forthright48

/cpps

Interval Scheduling

In "Interval Scheduling" problems, we are given a set of intervals with various properties like starting time, ending time, priority, cost and etc. We also have machines that can process these intervals. We have to choose a subset of these intervals while optimizing some parameters.

A set of intervals chosen must be **compatible** - a property which will be provided in the problem, e.g, the intervals cannot overlap.

Interval Scheduling Maximization Problem

The interval scheduling maximization problem (ISMP) is to find a largest compatible set - a set of non-overlapping intervals of maximum size. The goal here is to execute as many tasks as possible.

Selecting tasks that starts earliest, has shortest intervals or fewest conflicts does not give optimal solution.

The proper greedy solution has 3 steps:

1. Select the interval, X, with the **earliest** finishing time.
2. Remove X, and all intervals intersecting X, from the set of candidate intervals.
3. Continue until the set of candidate intervals is empty.

Resources: Wiki

Interval Scheduling With Deadline

Given a N intervals with profit and deadline and a single machine, we want to maximize the profit. Each interval takes unit time to process. Machine cannot process two intervals at the same time. An interval cannot be process after it has crossed its deadline.

First sort the intervals in decreasing order of profit. Then for each interval we will try to delay processing (**Deffered Processing**) as long as possible without crossing the deadline. This is optimal cause it allows some other intervals with earlier deadline to get processed. If we cannot process it this way, then we ignore it.

1. Sort in decreasing order of profit.
2. Keep track of free slots of the machine in a separate array. Let it be `arr[]`.
3. For each interval, look for a free slot in machine starting from its deadline towards 1.
    1. If found, assign the job to that slot.

2. Else ignore it.

Complexity

If we use a loop at step 3, then this has a complexity of $O(N^2)$. Advantage of this solution is that we are able to print the allocations of the jobs.

If we don't need to print the allocations and require to find the profit only, then a faster solution using Disjoint Set Union is possible. Initially each slot is isolated. Every time we use a slot, that slot points to the next free slot smaller than it. This brings down the complexity of allocation procedure to $O(N)$. But we still have to sort the jobs, so complexity is $O(NlogN + N)$.

Problem: MSCHED - Milk Scheduling | MIA14B - Work scheduling

Resources: GeeksForGeeks

Weighted Interval Scheduling Problem

Given a set of N intervals with start time, end time and profit, we need to choose a subset of compatible (non-overlapping) intervals such that profit is maximum.

O(N^3) Solution

First sort all intervals according to increasing deadline. Then, we can run a DP with parameters `start` and `end`. Initially, we want to find the answer of `dp(1,N)`, which gives the profit for processing intervals from 1 to N.

For each state, we select an interval between `start` and `end`. Let that be the interval k. If we take the k interval, then this will make some intervals incompatible on left and right of it. Therefore, we simply call `dp(start,l)` and `dp(r,end)` after choosing k interval. We choose the k which gives maximum result.

O(NlogN) Solution

First sort all intervals according to increasing deadline. Now, let `dp(x)` be the maximum profit we can get by processing intervals from 1 to x. We want to find the answer to $dp(N)$. Now, we simply perform a knapsack dp.

When at `dp(x)`, we can either include the interval x or not. If we don't take it, we call `dp(x-1)`. Otherwise, we call `dp(y)`, where y is the largest interval whose finish time is smaller or equal start time of x. This can be found using binary search.

Resources: Wiki

Minimum Point Interval Cover

Given a set of N intervals, find minimum number of points which covers every interval. A point covers an interval if it lies between the interval.

If there is a point that covers a set of intervals, then those set of intervals must have a common overlapping segment. We simply need to find these overlapping segments.

1. Sort the segments according to start position.
2. Take the first segment and initiate $res = 1$. Initially, placing a point anywhere within this segment bound covers the whole segment. Let us call this segment, Overlapping Segment. Note that, it is

sufficient to keep track of right bound of this segment. Let it be X.
3. Loop over other segments. For each segment, it will either intersect with the overlapping segment or not.
    1. If it overlaps, update the overlapping segment. $X = min(X, r)$, where r is the right bound of current segment.
    2. If it doesn't overlap, then it means we must create a new overlapping segment. So we increment res.

Problem: POJ 1328 Radar Installation

Interval Partition Problem

TODO

Maximum Independent Set

The Interval Scheduling problem can be converted to a graph. Each task is considered as node and two nodes are connected if the two tasks are incompatible. Finding the largest subset of task that are compatible is same as finding Maximum Independent Set in the graph.