Learn DAA : From B K Sharma

TCS-503: Design and Analysis of Algorithms

Advanced Design and Analysis Techniques: Greedy Algorithms

Learn DAA: From B K Sharma

Advanced Algorithm Design Techniques: Optimization Techniques

Dynamic Programming

Greedy Algorithms

Backtracking

Advanced Algorithm Analysis Technique

Amortized Analysis

Learn DAA: From B K Sharma

Advanced Algorithm Design Techniques: Optimization Techniques

Dynamic Programming

Optimal Substructure Property + Overlapping Substructure Property

Programming means "**Tabular Method**", not computer programming.

Works for more problems

In between, not as fast as greedy

Greedy Algorithms

Greedy Choice Property + Optimal Substructure Property

Can be applied to few problems

but gives fast algorithms

Backtracking

Can be applied to almost all problems

but gives very slow algorithms

Why Greedy Algorithm?

No direct solution available

Easy-to-implement solutions to complex, multi-step problems.

When Greedy Algorithm?

When the problem has:

Greedy Choice Property + Optimal Substructure Property

a.k.a.

Elements of GA

A good clue that a greedy strategy will solve the problem.

Greedy Choice Property

When we have a choice to make, make the one that looks best right now.

A locally greedy choice will lead to a globally optimal solution.

Make a locally optimal choice in hope of getting a globally optimal solution.

A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.

How Greedy Algorithm? What are the Steps of Greedy Algorithm?

1. Formulate the optimization problem in the form:

   we make a choice and we are left with one sub-problem to solve.

2. Show that the greedy choice can lead to an optimal solution:

   so that the greedy choice is always safe.

3. Demonstrate that an optimal solution to original problem =

   greedy choice + an optimal solution to the sub-problem

4. Make the greedy choice and solve top-down.

5. May have to preprocess input to put it into greedy order : e.g. Sorting activities by finish time.

Learn DAA: From B K Sharma

Problems to be solved using Techniques of Greedy Algorithm.

Fractional Knapsack Problem

Activity Selection Problem

Huffman Code

Set of activities $S = \{a_1, a_2, \ . \ . \ . \ , a_n\}$.

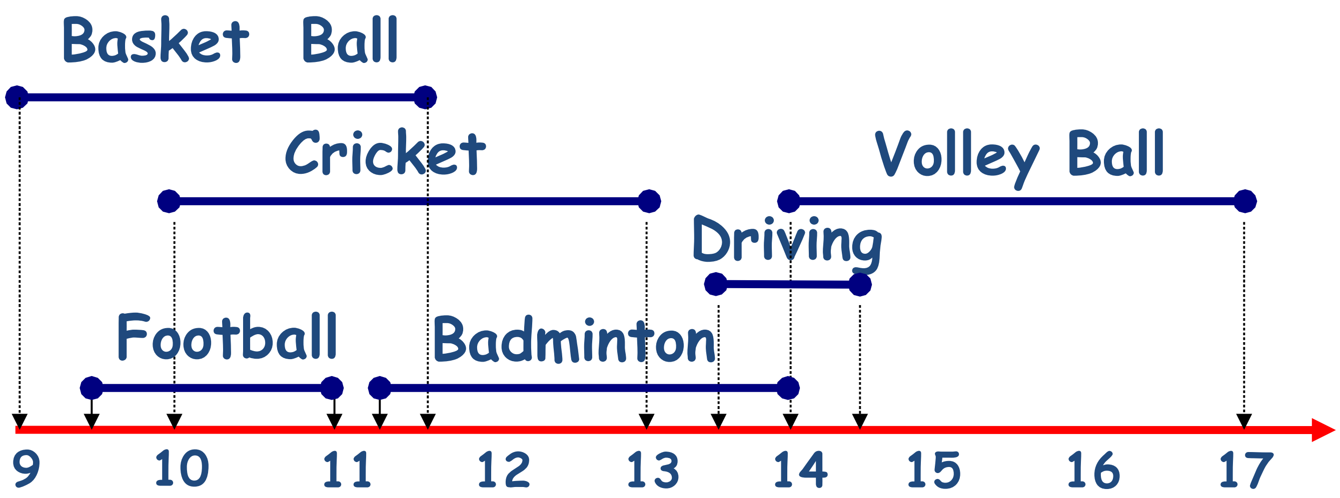
S={ Basket Ball, Football, Volley Ball , Badminton, Cricket, Driving }

n activities require *exclusive* use of a common resource.

a_i needs resource during period $(s_i \ , \ f_i)$, where s_i is start time and f_i is finish time.

Learn DAA: From B K Sharma

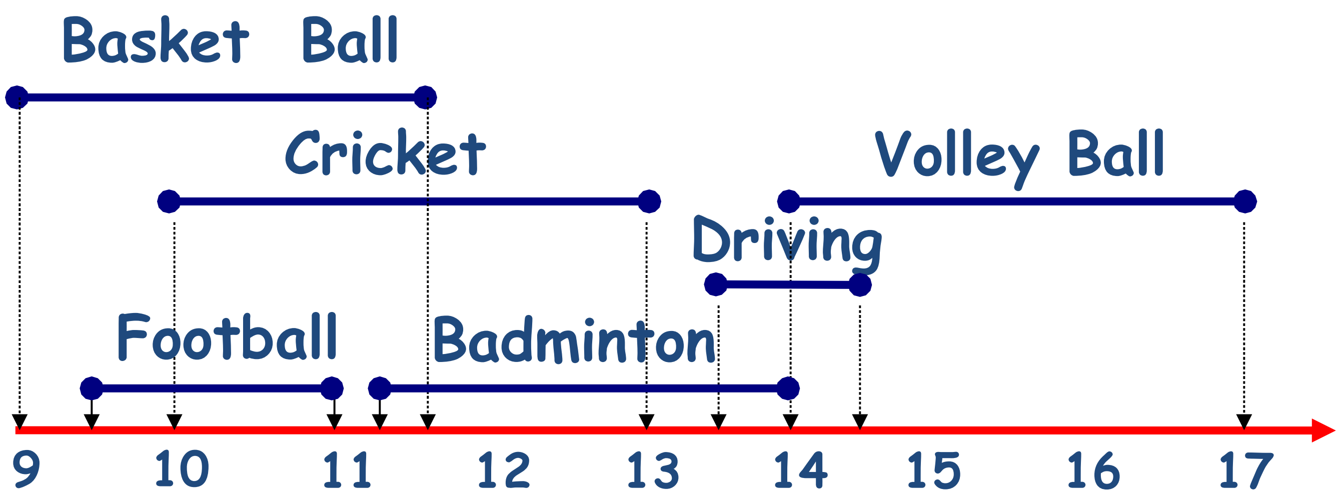
Activity Selection Problem



Goal: **Select the largest possible set of non-overlapping (*mutually compatible*) activities.**

Learn DAA: From B K Sharma

Activity Selection Problem



How to make an arrangement to have the more activities?

S1. Shortest activity first: Football , Driving

S2. First starting activity first: Basket Ball , Driving

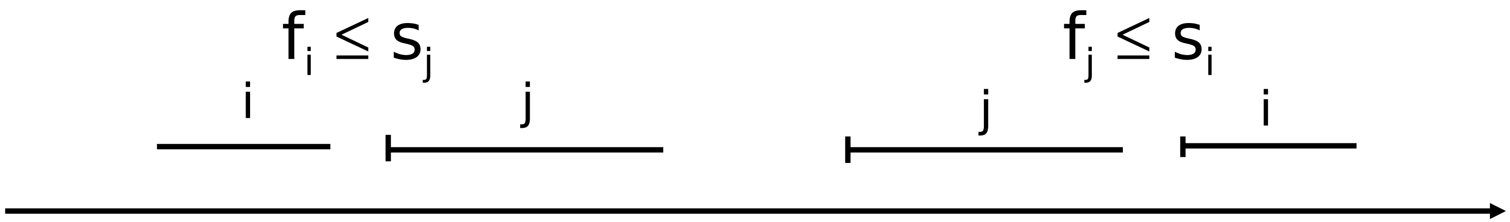
S3. First finishing activity first: Football , Badminton , Volley Ball

Learn DAA: From B K Sharma

Activity Selection Problem

Compatible Activities(Non-overlapping Activities)

Activities a_i and a_j are compatible if the intervals $(s_i\ ,\ f_i)$ and $(s_j,\ f_j)$ do not overlap.



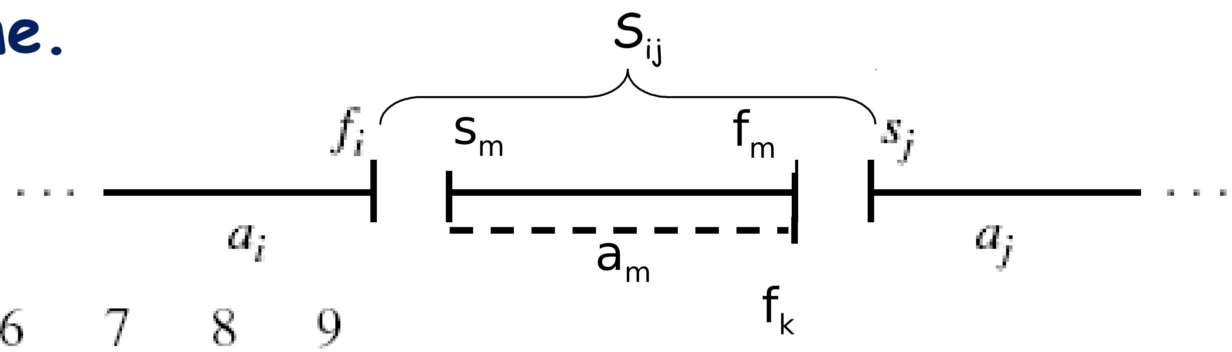
Activity Selection Problem

Greedy Choice Property of Problems

There exists an optimal solution that includes the greedy choice:

The activity a_m with the earliest finish time in S_{ij}

Always choose an activity with the earliest finish time.



i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

Learn DAA: From B K Sharma

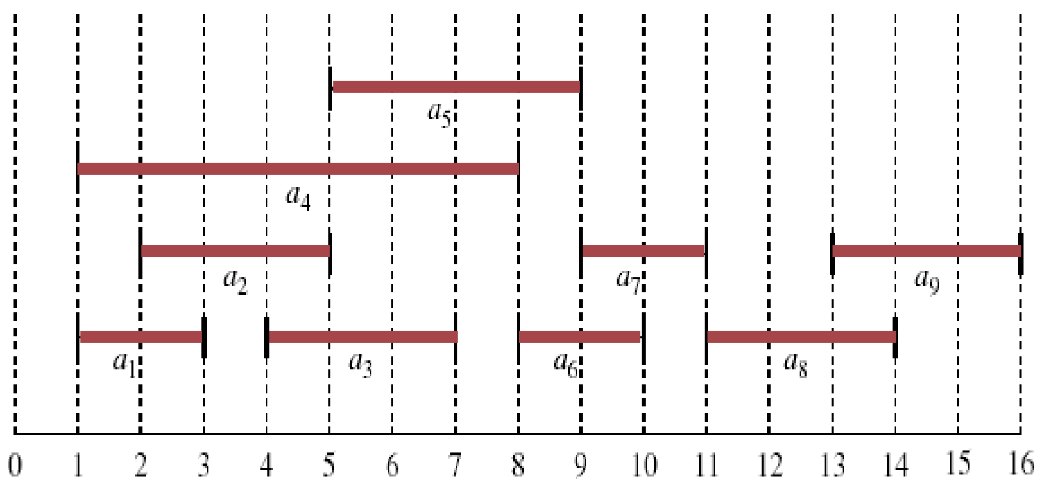
Activity Selection Problem

Greedy Choice Property of Problems

Always choose an activity with the earliest finish time.

Example: S sorted by finish time

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



Maximum-size mutually compatible set:

$\{a_1, a_3, a_6, a_8\}$

Not unique: Also

$\{a_2, a_5, a_7, a_9\}$

Activity Selection Problem

Greedy Choice Property of Problems

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 sub-problem $S_{m,j}$

To solve $S_{1,9}$ we select a_1 that will finish earliest, and solve for $S_{1,9}$.

To solve $S_{1,9}$ we select a_3 that will finish earliest, and solve for $S_{3,9}$.

To solve $S_{3,9}$ we select a_6 that will finish earliest, and solve for $S_{6,9}$.

To solve $S_{6,9}$ we select a_8 that will finish earliest, and solve for $S_{8,9}$.

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

Learn DAA: From B K Sharma

Activity Selection Problem

Greedy Choice Property of Problems

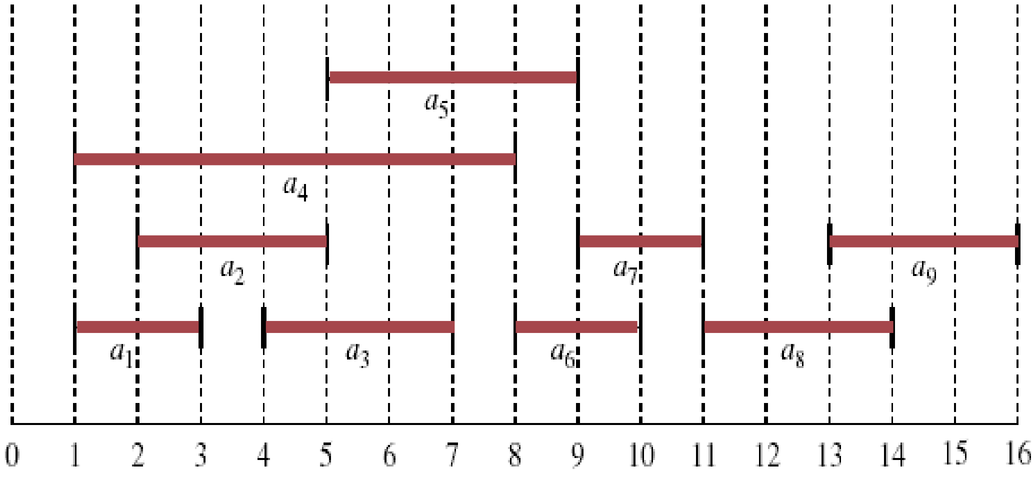
i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

To solve $S_{1,9}$ we select a_1 that will finish earliest, and solve for $S_{1,9}$.

To solve $S_{1,9}$ we select a_3 that will finish earliest, and solve for $S_{3,9}$.

To solve $S_{3,9}$ we select a_6 that will finish earliest, and solve for $S_{6,9}$.

To solve $S_{6,9}$ we select a_8 that will finish earliest, and solve for $S_{8,9}$.



a_2, a_4, a_5 overlaps with a_1

a_2, a_4, a_5 overlaps with a_3

a_7 overlaps with a_6

a_9 overlaps with a_8

Activity Selection Problem

Greedy Choice Property of Problems

To solve $S_{1,9}$ we select a_1 that will finish earliest, and solve for $S_{1,9}$.

To solve $S_{1,9}$ we select a_3 that will finish earliest, and solve for $S_{3,9}$.

To solve $S_{3,9}$ we select a_6 that will finish earliest, and solve for $S_{6,9}$.

To solve $S_{6,9}$ we select a_8 that will finish earliest, and solve for $S_{8,9}$.

Greedy Choices (Locally optimal choice)

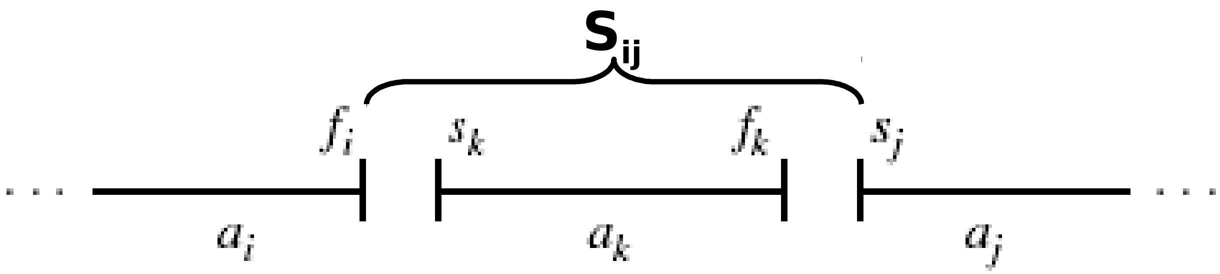
To leave as much opportunity as possible for the remaining activities to be scheduled.

Solve the problem in a top-down fashion

Activity Selection Problem

Optimal Sub-Structure Property of Problems

Suppose a solution to $S_{i,j}$ includes activity a_k,



then,

2 sub-problems are generated: $S_{i,k}$, $S_{k,j}$

Suppose a solution to $S_{1,9}$ contains a_3,

then,

2 sub-problems are generated: $S_{1,3}$ and $S_{3,9}$

Solution to S_{ij} = (Solution to S_{ik}) $\cup$ $\{a_k\}$ $\cup$ (Solution to S_{kj})

$|$Solution to $S_{ij}|$ = $|$Solution to $S_{ik}|$ + 1 + $|$Solution to $S_{kj}|$

Activity Selection Problem

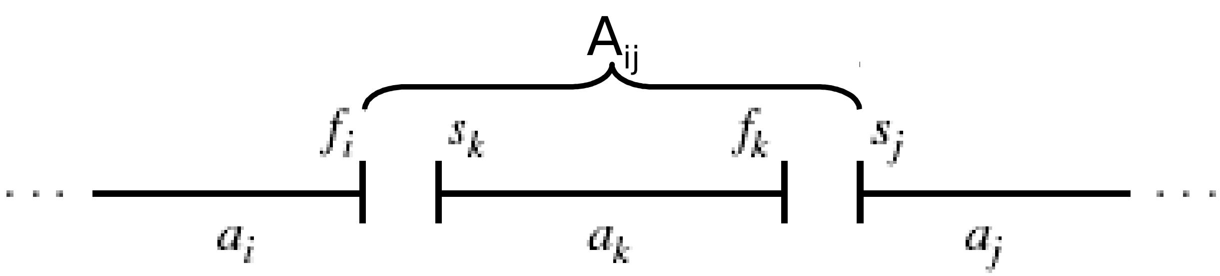
Optimal Sub-Structure Property of Problems

If an optimal solution to sub-problem S_{ij} includes activity a_k

$\Rightarrow$ it must contain optimal solutions to S_{ik} and S_{kj}

The maximum-size subset $A_{i,j}$ of compatible activities is:

$$A_{i,j} = A_{i,k} \cup \{a_k\} \cup A_{k,j}$$



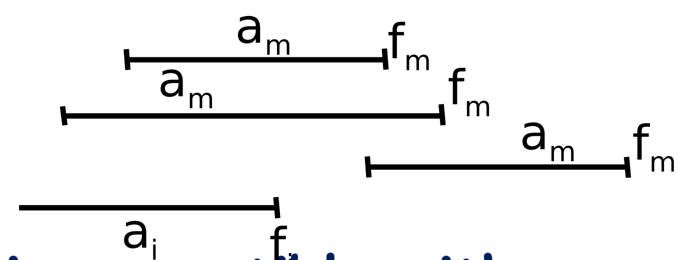
Learn DAA: From B K Sharma

An Iterative Greedy Algorithm

Alg.: GREEDY-ACTIVITY-SELECTOR(s, f)

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

```
n ← length[s]
A ← {a1}
i ← 1
for m ← 2 to n do
    if s_m ≥ f_i then        ► activity a_m is compatible with a_i
        A ← A ∪ {a_m}
        i ← m                ► a_i is most recent addition to A
return A
```



$A = \{ a_1, a_3, a_6, a_8 \}$

Designing Greedy Algorithms

1. Cast the optimization problem as one for which:
   - we make a choice and are left with only one sub-problem to solve
2. Prove the GREEDY CHOICE
   - that there is always an optimal solution to the original problem that makes the greedy choice
3. Prove the OPTIMAL SUBSTRUCTURE:
   - the greedy choice + an optimal solution to the resulting sub-problem leads to an optimal solution

Dynamic Programming vs. Greedy Algorithms

- Dynamic programming
  - We make a choice at each step
  - The choice depends on solutions to sub-problems
  - Bottom up solution, from smaller to larger sub-problems
- Greedy algorithm
  - Make the greedy choice and THEN
  - Solve the sub-problem arising after the choice is made
  - The choice we make may depend on previous choices, but not on solutions to sub-problems
  - Top down solution, problems decrease in size

Steps Toward Our Greedy Solution

1. Determine the optimal substructure of the problem
2. Develop a recursive solution
3. Prove that one of the optimal choices is the greedy choice
4. Show that all but one of the subproblem resulted by making the greedy choice are empty.
   - For example if greedy choice is a_k then first schedule that activity and we skip all other activities that are not compatible with the a_k

Designing Greedy Algorithms

- More generally, we design the greedy algorithms according to the following steps:

1. Cast the optimization problem as one for which:

   we make a choice and are left with only one

   subproblem to solve.

2. Prove that there is always an optimal solution to the original problem that makes the greedy choice.
   - Making the greedy choice is always safe

3. Demonstrate that after making the greedy