Drzewa Link-Cut

Marcin Dublański

Wrocław, 13.12.2010

1 Przedstawienie struktury

Drzewa Link-Cut to struktura danych umożliwiająca szybkie wykonywanie operacji związanych ze spójnością grafu będącego zmieniającym się w czasie lasem ukorzenionych drzew. Jej twórcami są Daniel D. Sleator i Robert E. Tarjan. Struktura ta znajduje zastosowania w problemie maksymalnego przepływu oraz badaniu spójności zmieniających się grafów.

2 Zmieniające się drzewa

Problem zmieniających się w czasie drzew polega na dobrej reprezentacji grafu w postaci lasu, którego każde drzewo jest ukorzenione. Drzewa te nie muszą spełniać żadnych dodatkowych warunków, w szczególności każdy wierzchołek może mieć dowolną ilość wierzchołków potomnych. Taka struktura powinna potrafić przeprowadzać następujące operacje:

- *Make_ Tree(u)* – tworzy nowe drzewo złożone tylko z korzenia u; dzięki tej operacji las może się powiększać o nowe wierzchołki.
- *Link(u,v)* – podczepia drzewo, którego korzeniem jest wierzchołek u pod wierzchołek v, czyli dodaje krawędź (u, v). Wymagane jest, aby u było korzeniem swojego drzewa oraz żeby oba wierzchołki pochodziły z różnych drzew.
- *Cut(u)* – Usuwa krawędź między wierzchołkiem u a jego bezpośrednim przodkiem. Wierzchołek u nie może być korzeniem swojego drzewa - staje się nim dopiero po wykonaniu tej operacji.
- *Find_ Root(u)* – Zwraca korzeń drzewa zawierającego wierzchołek u. Dzięki tej operacji można np. sprawdzać, czy dane dwa wierzchołki a, b należą do tego samego drzewa - wystarczy wtedy sprawdzić, czy *Find_ Root(a)* = *Find_ Root(b)*.

3 Drzewa Link-Cut

Drzewa Link-Cut realizują wszystkie powyższe operacje w czasie logarytmicznym od liczby wierzchołków lasu.

3.1 Definicje

Definicja 1. *Las, który będzie przedstawiany przez drzewo Link-Cut, będziemy nazywać* ***reprezentowanym****. Każde drzewo reprezentowanego lasu również nazwiemy* ***reprezentowanym****.*

Problem reprezentacji zmieniającego się w czasie lasu zabrania jakichkolwiek modyfikacji reprezentowanego lasu. Nie możemy zatem zmieniać jego struktury. Ponieważ drzewa z reprezentowanego lasu mogą mieć strukturę dowolną, nie możemy polegać jedynie na krawędziach. Potrzebujemy dostępu o kilka krawędzi wyżej, a nie tylko od syna do ojca. Z tego powodu drzewa reprezentowanego lasu są dzielone na ścieżki, które w naszej strukturze będą pamiętane w dość nietypowy sposób.

Definicja 2. *Każdy wierzchołek ma prawo mieć dokładnie jednego* ***preferowanego syna****. Krawędź między wierzchołkiem a jego preferowanym synem również będziemy nazywać* ***preferowaną****. Przez* ***preferowaną ścieżkę*** *rozumieć będziemy maksymalną (w sensie zawierania) ścieżkę złożoną jedynie z preferowanych krawędzi. W szczególności może być to ścieżka pusta - złożona tylko z jednego wierzchołka, który nie jest incydentny z żadną preferowaną krawędzią.*

Z definicji tej widać, że preferowane ścieżki tworzą podział wierzchołków reprezentowanego lasu (bo każdy wierzchołek należy do dokładnie jednej preferowanej ścieżki).

Drzewa Link-Cut reprezentują każde drzewo z reprezentowanego lasu za pomocą *drzew pomocniczych*. Każde jedno drzewo pomocnicze reprezentuje dokładnie jedną preferowaną ścieżkę. Drzewa pomocnicze to po prostu drzewa splay, w których wartościami wierzchołków są ich głębokości w reprezentowanym lesie (czyli odległości, liczone w krawędziach, do korzenia swojego drzewa). Wynika z tego bezpośrednio, że dla każdego wierzchołka u wszystkie wierzchołki będące w jego lewym poddrzewie (patrząc na drzewo pomocnicze) leżą bliżej korzenia niż u, a wierzchołki z prawego poddrzewa - dalej od niego. Co więcej, w drzewie pomocniczym każdy wierzchołek v z prawego poddrzewa wierzchołka u ma tę własność, że ścieżka łącząca v z korzeniem swojego drzewa przechodzi przez u.

Drzewa pomocnicze są połączone siecią *ścieżkowych wskaźników*. Każde drzewo pomocnicze posiada dokładnie jeden ścieżkowy wskaźnik, który jest przechowywany w aktualnym korzeniu drzewa pomocniczego. Wskazuje on na wierzchołek, który, patrząc na reprezentowany las, jest ojcem wierzchołka z danej ścieżki najbliżej położonego korzenia. Nie możemy pozwolić sobie na wskaźniki w drugą stronę, gdyż dany wierzchołek może mieć dowolną liczbę drzew pomocniczych na niego wskazujących. Korzeń drzewa pomocniczego, które ma pusty ścieżkowy wskaźnik, nazwiemy *korzeniem głównym*.

Zestaw drzew pomocniczych wraz ze ścieżkowymi wskaźnikami tworzy drzewo drzew pomocniczych, których zbiór będziemy rozumieć właśnie jako drzewo Link-Cut. Pojedyncze drzewo drzew pomocniczych odpowiada za jedno drzewo w reprezentowanym lesie. Drzewo drzew pomocniczych będziemy od teraz nazywać *drzewem nadrzędnym*. Taka reprezentacja dobrze odzwierciedla całą strukturę reprezentowanego lasu - każde jedno drzewo Link-Cut odzwierciedla dokładnie jeden las. Stwierdzenie to w drugą stronę nie jest prawdą - las może być reprezentowany przez wiele drzew Link-Cut, bo istnieje wiele sposobów rozbicia pojedynczego drzewa na preferowane ścieżki.

3.2 Operacje na drzewach Link-Cut

3.2.1 Operacja podstawowa

Wszystkie wymagane operacje będą korzystały z procedury *Access(u)*. Zadaniem tej procedury jest odpowiednia reorganizacja drzewa nadrzędnego zawierającego wierzchołek u w taki sposób, aby u stał się korzeniem głównym. Wymaganie to natychmiast pociąga za sobą fakt, że u musi leżeć na tej samej ścieżce preferowanej co korzeń R reprezentowanego drzewa zawierającego u. Procedura Access(u) tworzy więc ścieżkę preferowaną od R do u. Każda krawędź preferowana nienależąca do tej ścieżki, a incydentna z którymś z jej wierzchołków, musi przestać być krawędzią preferowaną - zamiast tego zostanie przekształcona na ścieżkowy wskaźnik. Wykonanie tych dwóch rzeczy - uczynienie ścieżki R do u preferowaną oraz odpreferowanie krawędzi incydentnych spoza tej ścieżki - wystarcza, aby powstałe drzewo nadrzędne miało dobrą strukturę.

Skoro ścieżka od R ma kończyć się na u, u musi odpreferować swojego preferowanego syna, jeśli taki w ogóle istnieje. Spowoduje to rozbicie preferowanej ścieżki przechodzącej przez u na dwie części - górną i dolną. Część górna jest częścią ścieżki z R do u, natomiast ścieżka dolna musi stać się samodzielną, niezależną ścieżką, ze ścieżkowym wskaźnikiem wskazującym na u. Jednym ze sposobów na oddzielenie dolnej ścieżki jest wykonanie splay(u), czyli wymuszenie na u bycie korzeniem swojego drzewa pomocniczego, a następnie pozbycie się prawego poddrzewa - tam właśnie będą wszystkie wierzchołki o większej głębokości, a więc dokładnie te które chcemy odłączyć.

Po odłączeniu dolnej części ścieżki, potrzebna jest rozbudowa ścieżki od u w górę. W tym właśnie momencie skorzystamy ze ścieżkowych wskaźników. Po wykonaniu splay(u), u jest korzeniem swojego drzewa pomocniczego, ma więc bezpośredni dostęp do ścieżkowego wskaźnika, który wskazuje wierzchołek v będący na ścieżce od R do u. Pozostaje nam połączyć ścieżkę preferowaną zawierającą u ze ścieżką zawierającą v - innymi słowy, trzeba uczynić u synem preferowanym v. Dzieje się to w dwóch krokach: najpierw odłączamy część ścieżki preferowanej leżącą poniżej wierzchołka v (w ten sam sposób, jak to zrobiliśmy w przypadku wierzchołka u, czyli wykonujemy splay(v) i odłączamy prawe poddrzewo), a następnie łączymy ścieżkę zawierającą v ze ścieżką zawierającą u. Ponieważ wszystkie wierzchołki z drzewa pomocniczego zawierającego u mają większą głębokość niż wierzchołki z drzewa pomocniczego zawierającego v, wystarczy uczynić u prawym synem wierzchołka v. Oczywiście trzeba również pozbawić u ścieżkowego wskaźnika, jako że nie jest już korzeniem swojego drzewa pomocniczego. Po wykonaniu tych czynności jesteśmy gotowi do kontynuowania przedłużania ścieżki. Procedurę podsumowuje poniższy pseudokod:

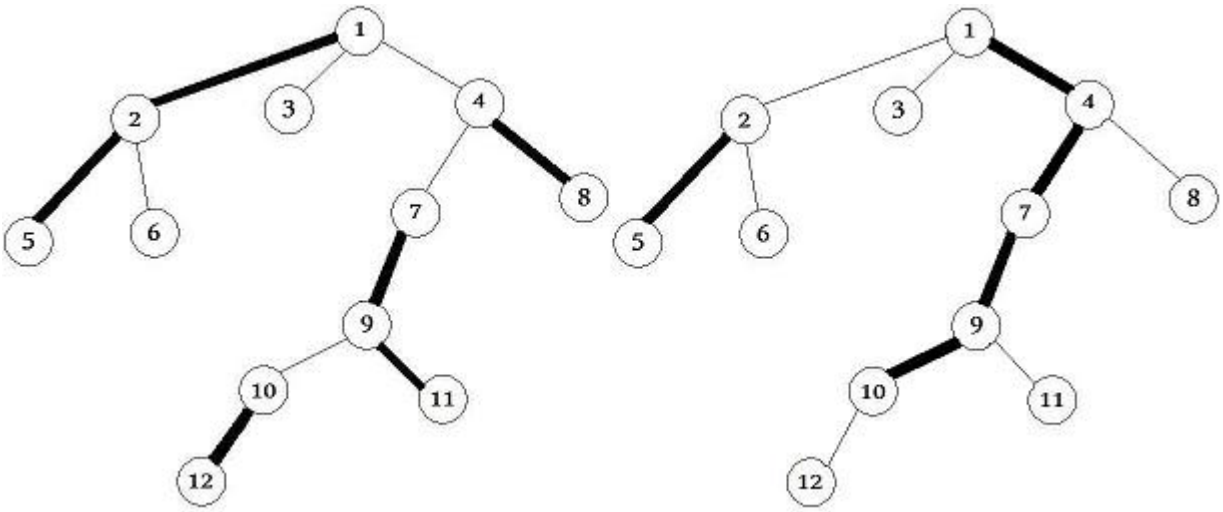
Algorithm 1. Access(u)

```
splay(u)
ścieżkowy_wskaźnik(right(u)) = u
right(u) = NULL
x = u
while x ≠ root do
  y = ścieżkowy_wskaźnik(x)
  splay(y)
  {Zmiana preferowanego syna}
  ścieżkowy_wskaźnik(right(y)) = y
  right(y) = x
  ścieżkowy_wskaźnik(x) = NULL
```

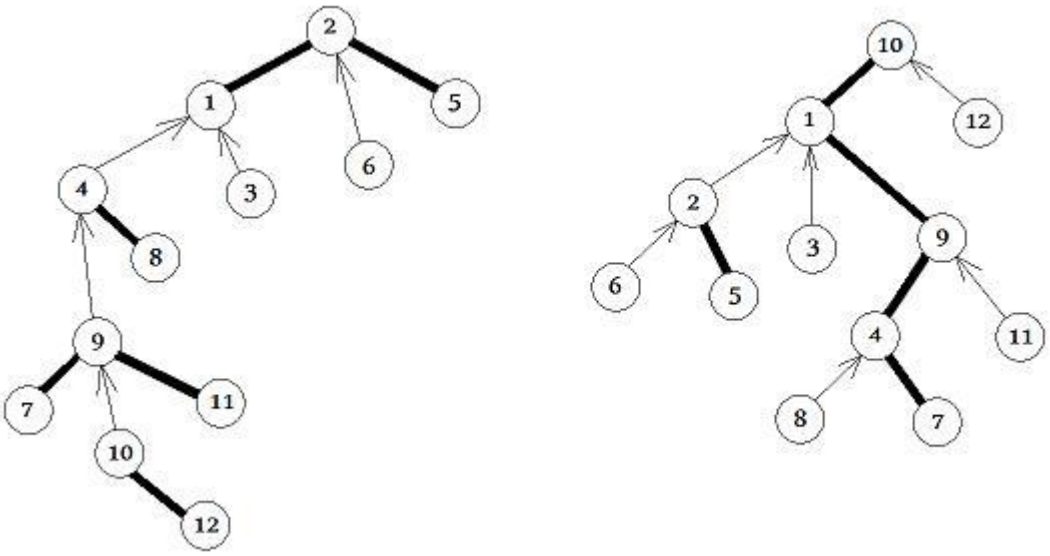
x = y
end while
splay(u)

end

a wyniki jej działania ilustruje poniższy przykład:



Drzewo reprezentowane: po lewej przed, po prawej po wywołaniu Access(10); pogrubione są krawędzie preferowane



Drzewo nadrzędne: po lewej przed, po prawej po wywołaniu Access(10); pogrubione są krawędzie preferowane, strzałki symbolizują ścieżkowe wskaźniki

Mając gotową procedurę Access(u), napisanie wymaganych funkcji nie jest już problemem.

3.2.2 Znajdowanie korzenia

Operacja Find_Root(u) wymaga niewiele więcej niż procedura Access(u). Po wykonaniu procedury mamy pewność, że u jest korzeniem głównym, oraz że w jego drzewie pomocniczym znajduje się także R, czyli szukany korzeń reprezentowanego drzewa. Ma on najmniejszą głębokość (równą 0), więc aby go znaleźć, wystarczy iść od u w lewo tak długo jak tylko się da. Na końcu czeka szukany korzeń R.

Algorithm 2. Find_Root(u)

```
Access(u)
while left(u) ≠ NULL do
  u = left(u)
end while
splay(u)
return u
```

end

3.2.3 Rozłączanie wierzchołków

Konsekwencją rozłączenia wierzchołka u od swojego bezpośredniego przodka w reprezentowanym drzewie jest rozłączenie u od swoich wszystkich, także tych dalszych, przodków. Po wykonaniu Access(u) wiemy, że wszyscy przodkowie u są w jego lewym poddrzewie (jako że mają mniejszą głębokość). Wystarczy więc odłączyć to lewe poddrzewo u.

Algorithm 3. Cut(u)

```
Access(u)
left(u) = NULL
```

end

3.2.4 Łączenie drzew

Łączenie dwóch reprezentowanych drzew też nie jest trudne. Po wywołaniu Access(u) i Access(v) wystarczy uczynić v lewym synem wierzchołka u.

Algorithm 4. Link(u)

```
Access(u)
Access(v)
left(u) = v
```

end

4 Analiza czasowa

Z pseudokodów widać, że wszystkie wykonywane w funkcjach operacje, poza być może procedurą Access(u), działają w czasie co najwyżej logarytmicznym. Pozostaje więc zbadać złożoność tej

procedury. W dalszym ciągu n oznacza liczbę wierzchołków reprezentowanego lasu.

4.1 Słabsze ograniczenie

Pokażemy najpierw, że:

Twierdzenie 1. *Access(u) działa w zamortyzowanym czasie* $O(\log^2 n)$.

Z pseudokodu tej procedury widać, że czas jej działania zależy od liczby iteracji głównej pętli oraz czasu wykonywania funkcji splay (wszystkie inne operacje obecne w pętli są wykonywane w czasie stałym, nie wnoszą więc nic wielkiego do złożoności). Znanym faktem jest, że funkcja splay w drzewach o tej samej nazwie działa w zamortyzowanym czasie logarytmicznym. Co więcej, każda kolejna iteracja pętli pracuje na coraz wyżej położonych preferowanych ścieżkach, i to na dokładnie jednej. Każda iteracja powoduje więc stałą liczbę zmian stanu "bycia preferowanym synem". Stąd wniosek, że jeśli pokażemy, że zmian tego stanu jest zamortyzowanie $O(\log n)$ na operację, czyli w sumie $O(m \log n)$ dla ciągu m kolejnych operacji, to będzie z tego już wynikać wymagane ograniczenie czasowe $O(\log^2 n)$ procedury Access(u).

4.1.1 Podział na krawędzie lekkie i ciężkie

Podział na krawędzie lekkie i ciężkie jest dość ogólną techniką przydatną przy analizowaniu złożoności algorytmów działających na dowolnych drzewach ukorzenionych.

Niech $size(u)$ oznacza liczbę wierzchołków całego poddrzewa u (rozważamy drzewa lasu reprezentowanego).

Definicja 3. *Krawędź łączącą wierzchołek u ze swoim ojcem v nazwiemy* ***ciężką*** *wtedy, gdy* $size(u) > \frac{1}{2} size(v)$, *a* ***lekką*** *w przeciwnym przypadku.*

Niech **lekkość(u)** oznacza liczbę lekkich krawędzi znajdujących się na ścieżce z u do korzenia. Łatwo zauważyć, że lekkość$(u) \leq \log n$. Istotnie, za każdym razem gdy idziemy lekką krawędzią w dół, liczba wierzchołków w poddrzewie zmniejsza się co najmniej dwukrotnie. Nie można jej zmniejszać w nieskończoność, po $\log n$ krokach na pewno zabraknie wierzchołków.

Co do krawędzi ciężkich, zauważmy, że z każdego wierzchołka wychodzi co najwyżej jedna taka krawędź, bo tylko jeden syn może mieć więcej niż połowę wierzchołków z poddrzewa swojego ojca. Jesteśmy już gotowi żeby przeprowadzić

Dowód. Zastosujmy podział na lekkie i ciężkie krawędzie do reprezentowanego lasu.

Pokazaliśmy, że wystarczy ograniczyć liczbę zmian stanu "bycia preferowanym". Zmiany stanu są wykonywane w procedurze Access(u) parami - z każdym odpreferowaniem wiąże się zapreferowanie. Wystarczy więc policzyć liczbę krawędzi stających się preferowanymi.

Jedno wykonanie Access(u) spowoduje preferowanie co najwyżej $\log n$ krawędzi lekkich. Wynika to stąd, że wszystkie krawędzie stające się preferowanymi leżą na ścieżce od u do korzenia a, jak zauważyliśmy wcześniej, na jednej ścieżce może być jedynie $\log n$ krawędzi lekkich.

Pozostaje ograniczyć liczbę preferowanych krawędzi ciężkich. Wywołanie Access(u) może spowodować preferowanie nawet liniowej liczby takich krawędzi, jednak rzeczywiście amortyzuje się to do ilości logarytmicznej. Rozważmy $m > n$ kolejnych wykonań procedury Access(u). Zauważmy, że liczbę zapreferowań krawędzi ciężkich możemy ograniczyć przez liczbę odpreferowań tego rodzaju krawędzi plus $n-1$. Wynika to stąd, że przed rozpoczęciem wykonywania naszych m operacji co najwyżej $n-1$ krawędzi ciężkich nie było preferowanych (bo co najwyżej tylko tyle krawędzi jest w całym lesie), a w trakcie wykonywania tego ciągu operacji zapreferowanie ponowne danej krawędzi ciężkiej musi wiązać się z odpreferowaniem jej w przeszłości (nie można przecież zapreferować krawędzi będącej już preferowaną). Stąd mamy nierówność:

$$\#\text{zapreferowań krawędzi ciężkich} \leq \#\text{odpreferowań krawędzi ciężkich} + (n-1)$$

Każde odpreferowanie krawędzi ciężkiej wiąże się jednak z zapreferowaniem krawędzi lekkiej - co wynika bezpośrednio stąd, że z danego wierzchołka wychodzi co najwyżej jedna krawędź ciężka. Ale przecież ograniczyliśmy już liczbę zapreferowań krawędzi lekkich. Ostatecznie więc mamy:

$$\#\text{zapreferowań krawędzi ciężkich} \leq \#\text{odpreferowań krawędzi ciężkich} + (n-1) \leq$$

$$\leq \#\text{zapreferowań krawędzi lekkich} + (n-1) \leq m \log n + (n-1)$$

Co daje liczbę $O(m \log n + n)$ zmian preferowań krawędzi w ciągu m operacji. Stąd i z naszych poprzednich rozważań wynika już żądane ograniczenie $O(\log^2 n)$. □

4.2 Lepsze ograniczenie

Poprzednie ograniczenie miało na celu przedstawienie ogólnej, ciekawej techniki wykorzystywanej przy badaniu algorytmów działających na drzewach, czyli podziału na krawędzie lekkie i ciężkie. Osiągnięty tam rezultat jest co prawda całkiem dobry, okazuje się jednak, że można go znacząco poprawić. Dokładniej pokażemy, że:

Twierdzenie 2. *Zamortyzowany czas wykonania procedury Access(u) jest* $O(\log n)$.

Dowód. Z pseudokodu procedury Access(u) można wywnioskować, że czas jej działania jest równy:

$$\textit{czas\_zapreferowania\_jednej\_krawędzi} * \textit{liczba\_zapreferowań\_krawędzi}$$

Z poprzedniego dowodu wiemy, że *liczba_zapreferowań_krawędzi* jest $O(\log n)$. Wystarczy więc pokazać, że zamortyzowany *czas_zapreferowania_jednej_krawędzi* jest stały. Wykorzystamy w tym celu odpowiednią funkcję potencjału.

Niech $s(u)$ oznacza liczbę wierzchołków znajdujących się pod wierzchołkiem u w drzewie nadrzędnym. Nasza funkcja potencjału będzie równa: $\Phi = \sum_u \log s(u)$. Podstawowe twierdzenie drzew splay mówi nam, że: $T(splay(u)) \leq 3(\log s(r) - \log s(u)) + 1$ gdzie r jest korzeniem drzewa pomocniczego zawierającego u. Procedura splay, będąca ważną częścią Access(u), zmienia co prawda strukturę pojedynczego drzewa pomocniczego, jednak nie ma wpływu na ogólną strukturę drzewa nadrzędnego. Wynika stąd, że wartości s zmieniają się jedynie wierzchołkom z pojedynczego drzewa

pomocniczego (które, przypomnijmy, jest drzewem splay) zawierającego u. Dalej, jeśli oznaczymy przez v wierzchołek, na który wskazuje ścieżkowy wskaźnik mieszczący się w r, to z definicji s wnioskujemy prawdziwość nierówności: $s(u) \leq s(r) \leq s(v)$. Stąd i z ograniczenia na zamortyzowany czas procedury splay dostajemy sumę teleskopową, która redukuje się i jest na pewno nie większa niż:

$$3(\log s(\text{korzeń\_reprezentowanego\_drzewa}) - \log s(u)) + O(\textit{liczba\_ zapreferowań\_ krawędzi})$$

co jest oczywiście $O(\log n)$. □

4.3 Podsumowanie

Dowiedliśmy, że zamortyzowany czas wykonania procedury Access(u) jest logarytmiczny ze względu na liczbę wierzchołków lasu. W połączeniu z naszymi poprzednimi spostrzeżeniami daje to ograniczenie logarytmiczne na wszystkie wymagane operacje. Drzewa Link-Cut są więc dobrą strukturą przechowującą zmieniające się w czasie lasy.

Literatura

[1] D. D. Sleator, R. E. Tarjan, *A Data Structure for Dynamic Trees*, Journal. Comput. Syst. Sci., 1983.