

Final Examination

CS 5114 (Spring 2009)

Assigned: May 5, 2009.

Due: at Torgerson 2160B by 5pm on May 12, 2009.

DO NOT EMAIL YOUR SOLUTIONS TO ME!

Name: _____

9-digit PID: _____

Instructions

1. For every algorithm you describe, prove its correctness and analyse its running time and space used. I am looking for clear descriptions of algorithms and for the most efficient algorithms you can come up with.
2. If you prove that a problem is \mathcal{NP} -Complete, remember to state how long the certificate is, how long it takes to check that the certificate is correct, and what the running time of the transformation is. All you need to show is that the transformation can be performed in polynomial time. Your transformation need not be the most efficient possible.
3. You may consult the textbook, your notes, or the course web site to solve the problems in the examination. You **may not** work on the exam with anyone else, ask anyone questions, or consult other textbooks or sites on the Web for answers. **Do not use** concepts from chapters in the textbook that we have not covered.
4. You must prepare your solutions digitally and submit a hard-copy.
5. I prefer that you use \LaTeX to prepare your solutions. However, I will not penalise you if you use a different system. To use \LaTeX , you may find it convenient to download the \LaTeX source file for this document from the link on the course web site. At the end of each problem are three commented lines that look like this:

```
% \solution{  
%  
% }
```

You can uncomment these lines and type in your solution within the curly braces.

6. Do not forget to staple the hard copy you hand in.

Good luck!

Problem 1 (10 points) Let us start with some quickies. For each statement below, say whether it is true or false.

1. A graph is 3-colourable if it has cycles of odd length.
2. $\sum_{i=1}^n i = \Theta(n^2)$.
3. In a directed graph, $G = (V, E)$ let $\pi(u, v)$ denote the length of the shortest path that starts at u and ends at v . If there is no such path, you can assume that $\pi(u, v) = \infty$. Since G is directed, $\pi(u, v) \neq \pi(v, u)$, in general. Then, for any three nodes $u, v, w \in V$, $\pi(u, v) + \pi(v, w) \geq \pi(u, w)$.
4. Suppose we have an algorithm that when given a graph G with n nodes and an integer k , checks if G has a vertex cover of size k or less in $O(k2^k n)$ time. Note that this time is polynomial in n (but exponential in k). Since we reduced INDEPENDENT SET to VERTEX COVER in class, we have an algorithm that can check if a graph G has an independent size of size at least l (for some integer $l > 0$) in time polynomial in n .
5. Green Kryptonite causes Superman to lose his powers.

Problem 2 (30 points) The flag of a certain populous country contains a symbol called the “Ashoka Chakra” (see the image below). This symbol has a central hub and 24 spokes. Naturally, this reminds us of a graph with 25 nodes and 48 edges, of which 24 nodes are connected by a cycle, and the 25th node is connected to each of the other 24 nodes. A *generalised k -chakra* is a graph with $k + 1$ nodes and $2k$ edges such that k nodes lie on a cycle and the $k + 1$ st node is connected to each of the other k nodes. Given an undirected graph G and an integer k , prove that the problem of determining if G contains a generalised k -chakra as a subgraph is \mathcal{NP} -Complete. (We say that a graph H is a *subgraph* of a graph G if every node in H is also a node in G and every edge in H is also an edge in G .)



Problem 3 (40 points) You are given an undirected graph $G = (V, E)$. Each vertex $v \in V$ has a label $l_v \in \{-1, 0, 1\}$. Each edge $e \in E$ has a weight $w_e > 0$. Consider the set V_0 of nodes in V whose label is 0. A *labelling* of V_0 is an assignment of 1 or -1 as the label of every node in V_0 . Note that there are $2^{|V_0|}$ possible labellings of V_0 . Your goal is to compute a labelling that takes the edge structure of G into account. For example, if a node v in V_0 has many more neighbours with label 1 than -1 , you may want to change v 's label to 1. Therefore, you decide to compute a labelling that maximises the *consistency* $c(G)$ of G , defined as

$$c(G) = \sum_{e=(u,v) \in E} w_e l_u l_v.$$

Either devise a polynomial time algorithm to maximise $c(G)$ or prove that the decision version of the problem (i.e., given a parameter κ , does G have a labelling of the nodes in V_0 such that $c(G) \geq \kappa$) is \mathcal{NP} -Complete.

Problem 4 (20 points) This problem deals with cost-effective purchase of candy. You enter a candy store. In front of you are glass jars, each filled with a different mouth-watering type of candy. You are on a strict budget: you have at most $\$b$ with you. There are n types of candy, each in a unique jar. For each $1 \leq i \leq n$, the entire jar containing the i th type of candy costs $\$c_i$ and weighs w_i kilogrammes. You are allowed to purchase as much candy as you want from each jar. You can purchase as tiny a portion of any piece of candy as you want. Put another way, you can think of the candy as being powdered¹ and you can scoop up whatever weight you want from each jar. Devise an algorithm that maximises the total weight of the candy you purchase, as long as the total cost of the candy you buy is at most $\$b$.



Solution: *I am using Mihai Alexe's solution.*

We take a greedy approach to solving this problem. We first sort the jars in decreasing order of their weight-cost ratio $\tau_i = w_i/c_i$. This step takes $\mathcal{O}(n \log n)$ time. Then, we buy as much candy as we can from the first jar (up to the entire jar); then, if we have any money left, we buy from the second jar, and so on. The algorithm returns the total weight W of the candy that we bought. As input, we provide the array of jars sorted by τ_i , along with w_i and c_i for each jar. We keep track of the money that we spend in a variable x . Naturally, $x \leq b$ at all times.

$W = \text{CostEffectiveCandyPurchase}(b)$

```

Sort the jars  $1, \dots, n$  in increasing order of  $\tau_i$ .
/*  $x = \text{money we have spent so far}$  */
/*  $W = \text{total amount of candy we bought so far}$  */
Initialize  $x = 0, W = 0$ .
For  $i = 1, \dots, n$ 
  /* Check if we're within the budget. */
  If  $x < b$ 
    /* Buy as much candy as possible from jar  $i$ . */
    Buy  $y = \min(b - x, c_i)$  dollars worth of candy from jar  $i$ .
     $W = W + y \cdot w_i/c_i$ 
     $x = x + y$ 
  Else
    Return  $W$ .
  End If
End For
Return  $W$ .

```

We claim that this greedy approach is optimal, i.e. maximizes the amount of candy we buy with $\$b$. We will give a proof based on an exchange argument. Suppose the algorithm buys a nonzero amount

¹I am not suggesting that anyone will actually want to eat powdered candy.

of candy w from jar i . We want to exchange a fraction f_i of the total amount of candy in jar i (w_i) for some candy in a jar $j > i$ (remember that if our algorithm buys candy from jar i , then it has already bought all available candy from jars $1, \dots, i - 1$). The cost of $f_i w_i$ kilograms of candy from jar i is $x = f_i c_i = f_i \frac{w_i}{\tau_i}$. Now, all the other jars j from where we could take candy instead have $\tau_j \leq \tau_i$, therefore with x dollars we can buy

$$\hat{w} = x \frac{w_j}{c_j} = f_i w_i \frac{\tau_j}{\tau_i} \leq f_i w_i = w \tag{1}$$

kilograms of candy from any such jar j . Hence x dollars buy us from jar j at most or less the amount of candy than they did for jar i , when $i < j$. Any modification to the algorithm output (i.e., any redistribution of candy purchases) would result in either a decrease of W , or in W remaining constant. Hence, the algorithm is optimal.

The run time of the algorithm is $\mathcal{O}(n \log n)$, given by the initial sorting stage. The time spent in the for loop is linear in n (each iteration takes a constant amount of time).

Note also that we are respecting the cost bounds, i.e. the total cost of the candy we buy cannot exceed b .