4b. Teoria drzew - przechodzenie

Grzegorz Kosiorowski

Uniwersytet Ekonomiczny w Krakowie

Drzewo uporządkowane

Choć drzewo poszukiwań binarnych jest bardzo praktyczne, gdy chcemy, by program miał szybki dostęp do opartej na nim bazy danych, to w praktyce nie zawsze takie bazy danych się tworzy (np. typowa struktura katalogów zazwyczaj dopuszcza więcej niż dwa podkatalogi każdego katalogu). Dlatego taką definicję warto uogólnić:

Drzewo uporządkowane

Jeśli dla każdego węzła drzewa z wyróżnionym korzeniem jego dzieci tworzą zbiór uporządkowany, to mówimy, że drzewo jest *uporządkowane*. Na rysunku, tak uporządkowane dzieci rysujemy od strony lewej do prawej. Kolejność dzieci u i v zapisujemy $u < v$ (nawet jeśli nie mają etykiet liczbowych).

Drzewo uporządkowane - uwagi

- Typowym porządkiem „nieliczbowym” ustawiania dzieci w kolejności może być porządek alfabetyczny.
- W szczególności, drzewo poszukiwań binarnych było uporządkowane - lewe dziecko poprzedzało prawe.
- Od tego momentu, o wszystkich drzewach, które się pojawią, zakładamy, że są uporządkowane.

Algorytmy przechodzenia drzewa

Algorytmy przechodzenia drzewa

Algorytm przechodzenia drzewa nazywamy algorytm, który odwiedza wszystkie wierzchołki skończonego uporządkowanego drzewa z wyróżnionym korzeniem i ustawia je w ciąg według pewnej reguły.

Warto zauważyć, że w praktyce przechodzenie drzewa sprowadza się do wykorzystania jednego z algorytmów przeszukiwania grafu (potencjalnie z dodatkowymi zasadami kolejności wypisywania wierzchołków) zastosowanego do drzew.
Trzy najbardziej popularne takie algorytmy są oparte na algorytmie przeszukiwania w głąb. Są to PREORDER, POSTORDER i INORDER (przy czym ten trzeci działa tylko dla drzew binarnych), ustawiające wierzchołki w porządkach odpowiednio prefiksowym, postfiksowym i infiksowym. Wszystkie te algorytmy są rekurencyjne.

Porządek prefiksowy

Porządek prefiksowy

Porządek prefiksowy drzewa z wyróżnionym korzeniem to takie uporządkowanie wierzchołków, w którym korzeń drzewa umieszczamy na początku listy, a dalej znajdują się poddrzewa uporządkowane w kolejności swoich korzeni (rysunkowo: kolejność od lewej do prawej). Innymi słowy, dzieci w tym porządku pojawiają się po rodzicach.

Tworzy się go algorytmem PREORDER.

Porządek prefiksowy - przykładowe zastosowania

Porządek prefiksowy używany jest w sytuacjach, gdy pewną operację musimy wykonać wpierw na rodzicach, by móc wykonać ją na dzieciach.

- Kopiowanie drzew: najczęściej drzewo będące kopią innego najłatwiej tworzyć „od góry", dodając wierzchołki dopiero wtedy, gdy można je przyłączyć do ich rodziców.
- Gdy drzewo z wyróżnionym korzeniem interpretujemy jako graf skierowany, porządek prefiksowy zadaje etykietowanie uporządkowane.
- *Notacji polska* (wynaleziona w 1929 roku przez Jana Łukasiewicza), czyli beznawiasowy zapis wyrażeń logicznych i arytmetycznych: najpierw działanie, a potem obiekty na którym jest wykonywane (na przykład $2 * (3 + 4)$ jest zapisywane jako $* \; 2 \; + \; 3 \; 4$). Warianty tej notacji są stosowane w niektórych językach programowania (i w starych kalkulatorach).

Algorytm PREORDER

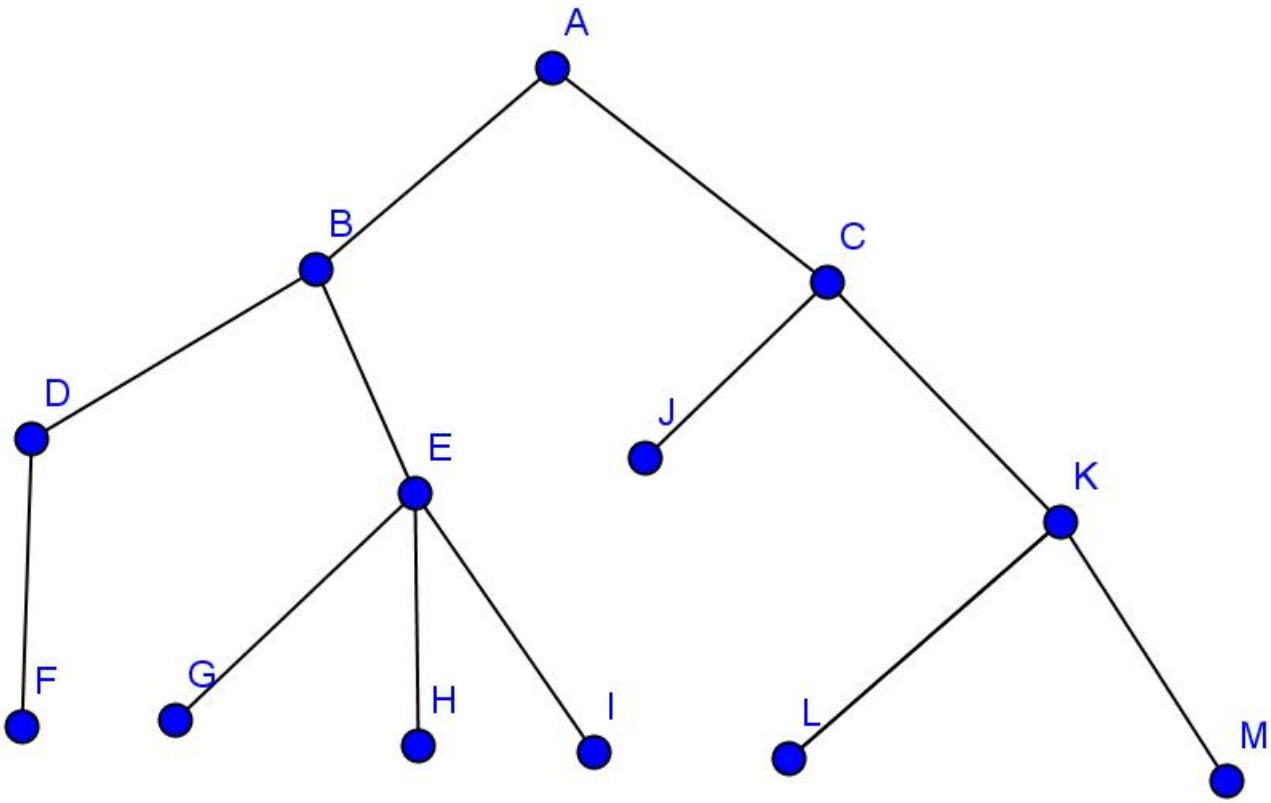
PREORDER(v)

Dane: Skończone, uporządkowane drzewo z wyróżnionym korzeniem v.

Zmienne: $L(v)$ - lista wierzchołków drzewa, w-węzeł.

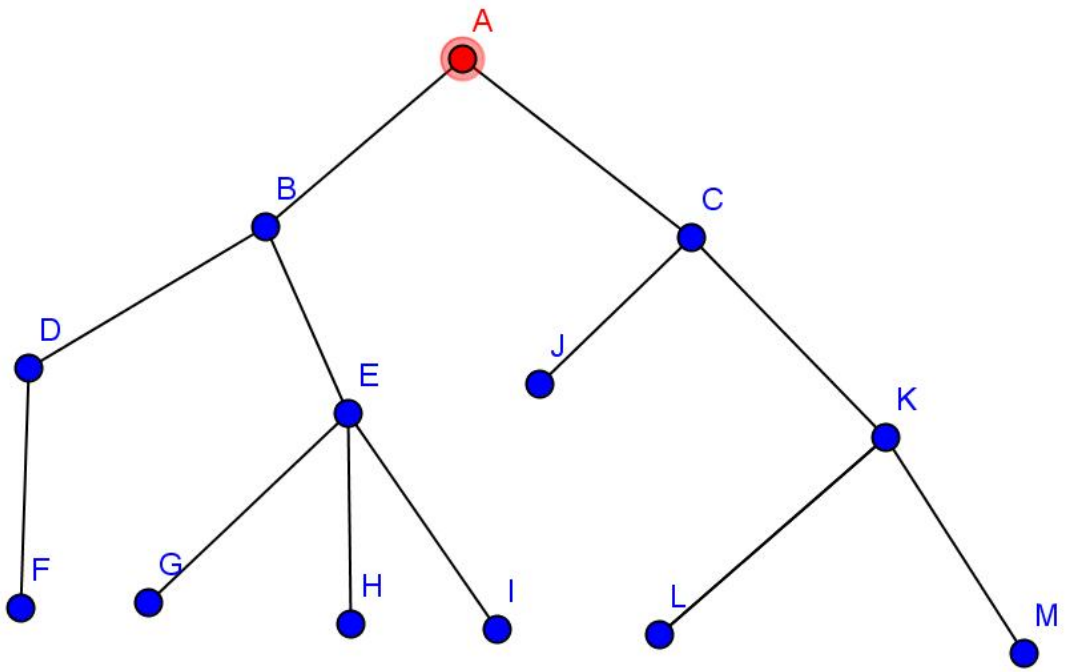
- I. Umieść v na liście $L(v)$.
- II. Dla każdego w - dziecka v (idąc od lewej do prawej) wykonaj: PREORDER(w) {otrzymujemy w ten sposób listę $L(w)$ złożoną z wierzchołka w i jego potomków}, a następnie dołącz otrzymaną listę $L(w)$ na końcu listy $L(v)$.
- **Rezultat**: $L(v)$ - lista wierzchołków drzewa, na której rodzice znajdują się zawsze przed swoimi dziećmi (czyli w porządku prefiksowym).

PREORDER - przykład



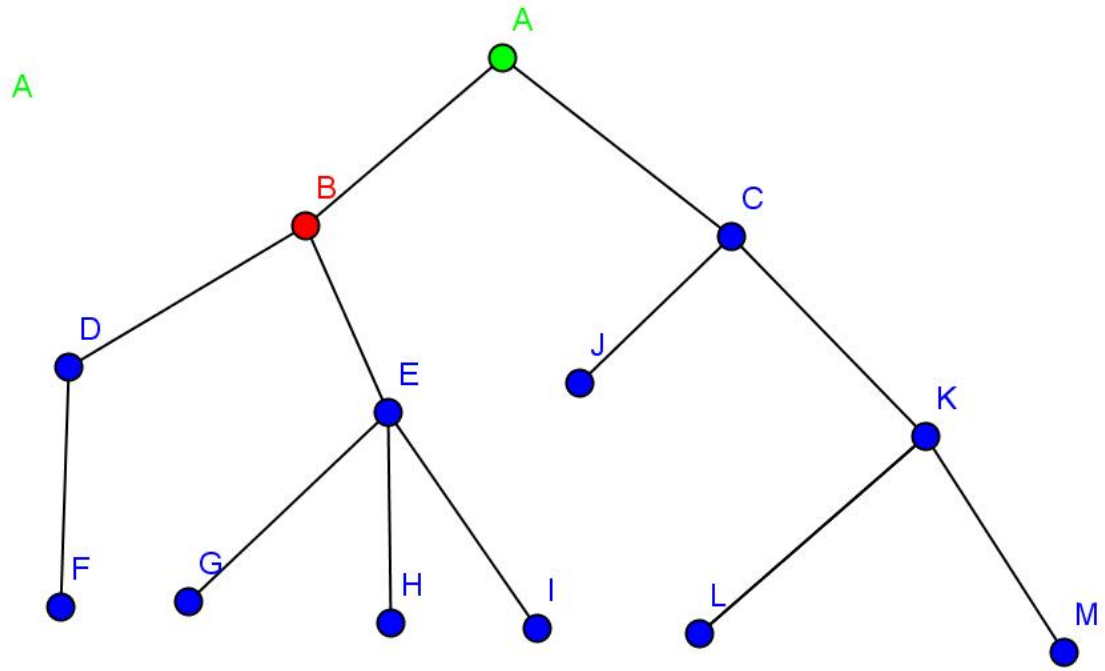
Spróbujemy algorytmem PREORDER prefiksowo uporządkować powyższe drzewo. Warto zauważyć, że komenda PREORDER(x) oznacza w przekładzie na ludzki: uporządkuj prefiksowo x i jego potomków.

PREORDER - przykład



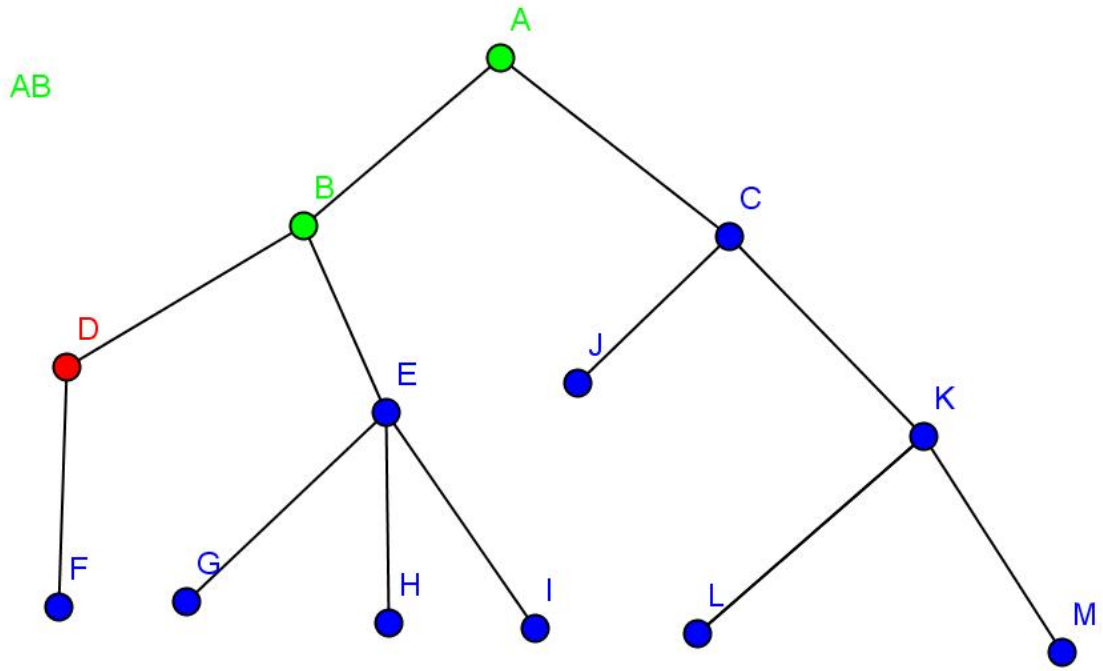
Zaczynamy od PREORDER(A), czyli umieszczamy A na początku (i jednocześnie końcu) listy (pojawi się w lewym górnym rogu) i wykonujemy PREORDER(B) (a jak skończymy, PREORDER(C)).

PREORDER - przykład



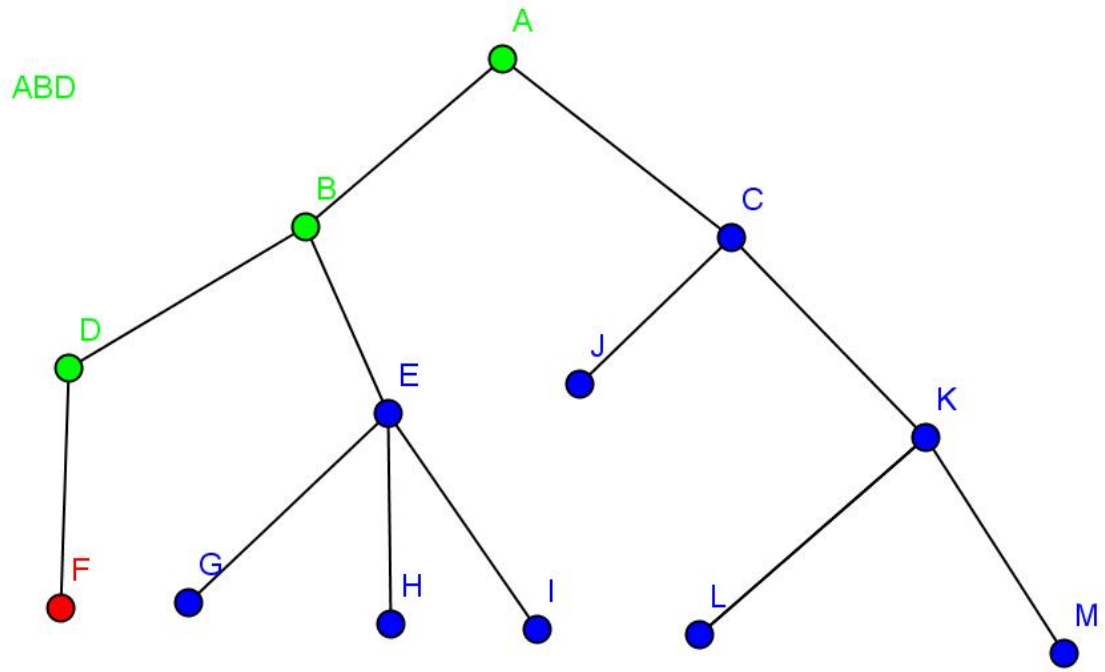
By wykonać PREORDER(B), umieszczamy B na końcu listy i wykonujemy PREORDER(D) (a jak skończymy, PREORDER(E)).

PREORDER - przykład



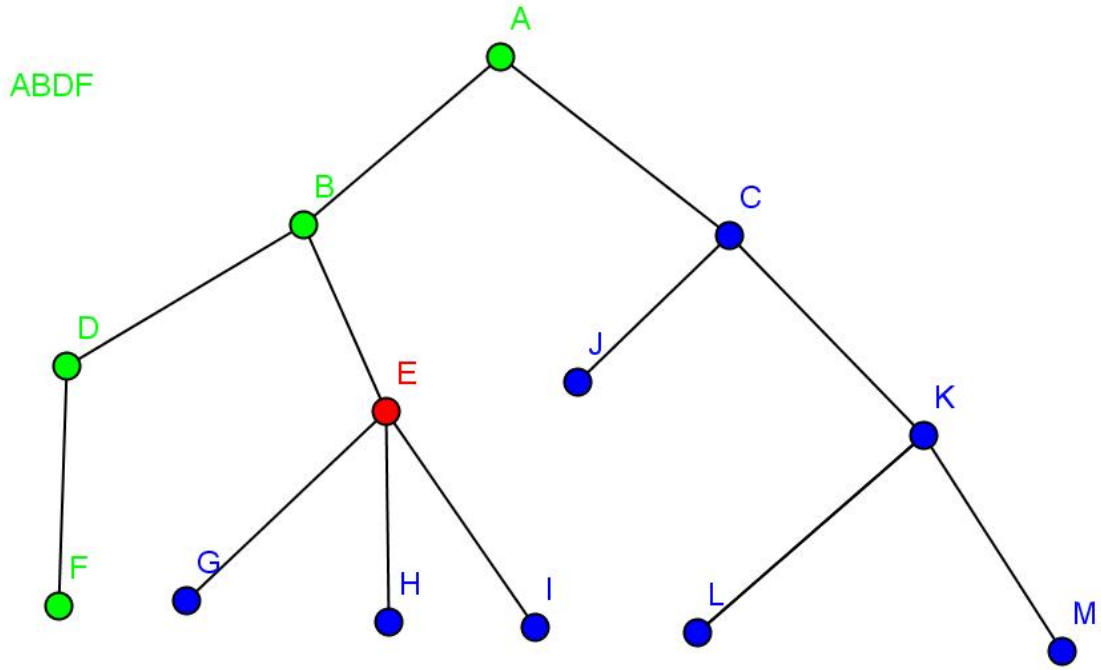
By wykonać PREORDER(D), umieszczamy D na końcu listy i wykonujemy PREORDER(F) (bo to jedyne dziecko D).

PREORDER - przykład



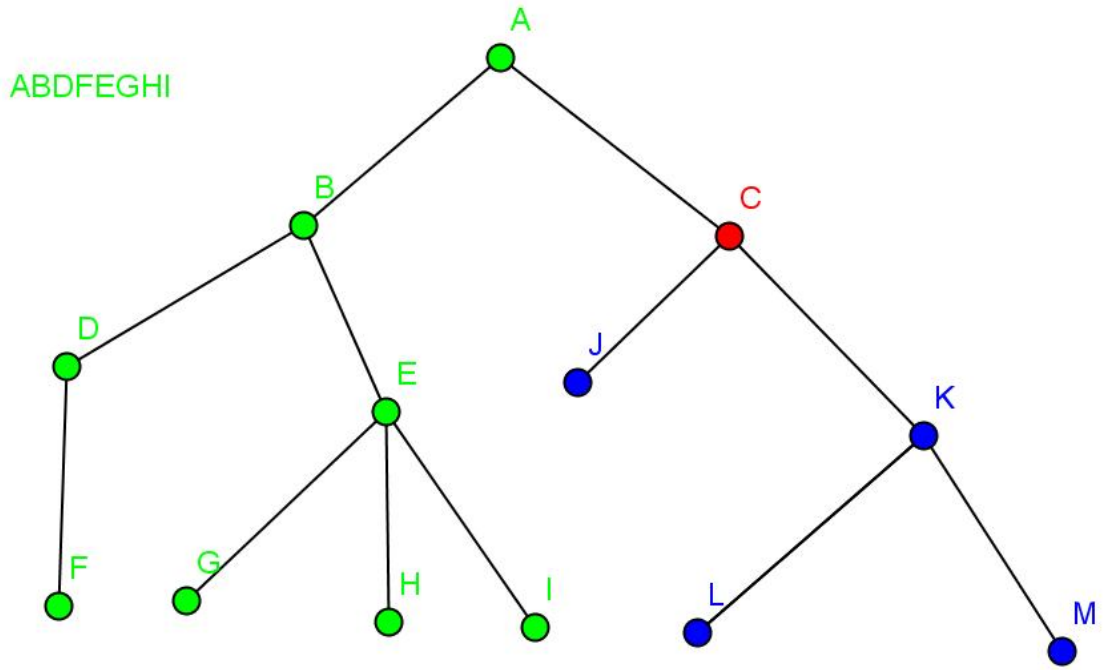
By wykonać PREORDER(F), umieszczamy F na końcu listy. Nie robimy nic więcej, bo F nie ma dzieci. Skoro tak, zakończyliśmy też PREORDER(D) i możemy przejść do PREORDER(E).

PREORDER - przykład



By wykonać PREORDER(E), umieszczamy E na końcu listy i wykonujemy kolejno PREORDER(G), PREORDER(H) i PREORDER (I). Jako, że G, H i I są liśćmi (nie mają dzieci), wykonanie tych trzech ostatnich procedur spowoduje jedynie dopisanie GHI na końcu listy. W ten sposób ukończyliśmy PREORDER (B) i przechodzimy do „zawieszonego" wykonania PREORDER (C).

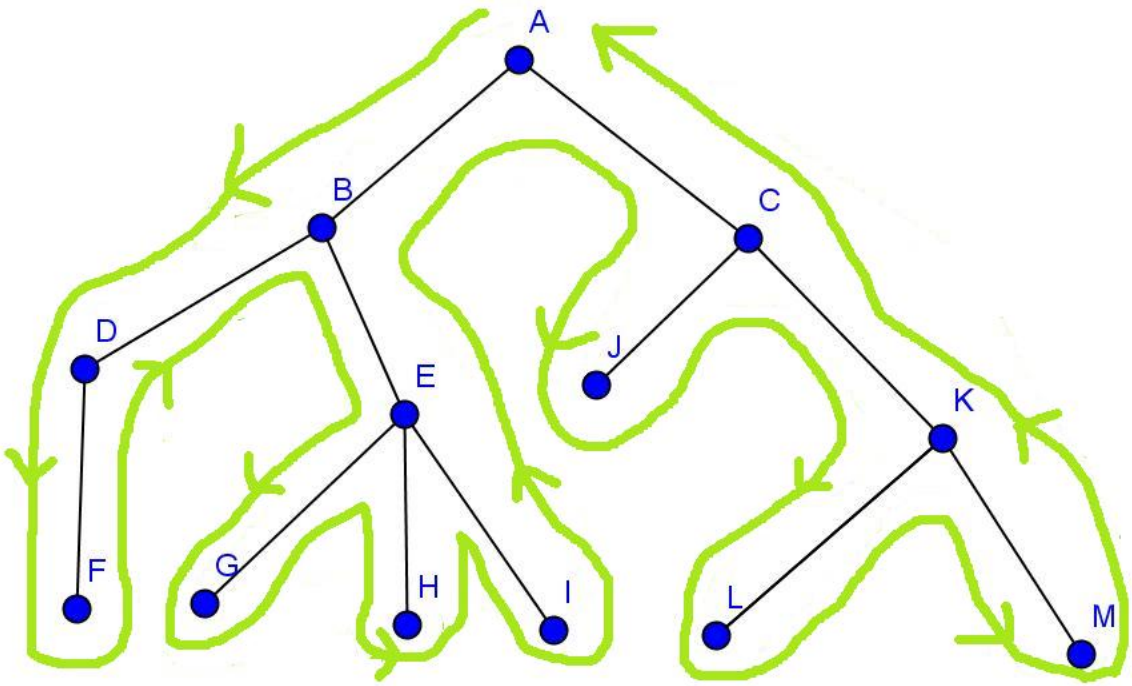
PREORDER - przykład



By wykonać PREORDER(C), umieszczamy C na końcu listy i wykonujemy kolejno PREORDER(J) (który spowoduje tylko dopisanie J na końcu listy) i PREORDER (K), który dopisze do listy kolejno wierzchołki K, L, M.

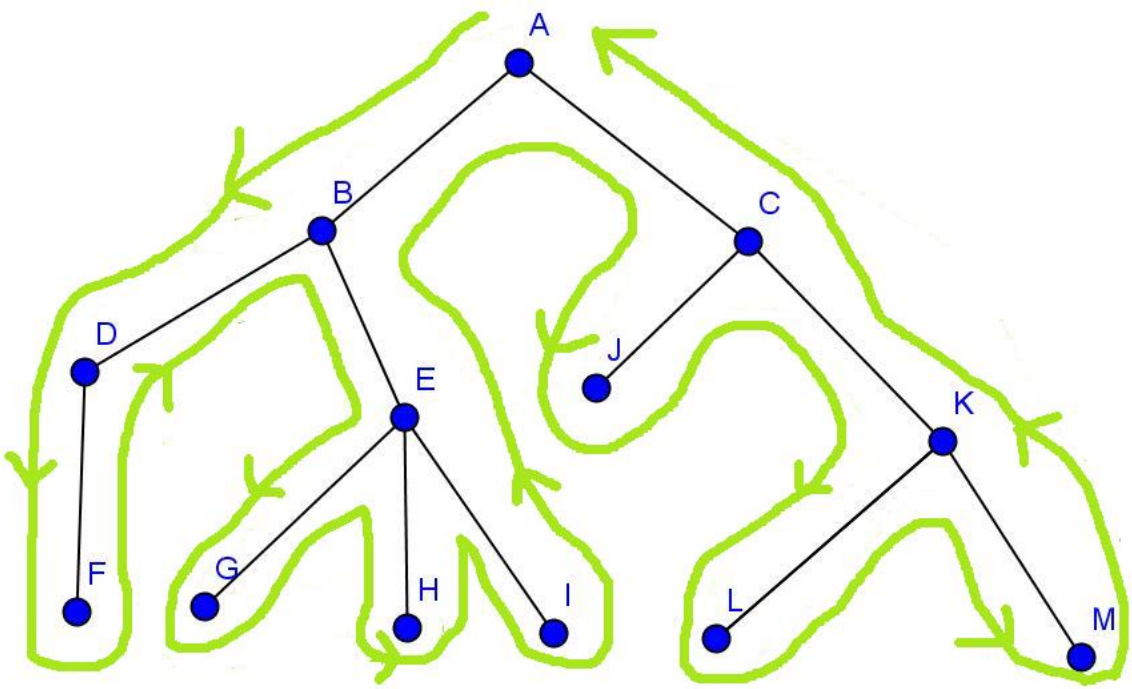
PREORDER - wynik i interpretacja graficzna

Algorytm PREORDER prowadzi nas zatem do następującego uporządkowania wierzchołków drzewa: *ABDFEGHICJKLM*. Ten ciąg łatwo można utworzyć „przechodząc" drzewo wzdłuż jego „obwodu" jak na rysunku poniżej. Startujemy od korzenia *A* i za każdym razem, kiedy przechodzimy koło wierzchołka, który jeszcze nie znalazł się na liście, dopisujemy ten wierzchołek na końcu listy.



PREORDER - wynik i interpretacja graficzna

Inna, bardziej zaawansowana interpretacja algorytmu PREORDER: jest to algorytm przeszukiwania drzewa w głąb, w którym zaczynamy od korzenia, w sytuacjach, gdy mamy wybór, zawsze idziemy „najbardziej w lewo jak się da” i każdy nowy wierzchołek, do którego dochodzimy zapisujemy na końcu dotychczas stworzonej listy.
ABDFEGHICJKLM



Porządek postfiksowy

Porządek postfiksowy

Porządek postfiksowy drzewa z wyróżnionym korzeniem to takie uporządkowanie wierzchołków, w którym poddrzewa są umieszczone najpierw (rysunkowo: kolejność od lewej do prawej), a potem dopiero korzeń. Innymi słowy, dzieci w tym porządku pojawiają się przed rodzicami.

Tworzy się go algorytmem POSTORDER.

Porządek postfiksowy - przykładowe zastosowania

Porządek postfiksowy używany jest w sytuacjach, gdy pewną operację musimy wykonać wpierw na dzieciach, by móc wykonać ją na rodzicach.

- Usuwanie drzew: zwłaszcza w językach bez „odśmiecania pamięci” (C, C++), kasując dane warto zadbać, by usunąć dzieci zanim usunie się rodziców (bo może potem nie być dostępu do dzieci i będą „zaśmiecać” pamięć).
- Zapisywanie w rodzicach informacji o tym, co się dzieje w poszczególnych poddrzewach ich dzieci.
- Częściej używany wariant notacji polskiej, tak zwana *odwrotna notacja polska*, w której zapisujemy najpierw obiekty, a potem działanie na nich wykonywane (na przykład $2 * (3 + 4)$ jest zapisywane jako $3\ 4\ +\ 2\ *$).

Algorytm POSTORDER

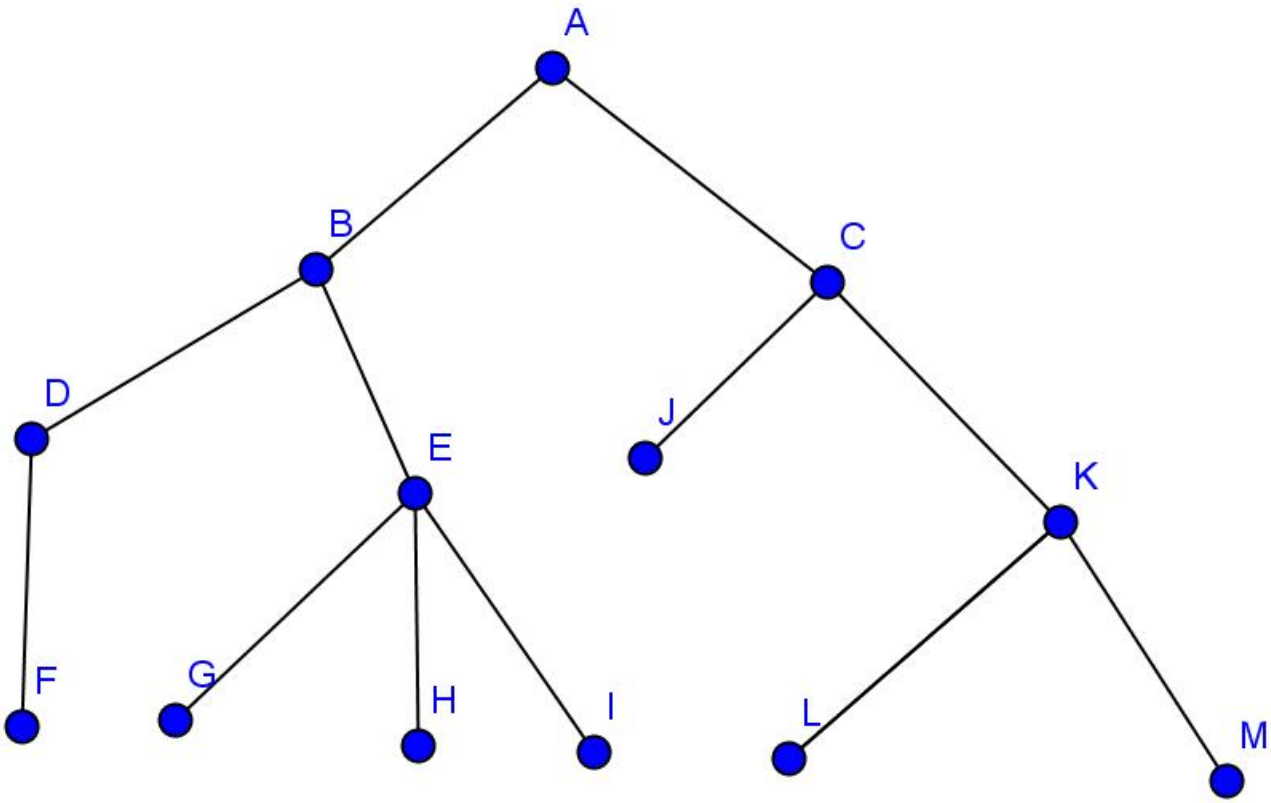
POSTORDER(v)

Dane: Skończone, uporządkowane drzewo z wyróżnionym korzeniem v.

Zmienne: $L(v)$ - lista wierzchołków drzewa, w-węzeł.

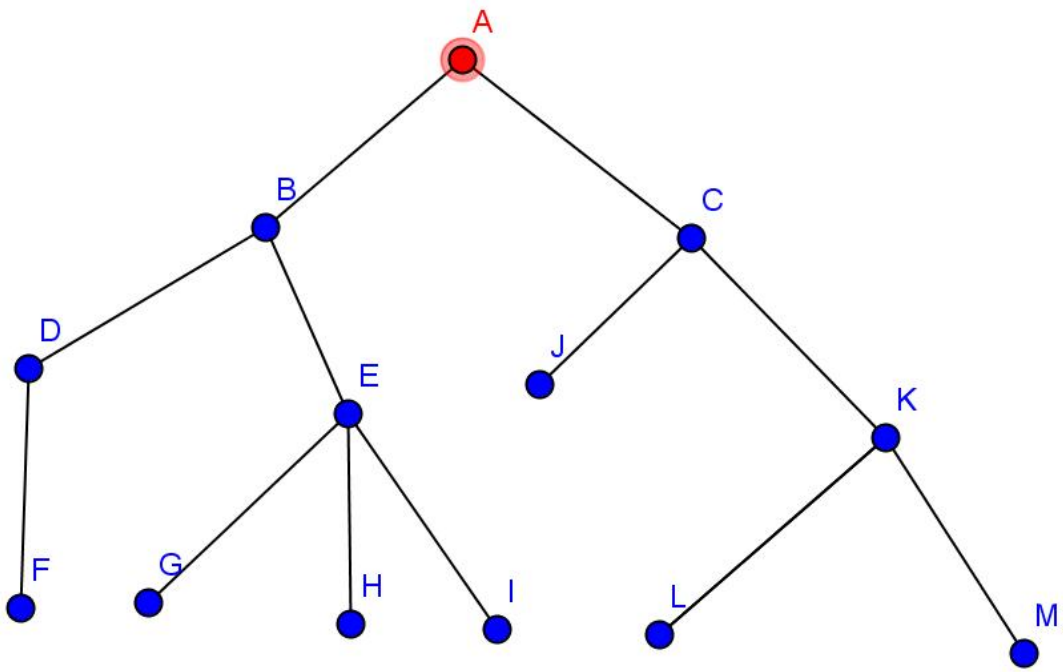
- I. Niech $L(v) = \emptyset$ (pusta lista).
- II. Dla każdego w - dziecka v (idąc od lewej do prawej) wykonaj: POSTORDER(w) {otrzymujemy w ten sposób listę $L(w)$ złożoną z wierzchołka w i jego potomków}, a następnie dołącz otrzymaną listę $L(w)$ na końcu listy $L(v)$.
- III. Dołącz wierzchołek v na końcu listy $L(v)$.
- **Rezultat**: $L(v)$ - lista wierzchołków drzewa, na której rodzice znajdują się zawsze po swoich dzieciach (czyli w porządku postfiksowym).

POSTORDER - przykład



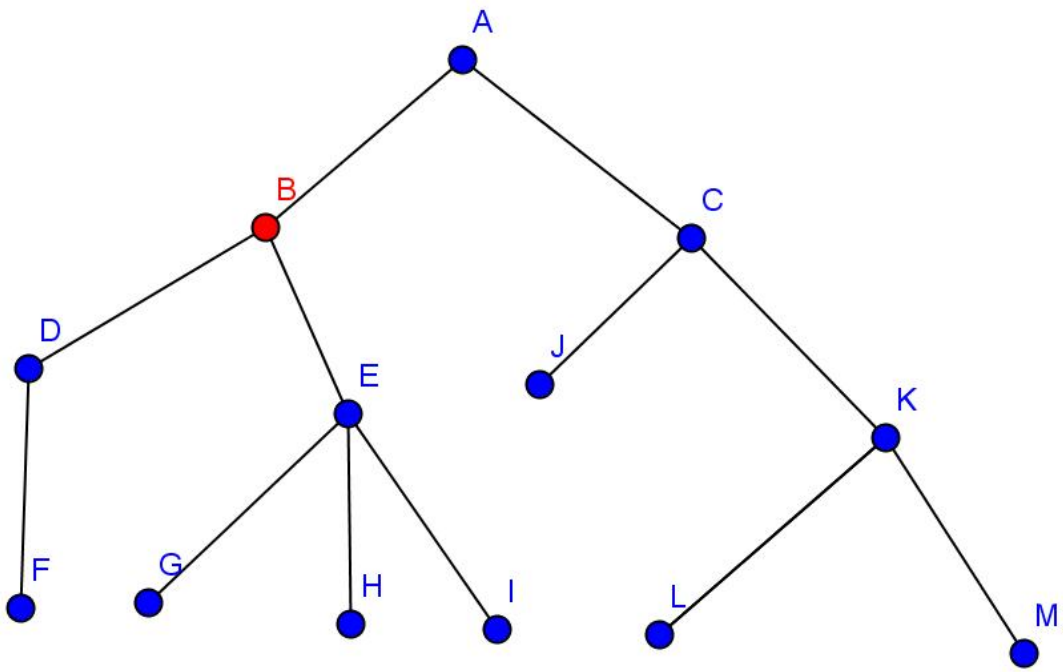
Spróbujemy algorytmem POSTORDER postfiksowo uporządkować powyższe drzewo. Warto zauważyć, że komenda POSTORDER(x) oznacza w przekładzie na ludzki: uporządkuj postfiksowo x i jego potomków.

POSTORDER - przykład



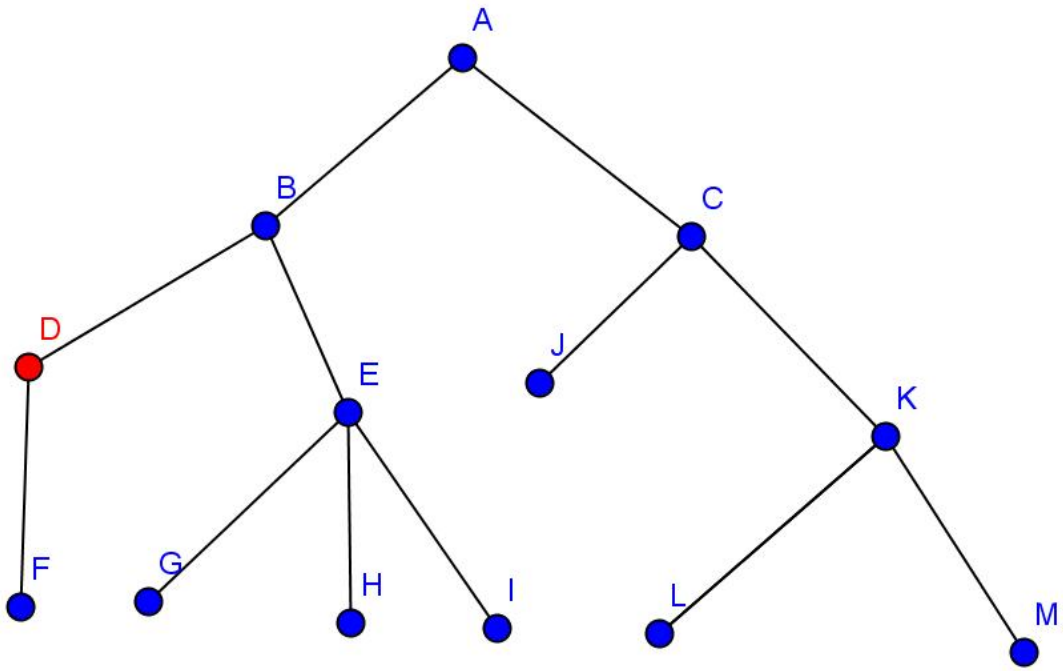
Zaczynamy od POSTORDER(A), co oznacza, że musimy wykonać POSTORDER(B), następnie POSTORDER (C) i dopiero potem dopisać A na końcu listy.

POSTORDER - przykład



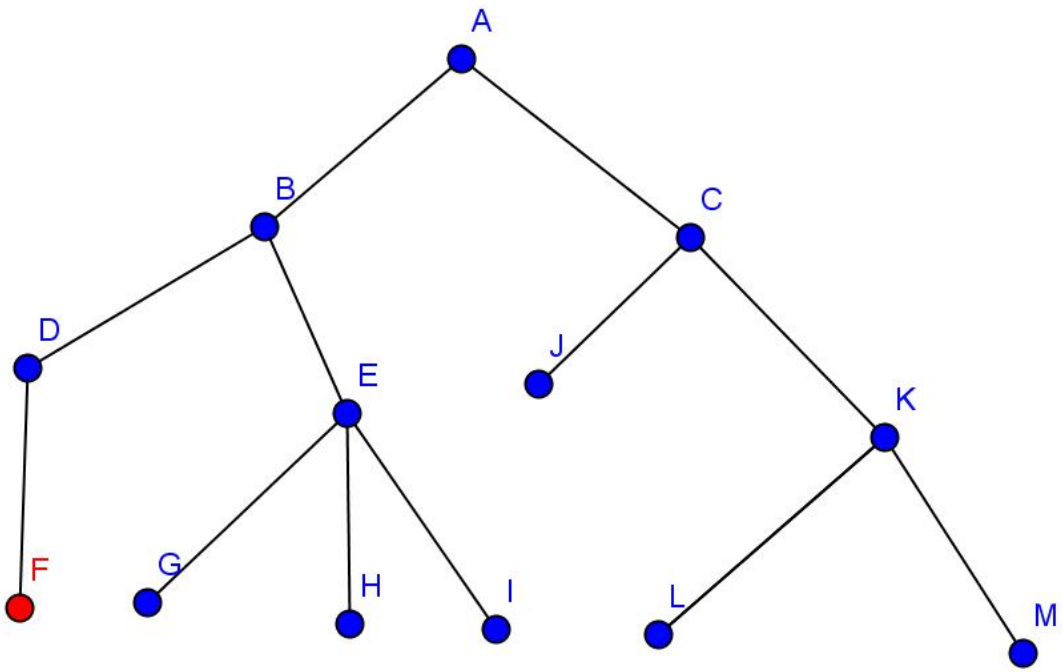
Wykonujemy POSTORDER(B), co oznacza, że musimy wykonać POSTORDER(D), następnie POSTORDER (E) i dopiero potem dopisać B na końcu listy.

POSTORDER - przykład



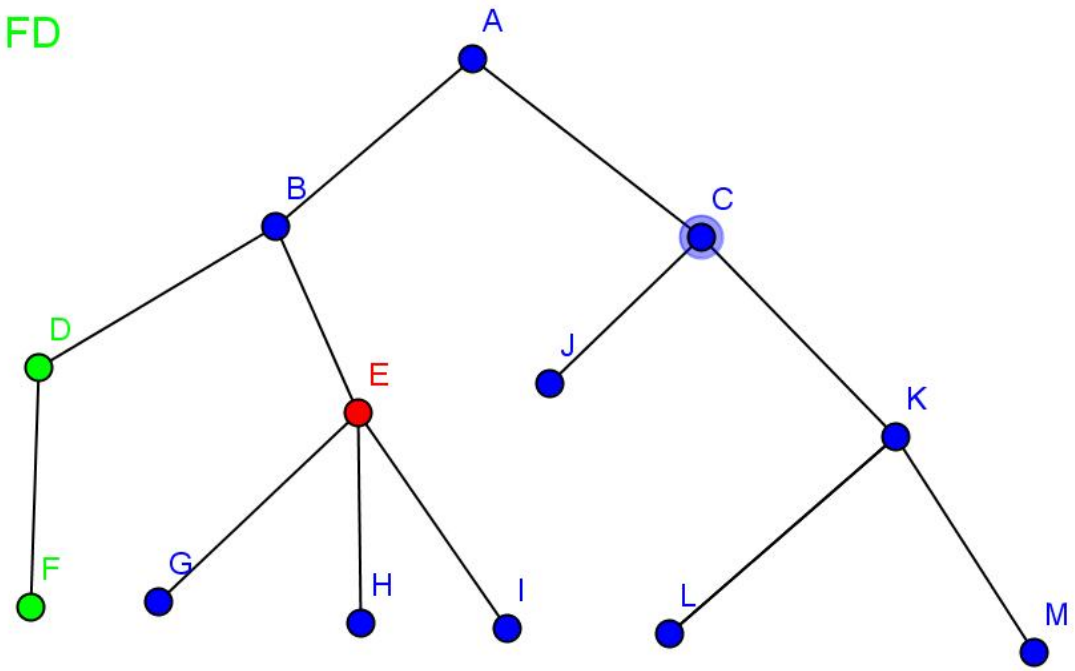
Wykonujemy POSTORDER(D), co oznacza, że musimy wykonać POSTORDER(F) i dopiero potem dopisać D na końcu listy.

POSTORDER - przykład



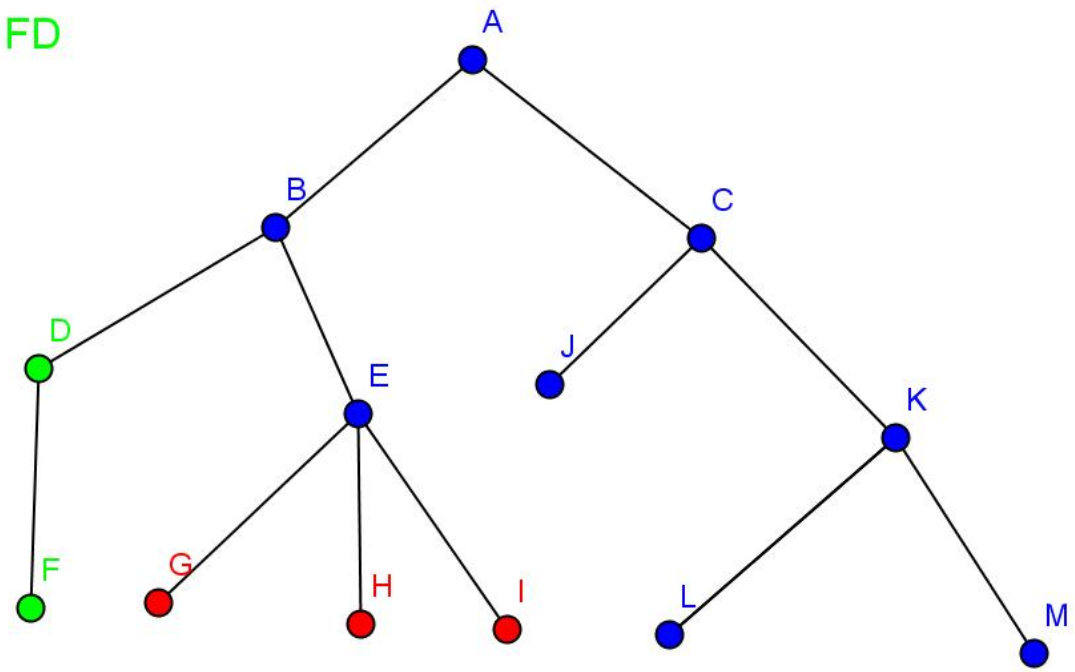
Wykonujemy POSTORDER(F), co oznacza dopisanie rozpoczęcie listy od F. Skoro POSTORDER(F) został wykonany, możemy dokończyć POSTORDER(D), czyli dopisać D na końcu listy (od tej pory lista pojawia się z lewej strony drzewa).

POSTORDER - przykład



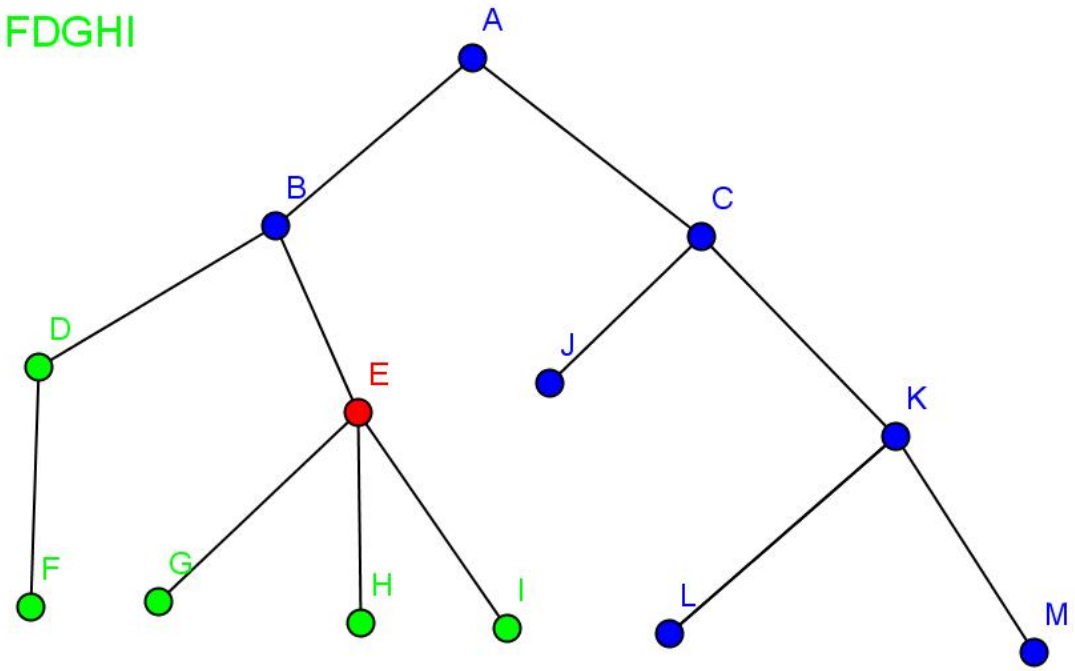
Skoro POSTORDER(D) został wykonany, przechodzimy do kolejnego kroku wykonania POSTORDER(B), czyli POSTORDER(E). Wykonujemy POSTORDER(E), co oznacza, że musimy wykonać POSTORDER(G), następnie POSTORDER (H), POSTORDER(I) i dopiero potem dopisać E na końcu listy.

POSTORDER - przykład



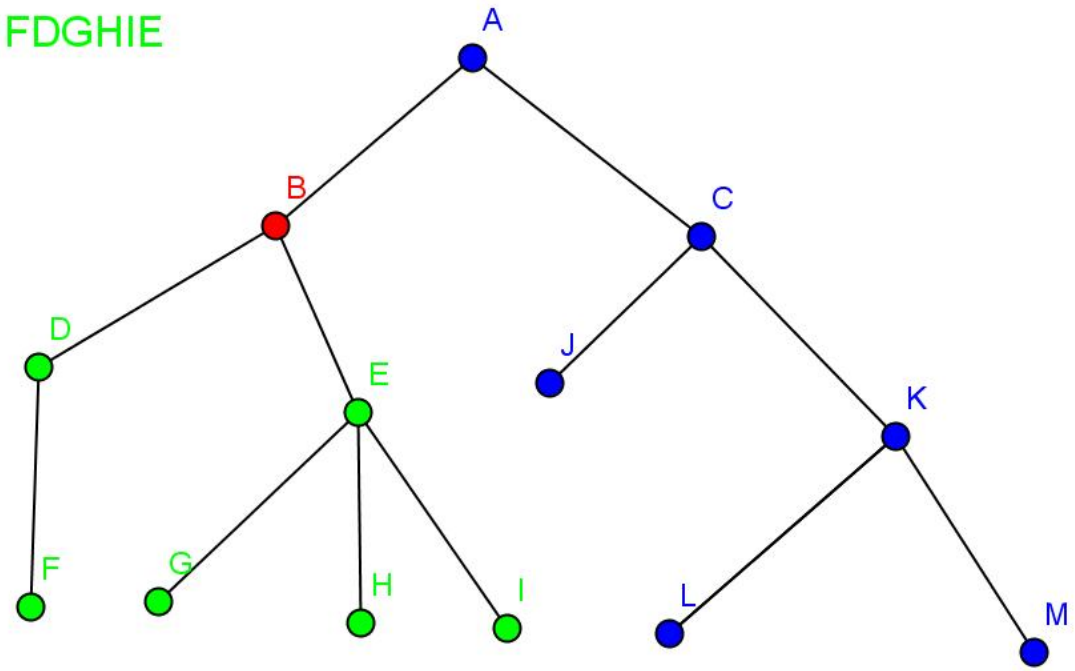
G, H, I nie mają dzieci, więc wykonanie POSTORDER(G), POSTORDER (H) i POSTORDER(I) sprowadza się do dopisania G, H, I na końcu listy.

POSTORDER - przykład



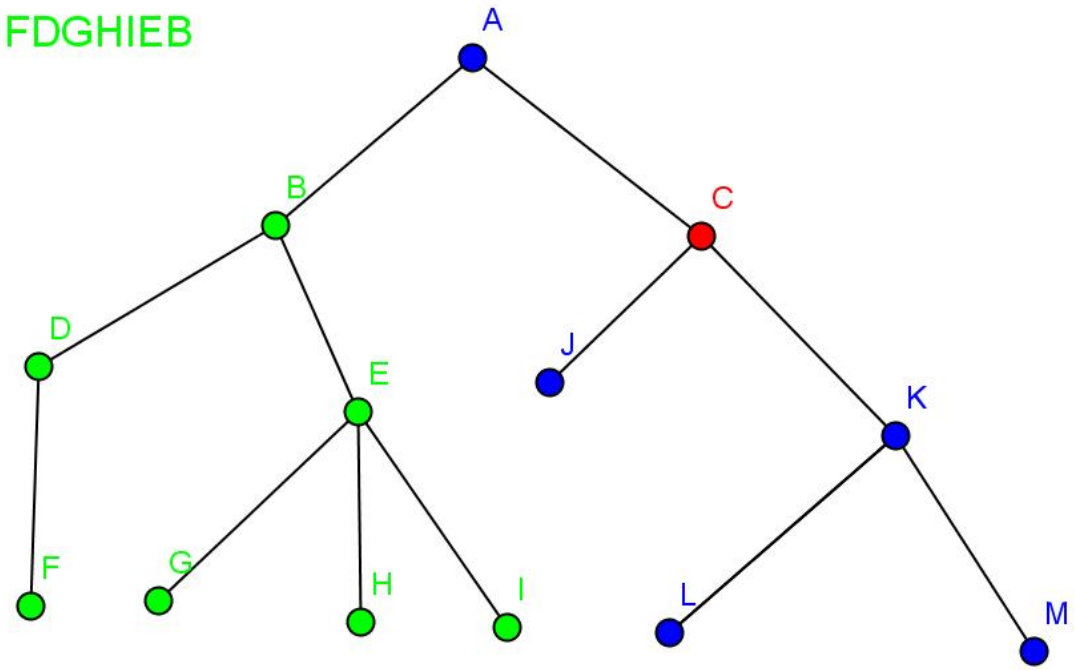
Skoro rozpatrzyliśmy już algorytm POSTORDER dla wszystkich dzieci E, kończymy POSTORDER(E) dopisując E na końcu listy.

POSTORDER - przykład



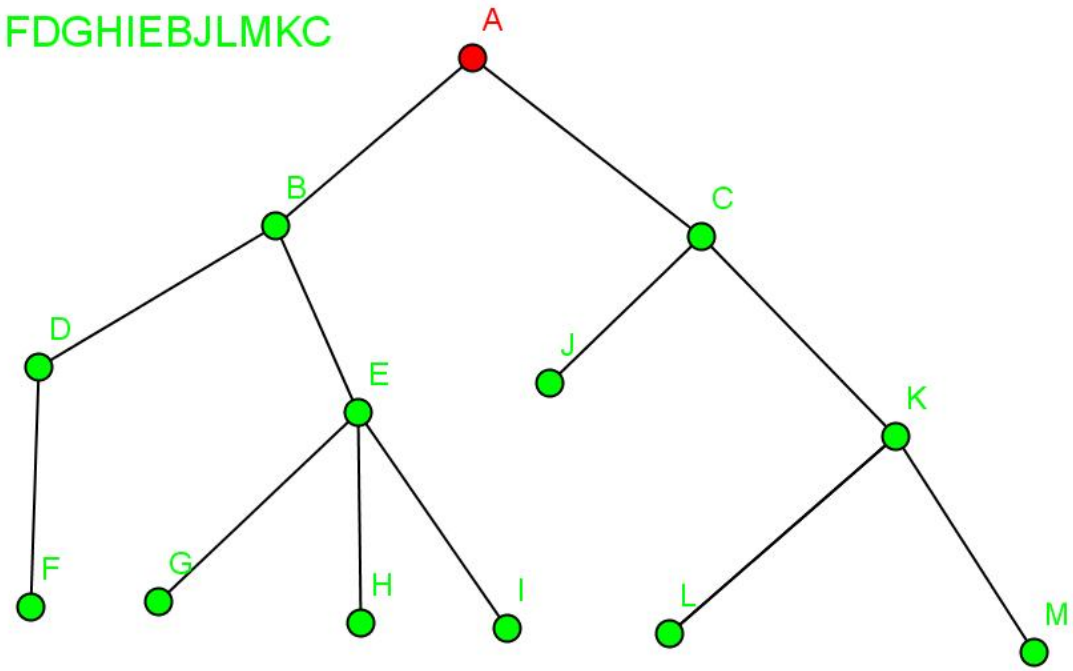
Skoro rozpatrzyliśmy już algorytm POSTORDER dla wszystkich dzieci B, kończymy POSTORDER(B) dopisując B na końcu listy.

POSTORDER - przykład



By kontynuować POSTORDER(A), musimy teraz wykonać POSTORDER (C). Nie będziemy już tego poddrzewa analizować szczegółowo: analogicznie do POSTORDER(B) do końca listy dopiszemy: $JLMKC$.

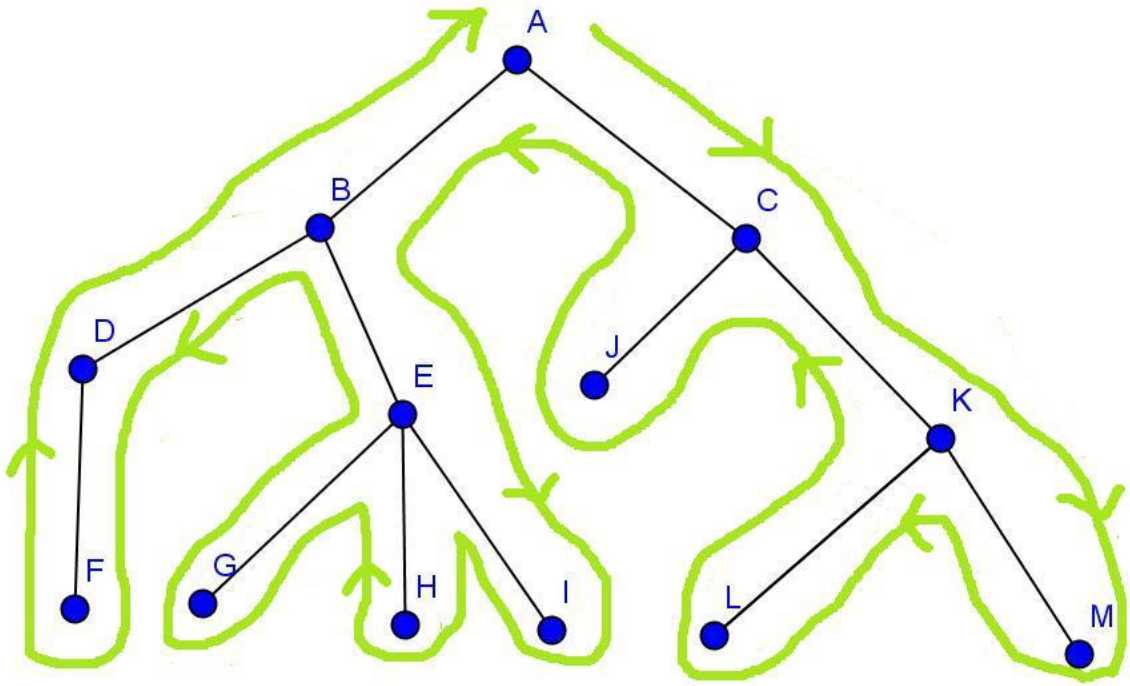
POSTORDER - przykład



Skoro rozpatrzyliśmy już algorytm POSTORDER dla wszystkich dzieci A, kończymy POSTORDER(A) dopisując A na końcu listy.

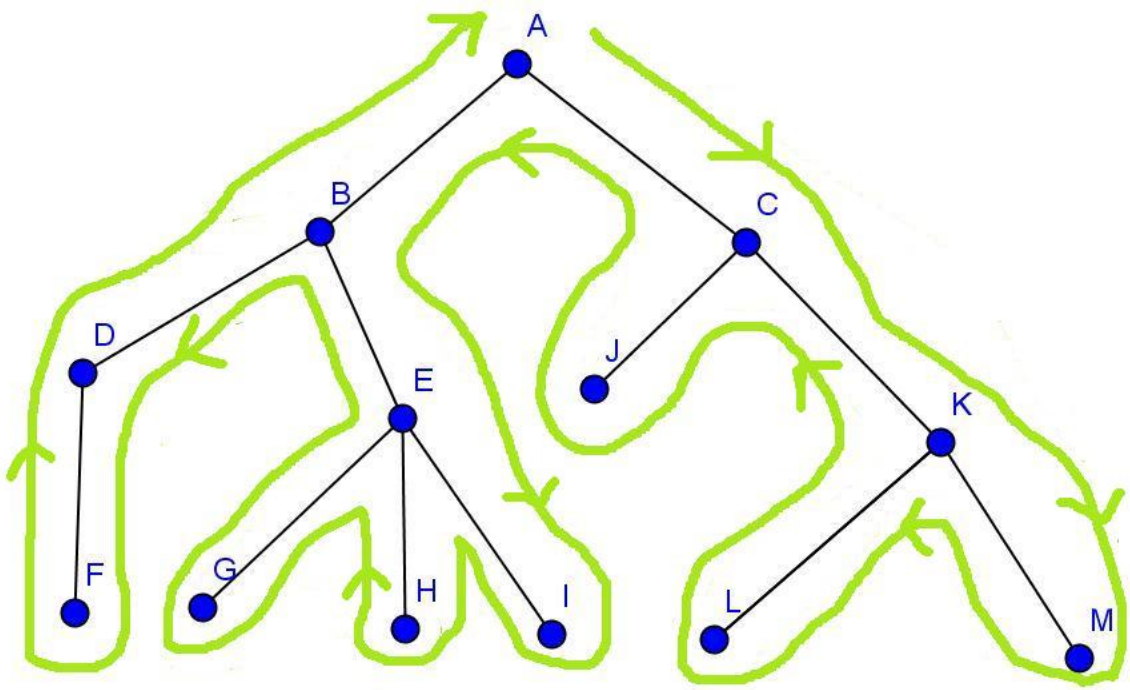
POSTORDER - wynik i interpretacja graficzna

Algorytm POSTORDER prowadzi nas zatem do następującego uporządkowania wierzchołków drzewa: *FDGHIEBJLMKCA*. Ten ciąg łatwo można utworzyć „przechodząc" drzewo wzdłuż jego „obwodu" jak na rysunku poniżej. Startujemy od korzenia *A* i za każdym razem, kiedy przechodzimy koło wierzchołka, który jeszcze nie znalazł się na liście, dopisujemy ten wierzchołek NA POCZĄTKU listy.



POSTORDER - wynik i interpretacja graficzna

Inna, bardziej zaawansowana interpretacja algorytmu POSTORDER: jest to algorytm przeszukiwania drzewa w głąb, w którym zaczynamy od korzenia, w sytuacjach, gdy mamy wybór, zawsze idziemy „najbardziej w prawo jak się da” i każdy nowy wierzchołek, do którego dochodzimy zapisujemy na początku dotychczas stworzonej listy.
FDGHIEBJLMKCA



Porządek infiksowy

Trzeci sposób przechodzenia drzewa dotyczy tylko uporządkowanych drzew binarnych z wyróżnionym korzeniem.

Porządek infiksowy

Porządek infiksowy drzewa binarnego z wyróżnionym korzeniem to takie uporządkowanie wierzchołków, w którym dany korzeń jest umieszczony pomiędzy wierzchołkami poddrzewa, którego korzeniem jest jego lewe dziecko, a wierzchołkami poddrzewa, którego korzeniem jest jego prawe dziecko. Innymi słowy, lewe dzieci w tym porządku pojawiają się przed rodzicami, a prawe po nich.

Tworzy się go algorytmem INORDER.

Porządek infiksowy - przykładowe zastosowania

Porządek postfiksowy używany jest zwłaszcza dla drzew poszukiwań binarnych, gdy ważna jest dla nas kolejność etykiet.

- Odtwarzanie wyjściowej listy etykiet z drzewa (w kolejności)
- Znajdowanie elementu listy o zadanym numerze (np. piątego wpisu w bazie danych)
- Klasyczna notacja logiczno-algebraiczna: działanie pomiędzy obiektami, na których je wykonujemy (wada - niejednoznaczność bez nawiasów)

Algorytm INORDER - część 1

INORDER(v)

Dane: Skończone, uporządkowane drzewo binarne z wyróżnionym korzeniem v.

Zmienne: $L(v)$ - lista wierzchołków drzewa, u, w-węzły.

- I. Niech $L(v) = \emptyset$ (pusta lista).
- II. Jeśli wierzchołek v ma lewe dziecko w, wykonaj: INORDER(w) {otrzymujemy w ten sposób listę $L(w)$ złożoną z wierzchołka w i jego potomków}, a następnie dołącz otrzymaną listę $L(w)$ na końcu listy $L(v)$.
- III. Dołącz v na koniec listy $L(v)$.

Algorytm INORDER - część 2

INORDER(v)

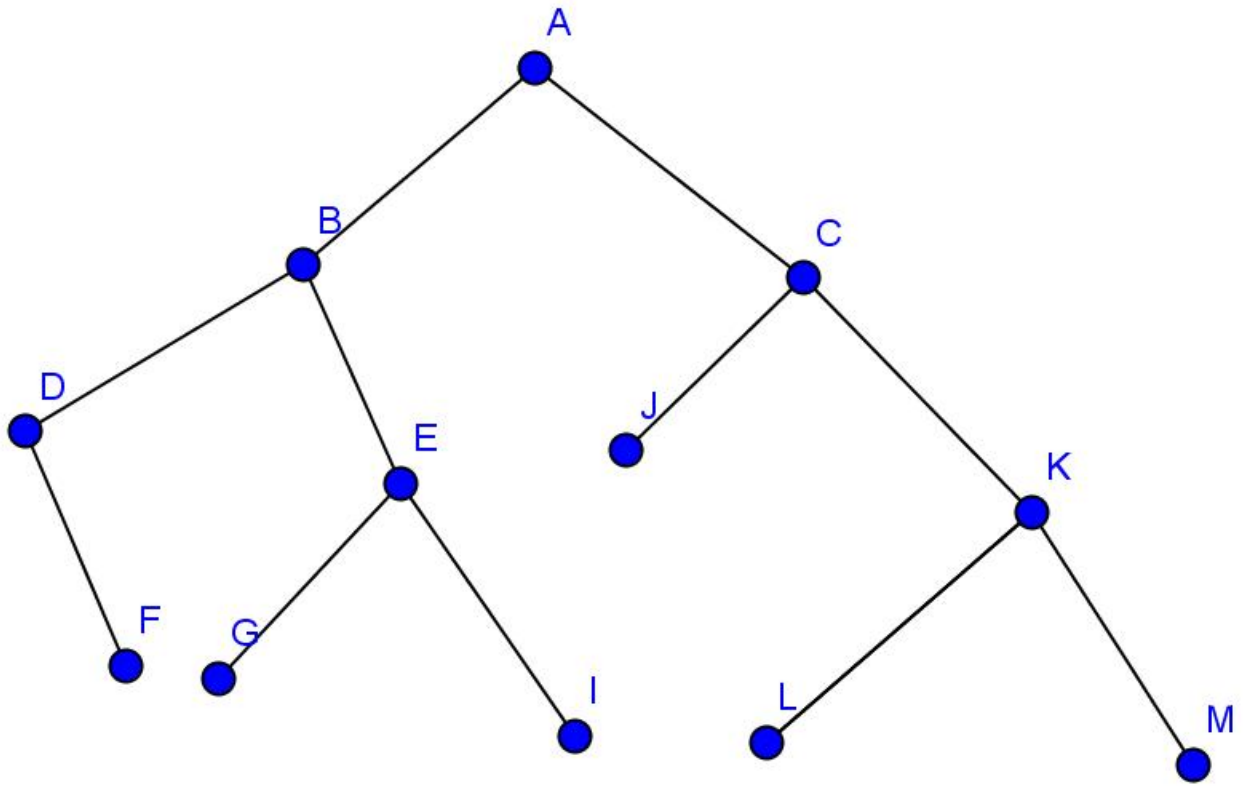
Dane: Skończone, uporządkowane drzewo binarne z wyróżnionym korzeniem v.

Zmienne: $L(v)$ - lista wierzchołków drzewa, u, w-węzły.

- IV. Jeśli wierzchołek v ma prawe dziecko u, wykonaj: INORDER(u) {otrzymujemy w ten sposób listę $L(u)$ złożoną z wierzchołka u i jego potomków}, a następnie dołącz otrzymaną listę $L(u)$ na końcu listy $L(v)$
- **Rezultat**: $L(v)$ - lista wierzchołków drzewa, na której lewe dzieci znajdują się przed rodzicami, a prawe - za nimi (czyli w porządku infiksowym).

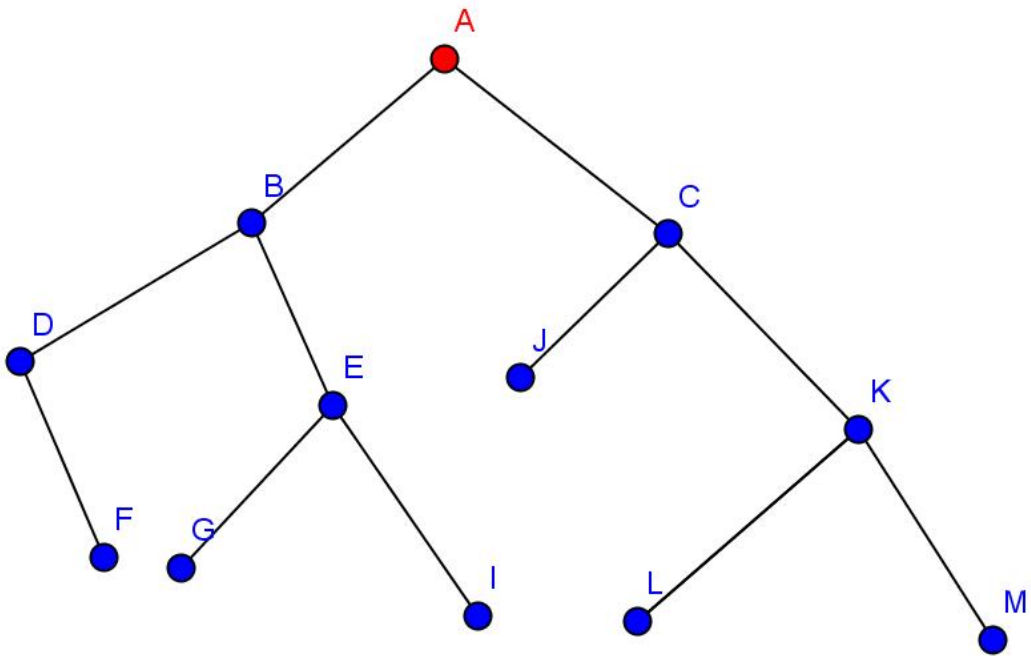
Warto zwrócić uwagę, że algorytm działa również jeśli niektórzy rodzice mają tylko jedno dziecko.

INORDER - przykład



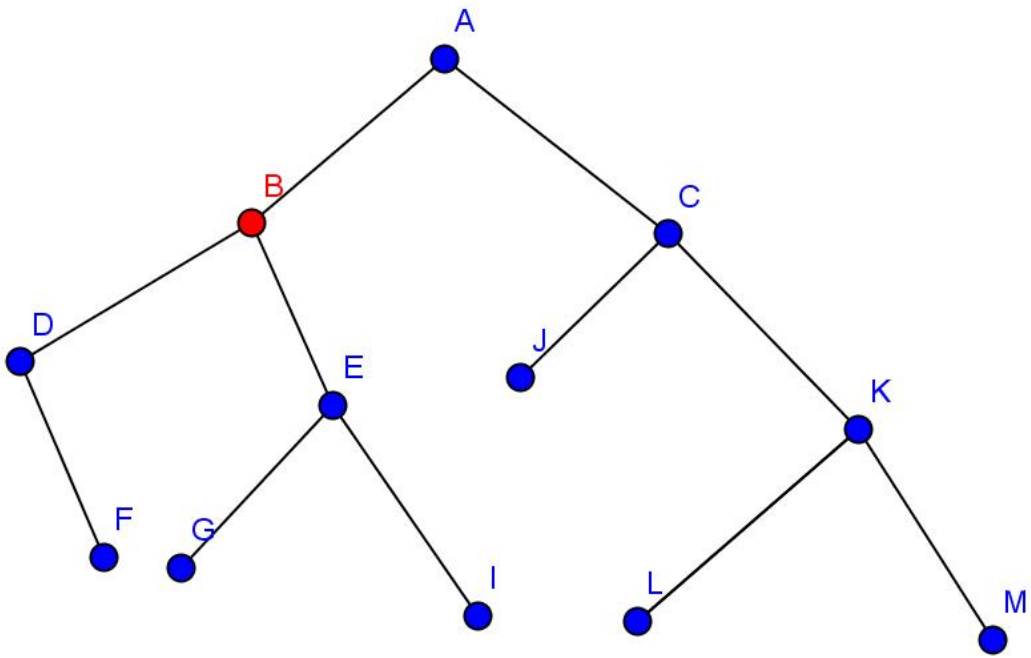
Spróbujemy algorytmem INORDER infiksowo uporządkować powyższe drzewo. Warto zauważyć, że komenda INORDER(x) oznacza w przekładzie na ludzki: uporządkuj infiksowo x i jego potomków.

INORDER - przykład



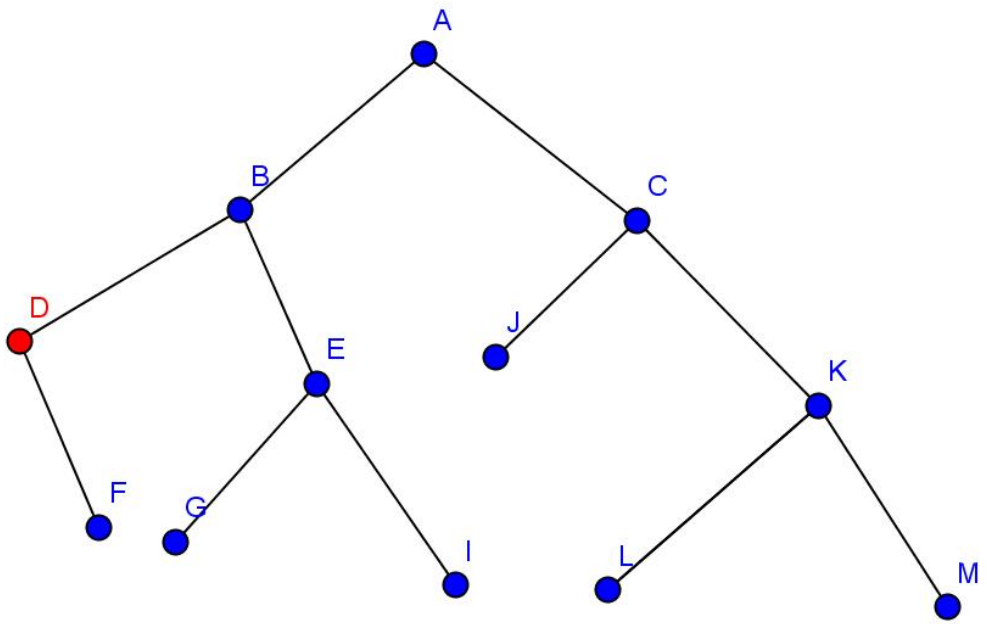
Zaczynamy od INORDER(A), co oznacza, że musimy wykonać INORDER(B), następnie dopisać A na końcu listy i dopiero potem INORDER (C).

INORDER - przykład



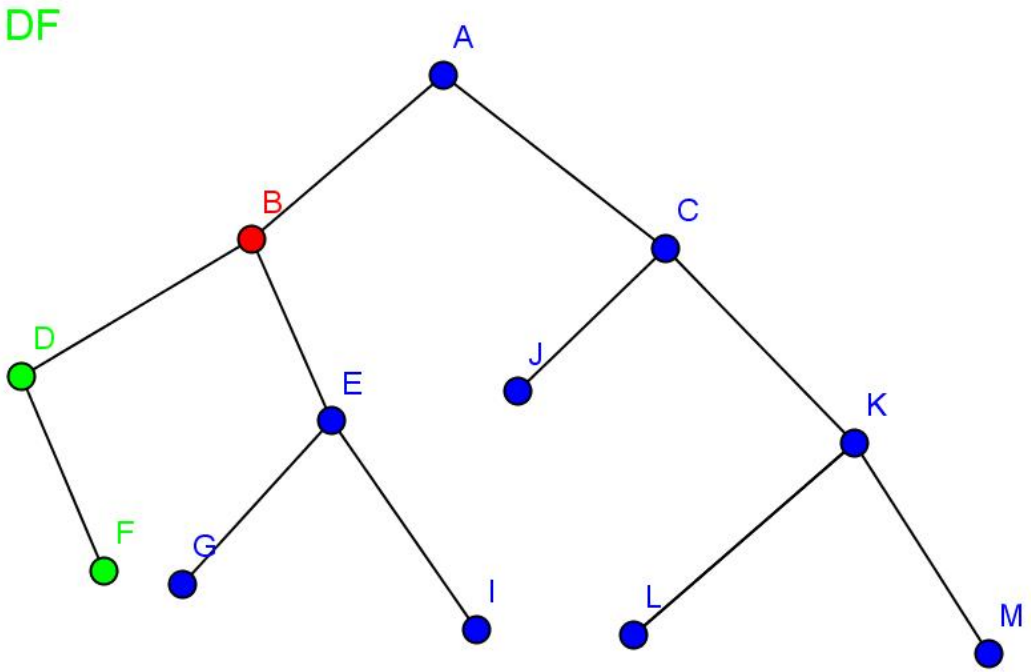
Wykonujemy INORDER(B), co oznacza, że mmusimy wykonać INORDER(D), następnie dopisać B na końcu listy i dopiero potem INORDER (E).

INORDER - przykład



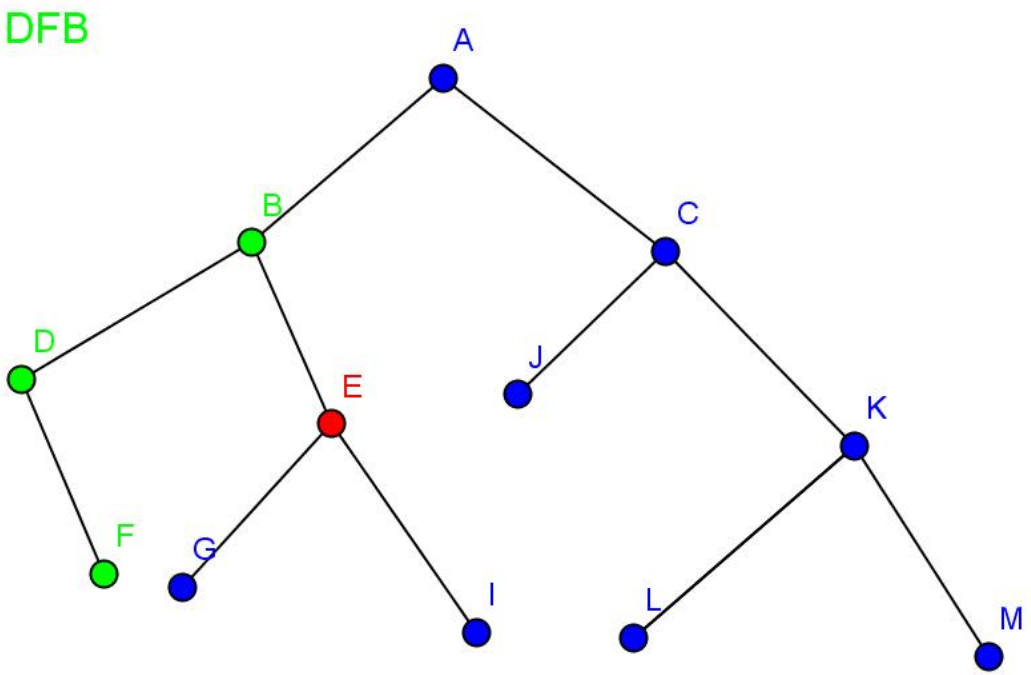
Wykonujemy INORDER(D). D nie ma lewych dzieci, więc najpierw dopisujemy D na końcu listy, a potem wykonujemy INORDER(F), co polega na dopisaniu F na końcu listy, bo F nie ma dzieci.

INORDER - przykład



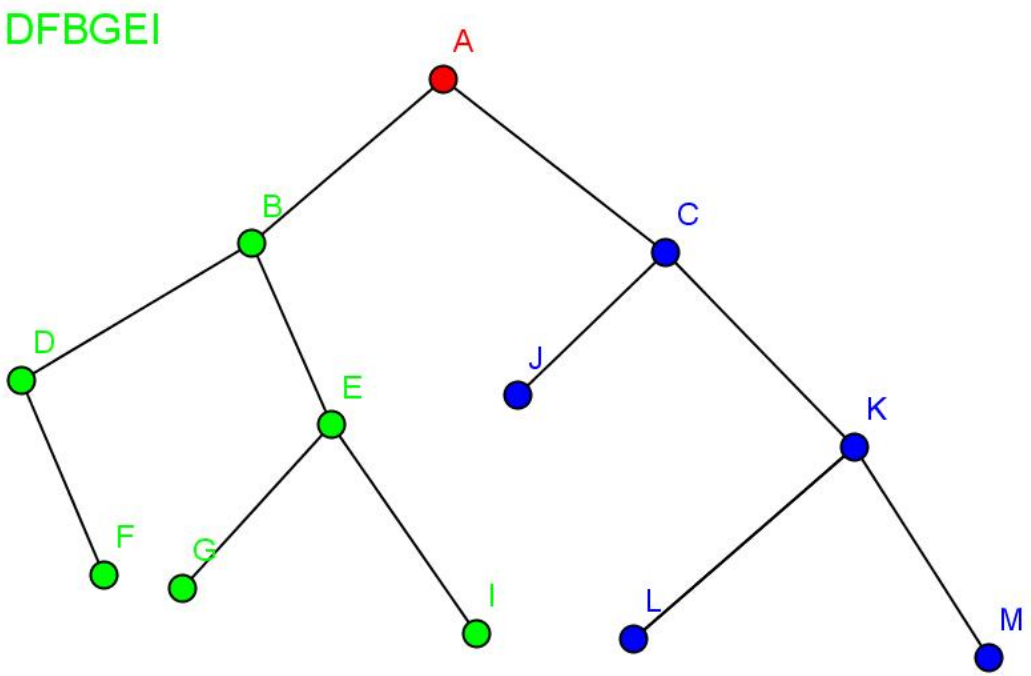
Kontynuujemy INORDER(B). Skoro wykonaliśmy już INORDER(D), to teraz czas dopisać B na koniec listy i wykonać INORDER(E).

INORDER - przykład



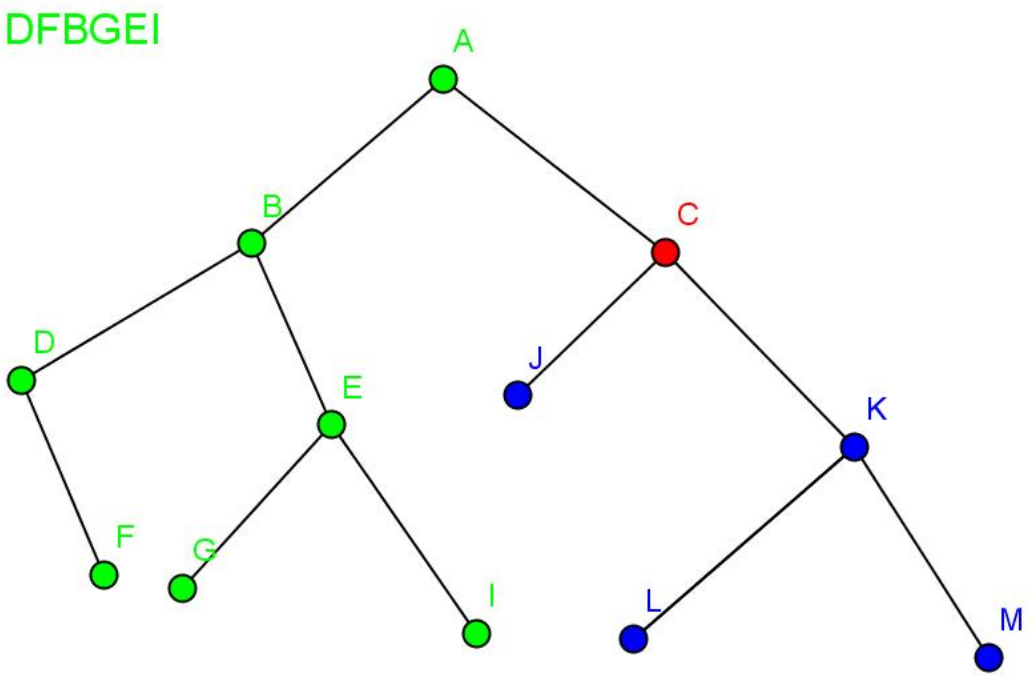
Wykoujemy INORDER(E), co oznacza, że mmusimy wykonać INORDER(G), następnie dopisać E na końcu listy i dopiero potem INORDER (I). Jako, że G i I nie mają dzieci, wykonanie INORDER(G) i INORDER(I) sprowadza się do dopisania odpowiednio G i I na końcu listy. Zatem INORDER(E) oznacza dopisanie na końcu listy: GEI.

INORDER - przykład



W ten sposób ukończyliśmy INORDER(B). Kolejny krok procedury INORDER(A) wymaga dopisania A na koniec listy i przejścia do wykonania INORDER(C), co zakończy cały algorytm.

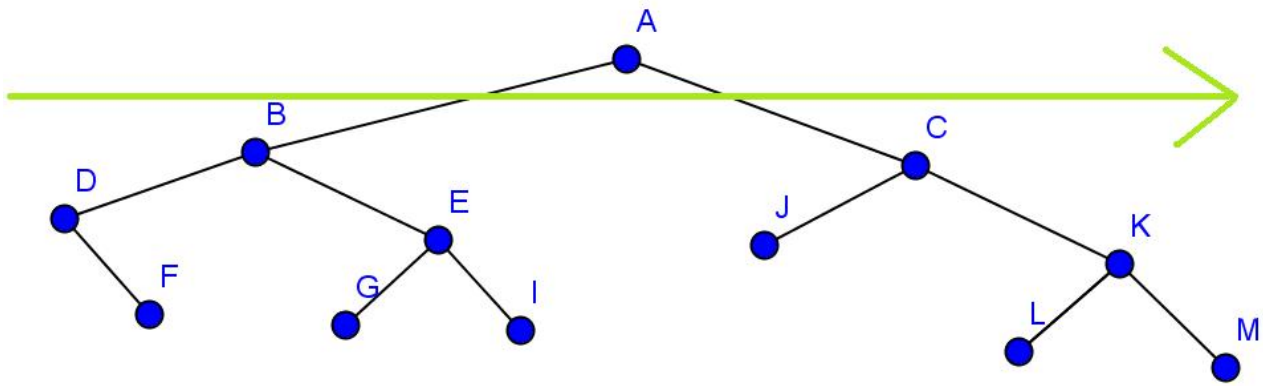
INORDER - przykład



Wykonujemy INORDER(C). Nie będziemy analizować tego szczegółowo. Analogicznie do poprzednich kroków, wynikiem INORDER(C) jest dopisanie na koniec listy: *JCLKM*.

INORDER - wynik i interpretacja graficzna

Algorytm INORDER prowadzi nas zatem do następującego uporządkowania wierzchołków drzewa: *DFBGEIAJCLKM*. Ten ciąg można utworzyć przerysowując drzewo tak, by każdy kolejny poziom drzewa miał coraz krótsze krawędzie, by lewi potomkowie zawsze byli na lewo od przodka, a prawi potomkowie na prawo. Następnie wystarczy odczytać wierzchołki od lewej do prawej, ignorując ich „wysokość".



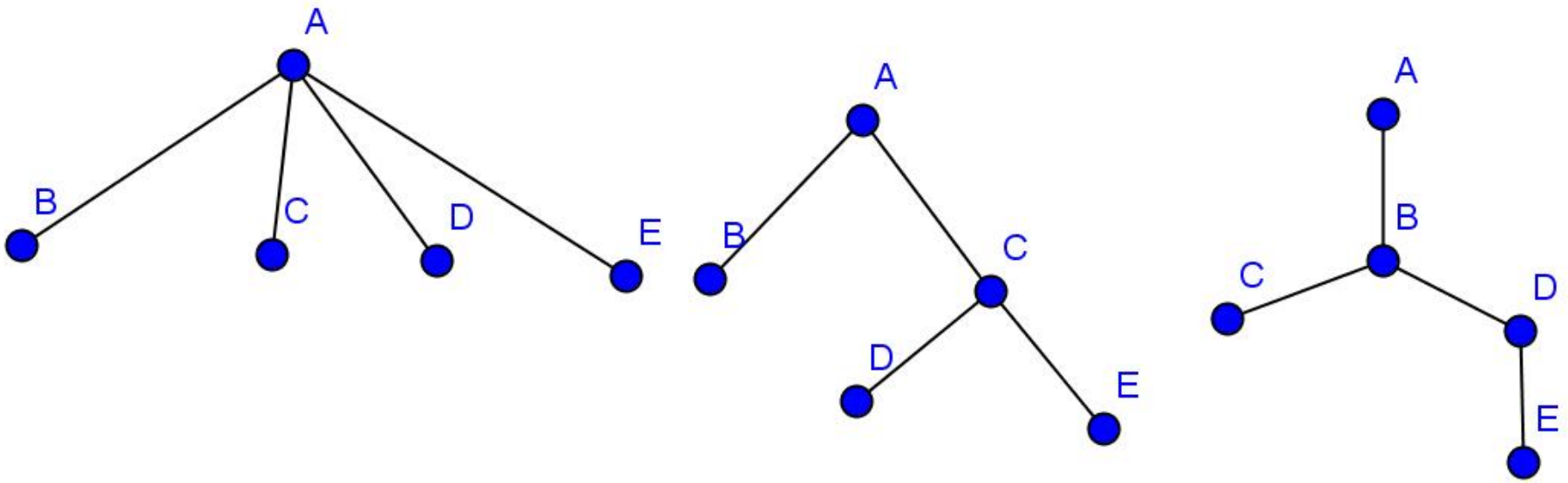
Algorytmy przechodzenia drzewa - uwagi

- Czas działania wszystkich trzech algorytmów przechodzenia drzewa to $O(|V|)$, gdzie V jest zbiorem wierzchołków drzewa.
- Istnieją inne sposoby przechodzenia drzew, na przykład „level-order", który wypisuje wierzchołki w kolejności rosnącej ich poziomów. Nie analizowałem szczegółowo tej metody, gdyż jest to po prostu przechodzenie drzewa wszerz (zaczynając od korzenia). Może być przydatne w przeszukiwaniu „płytkich" drzew: o wielu wierzchołkach lecz niewielkiej wysokości.

Dekodowanie drzew - wstęp

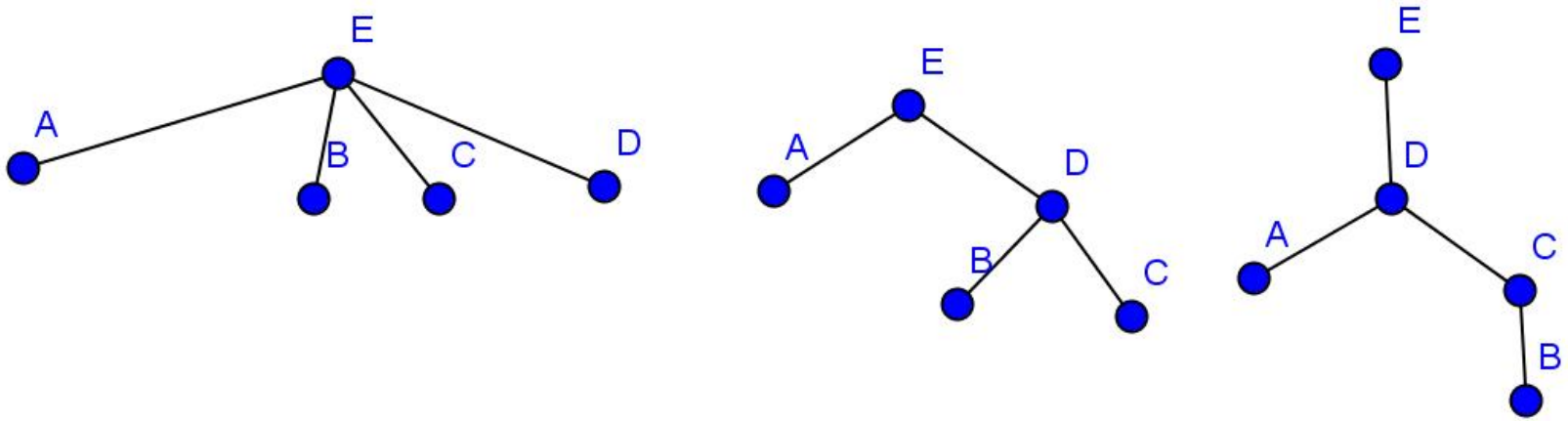
Wiemy już, jak można wierzchołki drzewa z wyróżnionym korzeniem zapisać w formie listy uporządkowanej w odpowiedni sposób. Jednak, czy można wykonać operację odwrotną? Czy możemy odtworzyć drzewo mając daną tylko listę wierzchołków i sposób powstania listy (prefiksowy, postfiksowy, infiksowy)? Na ogół odpowiedź jest przecząca.

Dekodowanie PREORDER - przykład



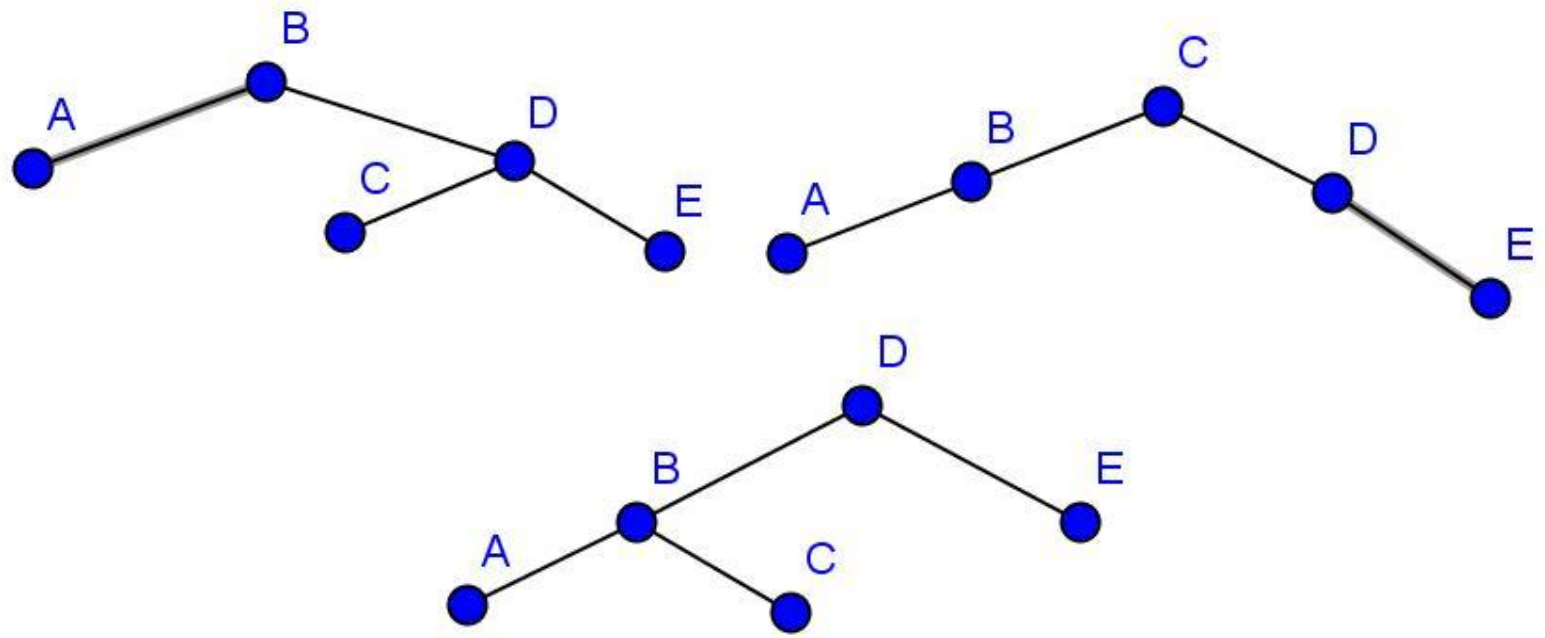
Wszystkie powyższe drzewa mają porządek prefiksowy *ABCDE*.

Dekodowanie POSTORDER - przykład



Wszystkie powyższe drzewa mają porządek postfiksowy *ABCDE*.

Dekodowanie INORDER - przykład



Wszystkie powyższe drzewa mają porządek infiksowy *ABCDE*.

Twierdzenie o odtwarzaniu drzew z list

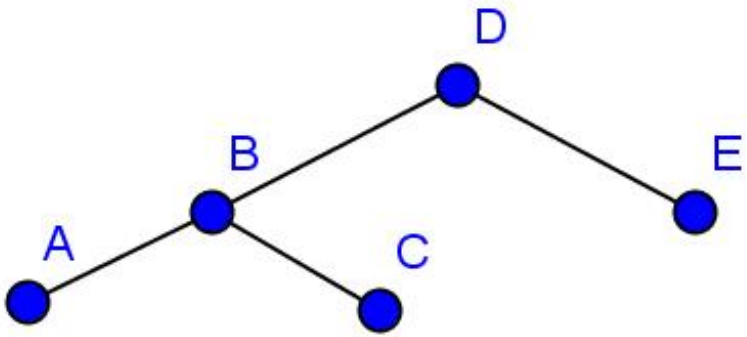
Otóż, odtworzenie jest możliwe, pod warunkiem, że mamy dodatkowe dane, które musimy sobie zapewnić podczas przechodzenia drzewa.

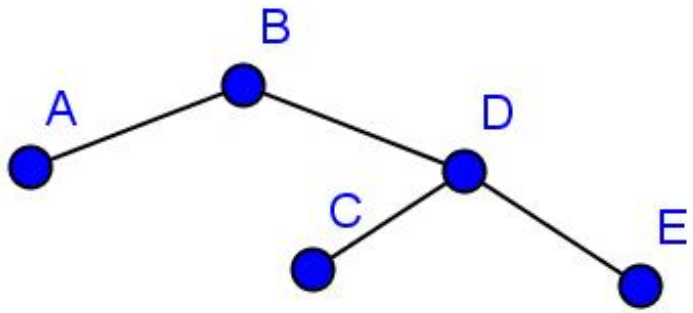
Twierdzenie o odtwarzaniu drzew z list

Niech T będzie skończonym, uporządkowanym drzewem z wyróżnionym korzeniem, którego wierzchołki są wypisane w porządku prefiksowym lub postfiksowym. Załóżmy, że znana jest liczba dzieci każdego wierzchołka. Wtedy to drzewo jest wyznaczone jednoznacznie i może być odtworzone z ciągu wierzchołków.

Niestety, nie da się uzyskać takiego wyniku dla porządku infiksowego, dlatego jest on mniej używany w komputerowych strukturach danych. Natomiast dla porządków pre- i post-fiksowych, twierdzenie to jest konstruktywne, czyli pozwala na algorytmiczną konstrukcję zakodowanego drzewa, co wkrótce zobaczymy.

Odtwarzanie drzewa z porządku infiksowego?





Dwa powyższe grafy nie tylko są kodowane w porządku infiksowym przez $ABCDE$, ale też ciąg dzieci każdego z wierzchołków w obydwu przypadkach można zapisać jako $(0, 2, 0, 2, 0)$ (czyli 0 dzieci węzła A, 2 dzieci węzła B itd.). Stąd wynika, że mając dany taki kod drzewa w porządku infiksowym i liczbę dzieci każdego węzła, nie jesteśmy w stanie odgadnąć, które z drzew reprezentuje podany kod. Dlatego twierdzenie o odtwarzaniu drzew dla porządku infiksowego byłoby nieprawdziwe.

Algorytmy odtwarzana drzewa - uwagi

Istnieją też inne twierdzenia o odtwarzaniu drzew z uporządkowanych list, których tu nie będziemy szczegółowo analizować.

- Lista wierzchołków w porządku prefiksowym lub postfiksowym wyznacza jednoznacznie całe drzewo o ile znamy poziom każdego wierzchołka w drzewie.
- Znajomość dwóch list wierzchołków: w porządku infiksowym oraz w jednym z dwóch pozostałych porządków wystarcza do odtworzenia w sposób jednoznaczny.

Algorytm DEPRE

Poniżej algorytm odzyskiwania drzewa z jego porządku prefiksowego (z zadaną liczbą dzieci każdego węzła).

DEPRE($v_1, \ldots v_n, c_1, \ldots c_n$)

Dane: $v_1, \ldots v_n$ - ciąg wierzchołków drzewa T w porządku prefiksowym, $c_1, \ldots c_n$ - ciąg liczb dzieci tych wierzchołków.
Zmienne: i, j - liczniki pętli wierzchołków. $S(v_i)$ - zbiory wierzchołków , $U(v_i)$ - liczby całkowite.

- I. Dla każdego i podstawiamy $U(v_i) := c_i$, $S(v_i) = \emptyset$. Następnie $i := 2$.
- II. Dopóki $i \leqslant n$ wykonujemy:
- Znajdujemy maksymalne możliwe $j < i$, takie, że $U(v_j) \neq 0$. Wtedy $U(v_j) := U(v_j) - 1$, $S(v_j) := S(v_j) \cup \{v_i\}$, $i := i + 1$.
- **Rezultat**: $S(v_i)$ - ciąg zbiorów, zawierających wierzchołki będące dziećmi kolejnych węzłów v_i.

DEPRE - uwagi

- $S(v_i)$ to zbiór dzieci węzła v_i.
- Jeśli znamy zbiory dzieci wszystkich węzłów, to znamy też wszystkie połączenia między węzłami, więc potrafimy odtworzyć całe drzewo.
- Algorytm DEPRE stworzy CO NAJWYŻEJ jedno drzewo. W szczególności, może się okazać, że dane, którymi go „nakarmiliśmy" nie są spełnione przez żadne drzewo. Przykładowo, żaden ciąg $(c_1, \ldots c_n)$ taki, że $c_n \neq 0$ nie może być ciągiem dzieci wierzchołków drzewa uporządkowanych prefiksowo, bo n-ty wierzchołek w tym porządku jest liściem, więc nie może mieć dzieci (które musiałyby się pojawić po swoim rodzicu).
- Podobnie, jeśli $n > 1$, to musi być $c_1 \neq 0$, bo pierwszym węzłem na liście uporządkowanej prefiksowo musi być korzeń, który musi mieć dzieci (chyba, że to jest drzewo o jednym wierzchołku).

DEPRE - przykład

Przeanalizujemy działanie algorytmu DEPRE na przykładzie listy wierzchołków *ABDIEFJCGH* i ciągu przypisanych im dzieci: $(2, 3, 1, 0, 0, 1, 0, 2, 0, 0)$. Wyniki będziemy zapisywać w takiej tabelce:

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$										
$S(v_k)$										

DEPRE - przykład

W pierwszym kroku, wpisujemy do tabelki c_i w miejsce $U(v_i)$, oraz $\emptyset$ w miejsce $S(v_i)$.

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	2	3	1	0	0	1	0	2	0	0
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	2	3	1	0	0	1	0	2	0	0
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Wierzchołek A, jako początek listy prefiksowej, jest oczywiście korzeniem drzewa, więc nie szukamy mu rodzica. „Szukanie rodziców" zaczynamy od wierzchołka B ($i = 2$). Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim A. Zatem podstawiamy $U(A) := 2 - 1 = 1$ i $S(A) = \emptyset \cup \{B\} = \{B\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	3	1	0	0	1	0	2	0	0
$S(v_k)$	B	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka D $(i = 3)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim B. Zatem podstawiamy $U(B) := 3 - 1 = 2$ i $S(B) = \emptyset \cup \{D\} = \{D\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	2	1	0	0	1	0	2	0	0
$S(v_k)$	B	D	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka I $(i = 4)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim D. Zatem podstawiamy $U(D) := 1 - 1 = 0$ i $S(D) = \emptyset \cup \{I\} = \{I\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	2	0	0	0	1	0	2	0	0
$S(v_k)$	B	D	I	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka E $(i = 5)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim B. Zatem podstawiamy $U(B) := 2 - 1 = 1$ i
$S(B) = \{D\} \cup \{E\} = \{D, E\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	1	0	0	0	1	0	2	0	0
$S(v_k)$	B	DE	I	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka F $(i = 6)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim B. Zatem podstawiamy $U(B) := 1 - 1 = 0$ i
$S(B) = \{D, E\} \cup \{F\} = \{D, E, F\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	0	0	0	0	1	0	2	0	0
$S(v_k)$	B	DEF	I	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka J $(i = 7)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim F. Zatem podstawiamy $U(F) := 1 - 1 = 0$ i $S(F) = \emptyset \cup \{J\} = \{J\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	1	0	0	0	0	0	0	2	0	0
$S(v_k)$	B	DEF	I	$\emptyset$	$\emptyset$	J	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka C $(i = 8)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim A. Zatem podstawiamy $U(A) := 1 - 1 = 0$ i
$S(A) = \{B\} \cup \{C\} = \{B, C\}$

DEPRE - przykład

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	0	0	0	0	0	0	0	2	0	0
$S(v_k)$	BC	DEF	I	$\emptyset$	$\emptyset$	J	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka G $(i = 9)$. Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim C. Zatem podstawiamy $U(C) := 2 - 1 = 1$ i $S(C) = \emptyset \cup \{G\} = \{G\}$

DEPRE - przykład

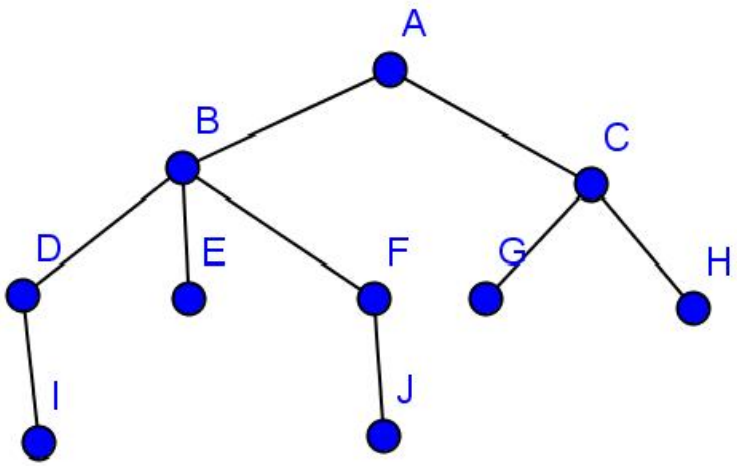
v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	0	0	0	0	0	0	0	1	0	0
$S(v_k)$	BC	DEF	I	$\emptyset$	$\emptyset$	J	$\emptyset$	G	$\emptyset$	$\emptyset$

Przechodzimy do wierzchołka H ($i = 10$). Szukamy poprzedniego wierzchołka na liście, dla którego $U \neq 0$. W tym przypadku jest nim C. Zatem podstawiamy $U(C) := 1 - 1 = 0$ i $S(C) = \{G\} \cup \{H\} = \{G, H\}$. W tym momencie doszliśmy do końca algorytmu - jako, że cały wiersz U naszej tabeli składa się teraz z zer, początkowy ciąg wierzchołków z liczbą ich dzieci faktycznie kodował drzewo.

DEPRE - wynik przykładu

v_k	A	B	D	I	E	F	J	C	G	H
$U(v_k)$	0	0	0	0	0	0	0	0	0	0
$S(v_k)$	BC	DEF	I	$\emptyset$	$\emptyset$	J	$\emptyset$	GH	$\emptyset$	$\emptyset$

Teraz drzewo łatwo można narysować:



Algorytm DEPOST

Poniżej algorytm dekodowania drzewa z jego porządku postfiksowego (z zadaną liczbą dzieci każdego węzła).

DEPOST($v_1, \ldots v_n$, $c_1, \ldots c_n$)

Dane: $v_1, \ldots v_n$ - ciąg wierzchołków drzewa T w porządku postfiksowym, $c_1, \ldots c_n$ - ciąg liczb dzieci tych wierzchołków.
Zmienne: i - licznik pętli wierzchołków. $S(v_i)$ - zbiory wierzchołków , $U(v_i)$- ciągi wierzchołków.

- I. Dla każdego $i = 1, 2, \ldots, n$ definiujemy:
- Ia. Jeśli $c_i = 0$, to $S(v_i) = \emptyset$, a $U(v_i)$ jest dowolne.
- Ib. Jeśli $c_i \neq 0$ to $U(v_i) = \{v_k : k < i\} \setminus \bigcup_{k<i} S(v_i)$ (uporządkowany w kolejności danych wejściowych). Wtedy definiujemy $S(v_i)$ jako c_i ostatnich elementów ciągu $U(v_i)$.
- **Rezultat**: $S(v_i)$ - ciąg zbiorów, zawierających wierzchołki będące dziećmi kolejnych węzłów v_i.

DEPOST - uwagi

- $S(v_i)$ to zbiór dzieci węzła v_i.
- Jeśli znamy zbiory dzieci wszystkich węzłów, to znamy też wszystkie połączenia między węzłami, więc potrafimy odtworzyć całe drzewo.
- Algorytm DEPOST stworzy CO NAJWYŻEJ jedno drzewo. W szczególności, może się okazać, że dane, którymi go „nakarmiliśmy" nie są spełnione przez żadne drzewo. Przykładowo, o ile $n > 1$, to żaden ciąg $(c_1, \ldots c_n)$ taki, że $c_n = 0$ nie może być ciągiem dzieci wierzchołków drzewa uporządkowanych postfiksowo, bo n-ty wierzchołek w tym porządku jest korzeniem, więc musi mieć dzieci, by drzewo istniało.
- Podobnie musi być $c_1 = 0$, bo pierwszym węzłem na liście uporządkowanej postfiksowo musi być liść (jakby miał dzieci, to musiałyby być przed rodzicem).

DEPOST - przykład

Przeanalizujemy działanie algorytmu DEPOST na przykładzie listy wierzchołków *ABCDEFGHIJ* i ciągu przypisanych im dzieci: $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$. Wyniki będziemy zapisywać w takiej tabelce:

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$										
$S(v_k)$										

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_1 = A$, $c_1 = 0$, zatem $S(A) = \emptyset$, a $U(A)$ nie ma znaczenia (co oznaczymy pauzą).

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-									
$S(v_k)$	$\emptyset$									

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_2 = B$, $c_2 = 0$, zatem $S(B) = \emptyset$, a $U(B)$ nie ma znaczenia (co oznaczymy pauzą). Podobna sytuacja zachodzi z węzłami C i D.

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-						
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$						

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_5 = E$, $c_5 = 3$. Po raz pierwszy obliczamy ciąg U (konkretnie $U(E)$).
Zgodnie z krokiem Ib algorytmu DEPOST, $U(E)$ jest to ciąg wszystkich wierzchołków poprzedzających E, które nie były w żadnym dotychczasowym zbiorze S. Czyli, jest to $ABCD$. $S(E)$ to jest c_5 (czyli 3) ostatnich wyrazów ciągu $U(E)$, czyli BCD.

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-	ABCD					
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD					

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_6 = F$, $c_6 = 2$. Obliczamy ciąg $U(F)$. Zgodnie z krokiem Ib algorytmu DEPOST, $U(F)$ jest to ciąg wszystkich wierzchołków poprzedzających F, które nie były w żadnym dotychczasowym zbiorze S. Czyli, jest to AE. $S(F)$ to jest c_6 (czyli 2) ostatnich wyrazów ciągu $U(F)$, czyli AE.

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-	ABCD	AE				
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD	AE				

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
Z wierzchołkami G i H postępujemy jak z A, B, C, D (bo też mają 0 dzieci).

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-	ABCD	AE	-	-		
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD	AE	$\emptyset$	$\emptyset$		

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_9 = I$, $c_9 = 2$. Obliczamy ciąg $U(I)$. Zgodnie z krokiem Ib algorytmu DEPOST, $U(I)$ jest to ciąg wszystkich wierzchołków poprzedzających I, które nie były w żadnym dotychczasowym zbiorze S. Czyli, jest to FGH. $S(I)$ to jest c_9 (czyli 2) ostatnich wyrazów ciągu $U(I)$, czyli GH.

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-	ABCD	AE	-	-	FGH	
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD	AE	$\emptyset$	$\emptyset$	GH	

DEPOST - przykład

Dane: $ABCDEFGHIJ$, $(0, 0, 0, 0, 3, 2, 0, 0, 2, 2)$.
$v_{10} = J$, $c_{10} = 2$. Obliczamy ciąg $U(J)$. Zgodnie z krokiem Ib algorytmu DEPOST, $U(J)$ jest to ciąg wszystkich wierzchołków poprzedzających J, które nie były w żadnym dotychczasowym zbiorze S. Czyli, jest to FI. $S(J)$ to jest c_{10} (czyli 2) ostatnich wyrazów ciągu $U(J)$, czyli FI.

v_k	A	B	C	D	E	F	G	H	I	J
$U(v_k)$	-	-	-	-	ABCD	AE	-	-	FGH	FI
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD	AE	$\emptyset$	$\emptyset$	GH	FI

DEPOST - wynik przykładu

v_k	A	B	C	D	E	F	G	H	I	J
$S(v_k)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	BCD	AE	$\emptyset$	$\emptyset$	GH	FI

Ponieważ wszystkie wierzchołki drzewa, poza korzeniem J, są wskazane jako dzieci innego wierzchołka, tabela ta pokazuje rzeczywiste drzewo, które łatwo można narysować: