

	R	NEXT	EXTERNAL		
			LIST	SIZE	_NAME
1	2	3	1	6	3
2	2	4	2	2	4
3	2	5	3	—	—
4	2	8	4	—	—
5	2	7			
6	1	0			
7	2	0			
8	2	1			

Zbiory (z nazwami zewnętrznymi)			INTERNAL	
			_NAME	
3 = {6}			1	—
4 = {1, 2, 3, 4, 5, 7, 8}			2	—
			3	1
			4	2

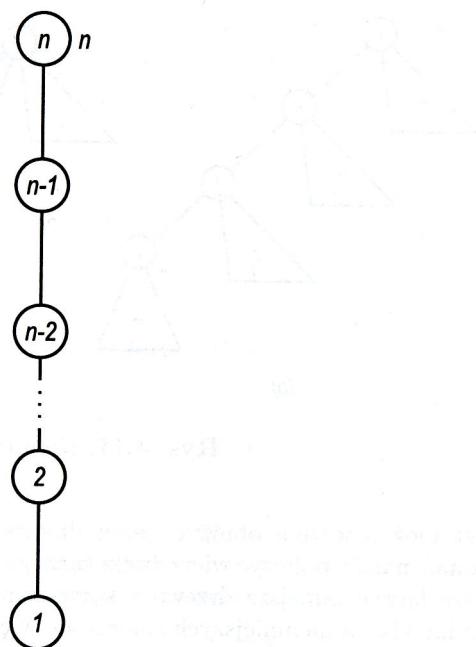
Rys. 4.15. Struktury danych po wykonaniu instrukcji UNION

Z twierdzenia 4.3 wynika, że gdy wykonuje się m instrukcji FIND i najwyżej $n - 1$ instrukcji UNION, czas tego wykonania jest $O(\text{MAX}(m, n \log n))$. Jeżeli m jest rzędu przynajmniej $n \log n$, algorytm jest optymalny z dokładnością do czynnika stałego. Zobaczmy, że często m jest $O(n)$ i bywa lepiej niż $O(\text{MAX}(m, n \log n))$.

4.7. Struktury drzew dla problemu UNION-FIND

W poprzednim punkcie przedstawiona jest struktura danych dla problemu UNION-FIND, która pozwala przetworzyć $n - 1$ instrukcji UNION i $O(n \log n)$ instrukcji FIND w czasie $O(n \log n)$. Obecnie przedstawimy strukturę, w której rodzina zbiorów jest reprezentowana przez las drzew. Ta struktura pozwala przetwarzać $O(n)$ instrukcji UNION i FIND w czasie prawie liniowym.

Założymy, że każdy ze zbiorów A jest reprezentowany przez drzewo nieskierowane z korzeniem T_A , tak że elementy A odpowiadają wierzchołkom T_A . Nazwę zbioru ma korzeń drzewa. Instrukcję o postaci UNION(A, B, C) można wykonać, czyniąc korzeń T_A synem korzenia T_B i zmieniając nazwę u korzenia T_B na C . Instrukcję o postaci FIND(i) można wykonać przez odszukanie wierzchołka, który reprezentuje i w pewnym drzewie T lasu, oraz przejście drogi z tego wierzchołka do korzenia T , gdzie jest nazwa zbioru, do którego należy i .



Rys. 4.16. Drzewo po wykonaniu wszystkich instrukcji UNION

W myśl tego schematu koszt łączenia dwóch drzew jest stały. Koszt instrukcji $\text{FIND}(i)$ jest jednak rzędu długości drogi z wierzchołka i do jego korzenia. Ta droga może mieć długość $n - 1$. Wobec tego koszt wykonania $n - 1$ instrukcji UNION, a potem n instrukcji FIND, może sięgnąć $O(n^2)$. Rozważmy na przykład koszt następującej sekwencji:

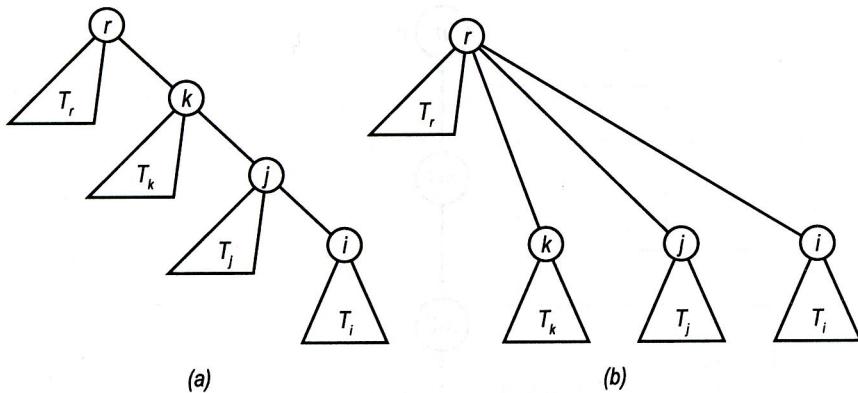
```

UNION(1, 2, 3)
UNION(2, 3, 3)
:
UNION(n - 1, n, n)
FIND(1)
FIND(2)
:
FIND(n)

```

Wynikiem wykonania $n - 1$ instrukcji UNION jest drzewo na rysunku 4.16. Koszt n instrukcji FIND jest proporcjonalny do:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}.$$



Rys. 4.17. Kompresja ścieżki

Koszt można jednak obniżyć, jeżeli drzewa mogą być zrównoważone. Aby tego dokonać, należy policzyć wierzchołki każdego drzewa i przy łączeniu dwóch zbiorów zawsze łączyć mniejsze drzewo z korzeniem większego. Stanowi to odpowiednik technikiłączania mniejszych zbiorów do większych, użytej w poprzednim punkcie.

Lemat 4.1. Jeżeli przy wykonaniu każdej z instrukcji UNION korzeń drzewa o mniejszej liczbie wierzchołków (równowagę można przełamać dowolnie) staje się synem korzenia większego drzewa, to wysokość żadnego drzewa lasu nie jest większa ani równa h , chyba że drzewo ma przynajmniej 2^h wierzchołków.

Dowód. Dowód jest indukcyjny względem h . Dla $h = 0$ lemat jest prawdziwy, gdyż każde drzewo ma przynajmniej jeden wierzchołek. Założymy indukcyjnie, że lemat jest prawdziwy dla wszystkich wartości mniejszych niż $h \geq 1$. Niech T będzie drzewem wysokości h przy najmniejszej liczbie wierzchołków. Zatem T zostało otrzymane przez łączenie drzew T_1 i T_2 , gdzie T_1 ma wysokość $h - 1$ i T_1 ma nie więcej wierzchołków niż T_2 . Z założenia indukcyjnego T_1 ma przynajmniej 2^{h-1} wierzchołków, więc T_2 ma ich przynajmniej 2^{h-1} , a stąd T ma przynajmniej 2^h wierzchołków. \square

Rozważmy w najgorszym przypadku czas wykonania ciągu n instrukcji UNION i FIND, który używa struktury danych lasu i w operacji UNION korzeń mniejszego drzewa staje się synem korzenia większego. Żadne drzewo nie może mieć wysokości większej, niż $\log n$. Wykonanie $O(n)$ instrukcji UNION i FIND kosztuje więc co najwyżej $O(n \log n)$ jednostek czasu. Oszacowanie to jest dokładne w tym sensie, że istnieją ciągi n instrukcji, które zużywają czas proporcjonalny do $n \log n$.

Kolejną modyfikację tego schematu nazywamy *kompresją ścieżki*. Ponieważ koszt naszych instrukcji FIND dominuje w ogólnym koszcie, postaramy się zmniejszyć koszt instrukcji FIND. Przy wykonaniu każdej instrukcji $\text{FIND}(i)$ przechodzimy drogę z wierzchołka i do jego korzenia r . Niech $i, v_1, v_2, \dots, v_n, r$ będą wierzchołkami na tej drodze. Każdy wierzchołek $i, v_1, v_2, \dots, v_{n-1}$ czynimy synem korzenia.

```

begin
  opróżnij listę LIST;
   $v \leftarrow \text{ELEMENT}[i]$ ;
  while  $\text{FATHER}[v] \neq 0$  do
    begin
      dodaj  $v$  do LIST;
       $v \leftarrow \text{FATHER}[v]$ 
    end;
    comment  $v$  jest teraz korzeniem;
    print  $\text{NAME}[v]$ ;
    for każdy  $w$  na liście LIST do  $\text{FATHER}[w] \leftarrow v$ 
  end

```

Rys. 4.18. Wykonanie instrukcji $\text{FIND}(i)$

Rysunek 4.17(b) pokazuje, jaki skutek dla drzewa z rysunku 4.17(a) ma instrukcja $\text{FIND}(i)$. Pełny algorytm łączenia drzew dla problemu UNION-FIND wraz z kompresją ścieżek jest wyrażony w następującym algorytmie.

Algorytm 4.3. Szybki algorytm sumy zbiorów rozłącznych.

Wejście. Ciąg σ instrukcji UNION i FIND dla kolekcji zbiorów, których elementami są liczby całkowite od 1 do n . Zakładamy, że nazwami zbiorów są także liczby całkowite od 1 do n oraz, że element i należy do zbioru jednoelementowego o nazwie i .

Wyjście. Ciąg odpowiedzi na instrukcje FIND należące do σ . Odpowiedź na instrukcję FIND trzeba podać przed przeczytaniem następnej instrukcji, która należy do σ .

Metoda. Algorytm opisujemy w trzech częściach: inicjowanie, odpowiedź na FIND i odpowiedź na UNION.

1. *Inicjowanie.* Dla każdego elementu i , $1 \leq i \leq n$, tworzymy wierzchołek v_i . Nastawiamy $\text{COUNT}[v_i] = 1$, $\text{NAME}[v_i] = i$ oraz $\text{FATHER}[v_i] = 0$. Początkowo każdy wierzchołek v_i jest samostanym drzewem. Aby znaleźć korzeń zbioru i tworzymy tablicę ROOT, w której $\text{ROOT}[i]$ wskazuje na v_i . Aby znaleźć wierzchołek z elementem i , tworzymy tablicę ELEMENT, w której na początku $\text{ELEMENT}[i] = v_i$.
2. *Wykonanie* $\text{FIND}(i)$. Program jest pokazany na rysunku 4.18. Wychodzimy z wierzchołka $\text{ELEMENT}[i]$ i przechodzimy drogę do korzenia drzewa, tworząc listę napotkanych wierzchołków. U korzenia drukujemy nazwę zbioru i synem korzenia czynimy każdy z wierzchołków pokonanej drogi.
3. *Wykonanie* $\text{UNION}(i, j, k)$. W tablicy ROOT znajdujemy korzenie drzew, reprezentujących zbiór i oraz zbiór j . Korzeń mniejszego drzewa czynimy synem korzenia większego. Patrz rys. 4.19. \square

```

begin
  wlg zakładamy, że COUNT[ROOT[i]] <= COUNT[ROOT[j]]
  otherwise zamieniamy i z j in
  begin
    LARGE  $\leftarrow$  ROOT[j];
    SMALL  $\leftarrow$  ROOT[i];
    FATHER[SMALL]  $\leftarrow$  LARGE;
    COUNT[LARGE]  $\leftarrow$  COUNT[LARGE] + COUNT[SMALL];
    NAME[LARGE]  $\leftarrow$  k;
    ROOT[k]  $\leftarrow$  LARGE
  end
end

```

Rys. 4.19. Wykonanie instrukcji UNION(*i, j, k*)

Pokażemy, że kompresja ścieżek znacznie przyspiesza algorytm. Aby obliczyć w jakiej mierze, wprowadzimy dwie funkcje *F* i *G*. Niech:

$$\begin{aligned} F(0) &= 1, \\ F(i) &= 2^{F(i-1)} \quad \text{dla } l > 0. \end{aligned}$$

Funkcja *F* ma niezwykle szybki wzrost, jak pokazuje tabela na rysunku 4.20. Funkcja *G* jest zdefiniowana jako najmniejsza liczba całkowita *k*, taka że $F(k) \geq n$. Funkcja *G* ma wzrost niezwykle wolny. Istotnie, $G(n) \leq 5$ dla wszystkich „praktycznych” wartości *n*, tzn. wszystkich n^{65536} .

Udowodnimy, że algorytm 4.3 wykona ciąg σ złożony z *cn* instrukcji UNION i FIND w czasie co najwyżej równym $c'nG(n)$, gdzie *c* oraz *c'* są stałymi i *c'* zależy od *c*. Dla prostoty zakładamy, że wykonanie instrukcji UNION zużywa jedną „jednostkę czasu” oraz, że wykonanie instrukcji FIND(*i*) zużywa liczbę jednostek czasu, która jest proporcjonalna do liczby wierzchołków na drodze z wierzchołka oznaczonego etykietą *i* do korzenia drzewa, które zawiera ten wierzchołek.⁵

Definicja. Dogodnie jest definiować *rzqd* wierzchołka ze względu na ciąg σ instrukcji UNION i FIND jak następuje:

1. usuń instrukcje FIND z σ ,
2. wykonaj tak otrzymany ciąg σ' instrukcji UNION,
3. rzqd wierzchołka *v* jest to wysokość *v* w tak otrzymanym lesie.

Wyprowadzimy teraz pewne istotne własności rzędu wierzchołka.

Lemat 4.2. Istnieje co najwyżej $n/2^r$ wierzchołków rzędu *r*.

⁵Jedna „jednostka czasu” wymaga w tym sensie pewnej stałej liczby kroków RAM. Ponieważ zaniedbujemy czynniki stałe, tezy o rzędach wielkości możemy także wyrażać, mówiąc o „jednostkach czasu”.

n	$F(n)$
0	1
1	2
2	4
3	16
4	65536
5	2^{65536}

Rys. 4.20. Pewne wartości F

Dowód. Z lematu 4.1 wynika, że każdy wierzchołek rzędu r ma przynajmniej 2^r potomków w lesie otrzymanym przez wykonanie σ' . Skoro zbiory potomków do wolnych dwóch różnych wierzchołków o tej samej wysokości w lesie są rozłączne i skoro istnieje co najwyżej $n/2^r$ rozłącznych zbiorów nie mniej niż 2^r wierzchołków, to może istnieć co najwyżej $n/2^r$ wierzchołków rzędu r . \square

Wniosek. Nie istnieje wierzchołek rzędu większego niż $\log n$.

Lemat 4.3. Jeżeli w czasie wykonania σ , w jest potomkiem właściwym v , to rząd w jest mniejszy niż rząd v .

Dowód. Jeżeli w czasie wykonania σ , w zostaje potomkiem v , to w jest potomkiem v w lesie otrzymanym przez wykonanie ciągu σ' . Wobec tego wysokość w musi być mniejsza niż wysokość v , a więc rząd w jest mniejszy niż rząd v . \square

Rzędy dzielimy teraz na *grupy*. Rząd r zalicza się do grupy $G(r)$. Na przykład rzędy 0 i 1 należą do grupy 0, rząd 2 do grupy 1, rzędy 3 i 4 do grupy 2, rzędy 5 – 16 do grupy 3. Dla $n > 1$, największy możliwy rząd $\lfloor \log n \rfloor$ należy do grupy rzędów $G(\lfloor \log n \rfloor) \leq G(n) - 1$.

Rozważmy koszt wykonania ciągu σ złożonego z cn instrukcji UNION i FIND. Skoro każda instrukcja UNION może być wykonana kosztem jednej jednostki czasu, wszystkie instrukcje UNION, należące do σ , mogą być wykonane w czasie $O(n)$. Aby oszacować koszt wykonania wszystkich instrukcji FIND posłużymy się ważną regułą „księgowania”. Kosztem wykonania pojedynczej instrukcji FIND obciążymy samą instrukcję FIND oraz te spośród wierzchołków drogi w strukturze danych lasu, które faktycznie przenosimy. Ogólny koszt obliczamy przez sumowanie po wszystkich instrukcjach FIND kosztu, którym są obciążone oraz sumowanie kosztu wierzchołków po wszystkich wierzchołkach w lesie.

Koszt instukcji $\text{FIND}(i)$ tworzymy jak następuje. Niech v będzie wierzchołkiem na drodze z wierzchołka, który reprezentuje i , do korzenia drzewa, które zawiera i .

1. Jeżeli v jest korzeniem lub jeżeli $\text{FATHER}[v]$ jest w innej grupie rzędów niż v , to jednostką czasu obciążamy instrukcję FIND .
2. Jeżeli v i jego ojciec są w tej samej grupie rzędów, to jednostką czasu obciążamy v .

Z lematu 4.3 wynika, że rzędy wierzchołków monotonicznie rosną, jeżeli idziemy ku górze drogi, mamy też co najwyżej $G(n)$ różnych grup rzędów, więc żadna instrukcja FIND nie jest obciążona kosztem większym niż $G(n)$ jednostek czasu wedle reguły 1. Jeżeli zastosowanie ma reguła 2., to wierzchołek v jest przenoszony i staje się synem wierzchołka wyższego rzędu niż ten, który jest rzędem byłego ojca tego wierzchołka. Jeżeli wierzchołek v należy do grupy $g > 0$, to v może być przeniesiony i obciążony kosztem co najwyżej $F(g) - F(g-1)$ razy, zanim otrzyma ojca z wyższej grupy rzędów. (W grupie rzędów 0 wierzchołek może być przeniesiony co najwyżej jeden raz, zanim otrzyma ojca z wyższej grupy.) Następnie koszt przenoszenia v będzie obciążać instrukcję FIND na mocy reguły 1.

Aby uzyskać górne ograniczenie kosztów, którymi obciążane są wierzchołki, mnożymy największy możliwy koszt wierzchołka z pewnej grupy rzędów przez liczbę wierzchołków tej grupy i sumujemy po wszystkich grupach rzędów. Niech $N(g)$ będzie liczbą wierzchołków w grupie rzędów $g > 0$. Wtedy z lematu 4.2:

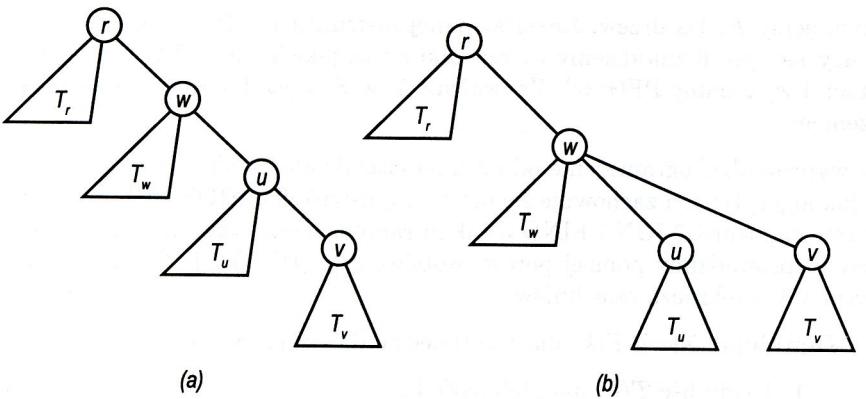
$$\begin{aligned} N(g) &\leq \sum_{r=F(g-1)+1}^{F(g)} n/2^r \\ &\leq (n/2^{F(g-1)+1}) [1 + \frac{1}{2} + \frac{1}{4} + \dots] \\ &\leq n/2^{F(g-1)} \\ &\leq n/F(g) \end{aligned}$$

Koszt maksymalny dowolnego wierzchołka z grupy rzędów $g > 0$ jest mniejszy lub równy $F(g) - F(g-1)$. Koszt maksymalny wszystkich wierzchołków z grupy rzędów g jest więc ograniczony przez n . Dotyczy to również $g = 0$. Skoro mamy co najwyżej $G(n)$ grup rzędów, koszt maksymalny wszystkich wierzchołków jest równy $nG(n)$. Czas, jakiego wymaga przetwarzanie cn instrukcji FIND , jest to najwyżej $cnG(n)$ z obciążenia instrukcji FIND i najwyżej $nG(n)$ z obciążenia wierzchołków. Mamy zatem następujące twierdzenie.

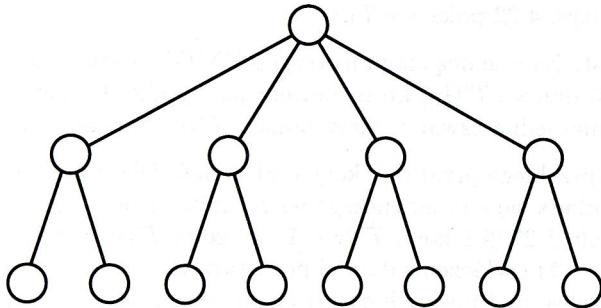
Twierdzenie 4.4. Niech c będzie dowolną stałą. Wtedy istnieje inna stała c' zależna od c taka, że algorytm 4.3 wykona na n elementach ciąg σ złożony z cn instrukcji UNION i FIND w co najwyżej $c'nG(n)$ jednostkach czasu

Dowód. Na podstawie powyższej dyskusji. □

Można pokazać, co pozostawiamy jako ćwiczenie, że jeżeli w ciągu σ dozwolone są prócz UNION i FIND również operacje pierwotne INSERT i DELETE , to σ nadal daje się wykonać w czasie $O(nG(n))$.



Rys. 4.21. Skutek operacji częściowego FIND

Rys. 4.22. Drzewo $T(2)$

Nie wiadomo, czy twierdzenie 4.4 opisuje dokładne oszacowanie czasu działania algorytmu 4.3. Poniżej udowodnimy jednak, jako wynik ciekawy teoretycznie, że czas działania algorytmu 4.3 nie jest liniowy względem n . Zrobimy to konstrując pewien ciąg instrukcji UNION i FIND, którego przetworzenie zabiera algorytmowi 4.3 więcej niż liniową ilość czasu.

Warto wprowadzić w tym celu nową operację na drzewach, którą nazwiemy *częściowym FIND*, bądź w skrócie PF (od *partial find*). Niech T będzie drzewem, w którym $v, v_1, v_2, \dots, v_m, w$ jest drogą z wierzchołka v do jego przodka w . (w nie musi być korzeniem.) Operacja $PF(v, w)$ z $v, v_1, v_2, \dots, v_{m-1}$ czyni synów wierzchołka w . Mówimy, że jest to częściowe FIND *długości* $m+1$ (jeżeli $v = w$, długością jest 0). Rys. 4.21(b) pokazuje, jaki skutek dla drzewa z rysunku 4.21(a) ma PF(v, w).

Przypuśćmy, że dany jest ciąg σ instrukcji UNION i FIND. Gdy w ciągu σ wykonujemy daną instrukcję FIND, znajdujemy wierzchołek v w pewnym drzewie T i przechodzimy drogę z v do korzenia w tego drzewa T . Przypuśmy teraz, że wykonujemy wtedy tylko instrukcje UNION, zaniedbując instrukcje FIND. Stąd

otrzymujemy F , las drzew. Rezultat danej instrukcji FIND z ciągu σ da się jednak uzyskać, jeżeli znajdziemy wierzchołki v i w , jakich używa FIND w pierwotnej postaci, i wykonamy $\text{PF}(v, w)$. Zauważmy, że w F wierzchołek w może już nie być korzeniem.

Aby wyprowadzić ograniczenie od dołu na czas działania algorytmu 4.3, rozważmy, jak algorytm ten zachowuje się dla ciągu instrukcji UNION i PF, który można zastąpić ciągiem UNION i FIND o takim samym czasie działania. Ze specjalnych drzew wyprowadzimy poniżej pewne osobliwe ciągi UNION i PF, dla których algorytm 4.3 przekracza czas liniowy.

Definicja. Niech $T(k)$ dla $k \geq 0$ będzie drzewem takim, że:

1. każdy liść $T(k)$ ma głębokość k ,
2. każdy wierzchołek wysokości h ma 2^h synów, $h \geq 1$.

Korzeń $T(k)$ ma więc 2^k synów, z których każdy jest korzeniem kopii drzewa $T(k-1)$. Rys. 4.22 pokazuje $T(2)$.

Lemat 4.4. Za pomocą ciągu instrukcji UNION można utworzyć dla dowolnego $k \geq 0$ drzewo $T'(k)$, które zawiera jako podgraf drzewo $T(k)$. Ponadto przynajmniej jedna czwarta wierzchołków $T'(k)$ jest liśćmi $T(k)$.

Dowód. Dowód przebiega przez indukcję względem k . Dla $k = 0$ lemat jest oczywisty, gdyż $T(0)$ składa się z pojedynczego wierzchołka. Konstrukcja $T'(k)$ dla $k > 0$ wymaga konstrukcji $2^k + 1$ kopii $T'(k-1)$. Drzewo $T'(k)$ powstaje przez wybór jednej kopii $T'(k-1)$ iłączenie do niej pozostałych, po jednej na raz. Korzeń tak otrzymanego drzewa ma (między innymi) 2^k synów, z których każdy jest korzeniem $T'(k-1)$.

Niech $N'(k)$ będzie liczbą wszystkich wierzchołków $T'(k)$ i niech $L(k)$ będzie liczbą liści $T(k)$. Wtedy:

$$\begin{aligned} N'(0) &= 1 \\ N'(k) &= (2^k + 1)N'(k-1) \quad \text{dla } k \geq 1, \end{aligned}$$

oraz

$$\begin{aligned} L(0) &= 1 \\ L(k) &= 2^k L(k-1) \quad \text{dla } k \geq 1; \end{aligned}$$

stąd

$$\frac{L(k)}{N'(k)} = \frac{\prod_{i=1}^k 2^i}{\prod_{i=1}^k (2^i + 1)} = \frac{2}{3} \prod_{i=2}^k \frac{1}{(1 + 2^{-i})} \quad \text{dla } k \geq 1 \quad (4.3)$$

Dla $i \geq 2$ mamy $\log_e(1 + 2^{-i}) < 2^{-i}$, zatem:

$$\log_e \left(\prod_{i=2}^k \frac{1}{1 + 2^{-i}} \right) \geq - \sum_{i=2}^k 2^{-i} \geq -\frac{1}{2} \quad (4.4)$$

Na podstawie (4.3) i (4.4) mamy:

$$\frac{L(k)}{N'(k)} \geq \frac{2}{3} e^{-1/2} \geq \frac{1}{4},$$

co kończy dowód lematu. \square

Podemy ciąg instrukcji UNION i PF, który zbuduje drzewo $T'(k)$ i wykona instrukcje PF na liściach podgrafa $T(k)$. Przedtem pokażemy, że dla każdego $l > 0$ istnieje takie k , że po kolej na każdym z liści $T(k)$ można wykonać PF długości l .

Definicja. Niech $D(c, l, h)$ będzie najmniejszą wartością k taką, że po zastąpieniu w $T(k)$ każdego poddrzewa, którego korzeń ma wysokość h , dowolnym drzewem, które ma l liści i wysokość przynajmniej 1, możemy wykonać PF długości c na każdym z liści tak otrzymanego drzewa.

Lemat 4.5. Wartość $D(c, l, h)$ jest określona (tj. skończona) dla wszystkich c, l i h większych od zera.

Dowód. Dowód wymaga podwójnej indukcji. Chcemy udowodnić wynik przez indukcję względem c . Jednak aby udowodnić wynik dla c na podstawie wyniku dla $c - 1$, musimy przeprowadzić indukcję względem l .

Przypadek bazowy, $c = 1$, jest łatwy. $D(1, l, h) = h$ dla wszystkich l i h , skoro PF długości 1 nie przenosi żadnych wierzchołków.

Założymy dla indukcji względem c , że dla wszystkich l i h określona jest wartość $D(c - 1, l, h)$. Musimy udowodnić, że dla wszystkich l i h określona jest wartość $D(c, l, h)$. Wykażemy to przez indukcję względem l .

Dla dowodu bazy tej indukcji musimy wykazać, że:

$$D(c, 1, h) \leq D(c - 1, 2^{h+1}, h + 1)$$

Zauważmy, że gdy $l = 1$, podstawiamy drzewa o jednym liściu za poddrzewa z koreniami na wierzchołkach wysokości h w $T(k)$ dla pewnego k . Niech zbiorem wierzchołków wysokości h w tym $T(k)$ będzie H . Jest jasne, że w tak zmienionym drzewie każdy liść jest potomkiem właściwym pewnego określonego elementu H . Stąd, jeżeli możemy wykonać instrukcje PF długości $c - 1$ na wszystkich elementach H , możemy z pewnością wykonać instrukcje PF długości c na wszystkich liściach.

Niech $k = D(c - 1, 2^{h+1}, h + 1)$. Z założenia indukcji względem c wiemy, że k istnieje. Jeżeli rozpatrujemy w $T(k)$ wierzchołki wysokości $h + 1$, widzimy, że każdy z nich ma 2^{h+1} synów, z których wszyscy należą do H . Jeżeli usuniemy z $T(k)$ wszystkich potomków właściwych wierzchołków w zbiorze H , to uzyskamy tyle, co

przez podstawienie za każde poddrzewo, które ma korzeń na wysokości $h+1$, drzewa wysokości 1 o 2^{h+1} liściach. Z definicji D wynika, że $k = D(c-1, 2^{h+1}, h+1)$ jest dostatecznie duże, by instrukcje PF długości $c-1$ dały się wykonać na wszystkich liściach tego drzewa (tj. elementach H).

Aby dokończyć indukcję względem c , musimy wykonać krok indukcyjny dla l . Pokażemy mianowicie, że:

$$D(c, l, h) \leq D(c-1, 2^{D(c, l-1, h)(1+D(c, l-1, h)/2)}, D(c, l-1, h)) \quad \text{dla } l > 1. \quad (4.5)$$

Dla dowodu (4.5) założymy, że $k = D(c, l-1, h)$ i k' jest prawą stroną (4.5). Musimy pokazać, jak za każdy wierzchołek wysokości h podstawić w $T'(k)$ drzewo o l liściach, a następnie wykonać PF długości c na każdym liściu. Zacznijmy od wykonania PF na $l-1$ spośród liści każdego podstawionego drzewa. Z założenia indukcyjnego dla indukcji względem l wynika, że można wykonać instrukcje PF na $l-1$ spośród liści każdego drzewa, podstawionego za poddrzewa.

Po wykonaniu instrukcji PF na $l-1$ spośród liści stwierdzamy, że l -ty liść każdego podstawionego drzewa ma innego ojca niż każdy l -ty liść dowolnego innego podstawionego drzewa. Zbiór tych ojców nazwiemy F . Jeżeli możemy wykonać instrukcje PF długości $c-1$ na tych ojcach, to możemy wykonać instrukcje PF długości c na liściach. Niech S będzie poddrzewem, którego korzeń miał wysokość k w $T(k')$. Łatwo sprawdzić, że S ma $2^{k(k+1)/2}$ liści w $T(k')$. Zatem po wykonaniu instrukcji PF liczba wierzchołków S , które należą także do F , jest co najwyżej równa $2^{k(k+1)/2}$. To co pozostało z S można uważać za dowolne drzewo o $2^{k(k+1)/2}$ liściach, czyli wierzchołkach ze zbioru F . (4.5) zachodzi założenie indukcyjnych dla c i l . \square

Twierdzenie 4.5. Algorytm 4.3 ma złożoność czasową, która jest większa niż cn dla dowolnej stałej c .

Dowód. Założymy, że istnieje stała c taka, że algorytm 4.3 wykona dowolny ciąg $n-1$ instrukcji MERGE⁶ i n instrukcji FIND w nie więcej niż cn jednostkach czasu. Wybierzmy $d > 4c$ i obliczmy $k = D(d, 1, 1)$. Skonstruujmy $T'(k)$ za pomocą ciągu instrukcji UNION. Skoro możemy wykonać PF długości d na każdym liściu zawartego w $T'(k)$ drzewa $T(k)$ i skoro liście $T(k)$ stanowią więcej niż jedną czwartą wierzchołków $T'(k)$, ten ciąg instrukcji UNION i PF wymaga więcej niż cn jednostek czasu; sprzeczność. \square

4.8. Zastosowania i rozszerzenia algorytmu UNION-FIND

Problem drzewa rozpinającego z przykładu 4.1. w sposób naturalny prowadził do ciągu pierwotnych instrukcji UNION i FIND. W tym punkcie przedstawiamy kilka

⁶Por. ćwiczenie *4.40 i tekst nad definicją $T(k)$ — przyp. tłum.