Assignment 3

Algorithm Design and Analysis

October 14, 2016

Notice:

1. - **Due** 9:00 a.m., Nov. 28, 2016 for graduate students in UCAS;
2. Please submit your answers in hard copy AND submit a digital version to UCAS website https://www2.ucas.ac.cn/.
3. Please choose at least two problems from Problem 1-4, and choose at least one problem from Problem 5-6.
4. When you're asked to give an algorithm, you should do at least the following things:
   - Describe the basic idea of your algorithm in natural language **AND** pseudo-code;
   - Prove the correctness of your algorithm.
   - Analyse the complexity of your algorithm.

1 Greedy Algorithm

Given a list of n natural numbers d_1, d_2,...,d_n, show how to decide in polynomial time whether there exists an undirected graph $G = (V,\ E)$ whose node degrees are precisely the numbers $d_1, d_2, \cdots, d_n$. G should not contain multiple edges between the same pair of nodes, or " loop" edges with both endpoints equal to the same node.

2 Greedy Algorithm

There are n distinct jobs, labeled $J_1, J_2, \cdots, J_n$, which can be performed completely independently of one another. Each jop consists of two stages: first it needs to be *preprocessed* on the supercomputer, and then it needs to be *finished* on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed on PCs at the same time. However, the supercomputer can only work on a single job a time without any interruption. For every job, as soon as the preprocessing is done on the supercomputer, it can be handed off to a PC for finishing.

Let's say that a *schedule* is an ordering of the jobs for the supercomputer, and the *completion time* of the schedule is the earlist time at which all jobs have finished processing on the PCs. Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

3 Greedy Algorithm

Assume the coasting is an infinite straight line. Land is in one side of coasting, sea in the other. Each small island is a point locating in the sea side. And any radar installation, locating on the coasting, can only cover d distance, so an island in the sea can be covered by a radius installation, if the distance between them is at most d.

We use Cartesian coordinate system, defining the coasting is the x-axis. The sea side is above x-axis, and the land side below. Given the position of each island in the sea, and given the distance of the coverage of the radar installation, your task is to write a program to find the minimal number of radar installations to cover all the islands. Note that the position of an island is represented by its x-y coordinates.

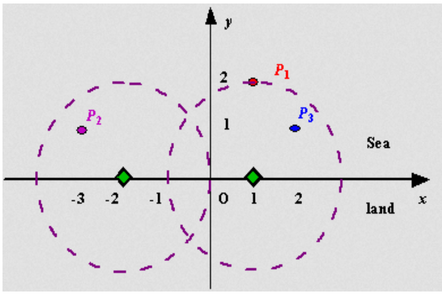

Figure 1

4 Greey Algorithm

Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an polynomial-time algorithm that will maximize your payoff.

5 Programming

Write a program in your favorate language to compress a file using Huffman code and then decompress it. Code information may be contained in the compressed file if you can. Use your program to compress the two files (*graph.txt* and *Aesop_Fables.txt*) and compare the results (Huffman code and compression ratio).

6 Programming

1. Implement Dijkstra's algorithm (using linked list, binary heap, binomial heap, and Fibonacci heap) to calculate the shortest path from node s to node t of the given graph (*graph.txt*), where s and t are randomly chosen. The comparison of different priority queue is expected.

   Note: you can implement the heaps by yourself or using Boost C++/STL, etc.

2. Figure out how many shortest paths is every node lying on in your program, except starting node s and finishing node t. For example, if there are in total three shortest paths $0 \to 1 \to 2 \to 10$, $0 \to 1 \to 3 \to 4 \to 10$ and $0 \to 1 \to 2 \to 6 \to 7 \to 10$, then 1 lies on 3 shortest paths, 2 lies on 2 shortest paths, and 3 lies on 1 shortest path, etc.