CSC373 Worksheet 2 Solution

July 24, 2020

1. **Notes:**

- Greedy Algorithm
  - Always makes the choice that looks best at the moment
    - Locally optimal solution leads to globally optimal solution
- Activity-selection Problem (Greedy algorithm using dynamic programming)
  - Goal: Selecting maximum size set of mutually compatible activities

    Example:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

  - Suppose a set exists $S = \{a_1 = [s_1, f_1), a_2 = [s_2, f_2), ..., a_n = [s_n, f_n)\}$
    - a_i represents an i^{th} activity
    - s_i represents starting time
    - f_i represents finishing time
    - $0 \leq s_i < f_i < \infty$
    - $a_1, ..., a_n$ sorted in monotonically increasing order of finish time

      i.e.

      $f_1 \leq f_2 \leq f_3 \leq ... \leq f_{n-1} \leq f_n$

    - a_i and a_j are **compatible**, if intervals $[s_i, f_i)$ and $[s_j, f_j)$ don't overlap

      i.e

      $s_i \geq f_j$ and $s_j \geq f_i$

- Steps
  1. Think about dynamic programming solution
     * Construct optimal solution using two subproblems

       S_{ij}**:** activities that start after activity a_i finishes and before activity a_j starts

       i.e.

       $S_{19} = \{a_4 = [5, 7), a_6 = [5, 9), a_7 = [6, 10)\}$

       A_{ij}**:** maximum set of mutually compatible activities in S_{ij} (including a_k)
       · $A_{ik} = A_{ij} \cap S_{ik}$
       · $A_{kj} = A_{ij} \cap S_{kj}$
       · $A_{ij} = A_{ik} \cup \{a_k\} \cup Akj$
       · So, $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

     * Verify that optimal solution A_{ij} must include optimal solution to the two subproblems for S_{kj}

       Let A'_{kj} be another mutually compatible activities in S_{kj} where $|A'_{kj}| > |A_{kj}|$.

       Then we could use A'_{kj} in a solution to subproblem of S_{ij}

       Then we have $|A_{ik}| + |A'_{kj}| + 1 > |A_{jk}| + |A_{kj}| + 1 = |A_{ij}|$ mutually compatible activites

       This contradicts assumption that A_{ij} is an optimal solution

     * Verify that optimal solution A_{ij} must include optimal solution to the two subproblems for S_{ik}

       The same applies for activities in S_{ik}

  2. Observe that only one choice - greedy choice, and that when we make the greedy choice, only one subproblem remains
     * Eliminates the need to solve subproblems using recurrences
     * Typically have top-down design
       · Make a choice and then solve a subproblem
       · Dynamic programming has bottom-up approach: Solve subproblems before making a choice
     * Dynamic programming
  3. Develop recursive greedy solution
  4. Convert the recursive algorithm into iterative one