C A J

CR 7

GRAPH THEORY

PART 6

Date: 02/10/2024

PROSIT 6: GRAPH THEORY

Roles

- ***Facilitator***: Ethan
- ***Resource and Time Manager***: Andrew
- ***Scribe***: Jeremy
- ***Secretary***: Larry

1. Keywords

- Gluttonous algorithm : is a problem-solving technique that makes the best local choice at each step in the hope of finding the global optimum solution.
- Chromatic number : the smallest number of colours needed to colour its vertices such that no two adjacent vertices share the same colour
- Cordial / perfect graph : if its chromatic number equals the size of the maximum clique, both in the graph itself and in every induced subgraph.
  Cordial graphs are a subclass of perfect graphs.
- NP-complete problem : if its solution can be verified in polynomial time by a deterministic Turing machine.
- Non-overlapping tasks
- Data center

2. Context

A company is trying to optimize the energy consumption of a data center by trying to optimize task scheduling for their servers.

3. Issue

How to optimize the energy efficiency of the servers?

How can we properly optimize task scheduling graphs?

How can we use graphs to represent the problem?

4. Constraints

- instances.zip resource file
- A single server can only perform a single task at a time
- Each task has a specific time interval

5. Generalization

- Graph theory

6. Deliverable

- Python script
- Written report (CER)

7. Hypothesis

- We will represent the graph using a matrix
- use a Breadth first search and Depth-first search
- The problem is NP-hard

8. Action Plan

1. Define keywords
2. Analyse resources

3. Analyse excel files
4. Colour-code servers
5. Represent graph
6. Learn how to compute the task assigning
7. Schedule tasks
8. Validate hypothesis
9. Conclusion

9. Resource Analysis

1. Basics of Graphs

- **Graph (G)**: A graph is a pair G=(V,E)G = (V, E)G=(V,E), where: (“Directed graph representations and traversals - Department of Computer ...”)
  - V is a set of vertices (or nodes).
  - E is a set of edges (or links) between vertices.
- **Directed Graph (Digraph)**: A graph where edges have a direction. An edge from vertex uuu to vvv is denoted as (u,v)(u, v)(u,v).
- **Undirected Graph**: A graph where edges have no direction. An edge between uuu and vvv is denoted as {u,v}
- **Weighted Graph**: A graph where edges have weights (values) associated with them, often representing costs, distances, or capacities.
- **Simple Graph**: A graph with no loops (edges from a vertex to itself) and no more than one edge between any

pair of vertices. (“Solution Verification: Maximum number of edges, given 8 vertices”)

- **Multigraph**: A graph that can have multiple edges between the same pair of vertices. (“END-TO-END NEURAL SEGMENTAL MODELS FOR SPEECH RECOGNITION”)
- **Complete Graph (K_n)**: A graph where every pair of vertices is connected by a unique edge. (“Complete Graph | Definition & Example - Lesson - Study.com”) It has n(n−1)2\frac{n(n-1)}{2}2n(n−1) edges.
- **Subgraph**: A graph formed from a subset of the vertices and edges of a larger graph. (“Introduction to Graph - Data Structures Tutorial | Study Glance”)

2. Graph Terminology

- **Degree**: The number of edges connected to a vertex. In a directed graph, we differentiate between **in-degree** (edges coming into a vertex) and **out-degree** (edges leaving a vertex).
- **"Path**: A sequence of edges that connects a sequence of vertices." (“MATH 101 - Study Guide Understanding Graph Theory”)
- **Cycle**: A path that starts and ends at the same vertex without repeating any other vertices. (“graph theory - When does a closed walk not have a cycle? - Mathematics …”)

- **Connected Graph**: An undirected graph is connected if there is a path between every pair of vertices. ("10.4 Connectivity - University of Hawai'i")
- **Disconnected Graph**: A graph that has at least two vertices without a path connecting them.
- **Tree**: A connected, acyclic graph.
- **Bipartite Graph**: A graph whose vertices can be divided into two disjoint sets UUU and VVV, such that every edge connects a vertex in UUU to a vertex in VVV. ("Bipartite Graph - Techie Delight")
- **Planar Graph**: A graph that can be drawn on a plane without any edges crossing. ("Planar Graph -- from Wolfram MathWorld")

3. Types of Graphs

- **Eulerian Graph**: A graph where there exists a path that visits every edge exactly once. A Eulerian circuit is a cycle that visits every edge exactly once and returns to the starting vertex.
    - **Euler's Theorem**: A connected graph has a Eulerian circuit if and only if every vertex has an even degree.
- **Hamiltonian Graph**: A graph where there exists a path that visits every vertex exactly once. ("12.8 Hamilton Paths - Contemporary Mathematics | OpenStax") A Hamiltonian cycle is a cycle that visits every vertex once and returns to the starting vertex. ("What is a Hamiltonian Cycle? - Online Tutorials Library")

  - There is no simple necessary and sufficient condition like Euler's theorem for Hamiltonian paths or cycles.
- **Complete Graph (K_n)**: A graph where every pair of vertices is connected by an edge. (“4.1: Definitions - Mathematics LibreTexts”)
- **Planar Graph**: A graph that can be embedded in the plane such that no edges intersect. (“Planar Graphs and Graph Coloring | Engineering Mathematics - GeeksforGeeks”)

4. Common Graph Algorithms

Traversal Algorithms

- **Depth-First Search (DFS)**: A traversal method that explores as far along a branch as possible before backtracking. (“Basic Algorithms - DEV Community”) It's used for cycle detection, pathfinding, and topological sorting.
- **Breadth-First Search (BFS)**: A traversal method that explores all neighbours of a vertex before moving on to their neighbours. It's used for shortest path algorithms in unweighted graphs.

Shortest Path Algorithms

- **Dijkstra’s Algorithm**: Finds the shortest path from a sole source vertex to all other vertices in a weighted graph (with non-negative weights).
- **Bellman-Ford Algorithm**: Finds the shortest path from a sole source to all vertices in a graph that may have negative edge weights. It also detects negative cycles.

- **Floyd-Warshall Algorithm**: A dynamic programming algorithm to find shortest paths between all pairs of vertices in a weighted graph. (“Floyd-Warshall Algorithm in C - GeeksforGeeks”)

Minimum Spanning Tree (MST) Algorithms

- **Kruskal’s Algorithm**: Finds the minimum spanning tree by adding edges in order of increasing weight, ensuring no cycles form.
- **Prim’s Algorithm**: Builds the minimum spanning tree by growing a single tree, adding the smallest edge that connects a new vertex. (“Prim's Algorithm - Vocab, Definition, and Must Know Facts - Fiveable”)

Graph Connectivity

- **Union-Find (Disjoint Set Union)**: Efficiently manages dynamic connectivity problems, such as finding which component a vertex belongs to and merging two components.
- **Tarjan's Algorithm**: Used to find strongly connected components in a directed graph in O(V+E) time.
- **Kosaraju's Algorithm**: Another algorithm to find strongly connected components using two DFS traversals.

Topological Sorting

- Used for directed acyclic graphs (DAGs). A topological sort is an ordering of vertices such that for every directed edge

u→vu to vu→v, vertex uuu comes before vvv in the ordering. (“Topological Sort | CodePath Cliffnotes”)

Eulerian Path/Circuit

- **Fleury’s Algorithm**: Finds a Eulerian path or circuit in an undirected graph by removing edges as they are traversed, ensuring no edge is removed that would disconnect the graph unless necessary.
- **Hierholzer’s Algorithm**: Finds a Eulerian circuit by constructing sub-cycles and merging them.

Hamiltonian Path/Cycle

- There is no efficient general algorithm for finding Hamiltonian paths or cycles, but methods such as backtracking and dynamic programming can be used in specific cases.

5. Theorems in Graph Theory

- **Euler’s Formula**: For planar graphs, V−E+F=2V - E + F = 2V−E+F=2, where V is the number of vertices, E is the number of edges, and F is the number of faces (regions). (“GeometrySpot Simplifying Complex Geometry Concepts: - Snokido”)
- "**Handshaking Lemma**: In any undirected graph, the sum of the degrees of all vertices is twice the number of edges." (“Handshaking Lemma and Interesting Tree Properties - Javatpoint”)

- **Menger's Theorem**: Relates the maximum number of independent paths between two vertices to the minimum number of vertices whose removal would separate them.
- **Hall's Marriage Theorem**: In bipartite graphs, gives a necessary and sufficient condition for a perfect matching (every vertex is matched).
- **Kuratoński's Theorem**: A graph is planar if and only if it contains no subgraph that is a subdivision of K5K_5K5 (complete graph on five vertices) or K3,3K_{3,3}K3,3 (complete bipartite graph on six vertices).
- **Graph Colouring**: Assigning colours to vertices of a graph such that no two adjacent vertices share the same colour. A famous result is the **Four Colour Theorem**, which states that any planar graph can be coloured using at most four colours.

6. Applications of Graph Theory

- **Networks**: Representing and optimizing computer networks, social networks, and transportation systems.
- **Pathfinding**: GPS systems use shortest path algorithms (e.g., Dijkstra) to find the best route.
- **Scheduling and Planning**: Topological sorting is used in project management tools like CPM (Critical Path Method).
- **Web Search**: Search engines use graph algorithms like PageRank to rank webpages based on their link structure.
- **Circuit Design**: Graph theory is used in the design and analysis of electronic circuits.

- **Molecular Biology**: Graph theory helps model interactions between proteins and genes, as well as pathways in metabolic networks.

7. Graph colouring

Graph colouring is the process of assigning colours to the vertices (or sometimes edges) of a graph such that no two adjacent vertices share the same colour. The goal is to minimize the number of colours used while still satisfying this constraint. It has applications in scheduling, register allocation in compilers, map colouring, and more.

There are several algorithms for graph colouring, each with different efficiencies and approaches. Here's an overview of the key algorithms for graph colouring:

1. Greedy Coloring Algorithm

This is one of the simplest graphs colouring algorithms. It attempts to assign the smallest possible colour to each vertex in a greedy manner, without looking ahead.

Steps:

1. Pick a vertex (arbitrary or based on some ordering).
2. Assign the smallest colour (starting with 1) that has not been assigned to its neighbours.
3. Repeat for all vertices.

Advantages:

- Simple and easy to implement.
- Works well for small graphs.

Disadvantages:

- Not optimal; it doesn't always produce the minimum number of colours.
- Its performance depends heavily on the ordering of vertices.

Time Complexity: O(V^2), where V is the number of vertices.

2. Welsh-Powell Algorithm

This is a greedy algorithm but improves performance by sorting vertices in decreasing order of their degrees (number of adjacent vertices). High-degree vertices are coloured first, reducing the chance of having to recolour them.

Steps:

1. Sort vertices in decreasing order of degrees.
2. Colour them using the greedy approach.

Advantages:

- Often uses fewer colours than the basic greedy algorithm.

Disadvantages:

- Still not guaranteed to use the minimum number of colours.

Time Complexity: O(V^2)

3. DSatur Algorithm

The **Degree of Saturation** (DSatur) algorithm is a more sophisticated greedy approach that considers both the degree of a vertex and how many distinct colours its neighbours have (the saturation degree).

Steps:

1. Assign a colour to the vertex with the highest degree.
2. For subsequent vertices, choose the one with the highest saturation (most neighbours with distinct colours).
3. If two vertices have the same saturation, choose the one with the highest degree.
4. Continue until all vertices are coloured.

Advantages:

- Often uses fewer colours than standard greedy methods.
- Works well on dense graphs.

Disadvantages:

- More computationally intensive compared to the greedy algorithm.

Time Complexity: O(V^2)

4. Backtracking Algorithm

This algorithm finds the optimal (minimum number of colours) solution but does so by exhaustively trying all possibilities. It's used for smaller graphs where finding the chromatic number is important.

Steps:

1. Start by assigning a colour to a vertex.
2. Recursively assign colours to the remaining vertices, backtracking when a conflict arises.
3. If all vertices are coloured without conflict, return the solution.

Advantages:

- Guarantees the optimal solution.

Disadvantages:

- Computationally expensive, exponential in nature.

Time Complexity: O(m^V) where mmm is the number of colours and V is the number of vertices.

5. Coloring Heuristics

Some heuristics attempt to improve efficiency and reduce the number of colours:

- **Largest First**: Colour vertices in order of decreasing degrees.
- **Smallest Last**: Colour vertices in reverse order of their degrees.
- **Random**: Randomly order vertices and apply greedy colouring.
- **Least-Saturation-First (LSF)**: Similar to DSatur but considers the least saturation for preliminary stages.

6. Advanced Techniques

Integer Linear Programming (ILP)

Graph colouring can be formulated as an optimization problem using ILP. This approach is used for finding exact solutions for graph colouring problems, particularly for small graphs.

Simulated Annealing & Genetic Algorithms

These are metaheuristic techniques used to approximate optimal solutions for large or complex graphs where exact solutions are computationally prohibitive.

7. Applications of Graph Coloring

- **Scheduling Problems**: Assigning time slots to exams or courses such that no two adjacent tasks (e.g., exams involving the same students) share the same time slot.

- **Register Allocation**: In compiler design, graph colouring is used to allocate variables to a limited number of CPU registers.
- **Map Coloring**: The Four-Color Theorem states that no more than four colours are needed to colour any map (planar graph) so that no adjacent regions share the same colour.
- **Frequency Assignment**: In wireless networks, graph colouring is used to assign frequencies to transmitters such that no two nearby transmitters use the same frequency.

8. Graph representations

An **adjacency matrix** is a 2D array (or matrix) of size V×V, where V is the number of vertices in the graph.

An **adjacency list** is an array (or list) of lists. Each element in the main list corresponds to a vertex, and each vertex has its own list of adjacent vertices (or edges).

An **edge list** is a list of all edges in the graph. Each edge is represented as a pair (or tuple) of vertices, and in a weighted graph, each edge also stores its weight.

An incidence matrix is a matrix of size V×E, where V is the number of vertices and E is the number of edges.

The **adjacency set** is a variation of the adjacency list, where instead of storing adjacent vertices in a list, we store them in a set. This makes lookups for adjacency faster (average O(1) time).

Representation	Space Complexity	Time Complexity (Check if edge exists)	Time Complexity (Find all neighbours)
Adjacency Matrix	O(V^2)	O(1)	O(V)
Adjacency List	O(V+E)	O(degree(v))	O(degree(v))
Edge List	O(E)	O(E)	O(E)
Incidence Matrix	O(V×E)	O(E)	O(V×E)
Adjacency Set	O(V+E)	O(1) on average	O(degree(v))

10. Conclusion

To conclude, there wasn't enough time to finish the solution. But I was able to draw the initial graph from it and implement the Dijkstra's algorithm because we hadn't done colouring algorithms. Therefore, I was in the middle of implementing the gluttonous and backtracking algorithm.