Course: CSC 2401 Introduction to Computational Complexity - Fall 2001
Lecture #12, Tuesday, October 23, 2001
Lecturer: Allan Borodin
Scribe: Jason Anderson

1 Subset Sum Revisited

Last time we discussed the subset sum problem, *SS*.

$$SS = \{a_1, ..., a_m; t\}$$

$$\exists I \subseteq \{1, ..., m\} : \sum_{i \epsilon I} a_i = t$$

SS is NP-complete in the weak sense. In fact, *SS* is NOT NP-hard in the strong sense. (Technically speaking we need to assume $P \neq NP$.) It is easy to reduce the *SS* problem to the following Knapsack problem. Then using dynamic programming we will show that there is a pseudo-polynmial time algorithm for the Knapsack problem.

2 The Knapsack Problem *(KP)*

The input to the problem is a set of weight/value pairs and a bound, B. Each of the weight/value pairs corresponds to an item. A feasible solution to the problem is a subset of items such that the total weight of the items in the subset does not exceed the specified bound. That is, an input, x, to the problem is:

$$x = (\{(w_i, v_i)\}, B) : 1 \leq i \leq n$$

Let L be the set of weight/value pairs, $L = \{(w_i, v_i)\}$. A feasible solution is a subset $I \subseteq \{1, ..., n\}$ such that:

$$\sum_{i \epsilon I} w_i \leq B$$

In optimization problems, we are seeking more than just a yes or no answer (as in decision problems). Specifically, in minimization problems we have a *cost function.* In maximization problems we have a *profit function.*

In the knapsack optimization problem, we have a profit function:

$$v_I = profit(I) = \sum_{i \epsilon I} v_i$$

In this optimization problem, the aim is to produce an optimal feasible solution. We use $v^*(x)$ to represent the maximum profit of any feasible solution for the input x.

Consider the knapsack optimization problem as a decision problem:

$$KP = (\{(w_i, v_i)\}, B, V) : 1 \leq i \leq n$$

$$\exists I \subseteq \{1, ..., n\} : \sum_{i \epsilon I} w_i \leq B \text{ and } \sum_{i \epsilon I} v_i \geq V$$

Clearly, if we can solve the KP optimization problem in polynomial time, we can solve the KP decision problem in polynomial time. Conversely, if we can solve the KP decision problem, we can solve the KP optimization problem.

3 The Knapsack Optimization Problem

To compute the optimal feasible solution, I^*, there are two steps:

1. Compute the optimal value (of an optimal feasible solution), $v^*(x)$. This is accomplished by using a binary search for $v^*(x)$ where the decision procedure is called as a subroutine.
2. Construct I^* item by item.

Step 2 is accomplished using the following algorithm:

$I^* = \phi$
$V = v^*(x)$ %this has been pre-computed in step 1
for each item i, $1 \leq i \leq n$
    ask if $(L - \{i\}, B - w_i, V - v_i)$ is in KP
    if YES then
        $I^* = I^* \bigcup \{i\}$
        $B = B - w_i$
        $V = V - v_i$
    $L = L - \{i\}$ % the weight/value pair corresponding to item i is removed from L

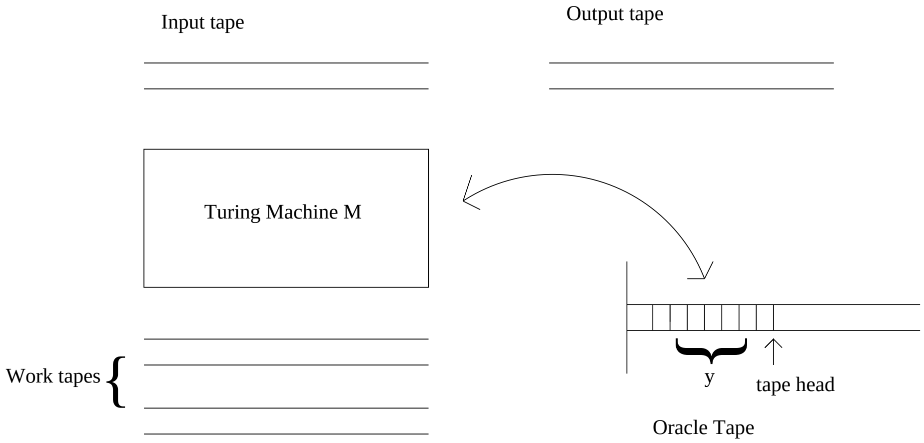
4 Polynomial Time Turing Reducibility (Cook Reducibility)

Previously we studied $\leq_P$ poly time transformability. This type of reducibility applies to languages and is also called Karp reducibility.

Intuitively, a problem $\prod_1$ is poly time Turing reducible to a problem $\prod_2$ if $\prod_1$ can be solved in poly time using a Turing machine which allows calls to a subroutine that computes $\prod_2$.

Oracle Turing machines are a formal notion, though they are essentially Turing machines that can have subroutines. Oracle Turing machines are also referred to as relativized Turing machines. The basic idea of an Oracle TM is shown in the figure below which shows a TM, M, that has access to an additional "Oracle Tape".

When the Oracle TM M enters a special query state $q?$ with say y on its oracle tape, the TM consults the oracle Π and returns the value (say z) of the



oracle for y on the oracle tape. It takes only a single step for z to appear on the Oracle tape. Then, M goes into a special "answer" state.

The notation M^{Π} is used to represent an Oracle TM where Π is a problem computed using the Oracle tape. Note that if Π is a decision problem we can use the symbol M^L for the Oracle TM, where L represents the language of the decision problem. In such cases, when M enters the special query state $q?$, it comes back in one of two special answer states, q_Y or q_N, that indicate YES or NO for the decision problem.

We used $\leq_p$ for poly time transformations. We will use $\leq_{p,T}$ for poly time Turing reductions.

In practice, when showing L_1 is poly time reducible to L_2, we only need $L_1 \leq_p L_2$ and in fact, $L_1 \leq_{logspace} L_2$ and even weaker transformations will suffice. This is especially true for showing NP-hardness.

Simple facts:

$\leq_{p,T}$ is transitive.
$L_1 \leq_{p,T} L_2$ and $L_2 \epsilon P \Rightarrow L_1 \epsilon P$

Open:

If L is NP-complete w.r.t. $\leq_{p,T}$ is it NP-complete w.r.t. $\leq_p$?

Simple fact:

$$L_1 \leq_{p,T} \overline{L_1}$$
$$\overline{L_1} \leq_{p,T} L_1$$

For $\leq_p$, we have:

If L is NP-complete then L transforms to $\overline{L}$ if and only if $NP = co - NP$ if and only if $\overline{L} \epsilon NP$.