Algorithm Design and Analysis

Assignment 3

Deadline: May 7, 2024

1. (25 points) Given an undirected graph $G = (V, E)$, a *perfect matching* is a set of edges that touches each vertex exactly once. Design an $O(n)$ time algorithm that, given a *tree* as input, decides if it has a perfect matching, where n is the number of vertices. Prove the correctness of your algorithm.

   Answer:

   Algorithm:

   - Check Tree Conditions: Verify that the tree T has an even number of vertices (n). If n is odd, return false as a perfect matching cannot exist.
   - Identify Leaf Nodes: Traverse the tree T to identify all leaf nodes. Leaf nodes are vertices with degree 1 (i.e., they are connected to only one other vertex).
   - Construct Perfect Matching: Initialize an empty set M to store the edges of the perfect matching.
   - Iteratively Remove Leaf Nodes:

     While there are leaf nodes remaining in T, For each leaf node v:

     - Let u be its neighbor (parent) in the tree.
     - Add the edge (u, v) to M (include this edge in the perfect matching).
     - Remove the edge (u, v) from T.
     - Remove all other edges incident to u (except the one connecting to v) to maintain acyclicity.
     - Remove the leaf node v and its incident edge from T.
   - Check Remaining Nodes:

     After all leaf nodes have been processed: If T is empty (no remaining vertices), return true indicating a perfect matching exists. If T is not empty, return false as there are unmatched vertices left.

   Time Complexity Analysis:

   - Traversal and Leaf Identification: $O(V)$ where V is the number of vertices.
   - Iteratively Removing Leaves and Edges: Each edge is processed once, so this step also takes $O(V)$ time.

   Correctness of the Algorithm:

   Necessity: If T has an odd number of vertices, then there cannot be a perfect matching. The process of removing leaf nodes and their connecting edges ensures that each vertex

is matched exactly once in the perfect matching, provided that T has an even number of vertices.

Sufficiency: If after removing all leaf nodes, the remaining tree T is empty, then a perfect matching exists. This is because each vertex in T must have been matched during the leaf removal process. If there are unmatched vertices remaining in T after processing all leaves, then it's impossible to form a perfect matching.

2. (25 points) A *matroid* is a pair $(U, \mathcal{I})$ where U is a set of finitely many elements and $\mathcal{I}$ is a collection of subsets of U, called *the independent sets*, that satisfies the following conditions:

   1. $\emptyset \in \mathcal{I}$.
   2. If $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$.
   3. If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there exists an element $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

   For example, the pair $(U, \mathcal{I})$ with $U = \{1, 2, 3\}$ and $\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ is a matroid, and it can be verified that the above three conditions are satisfied. The seven subsets of U in the collection $\mathcal{I}$ are called the independent sets.

   (a) Given a matroid $(U, \mathcal{I})$, an independent set $S \in \mathcal{I}$ is *maximal* if there does not exist $x \in U \setminus S$ such that $S \cup \{x\} \in \mathcal{I}$. Prove that every maximal independent set has the same size.

   (b) Given an undirected connected graph $G = (V, E)$, let $\mathcal{I}$ be the collection of the edge subsets $S \subseteq E$ such that the edges in each $S \in \mathcal{I}$ do not form any cycle. Prove that $(E, \mathcal{I})$ is a matroid and the maximal independent sets are the spanning trees.

   (c) Consider a matroid $(U, \mathcal{I})$ and a weight function $w : U \to \mathbb{Z}$ that assigns a weight to each element of U. Prove that the following algorithm correctly finds a maximal independent set S with minimum total weight $\sum_{i \in S} w(i)$.

      1. Sort the elements of U by the weight-ascending order, and let $(1, \ldots, n)$ be the sorted list.
      2. Initialize $S \leftarrow \emptyset$.
      3. **for** each $i = 1, \ldots, n$:
      4. if $S \cup i \in \mathcal{I}$, update $S \leftarrow S \cup \{i\}$; otherwise, do nothing.
      5. **return** S.

      Remark: If $(U, \mathcal{I})$ is the matroid defined in Part (b), then this algorithm is exactly Kruskal's algorithm.

Answer:

(a)To prove that every maximal independent set in a matroid $(U, \mathcal{I})$ has the same size, consider the following argument:

Suppose S and T are two different maximal independent sets in $\mathcal{I}$ with $|S| < |T|$. By the definition of a matroid, for any $x \in T \setminus S$, we must have $S \cup x \notin \mathcal{I}$, otherwise S would not be maximal. However, this contradicts the maximality of T because T should be the largest independent set. Therefore, $|S|$ cannot be less than $|T|$.

Now suppose $|S| > |T|$. By a similar argument, for any $y \in S \setminus T$, we must have $T \cup y \notin \mathcal{I}$, contradicting the maximality of S. Hence, $|S|$ cannot be greater than $|T|$.

Therefore, if S and T are both maximal independent sets, they must have the same size.

(b)To prove that $(E, \mathcal{I})$ where $\mathcal{I}$ consists of subsets of edges that do not form any cycle is a matroid, and that the maximal independent sets are the spanning trees:

Matroid Properties:

Condition 1: Clearly, the empty set $\emptyset \in \mathcal{I}$ as it contains no edges and thus no cycles.

Condition 2: If $A \in \mathcal{I}$ and $B \subseteq A$, then B also does not form any cycle, hence $B \in \mathcal{I}$.

Condition 3: Suppose $A, B \in \mathcal{I}$ with $|A| < |B|$. Adding an edge $e \in B \setminus A$ to A does not form a cycle (because B itself does not have a cycle), thus $A \cup e \in \mathcal{I}$.

Maximal Independent Sets as Spanning Trees:

An independent set $S \subseteq E$ in $\mathcal{I}$ (i.e., S contains no cycles) that spans all vertices (i.e., S forms a connected subgraph) is a spanning tree of G.

To see why, note that a spanning tree is a tree that includes all vertices of G and is acyclic. If S were not a tree (e.g., contained a cycle), then removing any edge from this cycle would keep the graph connected but without the cycle, thus contradicting maximality. Therefore, every maximal independent set (in this case, a spanning tree) is indeed a spanning tree of G.

(c) Consider a matroid $(U, \mathcal{I})$ with a weight function $w : U \to \mathbb{Z}$. We want to prove that the algorithm correctly finds a maximal independent set S with minimum total weight $\sum_{i \in S} w(i)$.

Sorting and Initialization: We start by sorting the elements of U in weight-ascending order and initialize S as an empty set.

Greedy Selection: We iterate through the sorted list of elements. For each element i: If adding i to S keeps S in $\mathcal{I}$ (i.e., $S \cup i \in \mathcal{I}$), we update S to include i.

Correctness of Algorithm: The algorithm essentially implements a greedy strategy to construct a maximal independent set S. By selecting elements in ascending order

of weight and adding them to S if they maintain independence, we ensure that S is maximal within $\mathcal{I}$. Since $(U, \mathcal{I})$ is a matroid, the exchange property guarantees that adding the next smallest weighted element that maintains independence will not violate the maximality of S. At the end of the algorithm, S will be a maximal independent set, and the total weight $\sum_{i \in S} w(i)$ will be minimized because of the greedy selection based on element weights.

3. (25 points) Given an undirected graph $G = (V, E)$, a *vertex cover* is a subset of vertices $S \subseteq V$ such that each edge has at least one endpoint included in S. The *minimum vertex cover problem* takes an undirected graph $G = (V, E)$ as input and outputs a vertex cover S with minimum size $|S|$.

   Prove that the following algorithm is a $(\ln |E|)$-approximation algorithm.

   1. Initialize $S \leftarrow \emptyset$.
   2. **while** S is not a vertex cover
   3. find a vertex x with the largest degree
   4. update $S \leftarrow S \cup \{x\}$
   5. update G by removing vertex x and all the edges incident to x
   6. **endwhile**
   7. **return** S.

   Answer:

   To prove that the given algorithm is a $(\ln |E|)$-approximation algorithm for the minimum vertex cover problem, we will follow these steps:

   1. **Define the Algorithm and Notation:** Let OPT be the size of the optimal (minimum size) vertex cover. Let S be the set of vertices maintained by the algorithm during its execution. $G = (V, E)$ is the input graph.

   2. **Algorithm Analysis:** The algorithm maintains a set S which starts empty and iteratively adds vertices until S forms a vertex cover. In each iteration, the algorithm adds a vertex x to S which has the maximum degree among the remaining vertices in G. After adding x to S, the algorithm removes x and all edges incident to x from G.

   3. **Properties of the Algorithm:** Let x be the vertex added to S in an iteration. The degree of x, denoted as $\deg(x)$, is the number of edges incident to x. The algorithm ensures that each edge removed from G by adding x to S is covered by S, thus maintaining the vertex cover property.

4. **Analysis of Approximation Factor:** Let $x_1, x_2, \ldots, x_k$ be the sequence of vertices added to S during the execution of the algorithm. At each step i, the degree of the vertex x_i, $\deg(x_i)$, is at least the number of edges incident to x_i that are not covered by the vertices already in S. Therefore, after adding x_i to S, at least $\deg(x_i)$ edges are covered by S.

5. **Bounding the Size of S:** Let m_i be the number of edges incident to x_i that are not yet covered by vertices in S at the time of adding x_i to S. Then, $m_i \geq \deg(x_i)$ since each edge incident to x_i contributes to its degree. The total number of edges covered by S after adding all vertices in the sequence $x_1, x_2, \ldots, x_k$ is at least $\sum_{i=1}^{k} \deg(x_i)$.

6. **Approximation Analysis:** Consider the total number of edges in E, denoted as $|E|$. The sum of degrees of all vertices in G (i.e., twice the number of edges) is $2|E|$. By adding vertices to S with the largest degree in each iteration, the total number of edges covered by S is at least $\sum_{i=1}^{k} \deg(x_i) \geq \frac{1}{2} \sum_{i=1}^{k} m_i$. Therefore, $|S| \leq 2 \sum_{i=1}^{k} \deg(x_i) \leq 2 \ln |E| \cdot OPT$.

7. **Conclusion:** The algorithm produces a vertex cover S such that $|S| \leq 2 \ln |E| \cdot OPT$. Thus, the algorithm is a $(\ln |E|)$-approximation algorithm for the minimum vertex cover problem, meaning that the size of S is at most $2 \ln |E|$ times the optimal size OPT of the minimum vertex cover.

4. (25 points) Consider the minimum vertex cover problem defined in the previous question and suppose G is a connected undirected graph. Consider the following algorithm. Find a DFS-tree T of G and output the set of all internal nodes of T (an internal node of a tree is a node that is not a leaf).

   (a) Prove that this is a 2-approximation algorithm.

   (b) Does the algorithm still work if we use BFS-tree instead?

   Answer:

   (a)

   1. **Algorithm Description:** The algorithm constructs a Depth-First Search (DFS) tree T of the graph G. It then outputs all internal nodes of T as the vertex cover.

   2. **Understanding DFS Tree:** - In a DFS tree, internal nodes (non-leaf nodes) correspond to vertices which are "visited" more than once during the DFS traversal. These nodes are part of the backbone of the tree structure.

   3. **Vertex Cover Size Bound:** - Consider any edge uv in G. In the DFS tree T, either u or v (or both) will be internal nodes (non-leaf nodes). Therefore, for each edge uv in G, at least one of u or v will be in the set of internal nodes of T. This means the set of internal nodes of T is a vertex cover of G.

4. **Approximation Ratio:** Let OPT be the size of a minimum vertex cover in G. Every edge in G is covered by the set of internal nodes of T. The number of edges in G is at most $2 \times$ number of vertices in T (since each internal node contributes to covering exactly two distinct vertices across the tree edges). Hence, the number of internal nodes in T, which is the size of the vertex cover produced by the algorithm, is at most $2 \times |OPT|$. Therefore, the algorithm provides a 2-approximation: $|C| \leq 2 \times |OPT|$.

(b)

1. **Algorithm Description:** Instead of a DFS tree, suppose we use a Breadth-First Search (BFS) tree T of G and output all internal nodes of T.

2. **BFS vs DFS for Vertex Cover:** The key idea of the original algorithm relies on internal nodes of the DFS tree due to the nature of DFS visiting. In DFS, the internal nodes are those visited more than once, indicating a different traversal pattern compared to BFS. In BFS, internal nodes (non-leaf nodes) might not directly correspond to a similar concept of traversal dependence as in DFS.

3. **Vertex Cover Using BFS Tree:** While BFS does ensure that all edges are covered (like DFS), the structure of the resulting tree and its internal nodes may not provide the same guarantee of a 2-approximation. The BFS tree's internal nodes might not cover the graph as efficiently as the DFS tree's internal nodes.

4. **Conclusion:** The algorithm's effectiveness relies on the specific properties of DFS and the structure of its resulting tree, which ensures a clear relationship between internal nodes and edge coverage. Using a BFS tree may not yield the same approximation guarantee or efficiency in terms of vertex cover size, as it lacks the same traversal characteristics that make the DFS tree's internal nodes effective for the vertex cover purpose.

Therefore, the algorithm is specific to DFS trees for its approximation properties, and substituting BFS trees would likely not result in a similar 2-approximation guarantee.

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

   It takes me about 6 hours to finish this assignment on a single day.For me,question 1 and 2 are easy to understand and think about solutions while question 3 is so confusing that I even have no idea whether my thinking is in the right direction.Though working out solution, Systematically describing my answers is sometimes Obstructed.Therefore,I turn to ChatGPT for organizing and presenting my thoughts.