2d. Algorytmy przeszukiwania grafów

Grzegorz Kosiorowski

Uniwersytet Ekonomiczny w Krakowie

Motywacja

Wiele algorytmów postępowania grafów opiera się na ich przechodzeniu lub **przeszukiwaniu**. Przez te pojęcia rozumiemy poruszanie się po grafie wzdłuż jego krawędzi tak, by odwiedzić wszystkie jego wierzchołki.
Czasem samo odwiedzenie wszystkich wierzchołków jest celem. Na przykład, jeśli startując z jednego wierzchołka i przechodząc tylko krawędziami odwiedzimy wszystkie wierzchołki (czyli algorytm przechodzenia zakończy się po odwiedzeniu wszystkich wierzchołków grafu) to oznacza, że graf wyjściowy jest spójny. Dlatego same algorytmy przeszukiwania są dobrymi testami spójności grafu.
Najczęściej jednak odwiedzając wierzchołki przy okazji wykonujemy na nich jakieś procedury.

Co można zrobić z odwiedzonymi wierzchołkami?

- Dopisać wierzchołek do listy. Można tworzyć listy wierzchołków uporządkowane według różnych kryteriów (szczególnie ważne w teorii drzew jako baz danych).
- Nadać etykiety, czyli nazwy odpowiednie dla konkretnego problemu opisywanego przez graf.
- Przypisać wskaźniki, czyli pewne wartości opisujące cechy wierzchołków. Przykładem była procedura testowania dwudzielności: odwiedzaliśmy wszystkie wierzchołki i każdemu przypisywaliśmy wskaźnik przynależności do zbioru V_1 lub V_2. W sieci społecznej tworzonej przez jakiś zespół zadaniowy, wskaźnikiem może być funkcja danej osoby w zespole. W sieci drogowej, odległość od zadanego punktu.
- Usunąć wierzchołek z grafu, gdy nie jest już potrzebny (np. *garbage collection*).
- Dołączyć wierzchołek do grafu, gdy jest potrzebny (np. wprowadzanie nowej osoby do sieci społecznej przez znajomego).

Algorytm przeszukiwania wszerz - założenia

Algorytm przeszukiwania wszerz polega na odwiedzeniu wszystkich wierzchołków wzdłuż krawędzi, porządkując je według wzrastającej odległości od zadanego wierzchołka startowego.
Oczywiście, uda nam się to tylko jeśli graf jest spójny.
Dla wygody, zaczynamy od ponumerowania w dowolny sposób wierzchołków takiego grafu $V(G) = \{1, \ldots, n\}$. Jeśli wierzchołki są oznaczone literami, zakładamy, że ich porządek jest zgodny z kolejnością alfabetyczną (np. A=1, B=2 itp.).
W rezultacie wykonania algorytmu, otrzymamy odwiedzone wierzchołki ustawione w ciąg. Podczas odwiedzania będziemy dzielić zbiór wierzchołków na 3 części: L - wierzchołki odwiedzone wcześniej (zaznaczane na zielono), S - wierzchołki odwiedzone ostatnio (czerwone) oraz resztę.

Algorytm przeszukiwania wszerz

Algorytm przeszukiwania wszerz

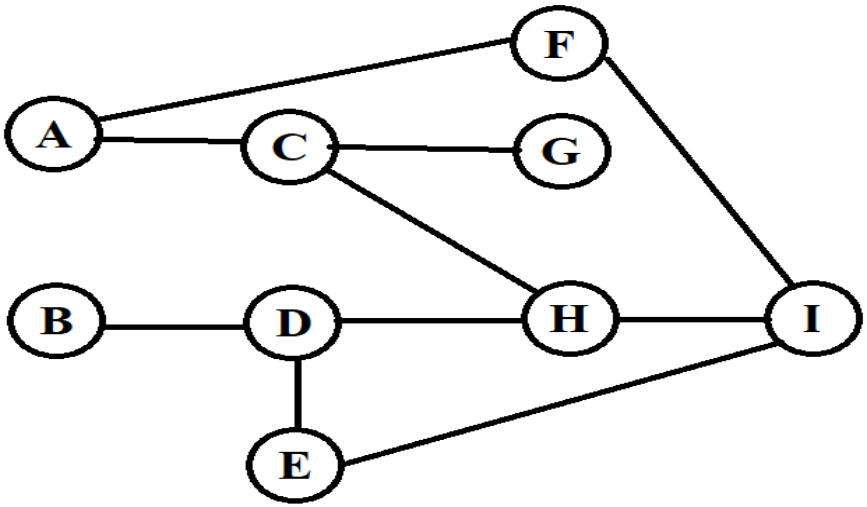
Dane: Graf spójny $G = (V(G), E(G))$, ze zbiorem wierzchołków $V(G) = \{1, \ldots, n\}$.
Zmienne: L, S i V - zbiory wierzchołków, i, j - liczby pomocnicze, a - ciąg wierzchołków.

- I. $S := \{1\}$, $V := \{2, \ldots n\}$, $L := \emptyset$, $a = (1)$.
- II. Dopóki $V \neq \emptyset$, dla każdego $i \in S$ (w kolejności dołączania do zbioru S) wykonuj:
- II.1 Przenieś i z S do L.
- II.2 Dopóki istnieje $j \in V$ takie, że $\{i, j\} \in E(G)$, przenieś j z V do S i dopisz j na końcu ciągu a.
- **Rezultat**: a to ciąg wszystkich wierzchołków grafu G ustawionych w kolejności wzrastającej odległości od wierzchołka 1.

Algorytm przeszukiwania wszerz - komentarz

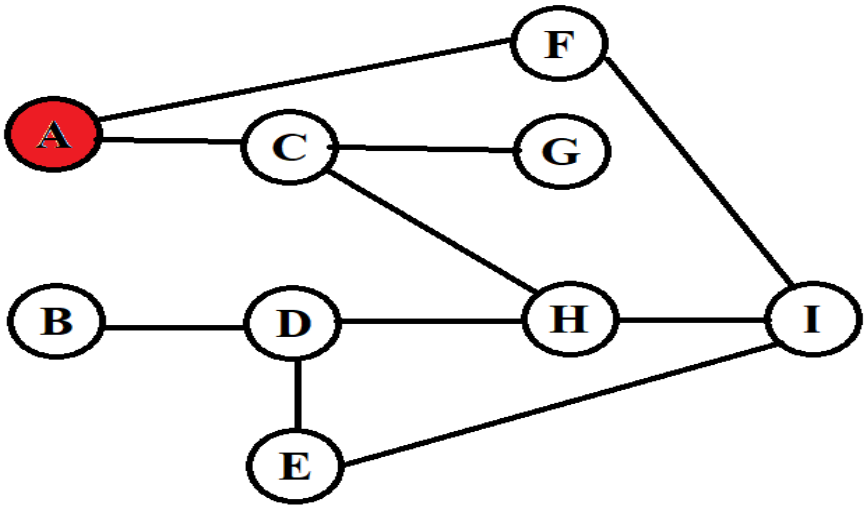
Potoczny opis algorytmu: najpierw odwiedzamy wierzchołek startowy. Następnie odwiedzamy kolejno sąsiadów wierzchołka startowego. Następnie szukamy wierzchołków, których jeszcze nie odwiedziliśmy, a sąsiadują z wierzchołkami, które ostatnio odwiedziliśmy, w kolejności odwiedzania (wymaga to zapisania ostatnio odwiedzonych wierzchołków w pomocniczej strukturze danych - kolejce). Te właśnie wierzchołki odwiedzamy i zapisujemy jako kolejkę nowoodwiedzonych wierzchołków, poprzednio odwiedzane wierzchołki usuwając z tej kolejki... i tak dalej, aż przejdziemy cały graf.

Algorytm przeszukiwania wszerz - przykład



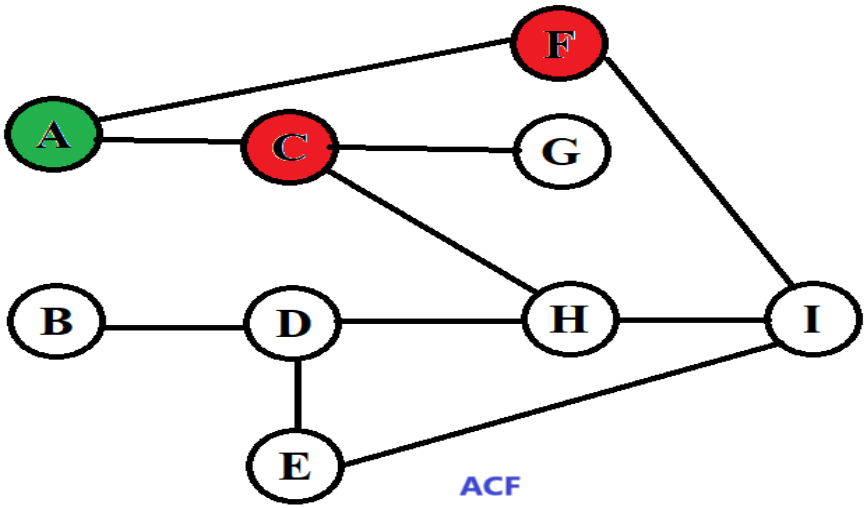
Spróbujemy zastosować algorytm przeszukiwania wszerz dla powyższego grafu nieskierowanego. Graf ten ewidentnie jest spójny, więc algorytm zadziała.

Algorytm przeszukiwania wszerz - przykład



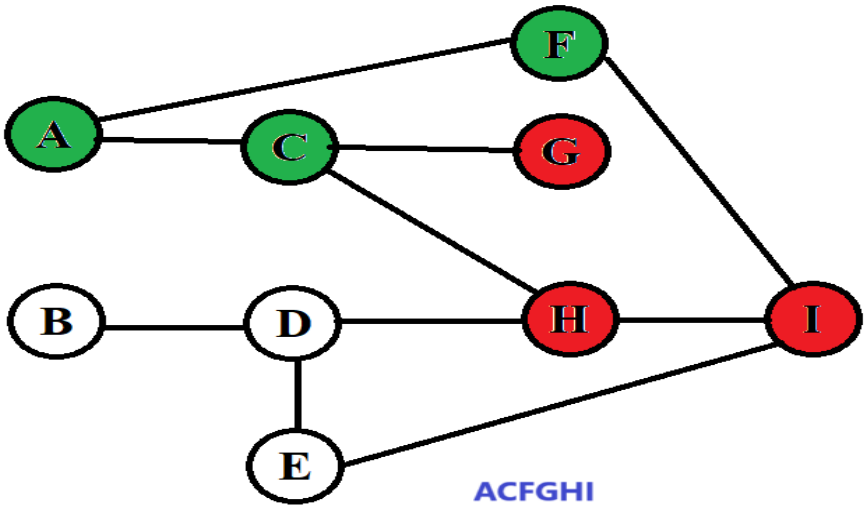
Algorytm zaczynamy od ponumerowania wierzchołków w kolejności alfabetycznej: A-1, B-2 itd... Zaczynamy przeszukiwanie od wierzchołka A, który na początku dopisujemy do zbioru S (zaznaczamy na czerwono) i staje się też pierwszym wierzchołkiem w odwiedzonym ciągu.

Algorytm przeszukiwania wszerz - przykład



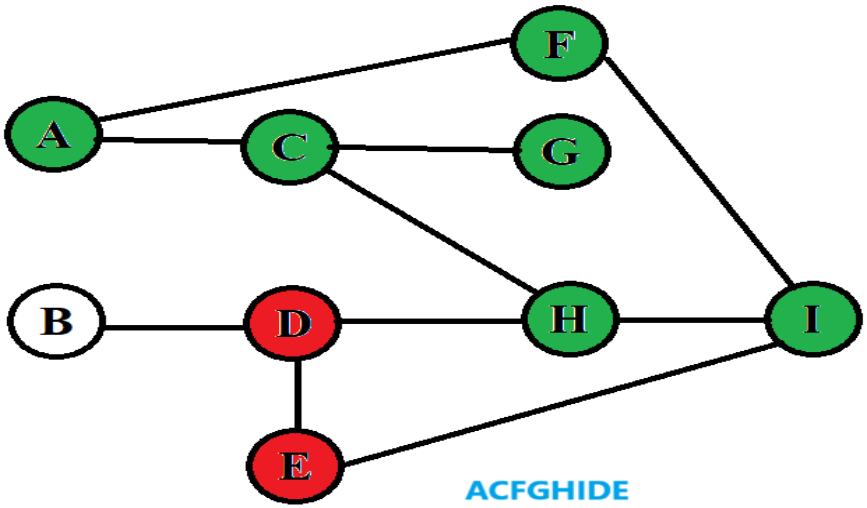
Wierzchołek A przenosimy z S do L (zaznaczamy na zielono). Sąsiedzi A, czyli C i F, należą do V, więc przenosimy ich do zbioru S i dopisujemy do ciągu odwiedzonych wierzchołków w kolejności alfabetycznej. V (zbiór białych wierzchołków) jest niepusty, więc kontynuujemy, szukając sąsiadów wierzchołków z S.

Algorytm przeszukiwania wszerz - przykład



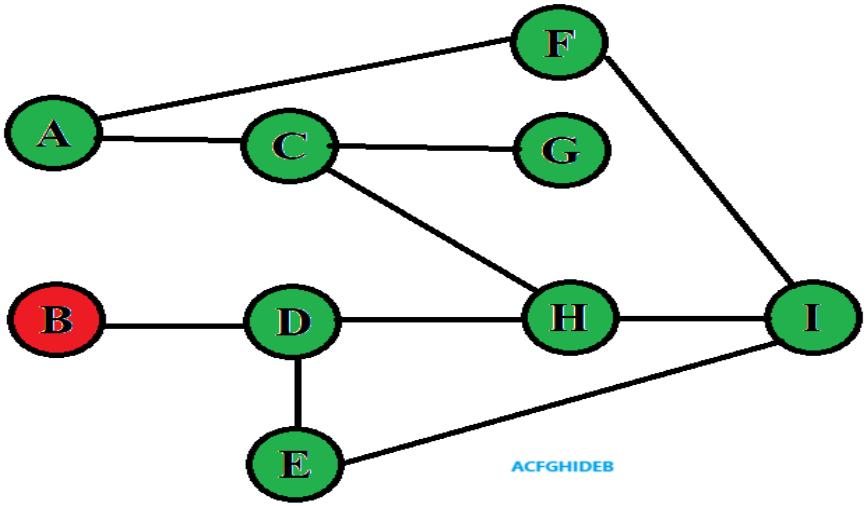
Wierzchołek C przenosimy z S do L (zaznaczamy na zielono). Sąsiedzi C, należący do V to G i H: dołączamy ich do zbioru S i dopisujemy do ciągu odwiedzonych wierzchołków w kolejności alfabetycznej. To samo czynimy z wierzchołkiem F, przeniesionym z S do L i jego sąsiadem I przeniesionym z V do S. A nie był w V, więc jego nie dopisujemy do S, mimo, że był sąsiadem C i F!

Algorytm przeszukiwania wszerz - przykład



Wierzchołek G nie ma sąsiadów z V, więc tylko przenosimy go do L. H ma takiego sąsiada (D), podobnie I (E), więc H i I zostają przeniesione do zbioru L, a D i E (w tej kolejności) zostają odwiedzone i znajdują się w zbiorze S. C i F nie były w V, więc ich nie dopisujemy do S, mimo, że były sąsiadami odpowiednio H i I!

Algorytm przeszukiwania wszerz - przykład



Wierzchołek D ma sąsiada w V (B), zatem przenosimy D do zbioru L, odwiedzamy B i przenosimy B do zbioru S. W tym momencie zbiór V staje się pusty, więc kończymy algorytm przeszukiwania wszerz, odwiedzając wszystkie wierzchołki w kolejności: ACFGHIDEB.

Dodatkowe informacje o przeszukiwaniu wszerz

- Dla grafu $G = (V, E)$ czas działania algorytmu przeszukiwania wszerz to $O(|V| + |E|)$.
- Założenie, że graf jest spójny umożliwia zakończenie pracy algorytmu. Alternatywnie, algorytm ten można wykorzystać do sprawdzania, czy dany graf jest spójny: jeśli w jakimś obiegu pętli zbiór V byłby niepusty, a na koniec kroku II zbiór S byłby pusty to oznaczałoby, że graf jest niespójny. Natomiast jeśli algorytm udałoby się doprowadzić do końca i zbiór V stałby się pusty, to oznacza, że graf jest spójny.
- Testowanie dwudzielności było przykładem zastosowania algorytmu przeszukiwania wszerz.
- Algorytm został stworzony w 1945 roku przez Konrada Zuse w jego pracy doktorskiej... która jednak nie została dopuszczona do obrony, więc opublikowany został dopiero wiele lat później.

Wyznaczanie najkrótszej drogi

Proste zastosowanie algorytmu przeszukiwania wszerz: jak wyznaczyć najkrótszą (o najmniejszej liczbie krawędzi) drogę z wybranego wierzchołka do każdego innego w spójnym grafie skierowanym lub nieskierowanym bez wag?
Rozwiązanie: **algorytm przeszukiwania wszerz** (formalnie można dodać: **ze wskaźnikami**): dzięki drobnej modyfikacji, przydzielając odwiedzonym wierzchołkom wskaźniki, łatwo zmodyfikować przeszukiwanie tak, by wyznaczało najkrótszą drogę z wierzchołka startowego do każdego innego.
Uwaga! Pozorne uproszczenie zadania do „znajdź najkrótszą drogę z zadanego wierzchołka do innego zadanego wierzchołka” (a nie do wszystkich innych wierzchołków) nie ułatwia rozwiązania - najprościej jest użyć algorytmu przeszukiwania wszerz i przerwać go, gdy znajdziemy szukaną drogę.

Wyznaczanie najkrótszej drogi (wszerz)

Wyznaczanie najkrótszej drogi (wszerz)

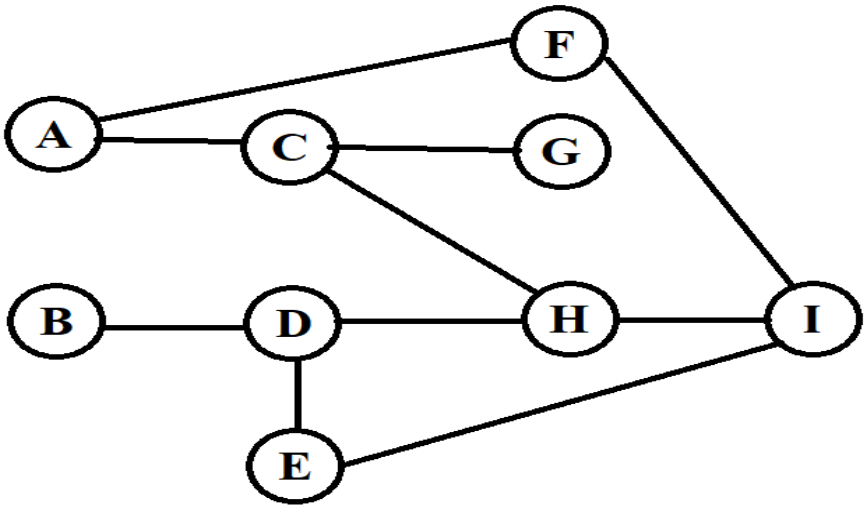
Dane: Graf spójny $G = (V(G), E(G))$, ze zbiorem wierzchołków $V(G) = \{1, \ldots, n\}$.
Zmienne: L, S i V - zbiory wierzchołków, d, p - funkcje, i, j - liczby pomocnicze.

- I. $L := \{1\}$, $V := \{2, \ldots n\}$. $d(1) = 0$, $p(1) = \emptyset$, $S := \emptyset$.
- II. Dopóki $L \neq V(G)$ wykonuj:
- II.1 Dla $j \in V \setminus L$ wykonuj: jeśli istnieje $i \in L$ takie, że $\{i, j\} \in E(G)$, dołącz j do S i zdefiniuj $d(j) = d(i) + 1$ oraz $p(j) = i$.
- II.2 $L := L \cup S$, $S := \emptyset$.
- **Rezultat**: $d(j)$ to minimalna długość drogi z 1 do j, a $p(j)$ to wskaźnik, który pokazuje poprzedni etap drogi z 1 do j.

Wyznaczanie najkrótszej drogi (wszerz) - komentarz

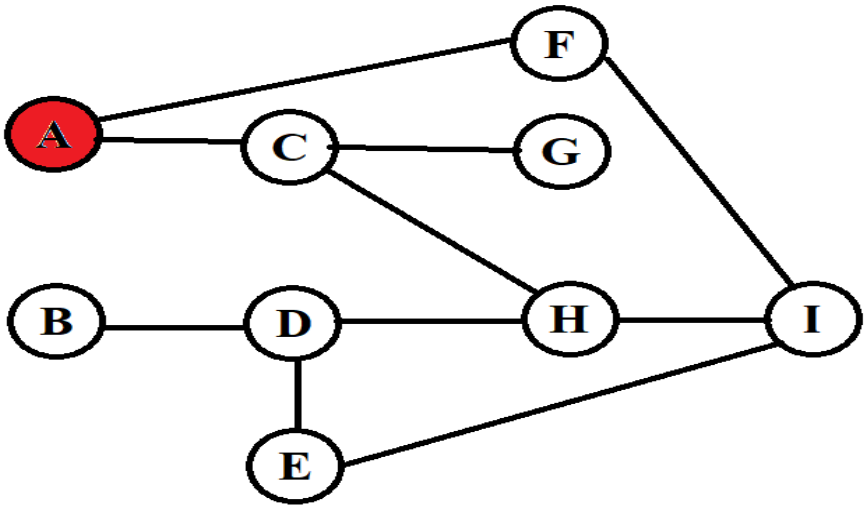
Minimalną długość drogi otrzymujemy od razu, natomiast nad samą drogą trzeba jeszcze chwilę popracować. W ciągu j, $p(j)$, $p(p(j))$, ... występują wierzchołki drogi minimalnej od 1 do j w odwrotnej kolejności - dlatego wystarczy ten ciąg wypisać od tyłu i już droga (jako ciąg wierzchołków) jest znaleziona.
Potocznie: Szukamy najpierw wierzchołków sąsiadujących ze startowym i przypisujemy im odległość 1. Następnie szukamy wierzchołków, których odległość nie została jeszcze wyznaczona, a sąsiadują z wierzchołkami o odległości 1 i przypisujemy im odległość 2. Następnie szukamy wierzchołków, których odległość nie została jeszcze wyznaczona, a sąsiadują z wierzchołkami o odległości 2... i tak dalej, aż przejdziemy cały graf. Za każdym razem, gdy przypisujemy wierzchołkowi jego odległość, zapisujemy z jakim wierzchołkiem o odległości o jeden mniejszej sąsiaduje - i to jest jego wskaźnik.

Wyznaczanie najkrótszej drogi (wszerz) - przykład



Spróbujemy zastosować algorytm wyznaczania najkrótszej drogi metodą przeszukiwania wszerz dla powyższego grafu nieskierowanego. Przedstawię tutaj sposób postępowania przyjęty w ramach tego kursu - obowiązujący na sprawdzianie/egzaminie. Będziemy obliczać minimalną długość drogi wszystkich wierzchołków od A, przyjmujemy alfabetyczny porządek wierzchołków.

Wyznaczanie najkrótszej drogi (wszerz) - przykład

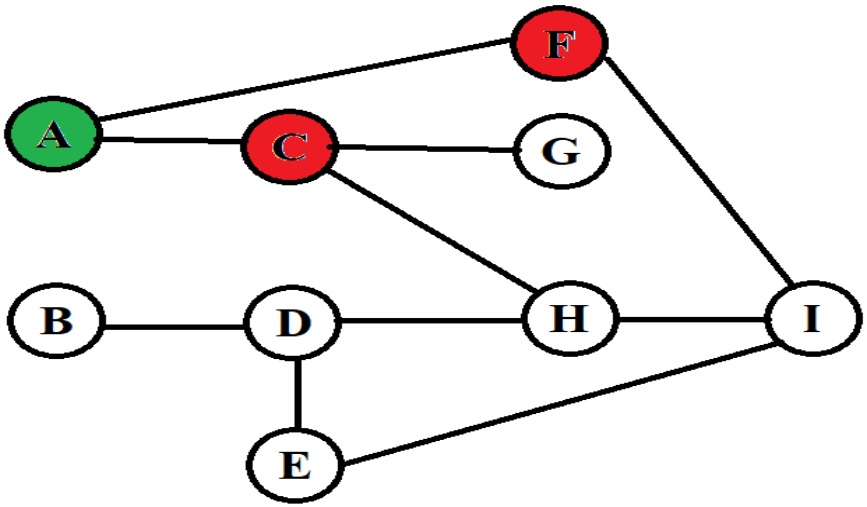


Odległość danego wierzchołka X od A to $d(X)$, a wskaźnik wierzchołka to $p(X)$. Zbiór L (zielone) to elementy, których odległości od A mamy policzone. Zbiór S (czerwone) to wierzchołki, których odległości obliczamy w danym etapie.

Działanie algorytmu zapisujemy w takiej tabeli:

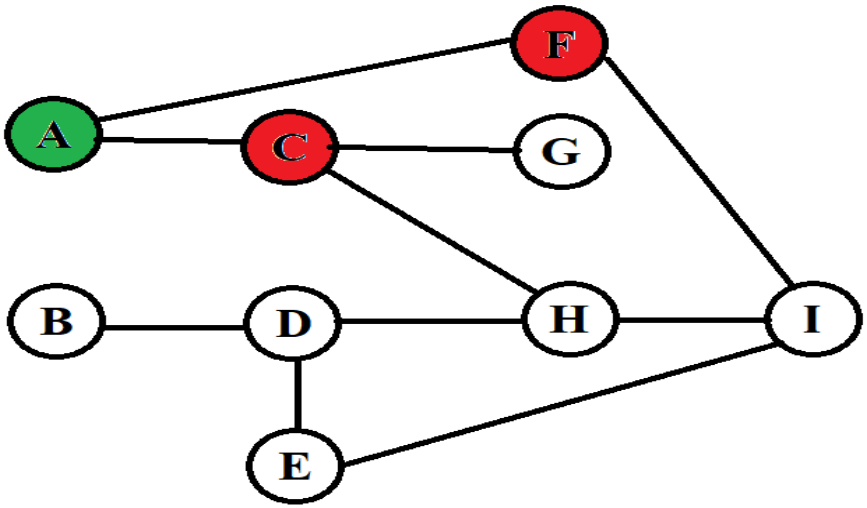
Nr etapu	zbiór L	zbiór S	uzyskane odległości	wskaźniki
1				
2				

Wyznaczanie najkrótszej drogi (wszerz) - krok 1



Na początku w zbiorze L jest tylko wierzchołek A. Szukamy sąsiadów wierzchołka A, którzy nie są w zbiorze L i przesuwamy ich do zbioru S.

Wyznaczanie najkrótszej drogi (wszerz) - krok 1

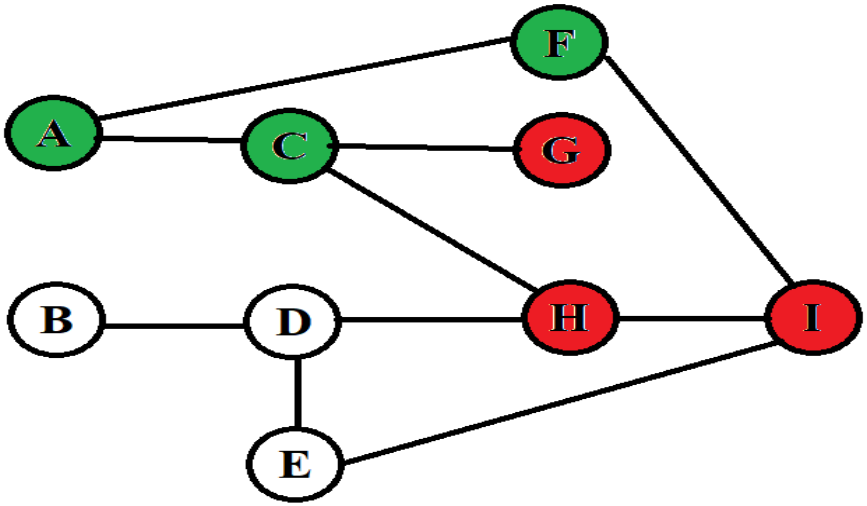


Skoro to pierwszy krok, obydwu sąsiadom A przypisujemy odległość 1. Ich sąsiadem będącym w zbiorze L jest A, więc ich wskaźnikiem jest A.

Zapisujemy to w tabeli:

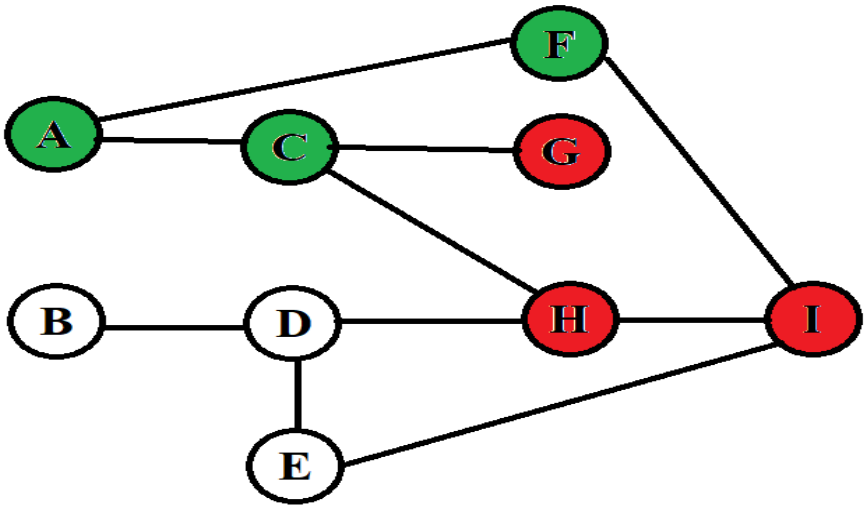
Nr etapu	zbiór L	zbiór S	uzyskane odległości	wskaźniki
1	A	C,F	$d(C) = d(F) = 1$	$p(C) = p(F) = A$

Wyznaczanie najkrótszej drogi (wszerz) - krok 2



Dołączamy wierzchołki z S do zbioru L i szukamy ich sąsiadów, którzy nie są w zbiorze L. Utworzą oni nowy zbiór S.

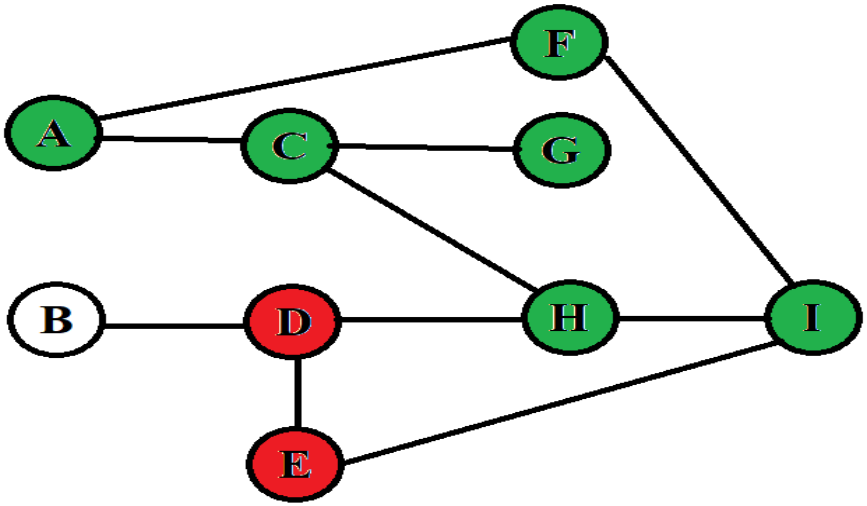
Wyznaczanie najkrótszej drogi (wszerz) - krok 2



Skoro to drugi krok, wierzchołkom z S przypisujemy odległość 2. Ich wskaźnikami są ich sąsiedzi ze zbioru L.

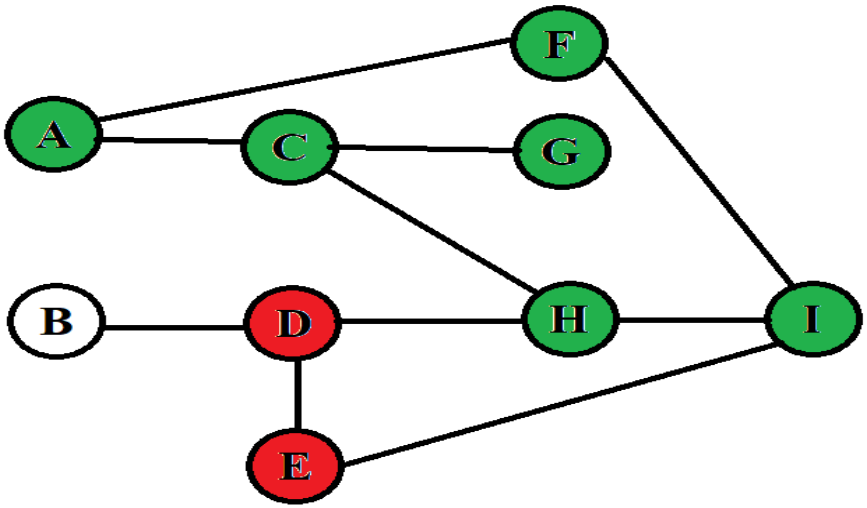
Nr	zbiór L	zbiór S	uzyskane odległości	wskaźniki
2	A,C,F	G,H,I	$d(G) = d(H) = d(I) = 2$	$p(G) = p(H) = C$, $p(I) = F$

Wyznaczanie najkrótszej drogi (wszerz) - krok 3



Dołączamy wierzchołki z S do zbioru L i szukamy ich sąsiadów, którzy nie są w zbiorze L. Utworzą oni zbiór S.

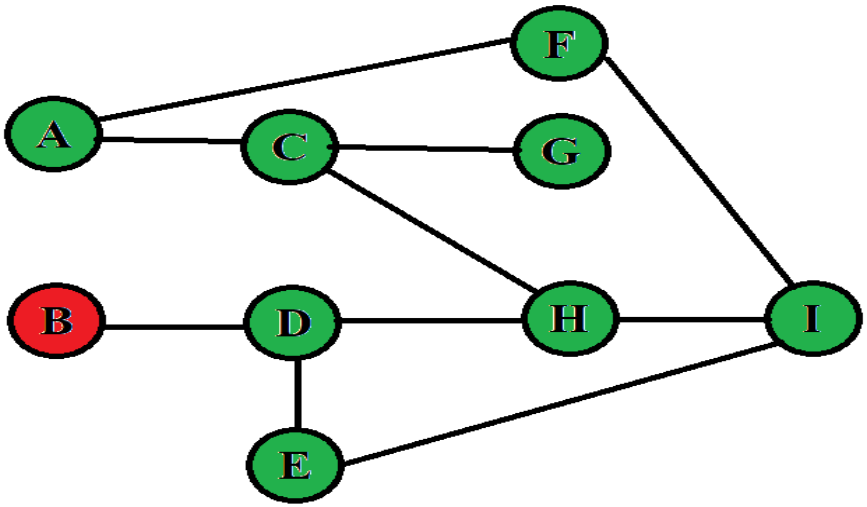
Wyznaczanie najkrótszej drogi (wszerz) - krok 3



Skoro to trzeci obieg pętli, wierzchołkom z S przypisujemy odległość 3. Ich wskaźnikami są ich sąsiedzi ze zbioru L.

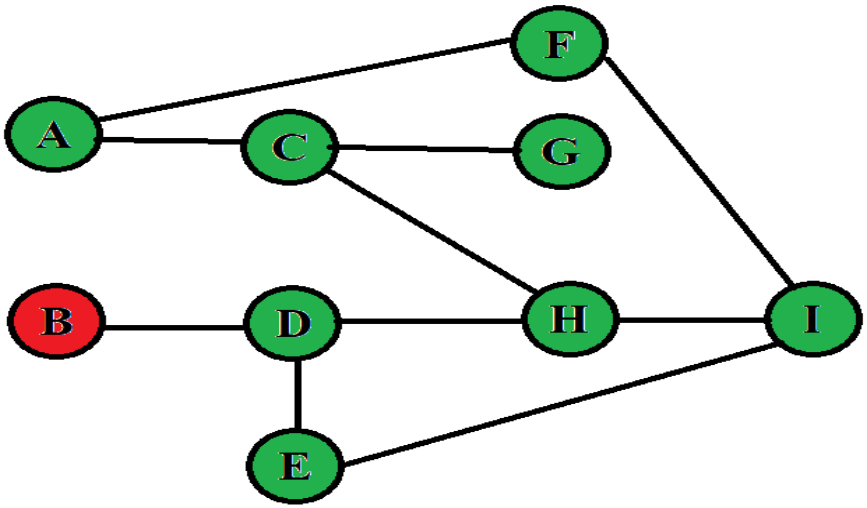
Nr	zbiór L	zbiór S	uzyskane odległości	wskaźniki
3	A,C,F,G,H,I	D,E	$d(D) = d(E) = 3$	$p(D) = H$, $p(E) = I$

Wyznaczanie najkrótszej drogi (wszerz)- krok 4



Dołączamy wierzchołki z S do zbioru L i szukamy ich sąsiadów, którzy nie są w zbiorze L. Został tylko jeden taki wierzchołek, który utworzy zbiór S.

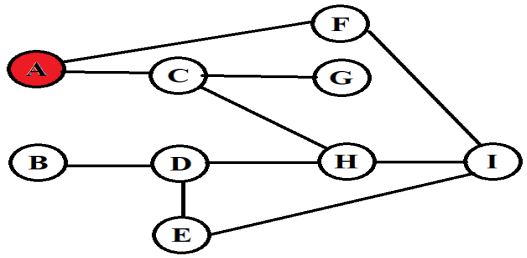
Wyznaczanie najkrótszej drogi (wszerz) - krok 4



Skoro to czwarty obieg pętli, wierzchołkowi z S przypisujemy odległość 4.

Nr	zbiór L	zbiór S	uzyskane odległości	wskaźniki
4	A,C,D,E, F,G,H,I	B	$d(B) = 4$	$p(B) = D$

Wyznaczanie najkrótszej drogi (wszerz) - tabela



Poniższa tabela jest wynikiem działania algorytmu:

Nr	zbiór L	zbiór S	obliczone odległości	wskaźniki
1	A	C,F	$d(C) = d(F) = 1$	$p(C) = p(F) = A$
2	A,C,F	G,H,I	$d(G) = d(H) = d(I) = 2$	$p(G) = p(H) = C$, $p(I) = F$
3	A,C,F,G,H,I	D, E	$d(D) = d(E) = 3$	$p(D) = H$, $p(E) = I$
4	A,C,D,E, F,G,H,I	B	$d(B) = 4$	$p(B) = D$

Jeśli chcemy znaleźć najkrótszą drogę np. z A do E, musimy zacząć od E, sprawdzić $p(E) = I$, następnie $p(I) = F$ i $p(F) = A$. Skoro dotarliśmy do A, droga się kończy i wystarczy wypisać jej wierzchołki w odwrotnej kolejności: $AFIE$.

Dodatkowe informacje o wyznaczaniu najkrótszej drogi

- Analizując grafy skierowane z wagami poznamy uogólnienie tego algorytmu: algorytm Dijkstry.
- W tej wersji algorytmu przeszukiwania grafu nie jest ważna dokładna kolejność wierzchołków dołączanych do S (wszystkie z tego samego obiegu pętli mają tę samą odległość od wierzchołka początkowego), więc też nie potrzebujemy dodatkowej struktury kolejki. Jeśli wierzchołek ma więcej niż jednego sąsiada w L, wybór wskaźnika spośród tych sąsiadów nie jest jednoznaczny i może być dowolny. Na potrzeby kursu wybieramy w takich przypadkach na wskaźnik wierzchołek występujący wcześniej w porządku liczbowym/alfabetycznym.

Algorytm przeszukiwania w głąb - założenia

Algorytm przeszukiwania w głąb lub **przeszukiwania z nawrotami** polega na odwiedzeniu wszystkich wierzchołków wzdłuż krawędzi, wędrując wzdłuż jednej drogi wychodzącej z wierzchołka początkowego. Jeśli dojdziemy do wierzchołka, który nie sąsiaduje z żadnym nieodwiedzonym cofamy się wzdłuż dotychczas utworzonej drogi do wierzchołka na ścieżce, który ma nieodwiedzonych sąsiadów i kontynuujemy odwiedziny stamtąd.
Tak jak przy każdym przeszukiwaniu, zaczynamy od ponumerowania w dowolny sposób wierzchołków takiego grafu $V(G) = \{1, \ldots, n\}$ i jeśli wierzchołki są oznaczone literami, zakładamy, że ich porządek jest zgodny z kolejnością alfabetyczną (np. A=1, B=2 itp.).
W rezultacie wykonania algorytmu, otrzymamy odwiedzone wierzchołki ustawione w ciąg. Podczas odwiedzania będziemy dzielić zbiór wierzchołków na 3 części: L - wierzchołki już odwiedzone (zaznaczane na zielono), S - wierzchołki, których sąsiadów właśnie badamy, oraz resztę.

Algorytm przeszukiwania w głąb

Przeszukiwanie w głąb

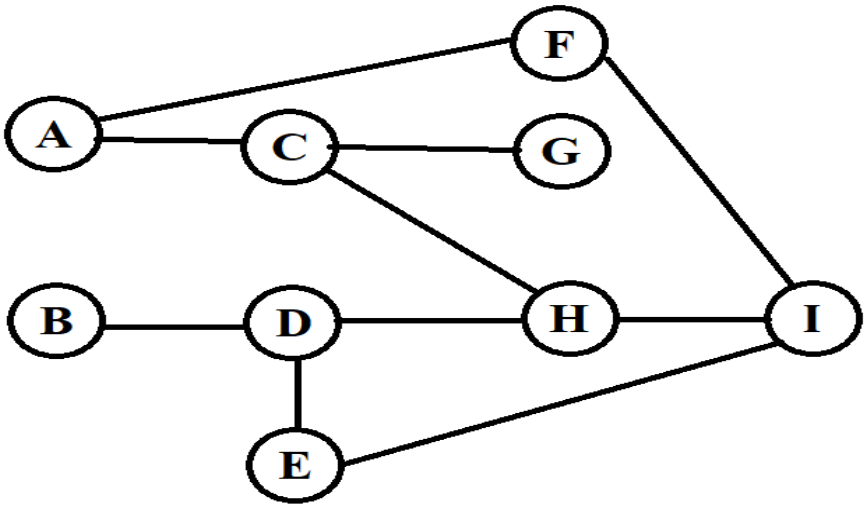
Dane: Graf spójny $G = (V(G), E(G))$, ze zbiorem wierzchołków $V(G) = \{1, \ldots, n\}$.
Zmienne: L, S i V - zbiory wierzchołków, i, j - liczby pomocnicze, a - ciąg wierzchołków.

- I. $S := \{1\}$, $V := \{2, \ldots n\}$, $L := \emptyset$, $a = (1)$.
- II. Dopóki $V \neq \emptyset$, $i \in S$ (S jest zawsze najwyżej jednoelementowy) wykonuj: Przenieś i z S do L.
- II.1 Jeśli $j \in V$ takie, że $\{i, j\} \in E(G)$, przenieś j z V do S i dopisz j na końcu ciągu a.
- II.2 Jeśli nie ma takiego j, cofnij się do poprzednika i w ciągu a i przenieś go z L do zbioru S.
- **Rezultat**: Ciąg wszystkich wierzchołków G odwiedzonych w kolejności „w głąb".

Algorytm przeszukiwania w głąb - komentarz

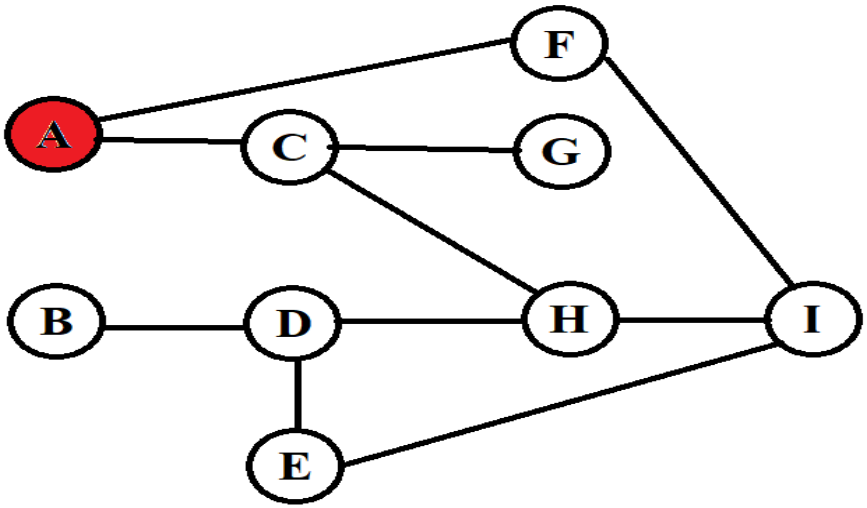
Potoczny opis algorytmu: najpierw odwiedzamy wierzchołek startowy i zapisujemy go na „stosie". Następnie odwiedzamy jego dowolnego sąsiada, dowolnego sąsiada tego sąsiada i tak dalej. Ciąg odwiedzonych wierzchołków jest jednocześnie pomocniczą strukturą danych - stosem. Jeśli dojdziemy do wierzchołka, który nie ma nieodwiedzonych sąsiadów, cofamy się wzdłuż tego ciągu do ostatniego wierzchołka, który takich sąsiadów ma i kontynuujemy procedurę, aż przejdziemy cały graf.
Algorytm przeszukiwania w głąb dużo łatwiej byłoby przedstawić jako rekurencyjny (wywołujący samego siebie). Pozostawiam to Państwu jako ćwiczenie.

Algorytm przeszukiwania w głąb - przykład



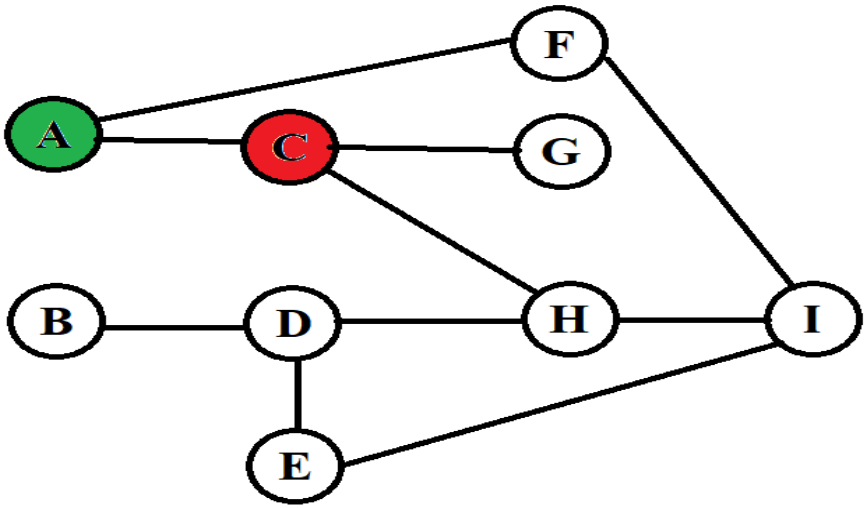
Spróbujemy zastosować algorytm przeszukiwania w głąb dla powyższego grafu nieskierowanego. Graf ten ewidentnie jest spójny, więc algorytm zadziała.

Algorytm przeszukiwania w głąb - przykład



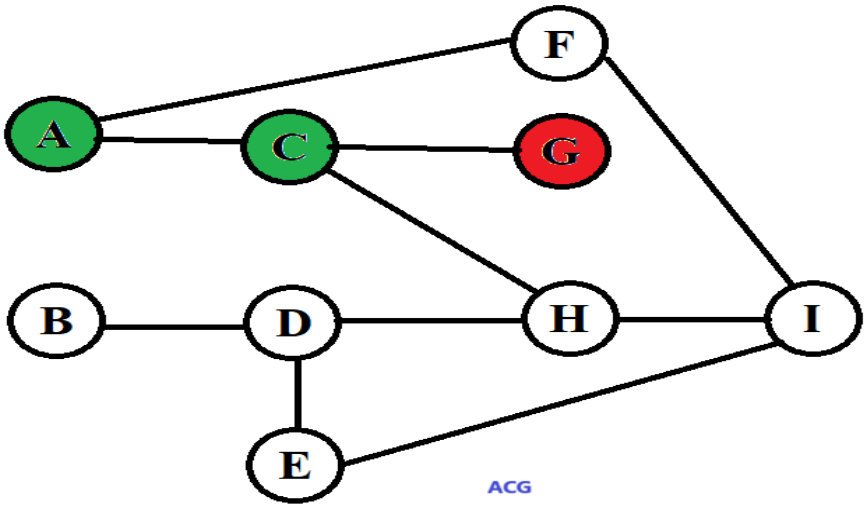
Algorytm zaczynamy od ponumerowania wierzchołków w kolejności alfabetycznej: A-1, B-2 itd... Zaczynamy przeszukiwanie od wierzchołka A, który na początku dopisujemy do zbioru S (zaznaczamy na czerwono) i staje się też pierwszym wierzchołkiem w odwiedzonym ciągu.

Algorytm przeszukiwania w głąb - przykład



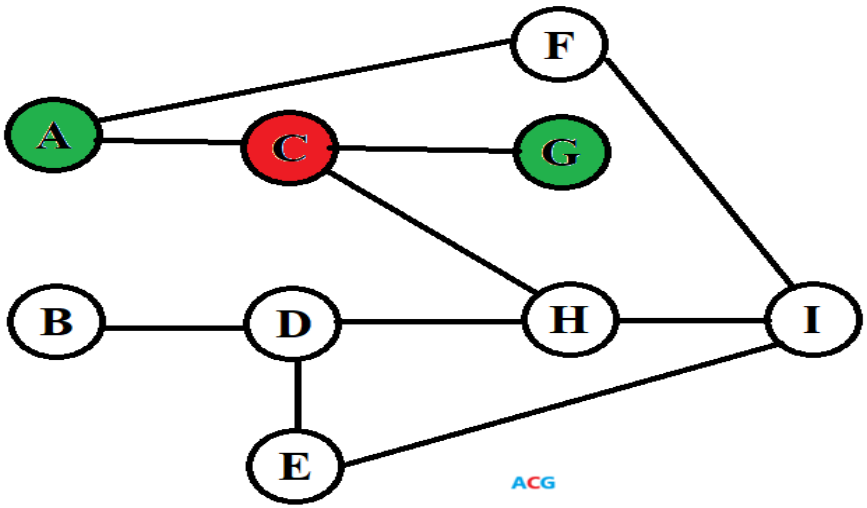
Wierzchołek A przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek C - nieodwiedzonego sąsiada A (moglibyśmy wybrać F, ale jest dalej w alfabecie...) i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków. V (zbiór białych wierzchołków) jest niepusty, więc kontynuujemy, szukając sąsiadów wierzchołka z S.

Algorytm przeszukiwania w głąb - przykład



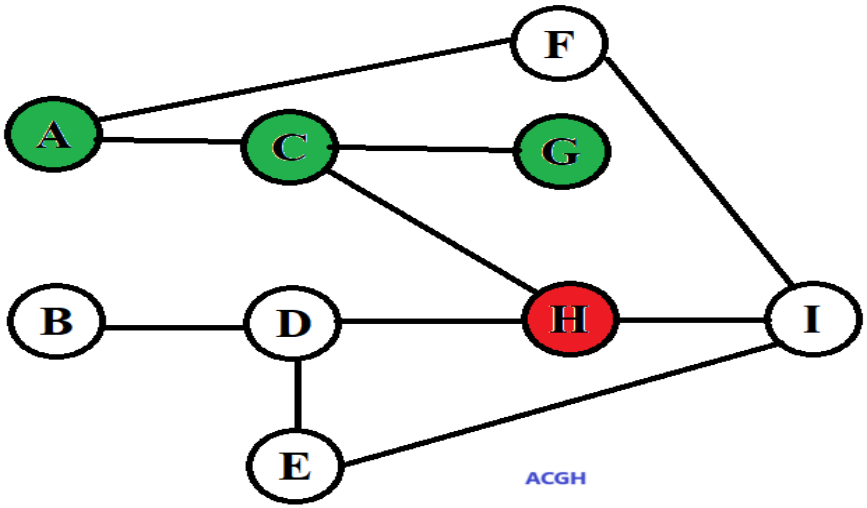
Wierzchołek C przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek G - nieodwiedzonego sąsiada C (moglibyśmy wybrać H, ale jest dalej w alfabecie...) i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków. V (zbiór białych wierzchołków) jest niepusty, więc kontynuujemy, szukając sąsiadów wierzchołka z S (w dalszych krokach nie będę tego powtarzać).

Algorytm przeszukiwania w głąb - przykład



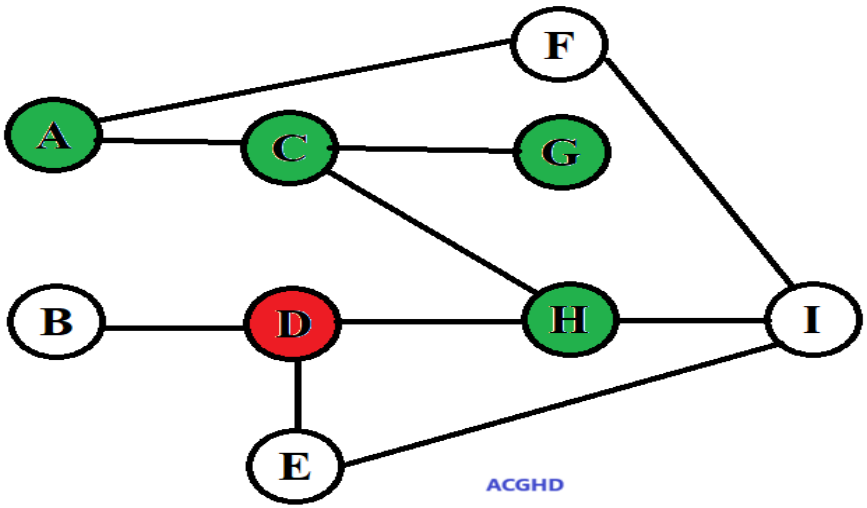
Wierzchołek G przenosimy z S do L (zaznaczamy na zielono). Nie ma on nieodwiedzonych sąsiadów, więc cofamy się do ostatniego wierzchołka, który ma nieodwiedzonych sąsiadów (C) i przywracamy go do zbioru S (nie usuwając go jednak z ciągu odwiedzonych wierzchołków!).

Algorytm przeszukiwania w głąb - przykład



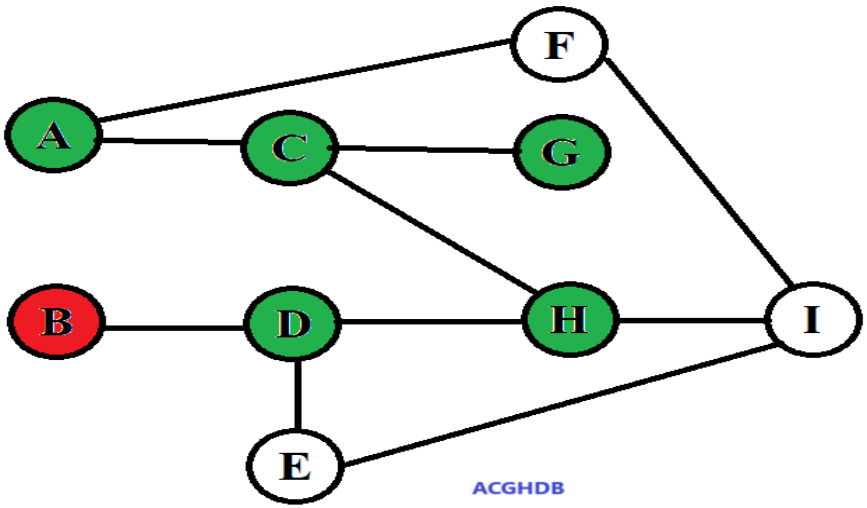
Wierzchołek C przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek H - ostatniego nieodwiedzonego sąsiada C i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków.

Algorytm przeszukiwania w głąb - przykład



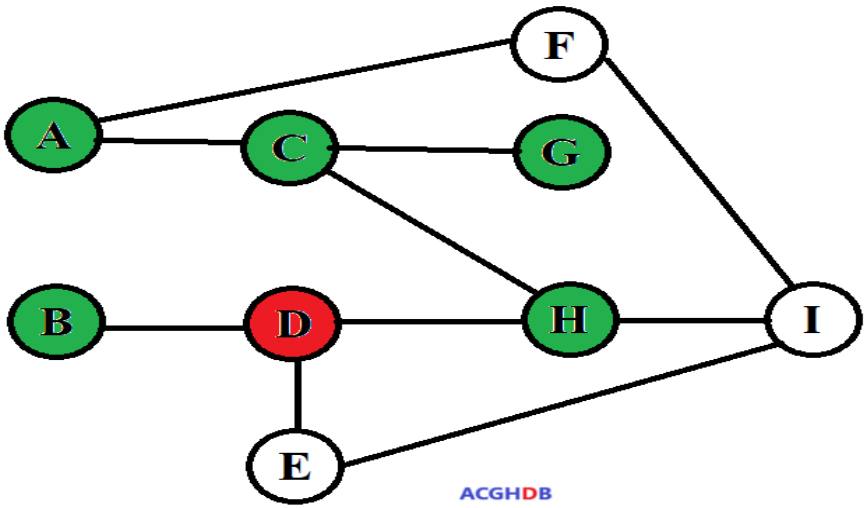
Wierzchołek H przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek D - najwcześniejszego w alfabecie nieodwiedzonego sąsiada H i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków.

Algorytm przeszukiwania w głąb - przykład



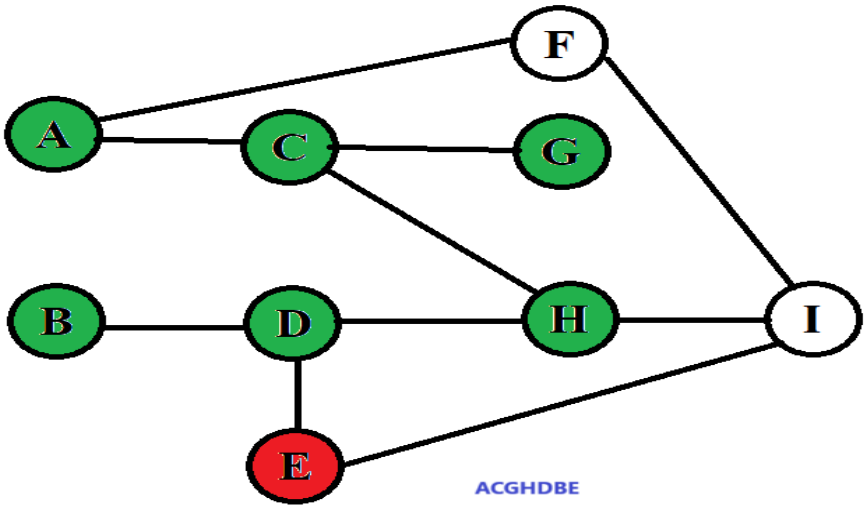
Wierzchołek D przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek B - najwcześniejszego w alfabecie nieodwiedzonego sąsiada D i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków.

Algorytm przeszukiwania w głąb - przykład



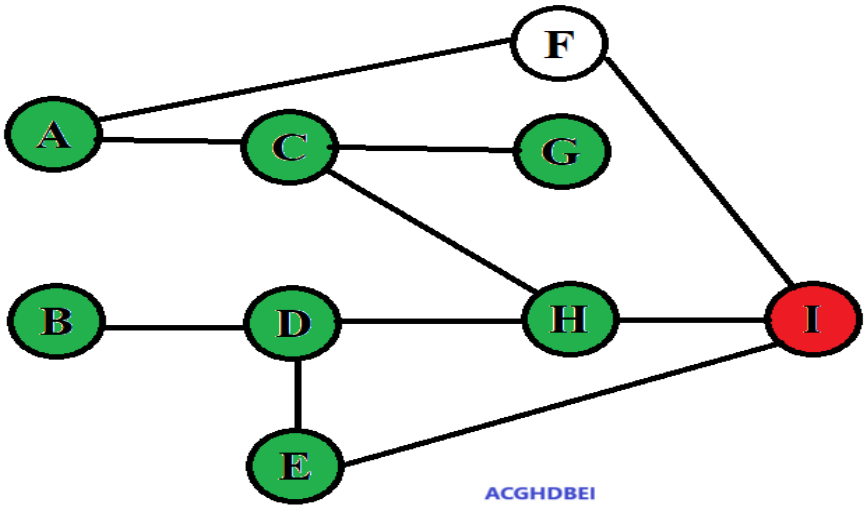
Wierzchołek B przenosimy z S do L (zaznaczamy na zielono). Nie ma on nieodwiedzonych sąsiadów, więc cofamy się do ostatniego wierzchołka, który ma nieodwiedzonych sąsiadów (D) i przywracamy go do zbioru S (nie usuwając go jednak z ciągu odwiedzonych wierzchołków!).

Algorytm przeszukiwania w głąb - przykład



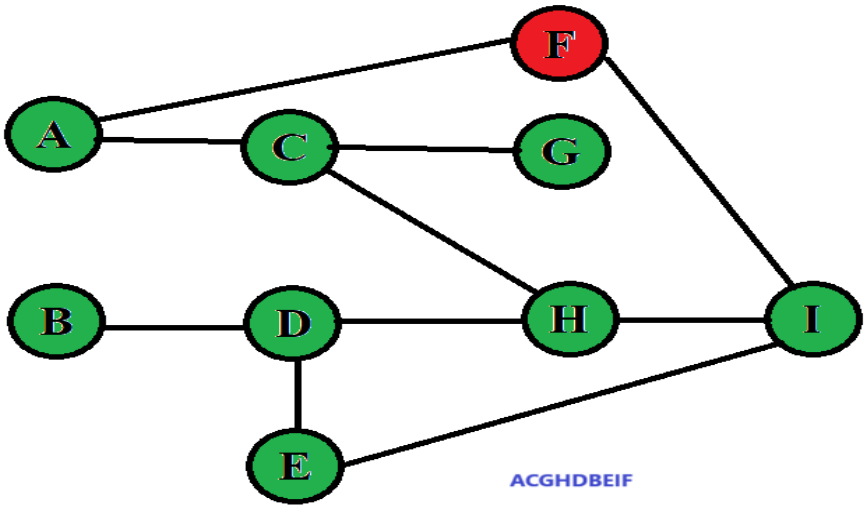
Wierzchołek D przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek E - ostatniego nieodwiedzonego sąsiada D i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków.

Algorytm przeszukiwania w głąb - przykład



Wierzchołek E przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek I - ostatniego nieodwiedzonego sąsiada E i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków.

Algorytm przeszukiwania w głąb - przykład



Wierzchołek I przenosimy z S do L (zaznaczamy na zielono). Wybieramy wierzchołek F - ostatniego nieodwiedzonego sąsiada I i przenosimy do zbioru S oraz dopisujemy do ciągu odwiedzonych wierzchołków. Ponieważ w tym momencie zbiór nieodwiedzonych wierzchołków V jest pusty, odwiedziliśmy wszystkie wierzchołki w kolejności zapisanego ciągu: ACGHDBEIF.

Dodatkowe informacje o przeszukiwaniu w głąb

- Dla grafu $G = (V, E)$ czas działania algorytmu przeszukiwania w głąb to również $O(|V| + |E|)$.
- Większość zastosowań, w szczególności badanie spójności, jest podobnych do stosowania algorytmu przeszukiwania wszerz.
- Przeszukiwanie w głąb jest szczególnie przydatne w przypadku drzew, gdzie drogi między wierzchołkami są jednoznaczne. Dzięki niemu tworzy się porządki wierzchołków zachowujące strukturę drzewa: prefiksowy, infiksowy i postfiksowy (wrócimy do nich przy teorii drzew).
- Algorytm został stworzony w XIX wieku przez Charlesa Tremaux jako narzędzie do rozwiązywania labiryntów.

Porównanie przeszukiwania wszerz i w głąb

- Obydwa algorytmy mają tę samą czasową złożoność obliczeniową.
- Algorytm przeszukiwania w głąb jest bardziej oszczędny pod względem wykorzystania pamięci: jako pomocnicza struktura danych wystarczy w nim stos złożony ze ścieżki od wierzchołka początkowego do badanego właśnie wierzchołka. Dla dużych grafów kolejka wierzchołków równoodległych od wierzchołka początkowego jest zwykle znacznie większa.

Porównanie przeszukiwania wszerz i w głąb

- Algorytm przeszukiwania wszerz jest zwykle bardziej wydajny w zagadnieniach, w których ważna jest odległość od wierzchołka początkowego (jak w znajdowaniu najkrótszej drogi) lub znalezienie jednego konkretnego wierzchołka - „rozwiązania problemu". Poszukując wygrywającej strategii rozegrania jakiejś pozycji w grze, w której jest wiele możliwości (np. szachach), przeszukiwanie wszerz typowo znajdzie ją szybciej.
- Szczególną przewagę przeszukiwanie wszerz ma w sytuacji, gdy szukamy danego wierzchołka w grafie, co do którego nie jesteśmy pewni, że jest skończony. Jeśli taki wierzchołek istnieje, algorytm wszerz znajdzie go w skończonym czasie, nawet jeśli sam graf jest nieskończony (a algorytm w głąb - niekoniecznie).