**Name:** Harshit Jain
**Access ID:** hmj5262
**Recitation:** 8

**Problem 0** **Points:**

**Acknowledgements**

(a) I did not work in a group.

(b) I did not consult with anyone in my group members.

(c) I did not consult any non-class materials.

**Problem 1**

**Points:**

**Algorithm 1:** GREEDY-HORN

**Input :** set of Horn clauses
**Output:** either the assignment or ”unsatisfiable”

Set all variables to 0;
**while** $\exists$ *an ”* $\implies$ *” that is not satisfied* **do**
  Set its RHS to 1;
**end while**
**if** *all pure negative clauses are* 1 **then**
  **return** *the assignment*
**end if**
**else**
  **return** *”unsatisfiable”*
**end if**

(a) According to the algorithm, first set all variables to 0 $\implies$

$$w = 0, x = 0, y = 0, z = 0$$

Now there are 5 clauses having ” $\implies$ ” out of which $4^{th}$ clause ($\implies x$) is not satified. So, we will set RHS of this clause to 1, that is, **x=1**.

This will lead to reconsidering the assignment of $y$ because according to $3^{rd}$ clause ($x \implies y$), if LHS is True then RHS should be set to 1, that is, **y=1**.

This will lead to reconsidering the assignment of $w$ because according to $5^{th}$ clause ($x \wedge y \implies w$), if LHS is True then RHS should be set to 1, that is, **w=1**.

This will lead to reconsidering the assignment of $z$ because according to $1^{st}$ clause ($w \wedge y \wedge z \implies x$), if RHS is True then LHS should be resolved to 1, that is, **z=1**.

Now, pure negative clauses are failed to satisfy, so there is no satisfying assignment, hence algorithm will return **”unsatisfiable”**.

(b) According to the algorithm, first set all variables to 0 $\implies$

$$w = 0, x = 0, y = 0, z = 0$$

Now there are 4 clauses having ” $\implies$ ” out of which $4^{th}$ clause ($\implies z$) is not satified. So, we will set RHS of this clause to 1, that is, **z=1**.

This will lead to reconsidering the assignment of $w$ because according to $2^{nd}$ clause ($z \implies w$), if LHS is True then RHS should be set to 1, that is, **w=1**.

Here, $x$ and $y$ need not to be changed since the implications are still satified with having **x=0, y=0**.

Now, pure negative clauses are still 1 and hence, satisfied. So, the algorithm will return the assigment

**w=1, x=0, y=0, z=1**

**Problem 2**

**Points:**

Subproblem → Notation: Sum($j$) = maximum sum of contigious subsequence ending at index $j$ in the list $S$ where $1 \leq j \leq n$.

Recurrence → Computing Sum($j$):

$$\text{Sum}(j) = \max\{\text{Sum}(j-1) + S[j], S[j]\} \text{ where } 1 \leq j \leq n$$

Base Case: $\text{Sum}(0) = 0$, $\text{Sum}(1) = S[1]$ (indexing starts from 1 and goes till $n$)

Pseudocode:

**Algorithm 2:** Contiguous Subsequence of Maximum Sum

**Input :** A list of numbers $S = a_1, a_2, a_3, \cdots, a_n$
**Output:** The contiguous subsequence of maximum sum

Set maximumSum = Sum[1] ; `/* maximum sum of subsequence */`
Set start = 1 ; `/* starting index of max subsequence */`
Set end = 1 ; `/* ending index of max subsequence */`
**for** *j = 2 to n* **do**
    Sum[$j$] = max(Sum[$j-1$] + $S[j]$, $S[j]$)
    **if** *Sum[j] > maximumSum* **then**
        maximumSum = Sum[$j$]
        **if** *Sum[j-1] ≤ 0* **then**
            start = $j$
        **end if**
        end = $j$
    **end if**
**end for**

Explanation:

**Problem 3**

**Points:**

Subproblem → Notation: $\text{LCS}(i, j)$ = length of the longest common substring for which there are indices $i$ and $j$ with $x_i x_{i+1} \cdots x_{i+k-1} = y_j y_{j+1} \cdots y_{j+k-1}$ where $1 \le i \le n$, $1 \le j \le m$.

Recurrence → Computing LCS($i$, $j$):

$$\text{LCS}(i, j) = \begin{cases} 1 + \text{LCS}(i-1, j-1) & ; \text{if } x[i] = y[j] \\ 0 & ; \text{otherwise} \end{cases} \tag{1}$$

Base Case: $\text{LCS}(i, 0) = 0$, $\text{LCS}(0, j) = 0$

Pseudocode:

**Algorithm 3:** Longest Common Substring

**Input :** $x = x_1 x_2 \cdots x_n, y = y_1 y_2 \cdots y_m$
**Output:** $k$ = length of the longest common string

```
Set k = 0;
for i = 0 to n do
    for j = 0 to m do
        if i == 0 or j == 0 then
            LCS[i][j] = 0 ;                                  /* Base Case */
        end if
        else if x[i] == y[j] then
            LCS[i][j] = 1+ LCS[i-1][j-1] ;                   /* Recurrence Formula */
            if LCS[i][j] ≥ k then
                k = LCS[i][j] ;                              /* Optimal Solution */
                solutionRow = i ;                            /* The row holding Optimal Solution */
                solutioncolumn = j ;                         /* The column holding Optimal Solution */
            end if
        end if
        else
            LCS[i][j] = 0
        end if
    end for
end for
return k
```

Explanation:

The approach is to find the length of the longest common substring for all substrings of both the strings $x$ and $y$ and store these lengths in a table LCS. Each cell $(i, j)$ of the table LCS, that is, $\text{LCS}[i][j]$ either holds 0 if $x[i] \neq y[j]$, or holds the length of the common substring of $x[0 \cdots i]$ and $y[0 \cdots j]$ if $x[i] = y[j]$, which is made up including $x[i]$ and $y[j]$. In that way, the table keeps track of all the common substrings available and returns the largest length of common substring available in the table.

If the character happened to be the same while iterating through both the strings $x$ (interator $i$) and $y$ (interator $j$), then this algorithm checks for the similarity of until previous character of both the strings ($x[i-1] == y[j-1]$) which has already been stored in the table, and add 1 to it which signifies $x[i] = y[j]$.