Problem Set 2 **Due: Feb 28** **cs2me3 Winter 2005**

Instructions

1. You are encouraged to work in groups of two. If you cannot find a partner, you can work alone.
2. Please submit one copy of the assignment; if you are working with a partner, both names should appear on the assignment.
3. For **Part A** of the assignment, you must submit an electronic copy of your C program via WebCT, and a hard copy of your program should be included when you hand in the assignment.

Part A

Write a C program which on the input pair $w \in \Sigma^*$ and G, where $\Sigma = [a-z]$ (i.e., the English alphabet of lower case letters), and G is a context-free grammar in Chomsky Normal Form, outputs a derivation of w in G (if one exists, and outputs a message saying that one does not exist, if that is the case).

Your program should work as follows: `chomsky inputfile.txt`, and it should output (to the standard output) the derivation. The file `inputfile.txt` should have the following format: first line is w, the second line is `S-->AB` or `S-->a`, where A, B are any variables, and a is any terminal; however, S has to be the start variable S. Each line after that should contain a single rule.

For example, on input:

```
aab
S-->AB
A-->AA
A-->a
B-->b
```

the output of `chomsky` should be:

```
S==>AB==>AAB==>aAB==>aaB==>aab
```

(Note that if the input is longer than, say, 70 characters, it should wrap around.)

Part B

1. Recall the *All Pairs Shortest Path Problem*. In the loop computing $B(i, j)$ we stated a certain pre-condition, a loop invariant, and a post-condition. State each of them, and prove that the loop invariant holds.

   Solution: The precondition is that $\forall i, j \in [n]\ B(i, j) = A(0, i, j)$. The postcondition is that $\forall i, j \in [n]\ B(i, j) = A(n, i, j)$. The loop invariant is that after the k-th iteration of the main loop, $B(i, j) = A(k, i, j)$. To prove the loop invariant note that $B(i, j)$ is given by $\min\{B(i, j), B(i, k) + B(k, j)\}$, so the only worry is that $B(i, k)$ or $B(k, j)$ was *already* updated, so we are not getting $A(k-1, i, k)$ or $A(k-1, k, j)$ as we should, but rather $A(k, i, k)$ or $A(k, k, j)$. But, it turns out that $A(k, i, k) = A(k-1, i, k)$ and $A(k, k, j) = A(k-1, k, j)$, because the shortest path from i to k (or k to j) does *not* contain k as an intermediate node.

2. Recall the *Simple Knapsack Problem*. Consider the loop computing $S(j)$. Again, state the pre-condition, the loop invariant, and the post-condition. Prove that the loop invariant holds; pay particular attention to the correctness of our overwriting technique.

   Solution: The precondition is that for all j, $S(j) = R(0, j)$. The postcondition is that for all j, $S(j) = R(d, j)$. Let the loop invariant be the assertion that after the i-th step, $S(j) = R(i, j)$. This loop invariant holds since we start "filling" S in from the right, and we only change false to true (never true to false—the reason is that if we could build an intermediate value j with the first $(i-1)$ weights, we can certainly still construct it with the first i weights).

3. Give a recurrence for computing the array $V(i, j)$ in the General Knapsack Problem.

   Solution: $V(i, j) = 0$ if $i = 0$ or $j = 0$.

   $$V(i,j) = \begin{cases} V(i-1,j) & \text{if } j < w_i \text{ or } R(i-1, j-w_i) = \mathsf{F} \\ \max\{v_i + V(i-1, j-w_i), V(i-1,j)\} & \text{otherwise} \end{cases}$$

   To see that this works, suppose that $j < w_i$. Then weight i cannot be included, so $V(i, j) = V(i-1, j)$. If $R(i-1, j-w_i) = \mathsf{F}$, then there is no subset $S \subseteq \{1, \ldots, i\}$ such that $i \in S$ and $K(S) = j$, so again weight i is not included, and $V(i, j) = V(i-1, j)$.

   Otherwise, if $j \geq w_i$ and $R(i-1, j-w_i) = \mathsf{T}$, then weight i may or may not be included in S. We take the case which offers more value: $\max\{v_i + V(i-1, j-w_i), V(i-1, j)\}$.

4. Write the algorithm for the *Activity Selection Problem*. Also, show how to find the actual set of activities, once $A(k)$ has been computed for all k.

   Solution: We show how to find the actual set of activities:

   Suppose $k > 0$. If $A(k) = A(k-1)$, then no activity has been scheduled to end at time u_k, so we proceed recursively to examine $A(k-1)$.

   If, on the other hand, $A(k) \neq A(k-1)$, then we know that some activity has been scheduled to end at time u_k. We have to find out which one it is. We know that in this case

$$A(k) = \max_{1 \le i \le n} \{p_i + A(H(i)) | f_i = u_k\}$$

so we examine all activities i, $1 \le i \le n$, and output the (first) activity i_0 such that $A(k) = p_{i_0} + A(H(i_0))$ and $f_{i_0} \le u_k$.

Now we repeat the procedure with $A(H(i_0))$.

We end when $k = 0$.

5. Recall the *Job Scheduling with Profits Problem*. Give a recurrence for $A(i,t)$. Then write the algorithm. Finally, verify your algorithm with suitable chosen pre-condition, loop invariant, and post-condition. Show that your loop invariant holds.

   Solution: Initialization: $A(0,t) = 0$, $0 \le t \le t_n$

   To compute $A(i,t)$ for $i > 0$ first define $t_{\min} = \min\{t, t_i\}$. Now

   $$A(i,t) = \begin{cases} A(i-1,t) & \text{if } t_{\min} < d_i \\ \max\{A(i-1,t), p_i + A(i-1, t_{\min} - d_i)\} & \text{otherwise} \end{cases}$$

   Justification: If job i is scheduled in the optimal schedule, it finishes at time $t_{\min}$, and starts at time $t_{\min} - d_i$.

   If it is scheduled, the maximum possible profit is $A(i-1, t_{\min} - d_i) + p_i$. Otherwise, the maximum profit is $A(i-1,t)$.