1 Dynamiczna spójność i ET-drzewa

1.1 Definicja problemu i lista znanych wyników

W tym wykładzie zajmiemy się następującym naturalnym problemem:

Problem (Dynamiczna spójność)**.** *Zaprojektować strukturę danych umożliwiającą wykonywanie następujących operacji na grafie G:*

ADD(u, v) *- łączy krawędzią wierzchołki u i v,*

DELETE(u, v) *- usuwa krawędź łączącą wierzchołki u i v,*

CONNECTED(u, v) *- zwraca TRUE wtw gdy u i v są w tej samej spójnej składowej G,*

CONNECTED$()$ *- zwraca TRUE wtw gdy G jest spójny.*

Tak jak w przypadku problemów statycznych za rozwiązanie efektywne z reguły uznaje się algorytm wielomianowy, tak w przypadku algorytmów dynamicznych dąży się do rozwiązań w których koszty wszystkich operacji są polilogarytmiczne. Dla problemu dynamicznej spójności znanych jest kilka takich rozwiązań:

1. update $O(\lg^3 n)$, query $O(\lg n/\lg\lg n)$, randomizowany (Henzinger, King (1995)),
2. update $O(\lg^2 n)$, query $O(\lg n/\lg\lg n)$, deterministyczny (Holm, de Lichtenberg, Thorup (1997)),
3. update $O(\lg n(\lg\lg n)^3)$, query $O(\lg n/\lg\lg\lg n)$, randomizowany (Thorup (2000)).

Wiadomo, że jeśli złożoność update'ów jest $O(\lg n \cdot x)$ to złożoność query musi być $\Omega(\lg n/\lg x)$. W tym sensie wszystkie wymienione wyżej algorytmy są optymalne (oczywiście drugi jest dużo bardziej optymalny niż pierwszy :)) Nie wiadomo, czy istnieje algorytm osiągający czasy $O(\lg n)$ dla update'ów i query (zapewne tak, bardzo ciekawy problem badawczy). Nie wiadomo też, czy da się uzyskać update'y $o(\lg n)$ przy polilogarytmicznych query (zapewne nie, ale pokazanie tego może być bardzo trudne).
W ramach tego wykładu omówimy drugie z wymienionych rozwiązań, tzn. algorytm Holma, de Lichtenberga i Thorupa. Algorytm Thorupa opiera się na tym samym pomyśle, ale jest dużo bardziej skomplikowany.

1.2 ET-drzewa

Zanim przejdziemy do opisu samego algorytmu rozwiążemy raz jeszcze problem z poprzedniego wykładu:

Problem (Dynamiczny las)**.** *Zaprojektować strukturę danych umożliwiającą wykonywanie następujących operacji na dynamicznym ukorzenionym lesie F:*

LINK(u, v) - *przyłącza drzewo o korzeniu u do wierzchołka v,*

CUT(u) - *usuwa krawędź łączącą wierzchołek u z jego ojcem, u staje się korzeniem swojego drzewa,*

ROOT(u) - *zwraca korzeń drzewa, w którym znajduje się u.*

Po co omawiać kolejne rozwiązanie tego samego problemu? Jest kilka powodów[1]:

1. ET-drzewa, które za chwilę omówimy są strukturą danych istotnie prostszą od od drzew link-cut Sleatora i Tarjana,
2. złożoności operacji w ET-drzewach są pesymistyczne, a nie zamortyzowane,
3. ET-drzewa lepiej nadają się do przechowywania dodatkowej informacji o drzewach (np. liczby wierzchołków w drzewie) i poddrzewach, co okaże się potrzebne w algorytmie HLT,
4. nauczenie się nowej sztuczki nikomu jeszcze nie zaszkodziło.[2]

Niech T będzie ukorzenionym drzewem o korzeniu r. Cyklem Eulera[3] T nazywamy ścieżkę $C(T)$ zaczynającą się i kończącą w r i przechodzącą każdą krawędź dokładnie dwukrotnie. ET-drzewem reprezentującym drzewo T nazywamy drzewo zrównoważone (np. 2-3-drzewo) $ET(T)$ przechowujące wierzchołki $C(T)$ w kolejności w jakiej występują one w $C(T)$. Zauważmy, że niektóre z wierzchołków mogą występować na cyklu wielokrotnie. W każdym z wystąpień dowolnego wierzchołka v przechowujemy dodatkowo wskaźnik do pierwszego i ostatniego jego wystąpienia.
Operacje LINK, CUT i ROOT można zrealizować za pomocą ET-drzew następująco:

CUT(u) - wycinamy z $ET(T)$ wierzchołki pomiędzy pierwszym a ostatnim wystąpieniem u (za pomocą operacji SPLIT i JOIN),

LINK(u, v) - wstawiamy do $ET(T(v))$ zawartość drzewa $ET(T(u))$ tuż pierwszym wystąpieniem v,

ROOT(u) - trywialne,

REROOT(u) - dodatkowa operacja, która zmienia korzeń reprezentowanego drzew — prosta implementacja za pomocą SPLIT i JOIN.

[1]Po przeczytaniu których każdy rozsądny człowiek zada pytanie "To po co omawialiśmy drzewa link-cut?". Na szczęście powód 4 działa w obie strony. Poza tym drzewa link-cut lepiej nadają się do przechowywania informacji o ścieżkach, co przydaje się w algorytmach przepływowych.

[2]Z wyjątkiem, być może, niektórych artystów cyrkowych.

[3]ET to skrót od ang. Extra-Terrestial, czyli pozaziemski.

Zwróćmy uwagę, że wszystkie operacje mają złożoność $O(\lg n)$, ale w sensie pesymistycznym, a nie zamortyzowanym jak w przypadku drzew link-cut.
Zwróćmy ponadto uwagę, że ET-drzewa można w standardowy sposób wzbogacić tak, aby przechowywały one informacje o poddrzewach, takie jak np. liczba wierzchołków (na podstawie której łatwo obliczyć liczbę wierzchołków w reprezentowanym drzewie).

1.3 Algorytm Holma, de Lichtenberga i Thorupa w ogólnym zarysie

Gdybyśmy w problemie dynamicznej spójności nigdy nie usuwali, a jedynie dodawali krawędzie, to problem byłby oczywiście trywialny.

Ćwiczenie 1. *Tak dla pewności: jak należałoby go wtedy rozwiązać?*

Usuwanie jest jednak wysoce problematyczne. Niech uv będzie usuwaną krawędzią. Musimy w jakiś sposób sprawdzić, czy u i v są nadal w tej samej spójnej składowej. Przypuśćmy, że nasz algorytm przechowuje las rozpinający G (co jest bardzo dobrym pomysłem, wszystkie wymienione we wstępie algorytmy to robią). Przed usunięciem krawędzi uv wierzchołki u i v znajdowały się w jednym drzewie T tego lasu. Po usunięciu krawędzi uv drzewo T rozbija się na dwa poddrzewa T_u, T_v i musimy sprawdzić, czy istnieje krawędź łącząca T_u z T_v, tzw. *krawędź zastępcza*. Naturalny pomysł przejrzenia wszystkich krawędzi wychodzących z T_u (bądź T_v) jest niestety zbyt kosztowny. Algorytm HLT radzi sobie z tym problemem dzieląc zbiór krawędzi na poziomy. Poziomy te z jednej strony pomagają szukać krawędzi zastępczej, z drugiej pozwalają dobrze oszacować zamortyzowany koszt operacji DELETE.
Poziomem $\ell(e)$ krawędzi e jest pewna liczba naturalna z przedziału $[0, \lg n]$. Oznaczmy przez G_i podgraf G zawierający krawędzie z poziomów $\geq i$. Mamy $G = G_0 \supseteq G_1 \supseteq \ldots$. Algorytm HLT utrzymuje las rozpinający F grafu G. Niech F_i będzie podlasem F składającym się z krawędzi na poziomie $\geq i$. Wtedy $F = F_0 \supseteq F_1 \supseteq \ldots$. Krawędzie F będziemy nazywali krawędziami *drzewowymi*, a pozostałe krawędzie krawędziami *niedrzewowymi*.
Będziemy utrzymywać następujące niezmienniki:

1. F_i jest lasem rozpinającym G_i,
2. drzewa w F_i mają $\leq \lfloor n/2^i \rfloor$.

Na początku wszystkie krawędzie mają poziom 0, a F jest dowolnym lasem rozpinającym. Oba niezmienniki są oczywiście prawdziwe. Poziomy krawędzi mogą wzrastać (podczas wykonywania operacji DELETE), ale nigdy nie maleją. Ponieważ $\lg n$ jest najwyższym możliwym poziomem, poziom krawędzi może wzrosnąć tylko $\lg n$ razy. Na tym spostrzeżeniu będzie oparta analiza złożoności algorytmu.
Przejdźmy do opisu poszczególnych operacji. W opisie tym ignorujemy szczegóły techniczne, którymi zajmiemy się w następnym podrozdziale.

ADD(u,v): Krawędzi uv przypisujemy poziom 0. Jeśli wierzchołki u, v nie były połączone, to dodajemy uv do F. Oba niezmienniki są oczywiście spełnione.

DELETE(u,v): Jeśli uv nie jest krawędzią drzewową to wystarczy ją usunąć. Jeśli natomiast uv jest drzewowa, to po jej usunięciu szukamy krawędzi zastępczej o najwyższym możliwym poziomie za pomocą pomocniczej operacji REPLACE(u, v, i). Szukana krawędź zastępcza nie może mieć poziomu wyższego niż $\ell(uv)$ (dlaczego?).

REPLACE(u,v,i)(operacja pomocnicza): Operacja ta szuka krawędzi zastępczej o najwyższym poziomie dla uv. Zakładamy przy tym, że nie ma takiej krawędzi na poziomach $> i$.

Niech T_u i T_v będą drzewami w F_i zawierającymi odpowiednio u i v. Załóżmy, że $|T_u| \leq |T_v|$. Ponieważ przed usunięciem uv spełniony był niezmiennik 2, to mamy $|T_u| \leq n/2^{i+1}$. Przeglądamy jedna po drugiej krawędzie niedrzewowe na poziomie i wychodzące z T_u. Możliwe są dwa przypadki:

- Przeglądana krawędź łączy dwa wierzchołki T_u. Wtedy zwiększamy poziom tej krawędzi (dzięki temu możliwa będzie amortyzacja) — możemy to zrobić, bo $|T_u| \leq n/2^{i+1}$.
- Przeglądana krawędź prowadzi do T_v. W tym przypadku znaleźliśmy krawędź zastępczą i dodajemy ją do F.

Jeśli żadna z krawędzi niedrzewowych na poziomie i nie okazała się być krawędzią zastępczą, wykonujemy REPLACE$(u, v, i - 1)$ o ile $i > 0$, wpp kończymy działanie algorytmu.

CONNECTED(u,v): Sprawdzamy, czy u i v znajdują się w tym samym drzewie F.

CONNECTED(): To można zaimplementować w czasie stałym. Wystarczy zliczać ADD-y, które łączą dwa drzewa i DELETE-y, które dzielą drzewo na dwie części.

1.4 Szczegóły implementacyjne algorytmu HLT

Każdy z lasów F_i jest reprezentowany za pomocą zbioru ET-drzew, po jednym dla każdego drzewa w F_i. Niech T będzie drzewem w F_i, a $ET(T)$ odpowiadającym mu ET-drzewem. Niech ponadto v będzie wierzchołkiem T, a $\hat{v}$ pierwszym jego wystąpieniem w $ET(T)$. Wtedy $\hat{v}$ przechowuje listę krawędzi drzewowych/niedrzewowych incydentych z v i mających poziom i. Ponadto wzbogacamy ET-drzewo w taki sposób, aby możliwe było przeglądanie krawędzi na poziomie i w czasie $O(\lg n)$ na krawędź. Łączna pamięć potrzebna na przechowywanie powyższych informacji jest $O(m + n \lg n)$ (algorytm Thorupa wymaga pamięci liniowej).

Przeanalizujmy teraz poszczególne operacje. Dodanie nowej krawędzi zajmuje oczywiście czas $O(\lg n)$ (uwaga na REROOT przy łączeniu drzew). Obciążamy je jednak dodatkowym kosztem $O(\lg^2 n)$ (po $O(\lg n)$ na każdą przyszłą zmianę poziomu), który pozwoli nam w przyszłości pokryć koszt nieudanego sprawdzania, czy krawędź jest krawędzią zastępczą — $O(\lg n)$ na szukanie kolejnej krawędzi na poziomie i oraz $O(\lg n)$ na przenoszenie krawędzi na wyższy poziom.
Usuwanie krawędzi niedrzewowej wymaga czasu $O(\lg n)$. Usuwając krawędź drzewową e musimy "pociąć" wszystkie lasy F_j dla $j \leq \ell(e)$, co wymaga czasu $O(\lg^2 n)$. Następnie co najwyżej $O(\lg n)$ razy wykonujemy algorytm REPLACE, każde wykonanie ma koszt $O(\lg n)$ plus koszt przeglądania krawędzi, za który placi operacja ADD. Jeśli krawędź zastępcza zostaje znaleziona, to musimy połączyć drzewa w $O(\lg n)$ lasach, co znów daje koszt $O(\lg^2 n)$.
Zapytanie CONNECTED(u, v) w oczywisty sposób wymaga czasu $O(\lg n)$.
Mamy zatem strukturę danych, która wykonuje operacje ADD i DELETE w czasie $O(\lg^2 n)$ i operacje CONNECTED w czasie $O(\lg n)$.

Ćwiczenie 2. *W jaki sposób uzyskać czas $O(\lg n / \lg \lg n)$ dla operacji CONNECTED?*

Wskazówka. Zmień arność ET-drzew.