COMP 355
Advanced Algorithms

Dynamic Programming:
Weighted Interval Scheduling
KT (Ch.6 Intro, 6.1-6.2)

Rhodes College

Google Interview Question

Given two sorted arrays with N elements each, find the median of their union in O(log n).

Rhodes College

Algorithmic Paradigms

Greed. Build up a solution incrementally, myopically optimizing some local criterion.

Divide-and-conquer. Break up a problem into two sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

Dynamic programming. Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

Dynamic Programming Applications

Areas.
- Bioinformatics.
- Control theory.
- Information theory.
- Operations research.
- Computer science: theory, graphics, AI, systems, ….

Some famous dynamic programming algorithms.
- Viterbi for hidden Markov models.
- Unix diff for comparing two files.
- Smith-Waterman for sequence alignment.
- Bellman-Ford for shortest path routing in networks.
- Cocke-Kasami-Younger for parsing context free grammars.

Dynamic Programming

DP relies are two important structural qualities:

- Optimal substructure: (principle of optimality)
  - For the global problem to be solved optimally, each subproblem should be solved optimally.
- Overlapping Subproblems
  - The number of distinct subproblems is reasonably small, ideally polynomial in the input size.

Generating Subproblems

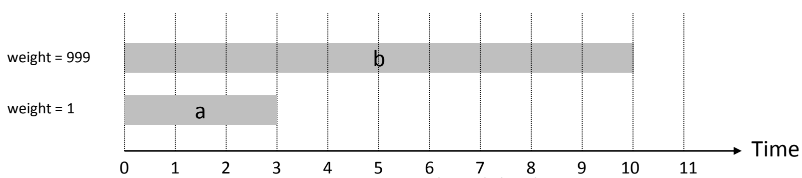
- Top-Down:
  - A solution to a DP problem is expressed recursively.
  - Applies recursion directly to solve the problem.
  - The same recursive call is often made many times.
  - Use ***memoization*** (record the results of recursive calls) so that subsequent calls to a previously solved subproblem are handled by table look-up.
- Bottom-up:
  - Formulate problem recursively, but solve iteratively
  - Combine the solutions to small subproblems to obtain the solution to larger subproblems.
  - The results are stored in a table.

Unweighted Interval Scheduling Review

Recall. Greedy algorithm works if all weights are 1.

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

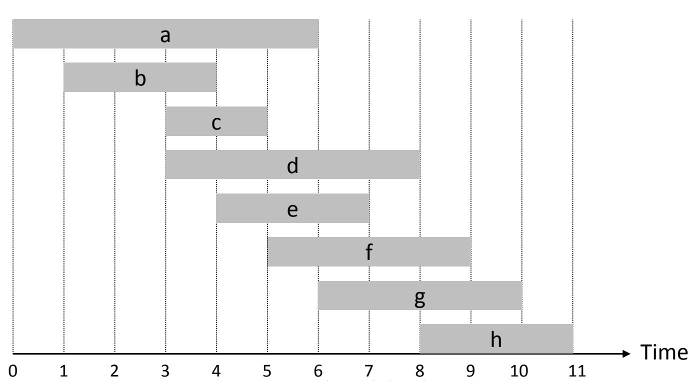
Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed.



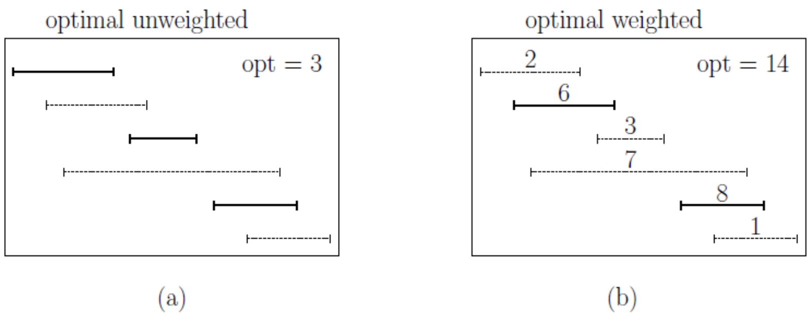
Weighted Interval Scheduling

Weighted interval scheduling problem.

- Job j starts at s_j, finishes at f_j, and has weight or value v_j .
- Two jobs compatible if they don't overlap.
- Goal: find maximum weight subset of mutually compatible jobs.



Weighted vs. Unweighted

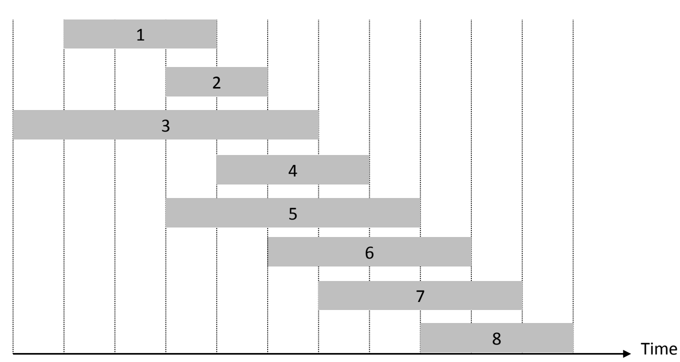


Weighted Interval Scheduling

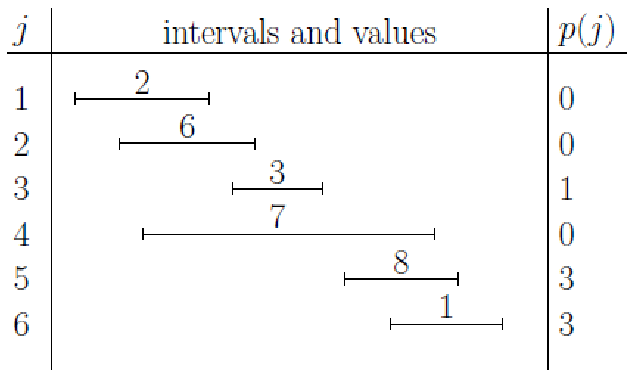
Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq . . . \leq f_n$.

Def. p(j) = largest index i < j such that job i is compatible with j.

Ex: p(8) = 5, p(7) = 3, p(2) = 0.



Weighted Input and P-Values



Rhodes College

Dynamic Programming: Recursive Formulation

Notation. OPT(j) = value of optimal solution to the problem consisting of job requests 1, 2, ..., j.

Case 1: OPT selects job j.

- can't use incompatible jobs { p(j) + 1, p(j) + 2, ..., j - 1 }
- must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., p(j)

↖ optimal substructure ↙

Case 2: OPT does not select job j.

- must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., j-1

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

DP Selection Principle:
When given a set of feasible options to choose from, try them all and take the best.

Weighted Interval Scheduling: Brute Force

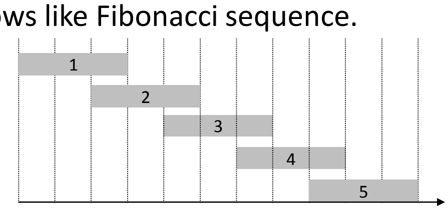
Brute force algorithm.

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Compute-Opt(j) {
   if (j = 0)
      return 0
   else
      return max(vj + Compute-Opt(p(j)), Compute-Opt(j-1))
}
```

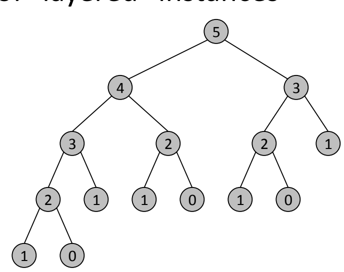

Weighted Interval Scheduling: Brute Force

Observation. Recursive algorithm fails spectacularly because of redundant sub-problems $\Rightarrow$ exponential algorithms.

Ex. Number of recursive calls for family of "layered" instances - grows like Fibonacci sequence.



p(1) = 0, p(j) = j-2



Weighted Interval Scheduling: Memoization

Memoization. Store results of each sub-problem in a cache; lookup as needed.

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n          ← global array
   M[j] = empty
M[0] = 0

M-Compute-Opt(j) {
   if (M[j] is empty)
      M[j] = max(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
}
```


Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes O(n log n) time.

- Sort by finish time: O(n log n).
- Computing p(·) : O(n) after sorting by start time.
- `M-Compute-Opt(j)`: each invocation takes O(1) time and either
  - (i) returns an existing value `M[j]`
  - (ii) fills in one new entry `M[j]` and makes two recursive calls
- Overall running time of `M-Compute-Opt(n)` is O(n).

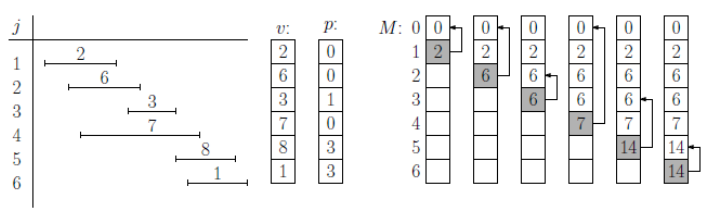
Remark. O(n) if jobs are pre-sorted by start and finish times.

Weighted Interval Scheduling: Bottom-Up

Bottom-up dynamic programming. Unwind recursion.

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Iterative-Compute-Opt {
   M[0] = 0
   for j = 1 to n
       if M[j-1] > vj + M[p(j)]:
              M[j] = M[j-1]; pred[j] = j-1;
       else:
              M[j] = vj + M[p(j)]; pred[j] = p[j];
}
```


Computing the Final Schedule



Example of iterative construction and predecessor values. The final optimal value is 14. By following the predecessor pointers back from M[6] we see that the requests that are in the schedule are 5 and 2.

Computing Weighted Interval Scheduling Schedule

```
get-schedule() {
    j = n
    sched = (empty list)
    while (j > 0) {
        if (pred[j] == p[j]) {
            prepend j to the front of sched
        }
        j = pred[j]
    }
}
```

Rhodes College