**Доклад**

**по**

**Алгоритми и структури от данни**

**Тема: Работа с графи в компютърна памет**

1. **Видове представяния на графите в компютърната памет**
2. **Обхождане в дълбочина на граф**
3. **Обхождане на ширина на граф**
4. **Най-кратък път в граф по алгоритъма на Дийкстра**
5. **Имплементация на дървета и графи**
6. **Източници**

Изготвил: Ваня Ванева 12а

1. **Видове представяния на графите в компютърната памет**

В компютърната наука, графите се представят чрез различни структури от данни в паметта, които могат да бъдат описани по различни начини. Някои от най-честите видове представяния на графи включват:

* **Списък на съседство**: Всяка връх в графа се представя като списък, който съдържа всички върхове, до които има директно ребро от този връх.

**class Graph**

**{**

**private Dictionary<int, List<int>> adjacencyList;**

**// Constructor**

**public Graph()**

**{**

**adjacencyList = new Dictionary<int, List<int>>();**

**}**

**// Method to add an edge between vertices**

**public void AddEdge(int source, int destination)**

**{**

**// If the source vertex is not in the adjacency list, create a new entry**

**if (!adjacencyList.ContainsKey(source))**

**{**

**adjacencyList[source] = new List<int>();**

**}**

**// Add the destination vertex to the adjacency list of the source vertex**

**adjacencyList[source].Add(destination);**

**}**

**// Method to access the adjacency list**

**public Dictionary<int, List<int>> GetAdjacencyList()**

**{**

**return adjacencyList;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create an instance of the Graph class**

**Graph graph = new Graph();**

**// Add some edges to the graph**

**graph.AddEdge(1, 2);**

**graph.AddEdge(1, 3);**

**graph.AddEdge(2, 3);**

**graph.AddEdge(3, 4);**

**graph.AddEdge(4, 2);**

**// Display the adjacency list**

**Console.WriteLine("Adjacency List:");**

**foreach (var kvp in graph.GetAdjacencyList())**

**{**

**Console.Write(kvp.Key + ": ");**

**foreach (var vertex in kvp.Value)**

**{**

**Console.Write(vertex + " ");**

**}**

**Console.WriteLine();**

**}**

**}**

**}**

* **Матрица на съседство**: Тук графът се представя чрез квадратна матрица, където стойностите на позиция [i, j] показват дали има ребро от връх i до връх j.

**class Graph**

**{**

**private int[,] adjacencyMatrix;**

**// Constructor**

**public Graph(int vertices)**

**{**

**adjacencyMatrix = new int[vertices, vertices];**

**}**

**// Method to add an edge between vertices**

**public void AddEdge(int source, int destination)**

**{**

**adjacencyMatrix[source, destination] = 1;**

**}**

**// Method to access the adjacency matrix**

**public int[,] GetAdjacencyMatrix()**

**{**

**return adjacencyMatrix;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create an instance of the Graph class**

**Graph graph = new Graph(5); // Creating a graph with 5 vertices**

**// Add some edges to the graph**

**graph.AddEdge(0, 1);**

**graph.AddEdge(0, 4);**

**graph.AddEdge(1, 3);**

**graph.AddEdge(1, 2);**

**graph.AddEdge(2, 4);**

**graph.AddEdge(3, 2);**

**graph.AddEdge(3, 4);**

**// Display the adjacency matrix**

**Console.WriteLine("Adjacency Matrix:");**

**int[,] matrix = graph.GetAdjacencyMatrix();**

**for (int i = 0; i < matrix.GetLength(0); i++)**

**{**

**for (int j = 0; j < matrix.GetLength(1); j++)**

**{**

**Console.Write(matrix[i, j] + " ");**

**}**

**Console.WriteLine();**

**}**

**}**

**}**

1. **Обхождане в дълбочина на граф**

Обхождането в дълбочина е метод за обхождане на граф, който използва стек за съхранение на върховете. При този вид обхождане, алгоритъмът преминава през всеки възможен връх преди да продължи с обхождането към следващия връх.

**class GraphTraversal**

**{**

**// Recursive method for depth-first search traversal**

**void DFSUtil(int vertex, bool[] visited, Graph graph)**

**{**

**visited[vertex] = true;**

**Console.WriteLine(vertex);**

**foreach (var adjacentVertex in graph.GetAdjacencyList()[vertex])**

**{**

**if (!visited[adjacentVertex])**

**DFSUtil(adjacentVertex, visited, graph);**

**}**

**}**

**// Method for depth-first search traversal**

**public void DFS(Graph graph)**

**{**

**int vertices = graph.GetAdjacencyList().Count;**

**bool[] visited = new bool[vertices];**

**for (int i = 0; i < vertices; ++i)**

**{**

**if (!visited[i])**

**DFSUtil(i, visited, graph);**

**}**

**}**

**}**

**class Graph**

**{**

**private Dictionary<int, List<int>> adjacencyList;**

**// Constructor**

**public Graph()**

**{**

**adjacencyList = new Dictionary<int, List<int>>();**

**}**

**// Method to add an edge between vertices**

**public void AddEdge(int source, int destination)**

**{**

**if (!adjacencyList.ContainsKey(source))**

**{**

**adjacencyList[source] = new List<int>();**

**}**

**adjacencyList[source].Add(destination);**

**}**

**// Method to access the adjacency list**

**public Dictionary<int, List<int>> GetAdjacencyList()**

**{**

**return adjacencyList;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create an instance of the Graph class**

**Graph graph = new Graph();**

**// Add some edges to the graph**

**graph.AddEdge(0, 1);**

**graph.AddEdge(0, 2);**

**graph.AddEdge(1, 2);**

**graph.AddEdge(2, 0);**

**graph.AddEdge(2, 3);**

**graph.AddEdge(3, 3);**

**// Create an instance of the GraphTraversal class**

**GraphTraversal traversal = new GraphTraversal();**

**// Perform DFS traversal on the graph**

**Console.WriteLine("Depth-First Search Traversal:");**

**traversal.DFS(graph);**

**}**

**}**

1. **Обхождане на ширина на граф**

При обхождането на ширина на граф, използва се опашка за съхранение на върховете. Този вид обхождане преминава през всички съседни върхове на текущия, преди да продължи към следващия възможен връх.

**class GraphTraversal**

**{**

**// Method for width-first search traversal**

**public void BFS(Graph graph, int startVertex)**

**{**

**int vertices = graph.GetAdjacencyList().Count;**

**bool[] visited = new bool[vertices];**

**Queue<int> queue = new Queue<int>();**

**visited[startVertex] = true;**

**queue.Enqueue(startVertex);**

**while (queue.Count != 0)**

**{**

**int currentVertex = queue.Dequeue();**

**Console.WriteLine(currentVertex);**

**foreach (var adjacentVertex in graph.GetAdjacencyList()[currentVertex])**

**{**

**if (!visited[adjacentVertex])**

**{**

**visited[adjacentVertex] = true;**

**queue.Enqueue(adjacentVertex);**

**}**

**}**

**}**

**}**

**}**

**class Graph**

**{**

**private Dictionary<int, List<int>> adjacencyList;**

**// Constructor**

**public Graph()**

**{**

**adjacencyList = new Dictionary<int, List<int>>();**

**}**

**// Method to add an edge between vertices**

**public void AddEdge(int source, int destination)**

**{**

**if (!adjacencyList.ContainsKey(source))**

**{**

**adjacencyList[source] = new List<int>();**

**}**

**adjacencyList[source].Add(destination);**

**}**

**// Method to access the adjacency list**

**public Dictionary<int, List<int>> GetAdjacencyList()**

**{**

**return adjacencyList;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create an instance of the Graph class**

**Graph graph = new Graph();**

**// Add some edges to the graph**

**graph.AddEdge(0, 1);**

**graph.AddEdge(0, 2);**

**graph.AddEdge(1, 2);**

**graph.AddEdge(2, 0);**

**graph.AddEdge(2, 3);**

**graph.AddEdge(3, 3);**

**// Create an instance of the GraphTraversal class**

**GraphTraversal traversal = new GraphTraversal();**

**// Perform BFS traversal on the graph starting from vertex 2**

**Console.WriteLine("Width-First Search Traversal:");**

**traversal.BFS(graph, 2);**

**}**

**}**

1. **Най-кратък път в граф по алгоритъма на Дийкстра**

Алгоритъмът на Дийкстра се използва за намиране на най-кратък път между два върха в тегловен граф. Той работи за графове без отрицателни тегла.

**class DijkstraAlgorithm**

**{**

**// Method for finding the shortest path**

**public void Dijkstra(Graph graph, int source)**

**{**

**int vertices = graph.GetAdjacencyList().Count;**

**int[] distance = new int[vertices];**

**bool[] shortestPathTreeSet = new bool[vertices];**

**for (int i = 0; i < vertices; ++i)**

**{**

**distance[i] = int.MaxValue;**

**shortestPathTreeSet[i] = false;**

**}**

**distance[source] = 0;**

**for (int count = 0; count < vertices - 1; ++count)**

**{**

**int u = MinDistance(distance, shortestPathTreeSet);**

**shortestPathTreeSet[u] = true;**

**foreach (var v in graph.GetAdjacencyList()[u])**

**{**

**if (!shortestPathTreeSet[v] && distance[u] != int.MaxValue &&**

**distance[u] + 1 < distance[v]) // Assuming each edge has weight 1**

**{**

**distance[v] = distance[u] + 1; // Assuming each edge has weight 1**

**}**

**}**

**}**

**PrintSolution(distance);**

**}**

**// Method for finding the minimum distance**

**int MinDistance(int[] distance, bool[] shortestPathTreeSet)**

**{**

**int min = int.MaxValue;**

**int minIndex = -1;**

**for (int v = 0; v < distance.Length; ++v)**

**{**

**if (shortestPathTreeSet[v] == false && distance[v] <= min)**

**{**

**min = distance[v];**

**minIndex = v;**

**}**

**}**

**return minIndex;**

**}**

**// Method for printing the solution**

**void PrintSolution(int[] distance)**

**{**

**Console.WriteLine("Vertex Distance from Source");**

**for (int i = 0; i < distance.Length; ++i)**

**Console.WriteLine($"{i}\t{distance[i]}");**

**}**

**}**

**class Graph**

**{**

**private Dictionary<int, List<int>> adjacencyList;**

**// Constructor**

**public Graph()**

**{**

**adjacencyList = new Dictionary<int, List<int>>();**

**}**

**// Method to add an edge between vertices**

**public void AddEdge(int source, int destination)**

**{**

**if (!adjacencyList.ContainsKey(source))**

**{**

**adjacencyList[source] = new List<int>();**

**}**

**adjacencyList[source].Add(destination);**

**}**

**// Method to access the adjacency list**

**public Dictionary<int, List<int>> GetAdjacencyList()**

**{**

**return adjacencyList;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create an instance of the Graph class**

**Graph graph = new Graph();**

**// Add some edges to the graph**

**graph.AddEdge(0, 1);**

**graph.AddEdge(0, 2);**

**graph.AddEdge(1, 2);**

**graph.AddEdge(1, 3);**

**graph.AddEdge(2, 3);**

**graph.AddEdge(3, 4);**

**// Create an instance of the DijkstraAlgorithm class**

**DijkstraAlgorithm dijkstra = new DijkstraAlgorithm();**

**// Perform Dijkstra's algorithm on the graph starting from vertex 0**

**Console.WriteLine("Dijkstra's Algorithm:");**

**dijkstra.Dijkstra(graph, 0);**

**}**

**}**

1. **Имплементация на дървета и графи**

Представянето и работата с дървета и графи включва различни структури от данни и алгоритми за тяхна обработка.

**class Tree**

**{**

**public int Value { get; set; }**

**public List<Tree> Children { get; set; }**

**// Constructor**

**public Tree(int value)**

**{**

**Value = value;**

**Children = new List<Tree>();**

**}**

**// Method to add children**

**public void AddChild(Tree child)**

**{**

**Children.Add(child);**

**}**

**// Method to display the tree structure**

**public void DisplayTree()**

**{**

**DisplayTreeUtil(this, 0);**

**}**

**private void DisplayTreeUtil(Tree node, int level)**

**{**

**Console.WriteLine($"{new string(' ', level \* 4)}{node.Value}");**

**foreach (var child in node.Children)**

**{**

**DisplayTreeUtil(child, level + 1);**

**}**

**}**

**// Method to access the children of a node**

**public List<Tree> GetChildren()**

**{**

**return Children;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Create the tree structure**

**Tree root = new Tree(1);**

**Tree child1 = new Tree(2);**

**Tree child2 = new Tree(3);**

**Tree grandchild1 = new Tree(4);**

**Tree grandchild2 = new Tree(5);**

**// Build the tree**

**root.AddChild(child1);**

**root.AddChild(child2);**

**child1.AddChild(grandchild1);**

**child1.AddChild(grandchild2);**

**// Display the tree structure**

**Console.WriteLine("Tree Structure:");**

**root.DisplayTree();**

**}**

**}**

1. **Източници**

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

<https://www.studysmarter.co.uk/explanations/computer-science/algorithms-in-computer-science/graph-traversal/>

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

<https://testbook.com/question-answer/when-using-dijkstras-algorithm-to-find-shor--5ec66d5df60d5d193e2d01f6#:~:text=Dijkstra's%20algorithm%20is%20used%20to,and%20graph%20must%20be%20connected.&text=So%2C%20we%20get%20the%20shortest,%3E%20f%20%2D%3E%20g>).

<https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>

<https://introprogramming.info/intro-csharp-book/read-online/glava17-durveta-i-grafi/>

<https://softuni.bg/blog/algorithms-advanced-graph-traversal>

<https://www.geeksforgeeks.org/graph-and-its-representations/>

<https://www.javatpoint.com/graph-theory-graph-representations>

<https://learnloner.com/memory-representation-of-graph/>