Design and Analysis of Algorithms (CS345A)

Practice-sheet : **Fobinacci Heap and NP-completeness**

Fibonacci Heaps

1. **Tinkering with Fibonacci heap**
   In the Fibonacci heap discussed in the class, as soon as a marked node v loses its second child, the subtree rooted at v is cut from its parent and added to the root list. What if the subtree rooted at a marked node v is cut from its parent and added to the root list only when it loses its 3rd child ? Will all the bounds still hold on the amortized time complexity of various operations on Fibonacci heap ? Give rigorous mathematical arguments to support your claim.

2. **Core property of Fibonacci heap**
   This problem is directly from the lectures. Let v be any node in Fibonacci heap. Show that the degree of v is $O(\log(s(v))$, where $s(v)$ is the size of the subtree rooted at v.

3. **A short and clean code for Decrease-key in Fibonacci Heap**
   Write a neat pseudo code for the Decrease-key(H, x) in a Fibonacci Heap ?

4. **Delete-key in a Fibonacci heap**
   Design an efficient algorithm for deleting an element from a Fibonacci Heap. The amortized cost must be $O(\log n)$.

5. **A surprising property for Fibonacci Heap**
   Let v be any node in a Fibonacci heap. We showed that if the size of the subtree rooted at v is m, then the degree of v is $O(\log m)$. Can we say the same thing about the height as well ? That is, will the height of v be bounded by $O(\log m)$? Note that all operations, including merging of Fibonacci heaps is allowed.
   Hint: There exists a sequence of operations that may result in a Fibonacci heap which will be a single tree that is just a vertical chain of m elements. Invent one such sequence.

1 NP-completeness

1. **Polynomial reduction $\leq_P$**

   Let A and B be any two computational problems. Let χ be any algorithm for solving B. Problem A is said to be reducible to problem B in polynomial time if each instance I of A can be solved by

   - A polynomial number of executions of χ on instances (of B) each of which are also polynomial of size of I,
   - and, if required, basic computational steps (each taking $O(1)$ time) which are also polynomial in the size of I.

   Convince yourself that this definition of $\leq_P$ subsumes the definition of polynomial time reducibility discussed in the class.

2. **Feedback set**

   Given an undirected graph $G = (V, E)$, a *feedback* set is a set $X \subseteq V$ with the property that $G - X$ has no cycle. The *Undirected Feedback Set Problem* asks: Given G and k, does there exist a feedback set of size at most k ? Prove that *Undirected Feedback Set Problem* is NP-complete.
   Hint: Reduce vertex cover problem to Feedback set problem. The reduction will be similar or same as used in some example discussed in the class.

3. **Subgraph Isomorphism**

   Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. G is said to be isomorphic to G' if we can obtain G' from G by renaming its vertices suitably. In formal words, it means the following.

   A 1-1 and onto function $f : V \to V'$ is said to be an isomorphism if for each pair of vertices $u, v \in V$, $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$.

   Subgraph-Isomorphism Problem is defined as follows. Given any two graphs $G = (V, E)$ and $G' = (V', E')$, does there exist any subgraph of G which is isomorphic to G'. Show that *Subgraph-Isomorphism Problem* is NP-complete.
   Hint: Reduce independent set problem or Hamiltonian cycle problem to subgraph isomorphism problem.

4. **Clique Problem**

   A clique is a complete graph (edge exists between each pair of its vertices). Consider the following problem: Given an undirected graph $G = (V, E)$ and an integer k, does G contain a clique of size k ?

   Show that this problem is NP-complete.

   Hint: Use the fact that *Independent Set* is NP-complete.

5. **Approximation Algorithm for Vertex Cover**

   In the course, we discussed bipartite-matching problem. The notion of matching can be extended naturally to any arbitrary undirected graph.

   Now consider the following algorithm for computing vertex cover for a given graph $G = (V, E)$:

(a) $S \leftarrow \emptyset$;

(b) Compute maximum matching $\mathcal{M}$ of G;

(c) For each edge $(u, v) \in \mathcal{M}$ do : $S \leftarrow S \cup \{u, v\}$

(d) return S.

Show that the above algorithm computes a vertex cover. Prove that the size of vertex cover returned by the algorithm is at most twice the size of the optimal (minimum size) vertex cover.

Important Note: We discussed the area of approximation algorithm very briefly. So only simple exercises on approximation algorithms and NP-completeness, if at all, may be expected in the exam.