Dokumentacja kodu - Algorytm Węgierski

Wprowadzenie Algorytm Węgierski, znany również jako metoda przypisania, jest algorytmem optymalnego przypisania węzłów między dwoma zbiorami, minimalizującym koszt całkowity przypisania. Jest używany w różnych dziedzinach, takich jak logistyka, planowanie tras, zarządzanie zadaniami, itp. Algorytm jest oparty na heurystyce i stosuje techniki pokrycia wierszy i kolumn w macierzy kosztów. Struktury danych Algorytm Węgierski korzysta z kilku struktur danych:

MineTypes - typ wyliczeniowy reprezentujący różne typy kopalń.

ZeroMark - typ wyliczeniowy reprezentujący oznaczenia zer w macierzy kosztów.

mine\_t - struktura reprezentująca kopalnię, zawierająca typ kopalni oraz jej pozycję.

dwarf\_t - struktura reprezentująca krasnoluda, zawierająca pozycję krasnoluda oraz mapę umiejętności krasnoluda w różnych typach kopalń.

Hungarian - główna klasa implementująca algorytm Węgierski.

Metody klasy Hungarian

subtract\_min\_col() - Metoda odjęcia wartości minimalnej dla każdej kolumny w macierzy kosztów.

subtract\_min\_row() - Metoda odjęcia wartości minimalnej dla każdego wiersza w macierzy kosztów.

mark\_starred\_zeros() - Metoda oznaczania zer w macierzy kosztów gwiazdkami. check\_cover\_is\_min() - Metoda sprawdzająca, czy pokrycie wierszy i kolumn jest minimalne.

exists\_uncovered\_zero(size\_t& rw, size\_t& cl) - Metoda sprawdzająca, czy istnieje nieoznaczone zero w macierzy kosztów i zwracająca jego indeksy.

zero\_in\_row(size\_t rw, size\_t& cl, ZeroMark mark\_type) - Metoda sprawdzająca, czy w danym wierszu istnieje zero o określonym oznaczeniu i zwracająca jego kolumnę. zero\_in\_col(size\_t& rw, size\_t cl, ZeroMark mark\_type) - Metoda sprawdzająca, czy w danej kolumnie istnieje zero o określonym oznaczeniu i zwracająca jego wiersz. augment\_path() - Metoda modyfikująca ścieżkę poprzez oznaczenie zer gwiazdkami i apostrofami.

clear\_covers() - Metoda czyszcząca oznaczenia pokrycia wierszy i kolumn. erase\_primes() - Metoda usuwająca oznaczenia apostrofów. F

ind\_min\_uncovered() - Metoda znajdująca minimalną wartość spośród niepokrytych zer w macierzy kosztów.

step\_four() - Krok 4 algorytmu, w którym sprawdzane są niepokryte zera w macierzy kosztów i podejmowane odpowiednie akcje.

step\_five() - Krok 5 algorytmu, w którym ścieżka jest rozszerzana na podstawie oznaczeń gwiazdkowych i apostrofowych.

update\_costs\_with\_min() - Krok 6 algorytmu, w którym aktualizowane są koszty z wykorzystaniem minimalnej wartości spośród niepokrytych zer. C

alc\_cost() - Metoda obliczająca całkowity koszt przypisania na podstawie oznaczeń gwiazdkowych w macierzy kosztów.

cost\_matrix\_to\_vectors() - Metoda konwertująca macierz kosztów na wektor wektorów, która jest przydatna do dalszej analizy i prezentacji danych.

get\_assignment() - Metoda zwracająca przypisanie w postaci pary indeksów, które reprezentują przypisanie węzłów między dwoma zbiorami.

create\_matrix() - Metoda pomocnicza do tworzenia macierzy o określonym rozmiarze i wartościach początkowych.

Metody pomocnicze

distance() - Metoda statyczna obliczająca odległość między dwoma punktami na podstawie ich współrzędnych.

Przepływ algorytmu Tworzenie instancji klasy Hungarian na podstawie danych wejściowych (krasnalów i kopalń). Wykonanie kroków 1-6 algorytmu w celu znalezienia optymalnego przypisania. Obliczenie całkowitego kosztu przypisania za pomocą metody calc\_cost(). Konwersja macierzy kosztów na wektor wektorów za pomocą metody cost\_matrix\_to\_vectors(). Zwrócenie przypisania za pomocą metody get\_assignment().

Podsumowanie Algorytm Węgierski jest używany do rozwiązania problemów optymalnego przypisania między dwoma zbiorami. Wykorzystuje on techniki pokrycia wierszy i kolumn oraz modyfikuje koszty w celu znalezienia optymalnego przypisania. Implementacja algorytmu w klasie Hungarian umożliwia wygodne korzystanie z niego i uzyskiwanie wyników przypisania oraz kosztów.

Dokumentacja kodu - Szukanie najkrótszego cyklu

Wprowadzenie Kod przedstawia implementację algorytmu szukającego najkrótszego cyklu w grafie kopalń. Algorytm oblicza sumę odległości między kolejnymi kopalniami na trasie cyklu i zwraca tę sumę jako wynik. Wykorzystuje on macierz odległości między kopalniami, obliczaną na podstawie ich współrzędnych.

Struktury danych Kod korzysta z następujących struktur danych:

mine\_t - struktura reprezentująca kopalnię, zawierająca jej pozycję.

distances - macierz przechowująca odległości między kopalniami.

route - wektor przechowujący indeksy kopalń na trasie cyklu.

visited - wektor przechowujący informację o odwiedzonych kopalniach.

Metoda find\_shortest\_cycle

Inicjalizacja zmiennych:

sum - suma odległości cyklu (początkowo równa 0).

counter - licznik (początkowo równy 0).

i i j - indeksy kopalni (początkowo równy 0). ores\_no - liczba kopalń.

min - minimalna odległość (początkowo ustawiona na maksymalną wartość).

route - wektor przechowujący indeksy kopalni na trasie cyklu.

visited - wektor przechowujący informację o odwiedzonych kopalniach (początkowo wszystkie ustawione na false).

Obliczenie macierzy odległości: Stworzenie macierzy distances o rozmiarze ores\_no x ores\_no. Iteracja po kopalniach i obliczenie odległości między nimi na podstawie współrzędnych. Przypisanie wartości symetrycznej do macierzy distances.

Przeszukiwanie cyklu: Dopóki nie przekroczone są granice liczby kopalń: Sprawdzenie, czy istnieje kopalnia, która nie została odwiedzona i ma mniejszą odległość od aktualnej minimalnej odległości. Aktualizacja minimalnej odległości i zapisanie indeksu kopalni na trasie. Inkrementacja indeksu kopalni. Jeśli osiągnięto koniec kolumny, dodanie minimalnej odległości do sumy, zresetowanie minimalnej odległości, przejście na początek kolumny, przejście do kolejnej kopalni na trasie, oznaczenie kopalni jako odwiedzonej i inkrementacja licznika. Ustalenie ostatniego indeksu kopalni na trasie.

4.Kontynuacja dokumentacji kodu - Szukanie najkrótszego cyklu Znalezienie minimalnej odległości dla ostatniej kopalni na trasie: Iteracja po kopalniach i sprawdzenie, czy istnieje kopalnia o mniejszej odległości niż aktualna minimalna odległość od ostatniej kopalni na trasie. Aktualizacja minimalnej odległości i zapisanie indeksu kopalni. Dodanie ostatniej minimalnej odległości do sumy. Wyświetlenie indeksów i pozycji kopalni na trasie: Iteracja po indeksach kopalni na trasie i wypisanie indeksu oraz pozycji kopalni na trasie. Zwrócenie sumy odległości cyklu jako wynik.

Podsumowanie Kod implementuje algorytm szukający najkrótszego cyklu w grafie kopalń. Na podstawie macierzy odległości między kopalniami oblicza sumę odległości między kolejnymi kopalniami na trasie cyklu. Algorytm wykorzystuje technikę przeszukiwania cyklu w grafie w celu znalezienia optymalnej trasy.

Dokumentacja kodu - Szukanie najkrótszego cyklu w grafie kopalń

Wprowadzenie Kod przedstawia główną funkcję main, w której jest realizowany proces szukania najkrótszego cyklu w grafie kopalń. Wykorzystuje on różne algorytmy i struktury danych zaimplementowane w innych plikach nagłówkowych, takich jak HungarianClass/hungarian.h, hamilton.h i brute-force.h. Kod zakłada istnienie klas Hungarian i Hamilton, które obsługują przypisanie kopalni do krasnoludów oraz szukanie najkrótszego cyklu.

Struktury danych Kod korzysta z następujących struktur danych:

mine\_t - struktura reprezentująca kopalnię, zawierająca jej typ (MineTypes) i pozycję. dwarf\_t - struktura reprezentująca krasnoluda, zawierająca jego pozycję i umiejętności w różnych typach kopalń.

Hungarian - klasa obsługująca przypisanie kopalni do krasnoludów.

Hamilton - klasa obsługująca szukanie najkrótszego cyklu.

brute\_force - funkcja wykonująca brute-force w celu znalezienia optymalnego przypisania.

Metoda main

Inicjalizacja danych:

Stworzenie wektora ores zawierającego kopalnie z ich typami i pozycjami.

Stworzenie wektora dwarfs zawierającego krasnoludów z ich pozycjami i umiejętnościami w różnych typach kopalń.

Utworzenie pary reprezentującej pozycję domu księcia.

Tworzenie obiektu Hungarian i przypisanie kosztów:

Utworzenie obiektu Hungarian na podstawie krasnoludów i kopalń.

Pobranie kosztów jako wektory.

Wyświetlanie stanu początkowego:

Wyświetlenie obiektu Hungarian, który reprezentuje stan początkowy przypisania.

Wykonanie przypisania opartego na efektywności:

Wywołanie metody make\_assignment na obiekcie Hungarian w celu wykonania przypisania kopalni do krasnoludów opartego na efektywności.

Wyświetlenie obiektu Hungarian po wykonaniu przypisania.

Wykonanie brute-force:

Wywołanie funkcji brute\_force z wykorzystaniem kosztów.

Wyświetlenie wyników brute-force.

Utworzenie obiektu Hungarian dla optymalnych odległości:

Pobranie zbioru zer z obiektu Hungarian (kopalnie przypisane dokrasnalów).

Wykonanie przypisania optymalizującego odległości od domu:

Utworzenie obiektu Hungarian dla optymalnych odległości na podstawie zbioru zer, krasnoludów i kopalń.

Wywołanie metody make\_assignment na obiekcie Hungarian w celu wykonania przypisania kopalni do krasnoludów optymalizującego odległości od domu. Wyświetlenie obiektu Hungarian po wykonaniu przypisania.

Pobranie przypisania:

Pobranie przypisania kopalni do krasnoludów z obiektu Hungarian.

Tworzenie wektora przypisanych kopalni:

Utworzenie wektora assigned\_ores zawierającego przypisane kopalnie na podstawie przypisania.

Dodanie kopalni domu księcia do wektora assigned\_ores.

Szukanie najkrótszego cyklu:

Wywołanie funkcji find\_shortest\_cycle z przekazaniem wektora assigned\_ores. Zapisanie wyniku jako zmienną distance.

Wyświetlanie najkrótszego cyklu:

Wyświetlenie informacji o najkrótszym cyklu, wypisując indeksy i pozycje kopalni na trasie.

Wyświetlenie całkowitej odległości cyklu.

Zakończenie programu.

Podsumowanie

Kod realizuje proces szukania najkrótszego cyklu w grafie kopalń.

Wykorzystuje różne metody i algorytmy, takie jak przypisanie oparte na efektywności, brute-force i optymalizacja odległości od domu.

Ostatecznie wyświetla informacje o najkrótszym cyklu oraz jego całkowitej odległości.