Grzegorz Krzeszowski
Projekt zaliczeniowy – Python
Rok akademicki: 2016/17

Generowanie oraz przechodzenie labiryntu

Wprowadzenie

Działanie programu polega na generowaniu labiryntu oraz przechodzeniu go. Problem można rozwiązać na wiele sposobów wieloma algorytmami.

Do stworzenia labiryntu użyłem wcześniej zaimplementowanych struktur przeze mnie na zajęciach (stos, randomQueue). Labirynt generuję za pomocą algorytmu DFS

Do przejścia labiryntu użyłem kolejki (zaimplementowanej na zajęciach). Wyjścia z labiryntu szukam za pomocą algorytmu BFS

Opis użytych struktur

Stos

Stos jest to jedna z podstawowych struktur danych. Polega ona na tym iż kładziemy zawsze elementy na jej szczyt funkcją push(value). Gdy chcemy zdjąć element ze stosu użyjemy metody pop(), zwraca ona element na samej górze stosu (ten co został ostatni położony). Aby sprawdzić czy stos jest pusty używamy funkcji empty() która zwraca wartość true gdy stos jest pusty lub false gdy stos nie jest pusty.

Kolejka

Kolejka jest to struktura danych do której wkładane elementy są kolejkowane. Na początku kolejki znajdują się elementy włożone na początku. Wstawiania elementów oraz wyjmowania z kolejki rzędu O(1).

Random Queue

Jest to specyficzna implementacja kolejki do której wkładamy elementy w czasie stałym oraz ściągamy z niej również w czasie stałym. Specyficzne w tej strukturze jest to iż ściągany element jest losowo wybrany z kolejki.

Algorytm DFS działa na strukturze opisanej wyżej (stos). Pseudokod:

1. Połóż element startowy na stosie.
2. Oznacz element jako odwiedzony.
3. Dopóki stos nie jest pusty wykonuj krok 4 - 7

4. Zdejmij element ze stosu
5. Przechodząc po wszystkich sąsiadach zdjętego przed chwilą elementu wykonaj krok 6 –7.
6. Jeśli sąsiad był nie odwiedzony połóż go na stos oraz oznacz jako odwiedzony
7. Ustaw dla sąsiada wskaźnik parent na wierzchołek który go wywołał.

Algorytm BFS działa na strukturze opisanej wyżej (kolejka). Pseudokod:

1. Połóż element startowy do kolejki.
2. Oznacz element jako odwiedzony.
3. Dopóki kolejka nie jest pusty wykonuj krok 4 – 7
4. Zdejmij element ze stosu
5. Jeśli element zdjęty jest wyjściem zwróć jego współrzędne i zakończ działanie
6. Przechodząc po wszystkich sąsiadach zdjętego przed chwilą elementu wykonaj krok 7 –8.
7. Jeśli sąsiad był nie odwiedzony połóż go do kolejki oraz oznacz jako odwiedzony
8. Ustaw dla sąsiada wskaźnik parent na wierzchołek który go wywołał.

Aby sprawdzić nasze wyjście z labiryntu wystarczy wskazać wierzchołek w którym znajdowały się „drzwi wyjściowe” po czym przeiterować po wskaźnikach parent które ustawialiśmy w algorytmie aby zobaczyć drogę wyjściową z labiryntu.

Uruchomienie programu

Aby uruchomić program wpisujemy w konsoli: python maze.py. Program wygeneruje labirynt ze startem w punkcie (0, 0) oraz wyjście z labiryntu losowo wygenerowane. Po wygenerowaniu labiryntu program wyświetla labirynt już z wyjściem.

Legenda:
- „S” - start
- „M” - wyjście z labiryntu
- „*” - obramowanie labiryntu
- „#” - ściany w labiryncie
- „O” = najkrótsza ścieżka wyjścia z labiryntu

Literatura

- http://users.uj.edu.pl/~ufkapano/algorytmy/index.html

- http://eduinf.waw.pl/inf/alg/001_search/0126.php
- http://eduinf.waw.pl/inf/alg/001_search/0125.php
- http://szymonsiarkiewicz.pl/poradniki/howto/howto-generator-labiryntow/