Podstawy wyszukiwania

Co to jest planowanie i wyszukiwanie?

Kiedy myślimy o tym, co czyni nas inteligentnymi, umiejętność planowania przed wykonaniem działań jest najważniejszym atrybutem. Przed wyruszeniem w podróż do innego kraju, przed rozpoczęciem nowego projektu, przed napisaniem funkcji w kodzie, następuje planowanie. Planowanie odbywa się na różnych poziomach szczegółowości w różnych kontekstach, aby dążyć do jak najlepszego wyniku podczas wykonywania zadań związanych z osiąganiem celów. Plany rzadko udają się idealnie w sposób, który wyobrażamy sobie na początku przedsięwzięcia. Żyjemy w świecie, w którym środowiska nieustannie się zmieniają, więc nie jest możliwe uwzględnienie wszystkich zmiennych i niewiadomych po drodze. Niezależnie od planu, od którego zaczynaliśmy, prawie zawsze odchodzimy ze względu na zmiany w przestrzeni problemowej. Musimy stworzyć nowy plan od naszego obecnego punktu, idąc naprzód, ale po tym, jak podejmiemy więcej kroków, pojawiają się nieoczekiwane zdarzenia, które wymagają kolejnej iteracji planowania, aby osiągnąć cele. W rezultacie ostateczny plan, który jest wykonywany, zwykle różni się od pierwotnego. Wyszukiwanie to sposób na pokierowanie planowaniem poprzez tworzenie kroków w planie. Na przykład, kiedy planujemy podróż, szukamy tras do obrania, oceniamy przystanki po drodze i ich ofertę oraz szukamy noclegów i zajęć, które odpowiadają naszym upodobaniom i budżetowi. W zależności od wyników tych poszukiwań plan się zmienia.

Załóżmy, że zdecydowaliśmy się na wycieczkę na plażę, która jest oddalona o 500 kilometrów, z dwoma przystankami: jednym w małym zoo, a drugim w pizzerii. W dniu przyjazdu będziemy spać w domku w pobliżu plaży i uczestniczyć w trzech zajęciach. Podróż do miejsca docelowego zajmie około 8 godzin. Za restauracją jedziemy również prywatną drogą na skróty, ale jest ona otwarta tylko do 14:00. Rozpoczynamy podróż i wszystko idzie zgodnie z planem. Zatrzymujemy się w małym zoo aby zobaczyć wspaniałe zwierzęta. Jedziemy dalej i zaczynamy być głodni; czas na postój w restauracji. Ale ku naszemu zdziwieniu restauracja niedawno wypadła z rynku. Musimy dostosować nasz plan i znaleźć inne miejsce do jedzenia, co wiąże się z poszukiwaniem pobliskiego lokalu według naszych upodobań i dostosowaniem naszego planu. Po chwili jeżdżenia po okolicy znajdujemy restaurację, jemy pizzę i ruszamy w drogę. Zbliżając się do prywatnej drogi skrótu, zdajemy sobie sprawę, że jest 14:20. Droga jest zamknięta; po raz kolejny musimy dostosować nasz plan. Szukamy objazdu i okazuje się, że wydłuży to naszą podróż o 120 kilometrów i będziemy musieli znaleźć nocleg w innym domku, zanim jeszcze dotrzemy na plażę. Szukamy miejsca do spania i wytyczamy nową trasę. Ze względu na stracony czas w miejscu docelowym możemy wziąć udział tylko w dwóch zajęciach. Plan został mocno skorygowany poprzez poszukiwanie różnych opcji, które zadowoliłyby nową sytuację, ale kończymy na wspaniałej przygodzie w drodze na plażę. Ten przykład pokazuje, jak wyszukiwanie jest wykorzystywane do planowania i wpływa na planowanie w kierunku pożądanych wyników. Ponieważ zmienia się otoczenie, nasze cele mogą się nieznacznie zmienić, a nasza droga do nich nieuchronnie wymaga dostosowania. Prawie nigdy nie można przewidzieć zmian w planach i należy je wprowadzać w razie potrzeby. Poszukiwanie obejmuje ocenę przyszłych stanów w kierunku celu w celu znalezienia optymalnej ścieżki stanów, aż do osiągnięcia celu. Ta część koncentruje się na różnych podejściach do wyszukiwania w zależności od różnych typów problemów. Wyszukiwanie to stare, ale potężne narzędzie do opracowywania inteligentnych algorytmów do rozwiązywania problemów.

Koszt obliczeń: przyczyna inteligentnych algorytmów

W programowaniu funkcje składają się z operacji, a ze względu na sposób działania tradycyjnych komputerów różne funkcje wykorzystują różne ilości czasu przetwarzania. Im więcej wymaganych obliczeń, tym droższa jest funkcja. Notacja Big O jest używana do opisu złożoności funkcji lub

algorytmu. Notacja Big O modeluje liczbę operacji wymaganych wraz ze wzrostem rozmiaru wejściowego. Oto kilka przykładów i związanych z nimi zawiłości:

• Pojedyncza operacja, która wyświetla Hello World - ta operacja jest pojedynczą operacją, więc koszt obliczeń wynosi O(1).

• Funkcja, która wykonuje iterację po liście i drukuje każdą pozycję na liście - liczba operacji zależy od liczby pozycji na liście. Koszt to O(n).

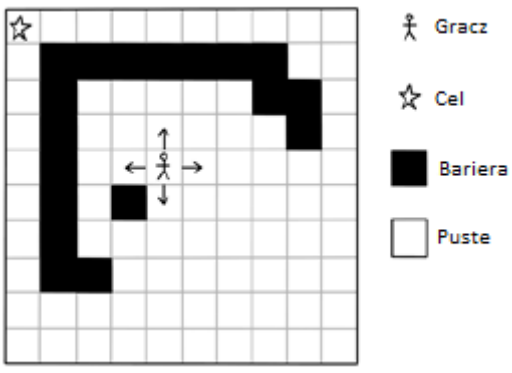
• Funkcja, która porównuje każdą pozycję na liście z każdą pozycją na innej liście - ta operacja kosztuje O (n^2).

Algorytmy, które wymagają operacji do zbadania wraz ze wzrostem wielkości danych wejściowych, są najgorsze; algorytmy wymagające stałej liczby operacji wraz ze wzrostem liczby wejść są lepsze.

Zrozumienie, że różne algorytmy mają różne koszty obliczeń, jest ważne, ponieważ rozwiązanie tego problemu jest głównym celem inteligentnych algorytmów, które dobrze i szybko rozwiązują problemy. Teoretycznie możemy rozwiązać prawie każdy problem, forsując każdą możliwą opcję, aż znajdziemy najlepszą, ale w rzeczywistości obliczenia mogą zająć godziny, a nawet lata, co sprawia, że jest to niewykonalne w rzeczywistych scenariuszach.

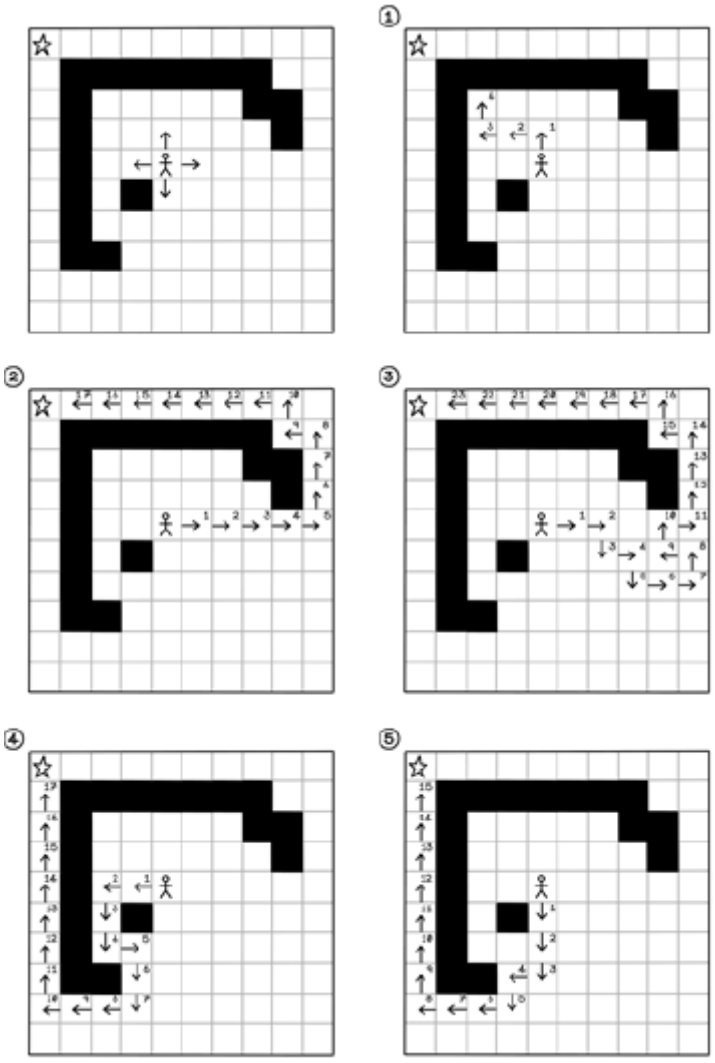
Problemy związane z algorytmami wyszukiwania

Prawie każdy problem, który wymaga podjęcia szeregu decyzji, można rozwiązać za pomocą algorytmów wyszukiwania. W zależności od problemu i rozmiaru przestrzeni poszukiwań można zastosować różne algorytmy, aby pomóc w ich rozwiązaniu. W zależności od wybranego algorytmu wyszukiwania i rozszerzenia zastosowanej konfiguracji, można znaleźć rozwiązanie optymalne lub najlepsze dostępne. Innymi słowy, znajdzie się dobre rozwiązanie, ale niekoniecznie będzie to najlepsze rozwiązanie. Kiedy mówimy o „dobrym rozwiązaniu" lub „rozwiązaniu optymalnym", mamy na myśli skuteczność rozwiązania w rozwiązywaniu problemu. Jednym ze scenariuszy, w którym przydatne są algorytmy wyszukiwania, jest utknięcie w labiryncie i próba znalezienia najkrótszej ścieżki do celu. Załóżmy, że znajdujemy się w kwadratowym labiryncie składającym się z 10 bloków na 10 bloków



Istnieje cel, do którego chcemy dotrzeć, i bariery, na które nie możemy wejść. Celem jest znalezienie drogi do celu, unikając barier, wykonując jak najmniejszą liczbę kroków, przesuwając się na północ, południe, wschód lub zachód. W tym przykładzie gracz nie może poruszać się po przekątnej. Jak znaleźć najkrótszą drogę do celu, unikając barier? Oceniając problem jako człowiek, możemy wypróbować każdą możliwość i policzyć ruchy. Korzystając z metody prób i błędów, możemy znaleźć najkrótsze

ścieżki, biorąc pod uwagę, że ten labirynt jest stosunkowo mały. Korzystając z przykładowego labiryntu, rysunek przedstawia niektóre możliwe ścieżki prowadzące do celu.



Patrząc na labirynt i licząc klocki w różnych kierunkach, możemy znaleźć kilka rozwiązań problemu. Podjęto pięć prób znalezienia czterech udanych rozwiązań z nieznanej liczby rozwiązań. Ręczne obliczenie wszystkich możliwych rozwiązań zajmie wiele wysiłku:

• Próba 1 nie jest prawidłowym rozwiązaniem. Wymagało to 4 działań, a cel nie został znaleziony.

• Próba 2 jest prawidłowym rozwiązaniem, podejmując 17 działań, aby znaleźć cel.

• Próba 3 to prawidłowe rozwiązanie, podejmowanie 23 działań, aby znaleźć cel.

• Próba 4 jest prawidłowym rozwiązaniem, podejmując 17 działań, aby znaleźć cel.

• Próba 5 jest najlepszym rozwiązaniem, podejmując 15 działań, aby znaleźć cel. Chociaż ta próba jest najlepsza, została znaleziona przypadkowo

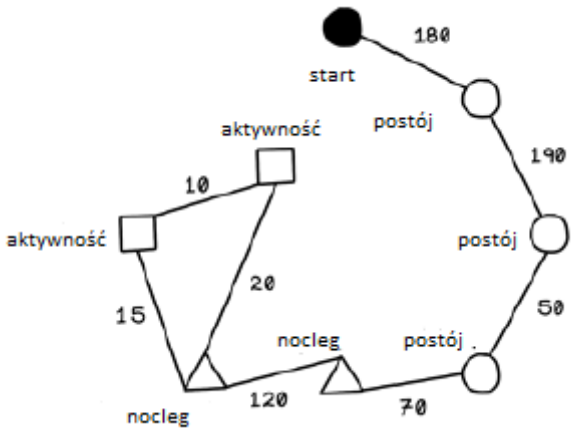
Gdyby labirynt był dużo większy, ręczne obliczenie najlepszej możliwej ścieżki zajęłoby ogromną ilość czasu. Algorytmy wyszukiwania mogą pomóc. Naszą mocą jako ludzi jest wizualne postrzeganie problemu, rozumienie go i znajdowanie rozwiązań na podstawie parametrów. Jako ludzie rozumiemy i interpretujemy dane i informacje w sposób abstrakcyjny. Komputer nie jest jeszcze w stanie

zrozumieć uogólnionych informacji w naturalnej formie, którą my robimy. Przestrzeń problemowa musi być przedstawiona w formie nadającej się do obliczeń i która może być przetwarzana za pomocą algorytmów wyszukiwania.

Wyjaśnijmy różnicę między danymi a informacjami. Dane to surowe fakty o czymś, a informacja to interpretacja tych faktów, która zapewnia wgląd w dane w określonej dziedzinie. Informacje wymagają kontekstu i przetwarzania danych, aby nadać znaczenie. Na przykład, każda indywidualna odległość przebyta w przykładzie labiryntu to dane, a suma całkowitej przebytej odległości to informacja. W zależności od perspektywy, poziomu szczegółowości i pożądanego rezultatu, klasyfikacja czegoś jako danych lub informacji może być subiektywna dla kontekstu i osoby lub zespołu. Struktury danych to pojęcia w informatyce używane do przedstawiania danych w sposób odpowiedni do wydajnego przetwarzania przez algorytmy. Struktura danych to abstrakcyjny typ danych składający się z danych i operacji zorganizowanych w określony sposób. Stosowana przez nas struktura danych zależy od kontekstu problemu i pożądanego celu. Przykładem struktury danych jest tablica, która jest po prostu zbiorem danych. Różne typy tablic mają różne właściwości, dzięki którym są wydajne w różnych celach. W zależności od używanego języka programowania tablica może pozwolić, aby każda wartość była innego typu lub wymagać, aby każda wartość była tego samego typu, lub tablica może nie zezwalać na zduplikowane wartości. Te różne typy tablic mają zwykle różne nazwy. Funkcje i ograniczenia różnych struktur danych umożliwiają również bardziej wydajne obliczenia. Inne struktury danych są przydatne podczas planowania i wyszukiwania. Drzewa i wykresy są idealne do przedstawiania danych w sposób, który mogą być wykorzystywane przez algorytmy wyszukiwania.

Wykresy: przedstawiające problemy wyszukiwania i rozwiązania

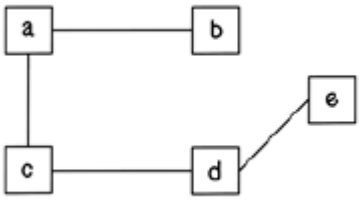
Wykres to struktura danych zawierająca kilka stanów z połączeniami między nimi. Każdy stan na wykresie nazywany jest węzłem (lub czasem wierzchołkiem), a połączenie między dwoma stanami nazywa się krawędzią. Wykresy pochodzą z teorii grafów w matematyce i są używane do modelowania relacji między obiektami. Grafy są użytecznymi strukturami danych, które są łatwe do zrozumienia dla ludzi ze względu na łatwość ich wizualnego przedstawienia, a także ich silny logiczny charakter, który idealnie nadaje się do przetwarzania za pomocą różnych algorytmów. Rysunek 2.10 to wykres wycieczki na plażę omówionej wcześniej.



Każdy przystanek jest węzłem na wykresie; każda krawędź między węzłami reprezentuje punkty przemieszczane między nimi; a wagi na każdej krawędzi wskazują przebytą odległość.

Przedstawianie wykresu jako konkretnej struktury danych

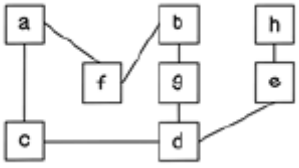
Wykres można przedstawić na kilka sposobów w celu wydajnego przetwarzania za pomocą algorytmów. W istocie wykres może być reprezentowany przez tablicę tablic, które wskazują relacje między węzłami, jak pokazano. Czasami przydatne jest posiadanie innej tablicy, która po prostu zawiera listę wszystkich węzłów na wykresie, tak aby nie trzeba było wywnioskować poszczególnych węzłów na podstawie relacji.



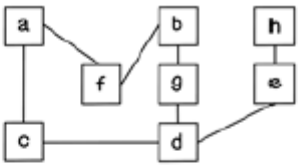

```
[  [a , b] ,
   [a , c] ,
   [c , d] ,
   [d , e] ]
```

ĆWICZENIE: PRZEDSTAWIANIE WYKRESU JAKO MATRYCY

Jak przedstawiłbyś następujący wykres za pomocą tablic krawędzi?



ROZWIĄZANIE: PRZEDSTAWIONO WYKRES JAKO MATRYCĘ




```
[ [a, c],
  [a, f],
  [b, g],
  [b, f],
  [c, d],
  [d, g],
  [d, e],
  [e, h] ]
```

Tablica krawędzi

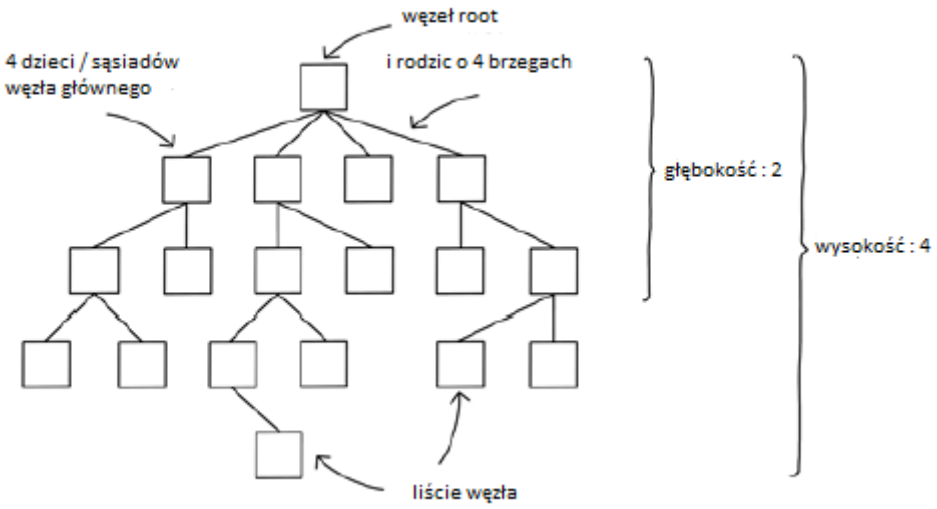
	a	b	c	d	e	f	g	h
a	0	0	1	0	0	1	0	0
b	0	0	0	0	0	1	1	0
c	1	0	0	1	0	0	0	0
d	0	0	1	0	1	0	1	0
e	0	0	0	1	0	0	0	1
f	1	1	0	0	0	0	0	0
g	0	1	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0

Macierz Sąsiedztwa

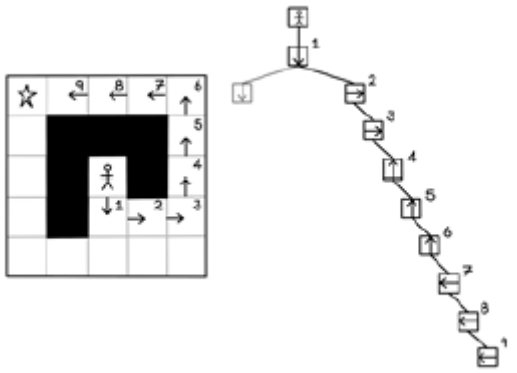
Drzewa: Konstrukcje betonowe używane do reprezentowania rozwiązań wyszukiwania

Drzewo to popularna struktura danych, która symuluje hierarchię wartości lub obiektów. Hierarchia to układ rzeczy, w którym pojedynczy obiekt jest powiązany z kilkoma innymi obiektami pod nim. Drzewo jest połączonym grafem acyklicznym - każdy węzeł ma krawędź do innego węzła i nie istnieją żadne cykle. W drzewie wartość lub obiekt reprezentowany w określonym punkcie nazywany jest węzłem. Drzewa zazwyczaj mają pojedynczy węzeł główny z zerem lub większą liczbą węzłów podrzędnych,

które mogą zawierać poddrzewa. Weźmy głęboki oddech i przejdźmy do terminologii. Gdy węzeł ma połączone węzły, węzeł główny nazywany jest węzłem nadrzędnym. Możesz zastosować to myślenie rekurencyjnie. Węzeł podrzędny może mieć własne węzły podrzędne, które mogą również zawierać poddrzewa. Każdy węzeł podrzędny ma jeden węzeł nadrzędny. Węzeł bez żadnych dzieci jest węzłem liścia. Drzewa mają również całkowitą wysokość. Poziom określonych węzłów nazywany jest głębokością. Terminologia używana w odniesieniu do członków rodziny jest często używana w pracy z drzewami. Pamiętaj o tej analogii, ponieważ pomoże ci ona połączyć pojęcia w strukturze danych drzewa. Zauważ, że na rysunku wysokość i głębokość są indeksowane od 0 od węzła głównego.



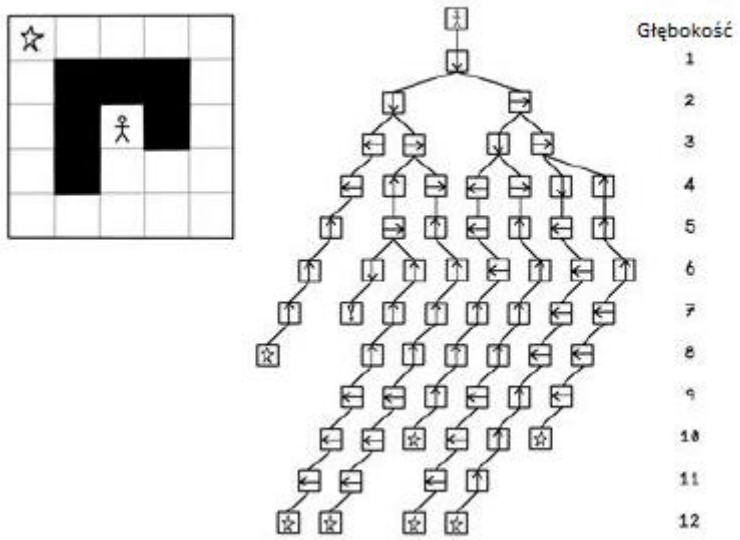
Najwyższy węzeł w drzewie nazywany jest węzłem głównym. Węzeł bezpośrednio połączony z co najmniej jednym innym węzłem nazywany jest węzłem nadrzędnym. Węzły połączone z węzłem nadrzędnym nazywane są węzłami potomnymi lub sąsiadami. Węzły podłączone do tego samego węzła nadrzędnego nazywane są rodzeństwem. Połączenie między dwoma węzłami nazywa się krawędzią. Ścieżka to sekwencja węzłów i krawędzi łączących węzły, które nie są bezpośrednio połączone. Węzeł połączony z innym węzłem poprzez podążanie ścieżką z dala od węzła głównego nazywany jest potomkiem, a węzeł połączony z innym węzłem przez podążanie ścieżką w kierunku węzła głównego nazywany jest przodkiem. Węzeł bez dzieci nazywany jest węzłem liścia. Termin stopień jest używany do opisania liczby dzieci, które ma węzeł; dlatego węzeł liścia ma stopień zerowy. Rysunek przedstawia ścieżkę od punktu początkowego do celu dla problemu labiryntu. Ta ścieżka zawiera dziewięć węzłów, które reprezentują różne ruchy wykonywane w labiryncie.



Drzewa są podstawową strukturą danych dla algorytmów wyszukiwania, nad którymi będziemy się teraz zastanawiać. Algorytmy sortowania są również przydatne w skuteczniejszym rozwiązywaniu pewnych problemów i rozwiązaniach obliczeniowych. Jeśli chcesz dowiedzieć się więcej o algorytmach sortowania, zapoznaj się z Grokking Algorithms autorstwa Manning Publications.

Niedoinformowane poszukiwanie: ślepe szukanie rozwiązań

Wyszukiwanie niedoinformowane jest również znane jako wyszukiwanie niekierowane, poszukiwanie ślepe lub wyszukiwanie siłowe. Niedoinformowane algorytmy wyszukiwania nie mają żadnych dodatkowych informacji o domenie problemu poza reprezentacją problemu, którą zwykle jest drzewo. Pomyśl o odkrywaniu rzeczy, których chcesz się nauczyć. Niektórzy ludzie mogą spojrzeć na szeroki wachlarz różnych tematów i nauczyć się podstaw każdego z nich, podczas gdy inni mogą wybrać jeden wąski temat i dogłębnie zbadać jego podtematy. Na tym właśnie polega odpowiednio przeszukiwanie wszerz (BFS) i przeszukiwanie w głąb (DFS). Przeszukiwanie w głąb bada określoną ścieżkę od początku do znalezienia celu na największej głębokości. Przeszukiwanie wszerz bada wszystkie opcje na określonej głębokości, zanim przejdzie do opcji głębiej w drzewie. Rozważ scenariusz labiryntu. Próbując znaleźć optymalną ścieżkę do celu, przyjmij następujące proste ograniczenie, aby zapobiec utknięciu w niekończącej się pętli i zapobiec cyklom w naszym drzewie. Ponieważ niedoinformowane algorytmy próbują każdej możliwej opcji w każdym węźle, utworzenie cyklu spowoduje katastrofalne niepowodzenie algorytmu. Gracz nie może wejść na blok, który zajmował wcześniej. To ograniczenie zapobiega cyklom na drodze do celu w naszym scenariuszu. Ale to ograniczenie wprowadzi problemy, jeśli w innym labiryncie z różnymi ograniczeniami lub regułami, w celu uzyskania optymalnego rozwiązania wymagane jest więcej niż jednokrotne przejście do poprzednio zajmowanego bloku. Na rysunku 2.15 przedstawiono wszystkie możliwe ścieżki w drzewie, aby wyróżnić różne dostępne opcje.



Ważne jest, aby zrozumieć, że w tym małym labiryncie przedstawienie wszystkich możliwości jest wykonalne. Cały sens algorytmów wyszukiwania polega jednak na tym, aby przeszukiwać lub generować te drzewa iteracyjnie, ponieważ generowanie całego drzewa możliwości z góry jest nieefektywne ze względu na to ,że są kosztowne obliczeniowo. To drzewo zawiera siedem ścieżek, które prowadzą do celu i jedną ścieżkę, która prowadzi do nieprawidłowego rozwiązania, biorąc pod uwagę ograniczenie polegające na nieprzenoszeniu się do poprzednio zajmowanych bloków. Należy również zauważyć, że termin odwiedzanie jest używany do określenia różnych rzeczy. Gracz odwiedza bloki w labiryncie. Algorytm odwiedza również węzły w drzewie. Kolejność wyborów wpłynie na

kolejność odwiedzanych węzłów w drzewie. W przykładzie labiryntu kolejność priorytetów ruchu jest północ, południe, wschód, a następnie zachód. Teraz, gdy rozumiemy idee kryjące się za drzewami i przykładem labiryntu, przyjrzyjmy się, jak algorytmy wyszukiwania mogą generować drzewa, które szukają ścieżek do celu.

Przeszukiwanie wszerz: Rozejrzyj się szeroko, zanim zajrzysz głęboko

Przeszukiwanie wszerz to algorytm używany do przechodzenia lub generowania drzewa. Algorytm ten zaczyna się w określonym węźle, zwanym korzeniem, i bada każdy węzeł na tej głębokości przed zbadaniem następnej głębokości węzłów. Zasadniczo odwiedza wszystkie elementy potomne węzłów na określonej głębokości przed odwiedzeniem następnej głębokości dziecka, aż znajdzie węzeł liścia celu. Algorytm wyszukiwania wszerz najlepiej jest implementować przy użyciu kolejki pierwszy na wejściu, pierwszy na wyjściu, w której przetwarzane są bieżące głębokości węzłów, a ich elementy podrzędne są umieszczane w kolejce do przetworzenia później. Taka kolejność przetwarzania jest dokładnie tym, czego wymagamy przy implementacji tego algorytmu. Oto kilka uwag i dodatkowych uwag dotyczących każdego etapu procesu:

1. Umieść węzeł główny w kolejce. Algorytm wyszukiwania wszerz najlepiej jest zaimplementować w kolejce. Obiekty są przetwarzane w kolejności, w jakiej są dodawane do kolejki. Ten proces jest również znany jako FIFO (First In, First Out). Pierwszym krokiem jest dodanie węzła głównego do kolejki. Ten węzeł będzie reprezentował początkową pozycję gracza na mapie.

2. Oznacz węzeł główny jako odwiedzony. Teraz, gdy węzeł główny został dodany do kolejki w celu przetworzenia, jest oznaczony jako odwiedzony, aby uniemożliwić ponowne odwiedzanie go bez powodu.

3. Czy kolejka jest pusta? Jeśli kolejka jest pusta (wszystkie węzły zostały przetworzone po wielu iteracjach), a ścieżka nie została zwrócona w kroku 12 algorytmu, nie ma ścieżki do celu. Jeśli w kolejce nadal znajdują się węzły, algorytm może kontynuować wyszukiwanie, aby znaleźć cel.

4. Powrót Brak ścieżki do celu. Ta wiadomość jest jedynym możliwym wyjściem z algorytmu, jeśli nie istnieje ścieżka do celu.

5. Usuń z kolejki węzeł jako bieżący węzeł. Wyciągając kolejny obiekt z kolejki i ustawiając go jako aktualnie interesujący nas węzeł, możemy zbadać jego możliwości. Po uruchomieniu algorytmu bieżący węzeł będzie węzłem głównym.

6. Pobierz następnego sąsiada bieżącego węzła. Ten krok polega na uzyskaniu następnego możliwego ruchu w labiryncie z bieżącej pozycji poprzez odniesienie do labiryntu i określenie, czy możliwy jest ruch na północ, południe, wschód lub zachód.

7. Czy sąsiad jest odwiedzany? Jeśli obecny sąsiad nie był odwiedzany, nie został jeszcze zbadany i można go teraz przetworzyć.

8. Oznacz sąsiada jako odwiedzonego. Ten krok wskazuje, że ten węzeł sąsiedni został odwiedzony.

9. Ustaw aktualny węzeł jako rodzica sąsiada. Ustaw węzeł początkowy jako rodzica bieżącego sąsiada. Ten krok jest ważny dla śledzenia ścieżki od bieżącego sąsiada do węzła głównego. Z punktu widzenia mapy punkt początkowy to pozycja, z której przeniósł się gracz, a bieżący sąsiad to pozycja, na którą przeniósł się gracz.

10. Umieść sąsiada w kolejce. Węzeł sąsiedni jest umieszczony w kolejce w celu późniejszego zbadania jego elementów podrzędnych. Ten mechanizm kolejkowania umożliwia przetwarzanie węzłów z każdej głębokości w tej kolejności.

11. Czy cel został osiągnięty? Ten krok określa, czy bieżący sąsiad zawiera cel, którego szuka algorytm.

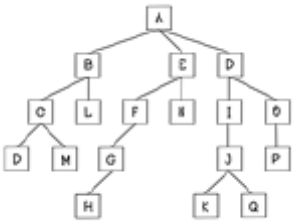
12. Ścieżka zwrotna za pomocą sąsiada. Odwołując się do rodzica sąsiedniego węzła, następnie do rodzica tego węzła itd., Zostanie opisana ścieżka od celu do katalogu głównego. Węzeł główny będzie węzłem bez rodzica.

13. Bieżący ma następnego sąsiada. Jeśli aktualny węzeł ma więcej możliwych ruchów do wykonania w labiryncie, przejdź do kroku 6 dla tego ruchu.

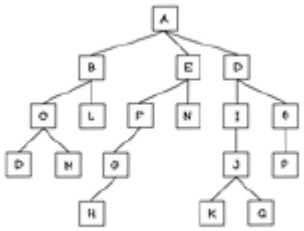
Zobaczmy, jak to wyglądałoby w prostym drzewie. Zauważ, że w miarę eksploracji drzewa i dodawania węzłów do kolejki FIFO, węzły są przetwarzane w żądanej kolejności poprzez wykorzystanie kolejki.

ĆWICZENIE:

Jaka byłaby kolejność odwiedzin przy użyciu wyszukiwania wszerz dla następującego drzewa?



ROZWIĄZANIE:

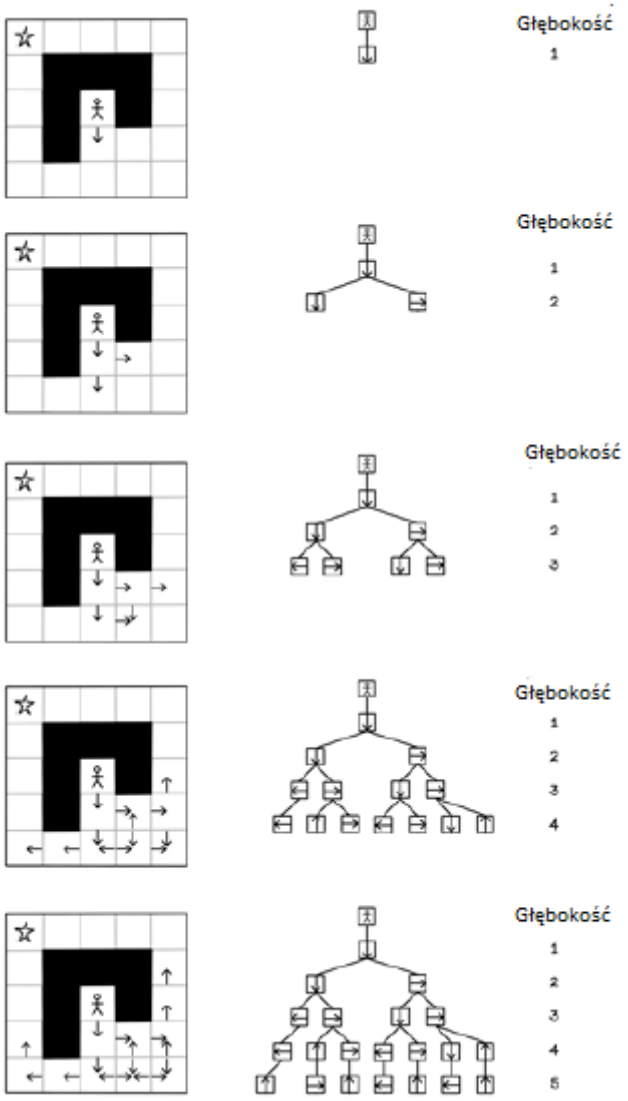


Kolejność wyszukiwania wszerz
A, B, E, D, C, L, F, N, I, O, D, M, G, J, P, H, K, Q

W przykładzie labiryntu algorytm musi zrozumieć aktualną pozycję gracza w labiryncie, ocenić wszystkie możliwe wybory ruchu i powtórzyć tę logikę dla każdego wyboru ruchu, aż do osiągnięcia celu. W ten sposób algorytm generuje drzewo z jedną ścieżką do celu. Ważne jest, aby zrozumieć, że procesy odwiedzania węzłów w drzewie służą do generowania węzłów w drzewie. Po prostu wyszukujemy powiązane węzły za pomocą mechanizmu. Każda ścieżka do celu składa się z serii ruchów prowadzących do celu. Liczba ruchów na ścieżce to odległość do osiągnięcia celu na tej ścieżce, który nazwiemy kosztem. Liczba ruchów jest również równa liczbie węzłów odwiedzonych na ścieżce, od węzła głównego do węzła-liścia zawierającego cel. Algorytm przesuwa się w dół drzewo po głębokości, aż znajdzie cel; następnie zwraca pierwszą ścieżkę, która doprowadziła go do celu jako rozwiązanie. Może istnieć bardziej optymalna ścieżka do celu, ale ponieważ wyszukiwanie wszerz jest niedoinformowane, nie ma gwarancji, że go znajdzie.

UWAGA : W przykładzie labiryntu wszystkie użyte algorytmy wyszukiwania kończą się, gdy znajdą rozwiązanie celu. Możliwe jest zezwolenie tym algorytmom na znalezienie wielu rozwiązań z niewielkimi zmianami w każdym algorytmie, ale najlepsze przypadki użycia algorytmów wyszukiwania znajdują jeden cel, ponieważ badanie całego drzewa możliwości jest często zbyt kosztowne.

Rysunek poniższy przedstawia generowanie drzewa za pomocą ruchów w labiryncie. Ponieważ drzewo jest generowane za pomocą przeszukiwania wszerz, każda głębokość jest generowana do końca przed spojrzeniem na następną głębokość.



Przeszukiwanie w głąb: Spojrzenie głęboko, zanim spojrzysz szeroko

Przeszukiwanie w głąb to kolejny algorytm używany do przechodzenia przez drzewo lub generowania węzłów i ścieżek w drzewie. Algorytm ten zaczyna się w określonym węźle i bada ścieżki połączonych węzłów pierwszego dziecka, robiąc to rekurencyjnie, aż dotrze do najdalszego węzła liścia przed cofnięciem i zbadaniem innych ścieżek do węzłów liści przez inne odwiedzone węzły potomne. Przyjrzyjmy się działaniu algorytmu wyszukiwania w pierwszej kolejności:

1. Dodaj węzeł główny do stosu. Algorytm przeszukiwania w głąb można zaimplementować za pomocą stosu, w którym ostatni dodany obiekt jest przetwarzany jako pierwszy. Ten proces jest znany jako ostatni w, pierwszy na wyjściu (LIFO). Pierwszym krokiem jest dodanie węzła głównego do stosu.

2. Czy stos jest pusty? Jeśli stos jest pusty i żadna ścieżka nie została zwrócona w kroku 8 algorytmu, nie ma ścieżki do celu. Jeśli w stosie nadal znajdują się węzły, algorytm może kontynuować wyszukiwanie, aby znaleźć cel.

3. Powrót Brak ścieżki do celu. Ten powrót jest jedynym możliwym wyjściem z algorytmu, jeśli nie istnieje ścieżka do celu.

4. Zdejmij węzeł ze stosu jako bieżący węzeł. Wyciągając kolejny obiekt ze stosu i ustawiając go jako aktualnie interesujący węzeł, możemy zbadać jego możliwości.

5. Czy odwiedzono aktualny węzeł? Jeśli bieżący węzeł nie był odwiedzany, nie został jeszcze zbadany i można go teraz przetworzyć.

6. Oznacz aktualny węzeł jako odwiedzony. Ten krok wskazuje, że ten węzeł został odwiedzony, aby zapobiec niepotrzebnemu powtarzaniu go.

7. Czy cel został osiągnięty? Ten krok określa, czy bieżący sąsiad zawiera cel, którego szuka algorytm.

8. Ścieżka zwrotna przy użyciu bieżącego węzła. Odwołując się do rodzica bieżącego węzła, następnie do rodzica tego węzła itd., Opisana jest ścieżka od celu do źródła. Węzeł główny będzie węzłem bez rodzica.

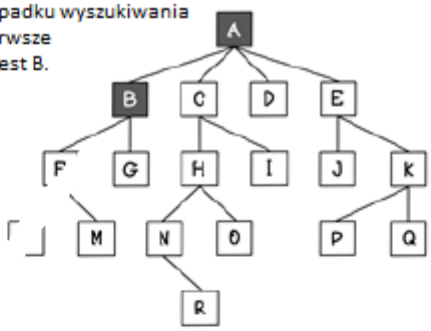
9. Bieżący ma następnego sąsiada. Jeśli bieżący węzeł ma więcej możliwych ruchów do wykonania w labiryncie, ruch ten można dodać do stosu, który ma zostać przetworzony. W przeciwnym razie algorytm może przejść do kroku 2, w którym następny obiekt w stosie może zostać przetworzony, jeśli stos nie jest pusty. Charakter stosu LIFO umożliwia algorytmowi przetwarzanie wszystkich węzłów do głębokości węzła liścia przed wykonaniem powrotu w celu odwiedzenia innych elementów podrzędnych węzła głównego.

10. Ustaw aktualny węzeł jako rodzica sąsiada. Ustaw węzeł początkowy jako rodzica bieżącego sąsiada. Ten krok jest ważny dla śledzenia ścieżki od bieżącego sąsiada do węzła głównego. Z punktu widzenia mapy punkt początkowy to pozycja, z której przeniósł się gracz, a bieżący sąsiad to pozycja, na którą przeniósł się gracz.

11. Dodaj sąsiada do stosu. Węzeł sąsiedni jest dodawany do stosu, aby jego elementy podrzędne mogły zostać później zbadane. Ponownie, ten mechanizm stertowania umożliwia przetwarzanie węzłów do największej głębokości przed przetwarzaniem sąsiadów na płytkich głębokościach.

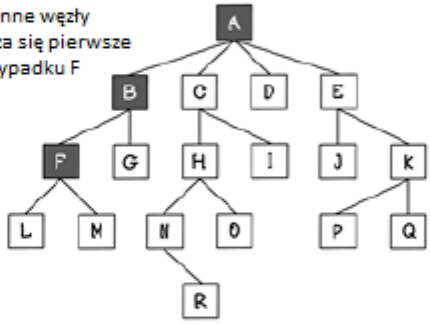
Dwa poniższe rysunki przedstawiają, w jaki sposób stos LIFO jest używany do odwiedzania węzłów w kolejności wymaganej przez przeszukiwanie w głąb. Zauważ, że węzły są wypychane i usuwane ze stosu wraz z postępem głębokości odwiedzanych węzłów. Push to termin używany do dodawania obiektów do stosu, a termin pop jest używany do usuwania najwyższego obiektu ze stosu.

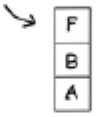
Podobnie jak w przypadku wyszukiwania wszerz, odwiedź pierwsze dziecko węzła A. To jest B.



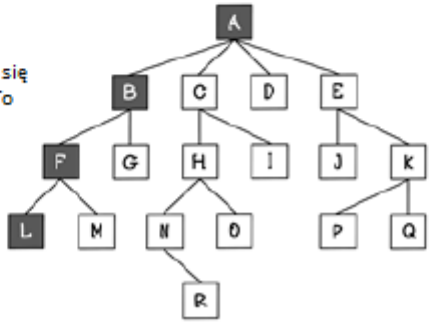
Sekwencja przetwarzania stosu

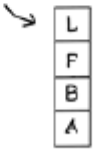
Zamiast odwiedzać inne węzły potomne A. Odwiedza się pierwsze dziecko B. W tym przypadku F



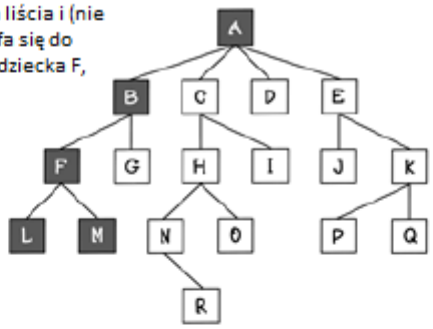


Ponownie odwiedza się pierwsze dziecko F. To jest L.

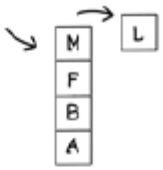




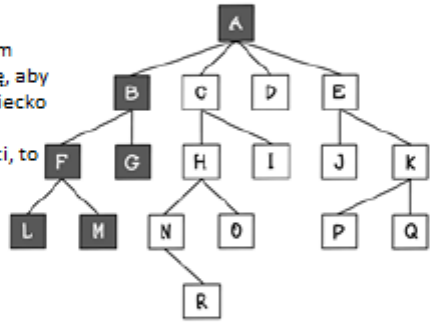
Ponieważ L jest węzłem liścia i (nie ma dzieci), algorytm cofa się do odwiedzin następnego dziecka F, to jest M

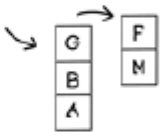


Sekwencja przetwarzania stosu

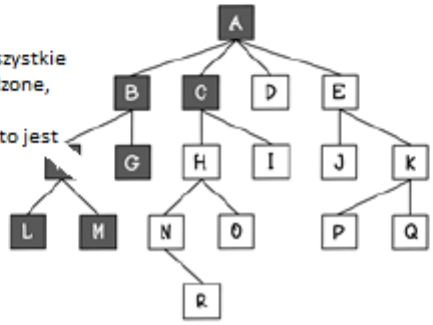


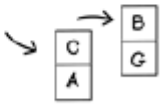
Ponieważ M jest węzłem liścia, algorytm cofa się, aby odwiedzić następne dziecko B, ponieważ F nie ma nieodwiedzonych dzieci, to jest G



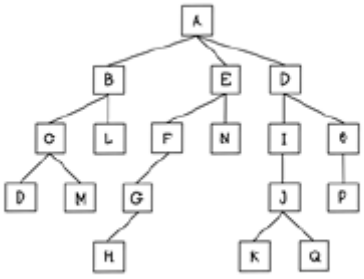


Wreszcie, ponieważ wszystkie dzieci B zostały odwiedzone, algorytm cofa się do następnego sąsiada A, to jest C.

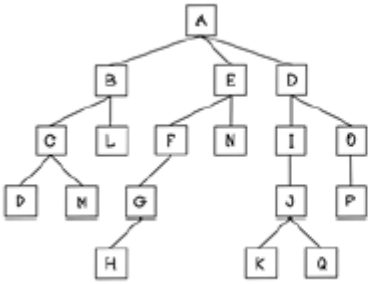




ĆWICZENIE: Jaka byłaby kolejność odwiedzin dogłębnego wyszukiwania w następującym drzewie?



ROZWIĄZANIE:



Depth-first Search Order:
A, B, C, D, M, L, E, F, G, H, N, D, I, J, K, Q, O, P

Ważne jest, aby zrozumieć, że kolejność potomków ma istotne znaczenie podczas korzystania z wyszukiwania w głąb, ponieważ algorytm bada pierwsze dziecko, dopóki nie znajdzie węzłów liści przed cofnięciem. W przykładzie labiryntu kolejność ruchu (północ, południe, wschód i zachód) wpływa na ścieżkę do celu znalezionego przez algorytm. Zmiana kolejności spowoduje inne rozwiązanie. Widelce przedstawione na rysunku nie mają znaczenia; liczy się kolejność wyboru ruchu w naszym przykładzie labiryntu.