BITS, PILANI – K. K. BIRLA GOA CAMPUS

Design & Analysis of Algorithms

(CS F364)

Lecture No.10

Activity Selection Problem

- **Input: A set of activities S = {a_1,…, a_n}**
- Each activity a_i has start time s_i and a finish time f_i such that $0 \le s_i < f_i < \infty$
- An activity a_i takes place in the **half-open interval** $[s_i, f_i)$
- Two activities are **compatible** if and only if **their interval does not overlap**

  i.e., a_i and a_j *are* **compatible** if $[s_i, f_i)$ *and* $[s_j, f_j)$ **do not overlap** *(i.e.* $s_i \ge f_j$ *or* $s_j \ge f_i$ *)*
- **Output: A maximum-size subset of mutually compatible activities**

Activity Selection Problem

Example

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

A = {1,4,8,11} is an optimal solution.

A = {2,4,9,11} is also an optimal solution

Recipe for Dynamic Programming

Step 1: Identify **optimal substructure**.

Step 2: Find a **recursive formulation** for the value of the optimal solution.

Step 3: Use **dynamic programming** to find the value of the optimal solution.

(bottom–up manner)

Notation

Notation

Define $S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$

= subset of activities that **start** after activity a_i finishes and **finish** before Activity a_j starts

To avoid special cases, add fictitious activities a_0 and a_{n+1} and define $f_0 = 0$ and $s_{n+1} = \infty$.

Assume activities are sorted in order of finish time:

$$f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n < f_{n+1}$$

Optimal Substructure Property

Exercise

Recursive Formulation

- Let **c[i,j]** be the number of activities in a maximum-size subset of mutually compatible activities in S_{ii}.

$$c[i, j] = c[i, k] + c[k, j] + 1$$

- So the solution is **c[0,n+1]**=$S_{0,n+1}$.
- Recurrence Relation for **c[i, j]** is :

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{i<k<j} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

We can solve this using **dynamic programming**

But there is **important property** which we can exploit.

We Show this is a Greedy Problem

Theorem:

Consider any nonempty subproblem S_{ij}, and let a_m be the activity in S_{ij} with earliest finish time: $f_m = \min\{f_k : a_k \in S_{ij}\}$, then

1. Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.

2. The subproblem S_{im} is empty, so that choosing a_m leaves S_{mj} as the only one that may be nonempty.

Proof : See Text book

We Show this is a Greedy Problem

Consequences of the Theorem?

1. **Normally we have to inspect all subproblems, here we only have to choose one subproblem**

2. What this theorem says is that we only have to find the first subproblem with the smallest finishing time

3. This means we can solve the problem top down by selecting the optimal solution to the local subproblem rather than bottom-up as in DP

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	8	9	10	11	12	13	14

To solve the $S_{i,j}$:

1. Choose the activity a_m with the **earliest finish time**.
2. Solution of $S_{i,j} = \{a_m\}$ **U Solution of subproblem** $S_{m,j}$

To solve $S_{0,12}$, we **select** a_1 **that will finish earliest**, and **solve for** $S_{1,12}$.

To solve $S_{1,12}$, we **select** a_4 **that will finish earliest**, and **solve for** $S_{4,12}$.

To solve $S_{4,12}$, we **select** a_8 **that will finish earliest**, and **solve for** $S_{8,12}$

And so on (Solve the problem in a top-down fashion)

Elements of greedy strategy

- Determine the **optimal substructure**
- Develop the **recursive solution**
- Prove one of the optimal choices is the greedy choice yet safe
- Show that all but one of subproblems are empty after greedy choice
- Develop a recursive algorithm that implements the greedy strategy
- Convert the recursive algorithm to an iterative one.

This was designed to show the **similarities and differences** between dynamic programming and the Greedy approach; These steps can be **simplified** if we apply the Greedy approach directly

Applying Greedy Strategy

Steps in designing a greedy algorithm

The optimal substructure property holds

(same as dynamic programming)

Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.

Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.