Design and Analysis of Algorithms

Practice-sheet 5 : **Amortized Analysis, Fibonacci Heap, and NP-completeness**

Date: *8th November, 2015*

1 Amortized Analysis

Note: There were 3-4 homework problems on amortized analysis given in the Lectures. Make sure you attempt them as well.

1. **Deleting half elements**

   Design a data structure to support the following two operations for a dynamic multiset S of integers, which allows the duplicate values:

   - *Insert*(S, x): inserts x into S.
   - *Delete-Larger-Half*(S): delete the largest $\lceil |S|/2 \rceil$ elements from S.

   Explain how to implement this data structure so that any sequence of m *Insert* and *Delete-Larger-Half* operations run in $O(m)$ time. Your implementation should also include a way to output the elements of S in $O(|S|)$ time.

2. **Multi-stack**

   A multistack consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2}.) Moving a single element from one stack to the next takes $O(1)$ time.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

3. **De-amortizing**

   Recall the dynamic tables we discussed in the class. Suppose the operations are only insertion of elements. We know how to achieve amortized $O(1)$ time per insertion. Dynamic tables have applications in In real systems. In these systems, it is not acceptable to achieve fast amortized $O(1)$ time per insertion. Instead, we need a worst case guarantee on the time per insertion. How will you achieve worst case $O(1)$ time per insertion ? Note that the space utilization must be at least a given constant at each stage.
   Hint: De-amortize the algorithm discussed in the class at the expense of a slight reduction in the utilization factor.

4. **Cyclic pattern matching**

   Give a linear time algorithm to determine whether a text T is a cyclic rotation of another string T'. For example, *arc* and *car* are cyclic rotations of each other.
   Hint: Reduce this problem to the pattern matching problem discussed in the class.

2 Fibonacci Heap

1. **Tinkering with the marked nodes**

   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 ?

2. **A short and clean code for Decrease-key in Fibonacci Heap**

   Write a neat pseudocode for the Decrease-key(H, x, Δ) in a Fibonacci Heap ?

3. **Delete-key in a Fibonacci heap**

   Design an efficient algorithm for deleting an element from a Fibonacci Heap. The amortized cost must be $O(\log n)$.

4. **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 m 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 Fibinacci heap which will be a single tree that is just a vertical chain of m elements. Invent one such sequence.

3 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. **Application of $\leq_P$**

   Let problem A be defined as follows. Given any undirected graph and an integer k, determine if the graph has an independent set of size at least k.

   Let problem B be defined as follows. Given any undirected graph and an integer t, determine if the graph has a vertex cover of size k.

   Using the definition of $\leq_P$ given in the previous exercise, show that $A \leq_P B$.

3. **Resolving whether $P = NP$?**

   For each of the two questions below, decide whether the answer is (i)**yes**, (ii)**no**, (iii) **unknown**, because it would resolve the question of whether "P=NP". Give a brief explanation of your answer.

   (a) Let us define the decision version of the Interval Scheduling Problem (discussed under the topic of Greed algorithms) as follows: Given a collection of Intervals on a time-line, and an integer k, does the collection contain a subset of nonoverlapping intervals of size at least k ?

   Question: Is it the case that *Interval Scheduling* $\leq_P$ *Vertex Cover* ?

   (b) Question: Is it the case that *Independent Set* $\leq_P$ *Interval Scheduling* ?

4. **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.

5. **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.

6. **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.