**1. Cel**

Celem zadania było napisanie programu implementującego różne algorytmy rozwiązywania popularnej układanki – „przesuwanki”, zwanej „piętnastką”, za pomocą różnych metod przeszukiwania przestrzeni stanów. Metody przeszukiwania to: przeszukiwanie wszerz (BFS, ang. breadth-first serach), przeszukiwanie w głąb (DFS, ang. depth-first serach), oraz strategii „najlepszy najpierw” A\* z dwoma heurystykami: metryką Hamminga oraz metryką Manhattan. Część badawcza stanowi zbadanie jak powyższe metody zachowują się przy rozwiązywaniu tego problemu.

**2. WPROWADZENIE**

**3. Opis implementacji**

Program został napisany w języku C++.

Aktualny stan układanki reprezentowany jest jako obiekt klasy Puzzle. Sama układanka przechowywana jest jako std::vector jednowymiarowy o nazwie board. Przechowuje ona również pozycje zera żeby przyspieszyć przeszukiwanie.

Kolejny stan układanki wraz z ruchem jaki wykonaliśmy aby dojść do tego stanu oraz wskaźnikiem na węzeł rodzica jest przechowywany w klasie Node. Klasa ta przechowuje również głębokość rekursji wykorzystywaną do ograniczenia algorytmu DFS.

Klasa Puzzle ma zaimplementowane operacje umożliwiające zamodelowanie przesuwania ‘zera’ po planszy - czyli zamianę pustego pola z którymś z sąsiadujących pól, określanie możliwych do wykonania ruchów, oraz sprawdzanie, czy układ jest układem docelowym.

Dodatkowo Wykorzystane zostały klasy – Solution – do przechowywania wyników i zapisywania ich do pliku.

Klasa Contex – do stworzenia konkretnego początkowego stanu układanki z danych podanych w pliku.

Klasa Configurator – do skonfigurowania programu zgodnie z parametrami wywołania

Każdy z algorytmów jest zaimplementowany jako osobna klasa dziedzicząca po abstrakcyjnej klasieMethods i implementująca metodę run.

Klasa implementująca algorytm DFS zawiera iteracyjną postać implementacji przeszukiwania w głąb. Struktura danych użyta w tej klasie to stos, na który odkładane są kolejne tworzone wierzchołki. Użycie stosu, który jest strukturą danych typu LIFO, zapewnia odpowiednią kolejność sprawdzania wierzchołków. Klasa implementująca algorytm BFS zawiera implementację przeszukiwania wszerz. Struktura danych użyta w tej klasie to kolejka typu FIFO, której użycie zapewnia odpowiednią kolejność sprawdzania wierzchołków. Klasa implementująca algorytm z heurystykami zawiera te same ciągi instrukcji co klasa DFS, jednakże, dodatkowo znajdują się tam fragmenty kodu odpowiadające za odpowiednie posortowanie i odłożenie na stos wierzchołków w założonej kolejności.

Pierwszymi instrukcjami po uruchomieniu programu jest odpowiednie ustawienie zmiennych w zależności od parametrów programu – odpowiada za to klasa Configurator, i jej metoda set(). Następnie za pomocą metody returnMethod() zwraca ona wybraną klasę która odpowiada za rozwiązywanie układanki według odpowiedniej metody przeszukiwania.

Następnie tworzony jest obiekt klasy Solution odpowiedzialny za przetwarzanie i zapisywanie wyników, obiekt ten jest przekazywany w wywołaniu metody run która uruchamia odpowiedni algorytm.

**4. Materiały i metody**

Na początku wygenerowaliśmy pliki tekstowe, zawierające wszystkie możliwe stany układanki o rozmiarach 4 na 4 do maksymalnej głębokości rozwiązania 7, korzystając z generatora udostępnionego na stronie przedmiotu, na platformie WIKAMP.

Algorytmy BFS i DFS były testowane z użyciemośmiuróżnychporządków

przeszukiwania sąsiedztwa: DRLU,DRUL, LUDR, LURD, RDLU, RDUL, ULDR, ULRD, gdzie odpowiednie litery oznaczają przesunięcie ‘zera’ w kierunku; D- do dołu, U-do góry, R-w prawo, L-w lewo.

Natomiast w algorytmie A\* testowane były dwie heurystyki: metryki Hamming-a

i Manhattan. Algorytm DFS miał na stałe ustawiona maksymalna

głębokość rekursji na 20. Badanie polegało na rozwiązywaniu przez algorytmy

w każdej z możliwych konfiguracji wszystkich 413 układów. Do uruchamiania

przeszukiwań oraz wyłuskania danych zostały wykorzystane skrypty

udostępnione na stronie przedmiotu. Celem eksperymentu było dokonanie

następującychporównań: porównanie różnychporządków przeszukiwania sąsiedztwa w algorytmachBFS i DFS, oraz różnych heurystyk w algorytmie A\*. Te wszystkie algorytmy i ich wersje porównywane były pod względem:

* długości znalezionego rozwiązania
* liczby stanów odwiedzonych
* liczby stanów przetworzonych
* maksymalnej osiągniętej głębokości rekursji
* czasu trwania procesu obliczeniowego

**5. Wyniki**

We wszystkich wykresach wartości 0d 1 do 7 na dole wykresu oznaczają głębokości przetwarzanych układanek.

**Wykresy danych ze względu na ilość stanów przetworzonych:**

**Wykresy danych ze względu na ilość stanów odwiedzonych:**

**Wykresy danych ze względu na długość znalezionego rozwiązania:**

**Wykresy danych ze względu na maksymalną głębokość osiągniętego poziomu rekursji :**

Wykresy danych ze względu na Średni czas działania w milisekundach:

Tabele podsumowująca czasy (w milisekundach) działania dla wszystkich algorytmów ( dodano tabele gdyż wykres nie uwidacznia szczegółów)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Głębokość | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| ASTR | 0,011 | 0,017625 | 0,02805 | 0,040625 | 0,056491 | 0,074262 | 0,092927 |
| BFS | 0,571313 | 0,603344 | 0,683088 | 0,692417 | 0,809116 | 1,151723 | 2,638839 |
| DFS | 0,009875 | 402,2305 | 415,1268 | 276,6128 | 489,7215 | 389,8097 | 589,7127 |

**6. Dyskusja**

**7. Wnioski**

Maksymalna głębokość rekurencji w przypadku algorytmów BFS i A\* nigdy nie przekraczała głębokości układu początkowego.

Algorytm DFS często nie znajdywałby rozwiązania, gdyby nie ograniczenie maksymalnej głębokości rekurencji (dochodziłoby do wyczerpania pamięci).

Liczba wierzchołków przetworzonych jest porównywalna do liczby wierzchołków odwiedzonych dla strategii DFS. Liczba wierzchołków przetworzonych 3 – krotnie większa niż wierzchołków odwiedzonych - dla BFS.

Algorytm BFS zachowywał się najbardziej przewidywalnie. Omówione wyżej wyniki dla poszczególnych kryteriów zmieniały się regularnie.

Dla większych odległości metryka Manhattana sprawdza się lepiej niż metryka Hamminga.

A\* i BFS znajdują zawsze najkrótsze rozwiązania, a DFS zazwyczaj dłuższe

— algorytm A\* jest zdecydowanie najbardziej optymalny, a DFS najmniej,

— wybrany porządek przeszukiwania sąsiedztwa ma wpływ na efektywność

działania algorytmów BFS i DFS, w DFS jest to wpływ znaczący,

— wybrana heurystyka ma znaczący wpływ na efektywność działania algorytmu

A\*,

— algorytmy BFS i DFS (w szczególności ten drugi) potrzebują znacznie

więcej zasobów do działania niż A\*,

11

— możesięzdarzyć, ze w pojedynczym uruchomieniu algorytm DFS zadziała

wydajniej niż BFS