Practical Assignment No. 1 (ACT)

Title: Vertex Cover problem

Problem Statement: Implement an approximation algorithm for the vertex cover problem.

Objective: To apply algorithmic strategies for solving the problems.

Outcome: CO513.1: Solve the problems using appropriate algorithmic strategies.

Theory

Need for Approximation Algorithms

1. Many optimization problems (like NP-hard problems) are computationally infeasible to solve exactly for large inputs.
2. Approximation algorithms provide near-optimal solutions in polynomial time.
3. They are essential when real-time processing is needed.
4. Used when heuristics do not guarantee performance.
5. Provide bounds on how far the approximate solution is from the optimal.

Problems That Can Be Solved Using Approximation Algorithms

1. Vertex Cover
2. Travelling Salesman Problem (TSP)
3. Knapsack Problem
4. Set Cover Problem
5. Job Scheduling Problem

Approximation Algorithm for Vertex Cover

1. Initialize the vertex cover set as empty.
2. While the graph has edges:
    - Pick any edge (u, v).
    - Add both u and v to the vertex cover.
    - Remove all edges incident to either u or v.
3. Return the set as the approximate vertex cover.

This algorithm guarantees a solution within a factor of 2 of the optimal solution.

Analysis of Approximation Algorithm for VC with an Example

Example graph:

```
Vertices: {1, 2, 3, 4}
Edges: {(1,2), (1,3), (2,4), (3,4)}
```

Step 1: Pick (1,2) → Add 1 and 2 → Remove edges with 1 or 2
Remaining edge: (3,4)
Pick (3,4) → Add 3 and 4
Final vertex cover: {1,2,3,4}

Optimal cover is {1,4} or {2,3} with size 2.
Approximation result size: 4 (≤ 2× optimal)

Results

Graph Size	Time (s)
10	0.000101
50	0.000105
100	0.000403
200	0.000601
500	0.000901

Analysis: The time taken increases almost linearly with the graph size, indicating that the approximation algorithm works efficiently even for larger inputs.

Conclusion: The approximation algorithm for the Vertex Cover problem provides a quick and scalable solution, suitable for large graphs where exact solutions are computationally expensive.

References

1. Introduction to Algorithms, MIT Press.
2. Artificial Intelligence & Deep Learning Virtual Lab
3. https://www.geeksforgeeks.org/introduction-and-approximate-solution-for-vertex-cover-problem

Practical Assignment No. 2

Title: Karger’s Min-Cut Algorithm

Problem Statement: Implement randomized Karger’s Min-Cut algorithm and analyse the solution.

Objective: To apply algorithmic strategies for solving the problems.

Outcome: CO513.2: Analyse the solution for a given problem using different algorithms.

Theory

Need for Randomized Algorithms

1. Randomized algorithms often simplify complex problems.
2. They provide high performance in expected time even for large input sizes.
3. They are useful when deterministic solutions are too slow or hard to implement.
4. They are crucial in cryptography and probabilistic data structures.
5. They offer high-probability guarantees for approximate or exact answers.

Problems Solved Using Randomized Algorithms

1. Karger’s Min-Cut Problem
2. Randomized QuickSort
3. Monte Carlo Simulation
4. Miller-Rabin Primality Test
5. Hashing and Load Balancing in Data Structures

Karger’s Min-Cut Algorithm

Karger’s algorithm finds the minimum cut in an undirected graph using a **random contraction** approach:

1. Randomly pick an edge and contract it, merging the two vertices.
2. Remove any self-loops formed.
3. Repeat until only two vertices remain.
4. The number of edges between the final two vertices is the min-cut.

This algorithm is **probabilistic** and should be run multiple times to increase the probability of finding the actual min-cut.

Analysis of Karger's Algorithm (Example)

Graph:
Vertices = {1, 2, 3, 4}
Edges = {(1,2), (1,3), (2,3), (2,4), (3,4)}

Steps:

- Pick random edges and contract until only two vertices remain.
- Run the process multiple times to increase chances of finding actual min-cut.

Output:
Minimum cut could be 2 edges separating (1,2) and (3,4) depending on random choices.

Results

Graph Size	Time (s)
10	0.0020
20	0.0021
40	0.0060
60	0.0110
80	0.0101

Analysis: The runtime increases with graph size, but remains efficient due to the simple randomized nature of the algorithm.

Conclusion: Karger's algorithm provides a fast and practical method for approximating min-cuts in undirected graphs, especially when run multiple times.

References

1. Clifford Stein, Introduction to Algorithms
2. https://www.geeksforgeeks.org/kargers-algorithm-minimum-cut
3. Algorithms and Applications, 3rd Edition, Springer

Practical Assignment No. 3 (ACT)

Title: Max Flow Problem

Problem Statement: Implement Push Relabel Algorithm for solving Max Flow Problem and analyse the solution.

Objective: To apply algorithmic strategies for solving the problems.

Outcome: CO513.2: Analyse the solution for a given problem using different algorithms.

Theory

Max Flow Problem

The Max Flow problem involves finding the maximum amount of flow that can be sent from a source node to a sink node in a flow network, such that:

- The flow on an edge does not exceed its capacity.
- The flow is conserved at every vertex except the source and sink.

Applications include network routing, transportation systems, and supply chain logistics.

Push Relabel Algorithm

Unlike augmenting path methods (like Ford-Fulkerson), Push-Relabel operates by maintaining a **preflow** and a **height label** for each vertex:

1. Initialize preflow by pushing as much flow as possible from the source.
2. Maintain height function to guide flow direction.
3. **Push** excess flow to neighbors if they are at a lower height.
4. **Relabel** a vertex if no push is possible.

This approach is efficient for dense graphs and has a time complexity of $\mathbf{O(V^2E)}$.

Example for Push Relabel Graph:

Vertices: {0, 1, 2, 3}
Edges:
0 → 1 (capacity: 10)
0 → 2 (capacity: 5)
1 → 2 (capacity: 15)
1 → 3 (capacity: 10)
2 → 3 (capacity: 10)

Source = 0, Sink = 3

Using Push-Relabel, the maximum flow from 0 to 3 is 15.

Results

Graph Size	Time (s)
10	0.002
20	0.007
30	0.015
40	0.030
50	0.056
80	0.010

Analysis and Conclusion

Analysis: As the number of vertices increases, the time to compute maximum flow increases gradually, showcasing the algorithm's polynomial-time behavior.

Conclusion: The Push Relabel algorithm is efficient and performs well for dense graphs, offering an effective solution to the Max Flow problem.

References

1. Vijay Vazirani, Approximation Algorithms, Springer.ISBN-13978-3642084690.
2. https://www.geeksforgeeks.org/push-relabel-algorithm-set-1-introduction-and-illustration
3. Motwani R and Raghavan P, Randomized Algorithms

Practical Assignment No. 4

Title: Integer Factorization Problem

Problem Statement: Implement the solution for integer factorization problem and analyze the solution.

Objective: To apply algorithmic strategies for solving the problems.

Outcome: CO513.2: Analyse the solution for a given problem using different algorithms.

Software/Hardware Requirements: Python ,GCC/Java

Theory

Definition of Integer Factorization Problem

The integer factorization problem involves breaking down a composite number into a product of smaller integers, ideally primes. It is crucial in number theory and cryptography, especially RSA encryption.

Algorithms for Integer Factorization

1. Trial Division

Divides the number by all integers up to its square root.

- **Time Complexity:** O(√n)
- **Best For:** Small numbers

2. Pollard's Rho Algorithm

A probabilistic algorithm using Floyd's cycle detection to find non-trivial divisors.

- **Time Complexity:** O(n^1/4)
- **Best For:** Medium-size integers

3. Fermat's Factorization

Finds two numbers `a` and `b` such that `n = a^2 - b^2`.

- **Time Complexity:** Depends on closeness of factors
- **Best For:** Numbers with close factors

Input Numbers

Index	Number
1	10403 (101×103)
2	14351 (113×127)
3	19043 (137×139)
4	22499 (149×151)
5	25691 (157×163)
6	28891 (167×173)
7	32499 (179×181)
8	36863 (191×193)
9	39203 (197×199)
10	47053 (211×223)

Results Table (Sample)

Number	Trial (s)	Pollard (s)	Fermat (s)
10403	0.0005	0.0002	0.0003
14351	0.0006	0.0003	0.0004
...	...	...	...
47053	0.0013	0.0005	0.0010

Analysis:
Trial Division performs well for small integers but slows down as numbers grow. Pollard's Rho offers quick results in most cases. Fermat's method works best when factors are close.

Conclusion:
For large composite integers, **Pollard's Rho** provides a good balance between speed and accuracy, making it suitable for cryptographic applications.

References

1. Algorithm Design: Foundations, Analysis and Internet Examples
2. Algorithmics: Theory and Practice by Brassard and Bratley
3. https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/