (3 warianty) (Wskazówka: który schemat: BFS czy DFS będzie lepszy i dlaczego? (oba mają tą samą złożoność):
  - kłębek nici (metoda Ariadny)
  - kredę do pisania na ścianach labiryntu
  - k operatorów (każdy z kredą) mogących działać niezależnie
- zaproponuj algorytm sprawdzania, czy dany graf nieskierowany jest dwudzielny
- planowany jest mecz pośród n szachistów, gdzie ustalone r par szachistów planuje rozegrać partie. W całym meczu każdy z graczy chciałby rozegrać wszystkie partie jednym kolorem. Zaproponuj algorytm, który sprawdzi, czy to jest możliwe. Czy da się to zrobić w czasie liniowym?

Dziękuję za uwagę