Michał Zadrożny

AAL – dokumentacja

Temat 14: Archipelag

Opis problemu:

Algorytm na wejściu otrzymuje macierz wypełnioną zerami oraz jedynkami. Na wyjściu podaje rozmiar największej „wysypy jedynek" – fragmentu macierzy, w którym jedynki ze sobą sąsiadują (w poziomie, w pionie lub po skosie).

Rozwiązanie:

- **Struktury danych**

Otrzymywana na wejściu mapa jest reprezentowana jako macierz – dwuwymiarowa tablica typu int wypełniana zerami oraz jedynkami. Macierz ta może być postrzegana jako pewna reprezentacja grafu: wierzchołki grafu są reprezentowane w macierzy przez jedynki. Jeśli dwa wierzchołki grafu są połączone krawędzią, to w macierzy jedynki reprezentujące te wierzchołki będą ze sobą sąsiadować. Wynika z tego, że problem znalezienia największej wyspy jedynek jest równoważny problemowi znalezienia największego spójnego podgrafu w odpowiadającym macierzy grafie. Do rozwiązania problemu mogą zatem posłużyć dwa znane algorytmy przeszukiwania grafu – DFS oraz BFS. Oba te algorytmy gwarantują odwiedzenie wszystkich wierzchołków osiągalnych dla danego wierzchołka startowego (czyli odwiedzenie wszystkich wierzchołków należących do spójnego podgrafu). Zliczanie odwiedzanych wierzchołków pozwoli na znalezienie szukanego rozmiaru największej wyspy.

Do działania obu algorytmów niezbędna będzie dwuwymiarowa tablica typu boolean visited[][], na początku wypełniona wartościami false, o takim samym rozmiarze co rozpatrywana macierz. Tablica visited posłuży do odznaczania już odwiedzonych przez algorytm wierzchołków (jedynek w macierzy). Dodatkowo w algorytmie BFS stosowana będzie kolejka, do której trafiać będą nowo odwiedzeni sąsiedzi aktualnie rozpatrywanego wierzchołka.

- **Algorytmy**
  - **DFS (Depth-first search)**

Algorytm przechodzi po kolejnych elementach macierzy. Gdy natrafi na jedynkę, która nie była jeszcze odwiedzona (wartość w tablicy visited pod odpowiadającym znalezionej jedynce indeksem równa false), to wywoływana jest funkcja visitIsland z rozpatrywaną jedynką jako wierzchołkiem startowym (odwiedzamy nową, potencjalnie największą, wyspę). Funkcja ta zapisuje fakt odwiedzenia wierzchołka w tablicy visited, a następnie przegląda wszystkich sąsiadów rozpatrywanej jedynki (sprawdzanie wartości w macierzy pod odpowiednimi

indeksami). Znalezienie wśród sąsiadów nieodwiedzonej jedynki wiąże się z rekurencyjnym wywołaniem funkcji visitIsland, tym razem z nową jedynką jako rozpatrywanym wierzchołkiem (teraz poszukujemy kolejnych jedynek wśród sąsiadów znalezionej jedynki). Rozmiar aktualnie rozpatrywanej wyspy jest równy liczbie wywołań rekurencyjnych funkcji visitIsland począwszy od jej wywołania dla startowej jedynki. Jeśli rozmiar ostatnio analizowanej wyspy jest do tej pory największy, to jest on zapisywany w zmiennej max. Po przejrzeniu całej macierzy zwracana jest wartość zmiennej max.

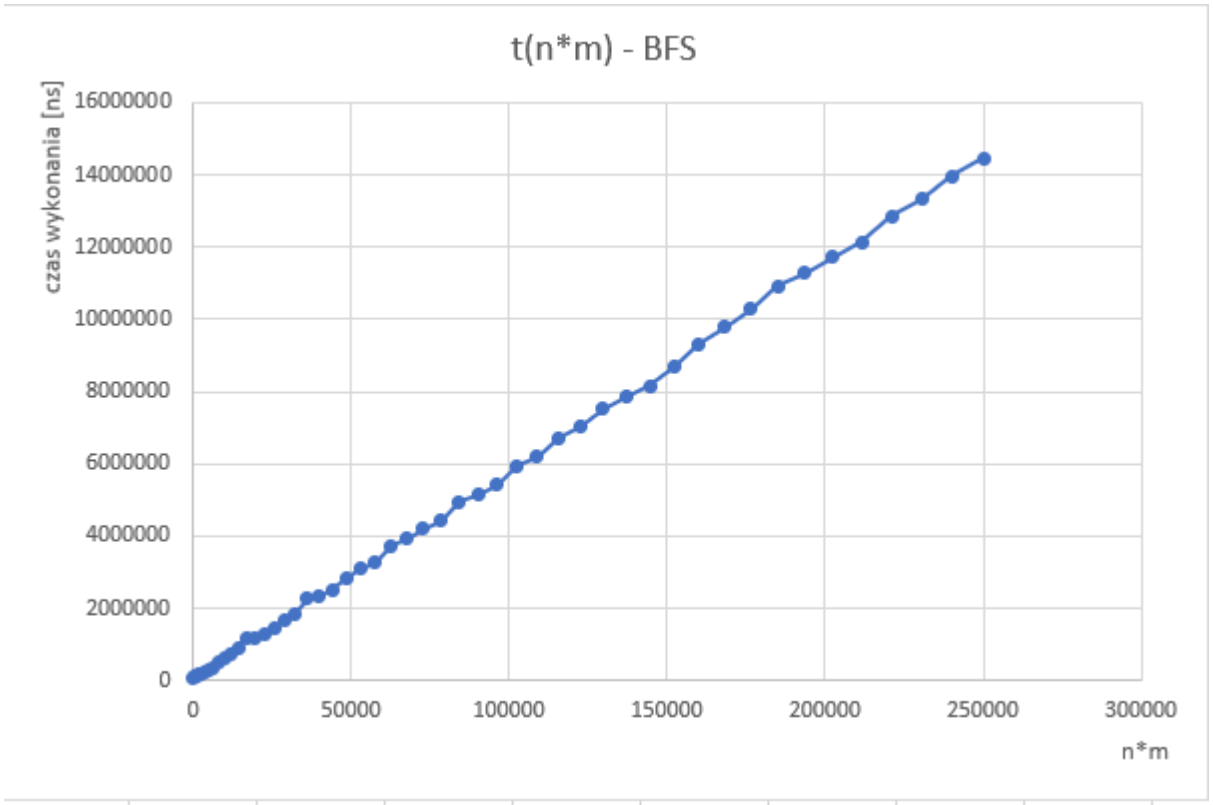
- **BFS (Breadth-first search)**

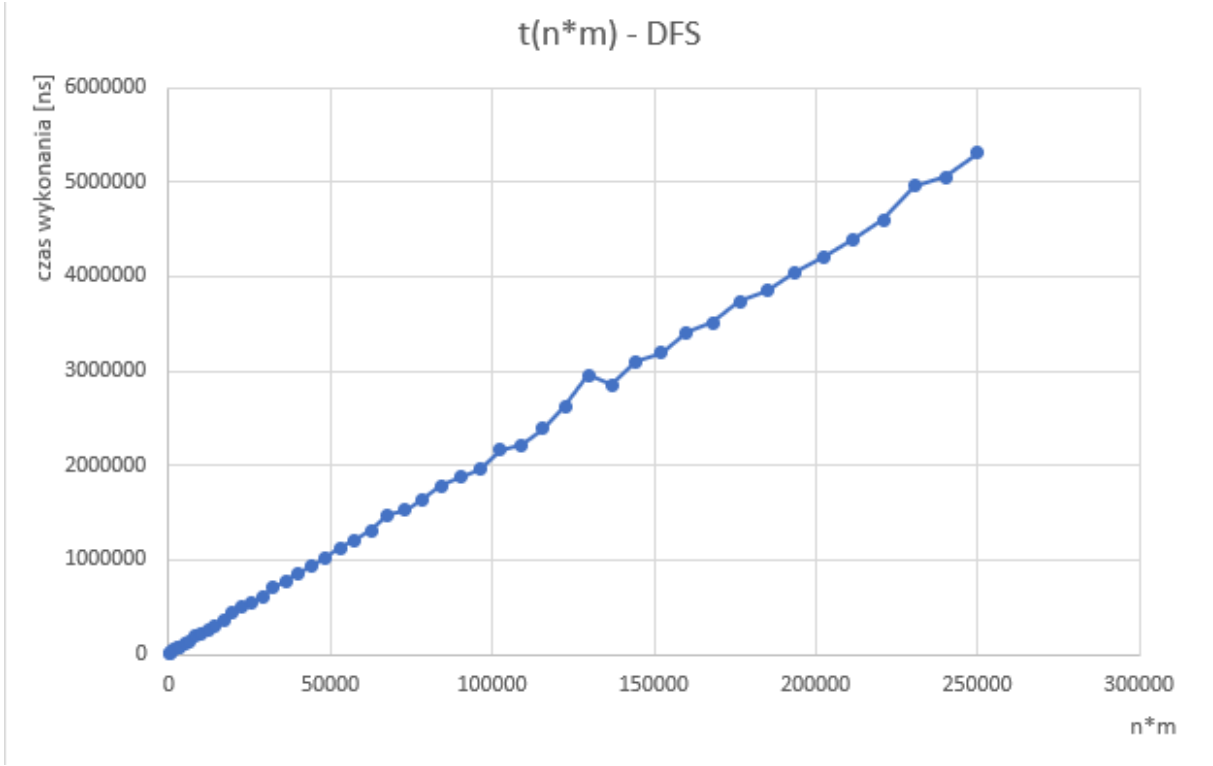
Algorytm działa podobnie do wcześniej opisanego algorytmu DFS. Różnica pojawia się w momencie znalezienia nowej jedynki wśród sąsiadów aktualnie rozpatrywanej jedynki. Zamiast rekurencyjnego wywołania funkcji, znaleziona jedynka (a dokładniej jej współrzędne w macierzy) są umieszczane w kolejce. Jedynki są rozpatrywane według kolejności kolejki (rozpatrywana jedynka, wśród której sąsiadów będziemy teraz poszukiwać, jest usuwana z kolejki). Analiza jednej wyspy kończy się, kiedy kolejka stanie się pusta. Rozmiar wyspy otrzymujemy dzięki zwiększaniu licznika przy każdym umieszczeniu jedynki w kolejce. Po przeanalizowaniu całej wyspy, porównujemy jej rozmiar do zmiennej max, i ewentualnie go nadpisujemy (analogicznie jak w algorytmie DFS).

Spodziewana złożoność algorytmów:

Spodziewana złożoność asymptotyczna obu opisanych algorytmów jest liniowa. W przypadku pesymistycznym (cała macierz wypełniona jedynkami) algorytm odwiedzi wszystkie elementy macierzy w ramach odwiedzenia wyspy, po czym zewnętrzna pętla ponownie przejdzie po wszystkich (jak się okaże – już odwiedzonych) elementach (dwukrotne przejście po wszystkich elementach macierzy). Dodatkowo, dla każdej rozpatrywanej jedynki wykonujemy 8 porównań – rozpatrujemy sąsiadów (z wyjątkiem elementów na brzegach, gdzie porównań będzie mniej). Ostatecznie daje nam to złożoność liniową z pewnym współczynnikiem.

Liniową złożoność algorytmów potwierdzają przeprowadzone testy.


t(n*m) - BFS
czas wykonania [ns]
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
0
50000
100000
150000
200000
250000
300000
n*m


t(n*m) - DFS
czas wykonania [ns]
6000000
5000000
4000000
3000000
2000000
1000000
0
0
50000
100000
150000
200000
250000
300000
n*m