Design and Analysis of Algorithms: Lecture 1

Ben Chaplin

Contents

1 Overview

1.1 Course details

Course Title: Design and Analysis of Algorithms
Teacher: Professors Erik Demaine, Srini Devadas & Nancy Lynch
School: MIT
Lectures: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/
Textbook: *Introduction to Algorithms* by Cormen, Leiserson, Rivest & Stein (3rd ed)

1.2 Complexity classes

Definition. P is the class of problems solvable in polynomial time.

Definition. NP is the class of problems verifiable in polynomial time.

Example. Given a graph, does there exist a Hamiltonian cycle? This problem is thought not to be in P, but is in NP. This is because no algorithm has been found to determine whether a graph contains a Hamiltonian cycle in $O(|V|^k)$ time for any $k \in \mathbb{N}$. However, given a graph and a path, it can be verified whether the path is a Hamiltonian cycle in polynomial time.

Definition. NP-complete problems are problems in NP which are "as hard" as any problem in NP.

2 Interval Scheduling

2.1 Definitions & Algorithm

Definition. A **request** is an ordered pair of numbers $(s, f) \in \mathbb{N} \times \mathbb{N}$ representing "start" and "finish" times.

Definition. Two requests $r_i = (s_i, f_i)$ and $r_j = (s_j, f_j)$ are **compatible** if $f_i \leq s_j$ and $f_j \leq s_i$ (they don't overlap). A set of requests is **compatible** if every pair of requests is compatible.

Definition. Given a set of requests R, the **optimal schedule of** R is the largest-cardinality compatible subset of R.

Algorithm 1 Greedy Algorithm for Interval Scheduling

Input
R a list of requests, ordered by finish time

Output
C the optimal schedule of R

$R' \leftarrow R$
$O \leftarrow \{\}$
while R' is not empty **do**
  $r \leftarrow$ the request in R' with the earliest finish time
  Add r to O
  Remove all requests from R' which are not compatible with r
end while
return O

2.2 Correctness

First, we'll prove the correctness of this algorithm.

Theorem 1. *Given a list of requests R (ordered by finish time), **Algorithm 1** produces an optimal schedule of R.*

Proof. (Induction on k^* = the size of the optimal schedule)

Base case: Suppose $k^* = 1$. Then **Algorithm 1** returned one request r, where r has the earliest finish time, and the rest of the requests are incompatible with r. Therefore, there can only be one compatible request, so **Algorithm 1** succeeded.

Inductive hypothesis: If a list of requests R has an optimal schedule of size k^*, then **Algorithm 1** will produce a schedule of size k^*.

Let S be a list of requests which has an optimal schedule O^* of size $(k^* + 1)$:

$$O^*[1, \ldots, k^* + 1] = (s_{j_1}, f_{j_1}), \ldots, (s_{j_{k^*+1}}, f_{j_{k^*+1}})$$

Suppose, when **Algorithm 1** is run on S, it produces the schedule O of size k:

$$O[1, \ldots, k] = (s_{i_1}, f_{i_1}), \ldots, (s_{i_k}, f_{i_k})$$

Note that $f_{i_1} \leq f_{j_1}$, because by definition, the algorithm chooses the request with the earliest finish time. Then, we can define a new schedule by taking O^*, and replacing $O^*[1]$ with $O[1]$:

$$O^{**}[1, \ldots, k^* + 1] = (s_{i_1}, f_{i_1}), (s_{j_2}, f_{j_2}), \ldots, (s_{j_{k^*+1}}, f_{j_{k^*+1}})$$

Note that O^{**} is also optimal, as it has length $(k^* + 1)$. We define a new list S':

$$S' = \{(s_i, f_i) \mid (s_i, f_i) \in S \text{ and } s_i \geq f_{i_1}\}$$

In other words, S' is the list of intervals in S which are compatible with (s_{i_1}, f_{i_1}).

Because O^{**} is optimal for S, $O^{**}[2, \ldots, k^* + 1]$ is optimal for S'. By the **Inductive hypothesis**, **Algorithm 1** should produce a schedule of k^* requests when run on S'. So, when run on S, **Algorithm 1** will:

1. choose (s_{i_1}, f_{i_1}) (we've stated this explicitly),
2. then choose another k^* requests (as now, the algorithm is effectively running on S').

So **Algorithm 1** will return the optimal schedule of size $(k^* + 1)$. $\square$

2.3 Runtime

Algorithm 1 runs in $O(n^2)$ time, as:

1. it iterates $O(n)$ times (no request can be processed more than once),
2. and for each selected request, every subsequent request is checked, resulting in another $O(n)$.