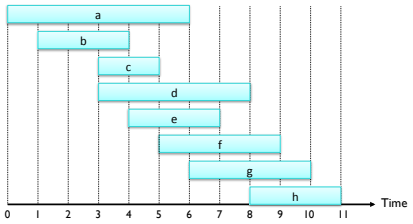
Objectives

- Dynamic Programming
  - Weighted interval schedule
  - Segmented Least Squares

Review: Weighted Interval Scheduling

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



Review: Weighted Interval Scheduling

- Jobs have start time, end time, value/weight
  - Goal: schedule compatible jobs with maximum weight
- What was the key insight to solving the weighted interval scheduling problem?

Binary decision:
- Optimal solution for jobs 1 through j includes j or doesn't

- How do we pick the solution?

Choose the larger value of
- [choose j and the best solution of compatible jobs] OR
[best solution if don't pick j]

Dynamic Programming: Binary Choice

- 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*)
  - Case 2: OPT does **not** select job *j*
    - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j*-1

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

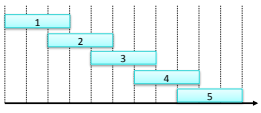
Basecase

Choose the "better" of the two solutions

Weighted Interval Scheduling: Recursive Algorithm

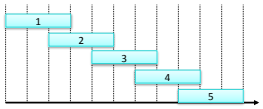
```
Input: n jobs (associated start time s_j, finish time f_j, and value v_j)
Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n
Compute p(1), p(2), …, p(n)        Closest compatible job
Compute-Opt(j):
   if j = 0
      return 0                     Picks j
   else                            Doesn't pick j
      return max(v_j + Compute-Opt(p(j)), Compute-Opt(j-1))
```

What is the runtime?
(Trace for *n* = 5)

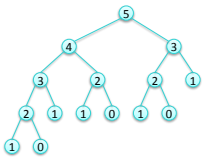


Weighted Interval Scheduling: Brute Force

- Observation. Redundant sub-problems ⇒ exponential algorithms
- Ex. Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.



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



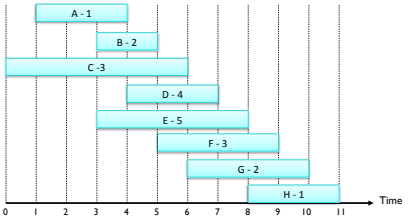
Weighted Interval Scheduling: Memoization

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

```
Input: n jobs (associated start time s_j, finish time f_j, and value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n
Compute p(1), p(2), …, p(n)

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

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]

M-Compute-Opt(n)         <-- Call function with initial input
```


Example

- Jobs labeled as name – weight

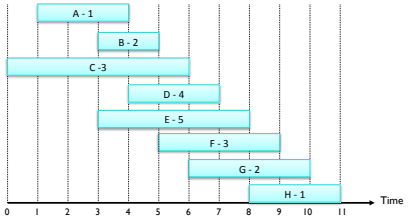


M	0	A	B	C	D	E	F	G	H
	0								

Example

What is the value of p for each job?

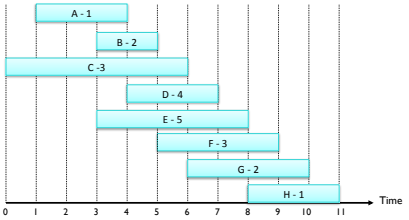
- Jobs labeled as name – weight



M	0	A	B	C	D	E	F	G	H
	0								

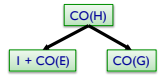
Example

P(j)
0
0
0
A
0
B
C
E

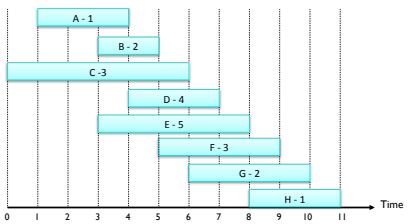


M	0	A	B	C	D	E	F	G	H
	0								

Example

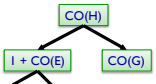


P(j)
0
0
0
A
0
B
C
E

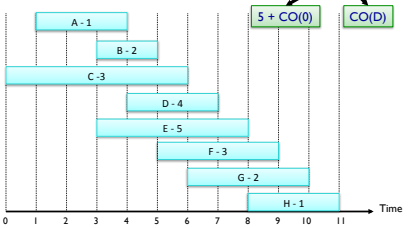


M	0	A	B	C	D	E	F	G	H
	0								

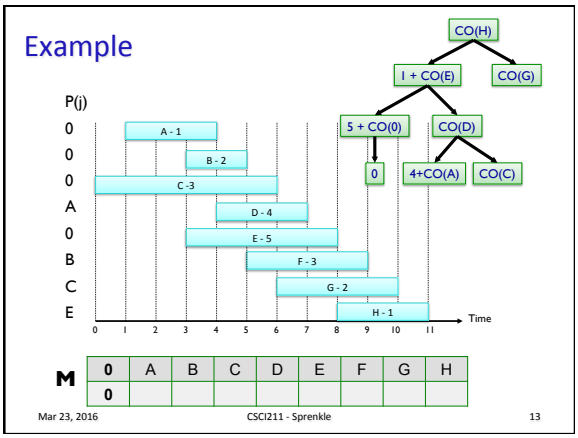
Example

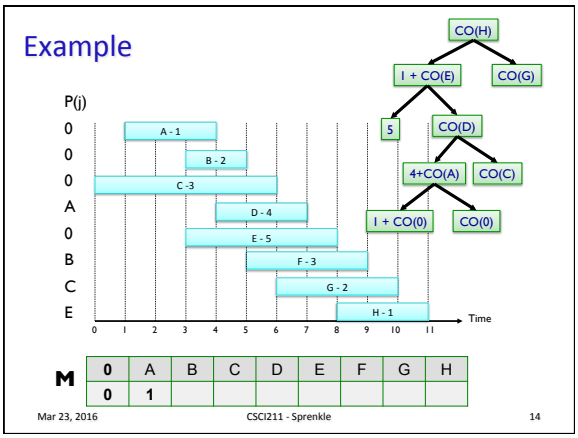


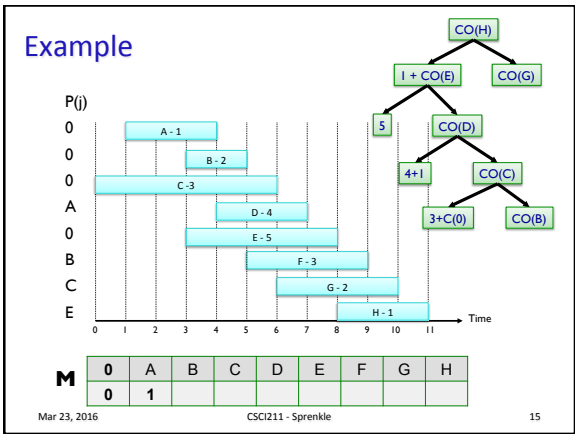
P(j)
0
0
0
A
0
B
C
E

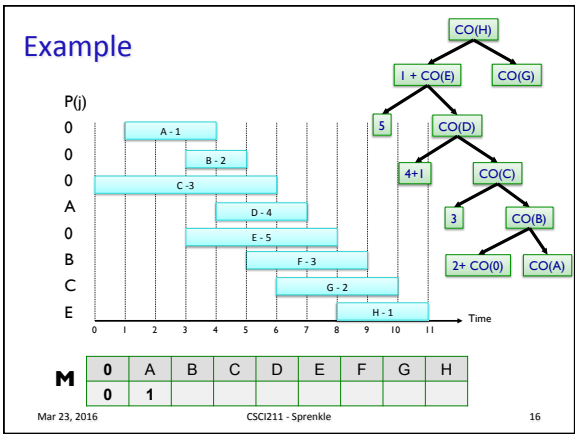


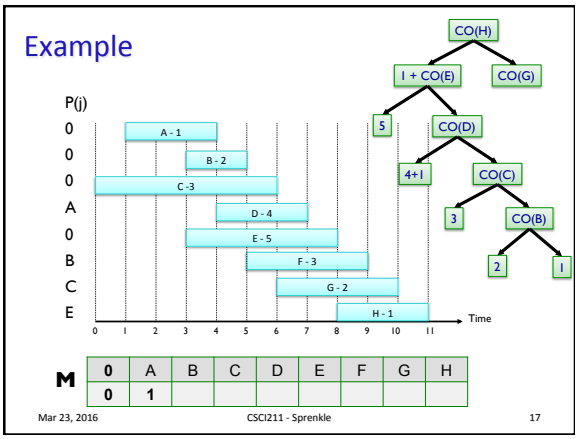
M	0	A	B	C	D	E	F	G	H
	0								

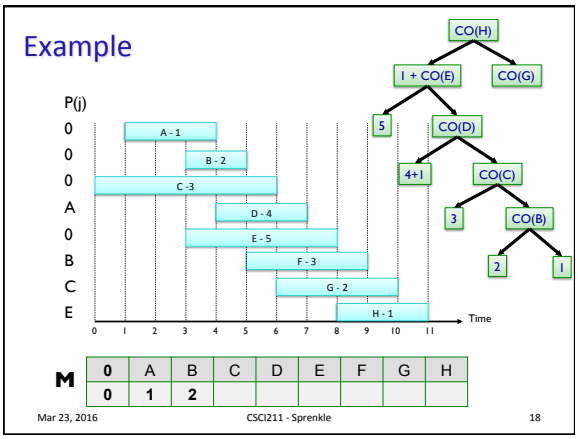

Example
CO(H)
I + CO(E)
CO(G)
5 + CO(0)
CO(D)
0
4+CO(A)
CO(C)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0
Mar 23, 2016
CSCI211 - Sprenkle
13

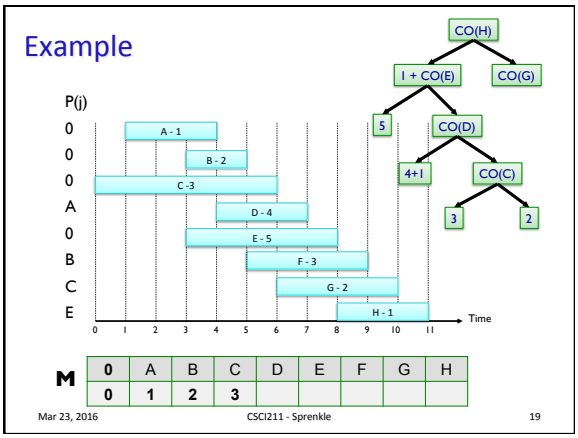

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+CO(A)
CO(C)
I + CO(0)
CO(0)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
Mar 23, 2016
CSCI211 - Sprenkle
14

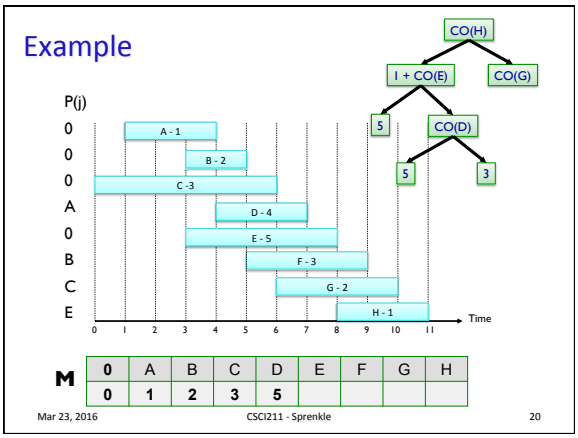

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+I
CO(C)
3+C(0)
CO(B)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
Mar 23, 2016
CSCI211 - Sprenkle
15

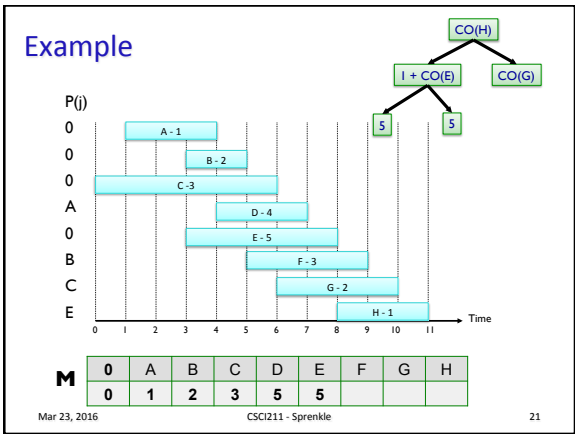

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+I
CO(C)
3
CO(B)
2+ CO(0)
CO(A)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
Mar 23, 2016
CSCI211 - Sprenkle
16

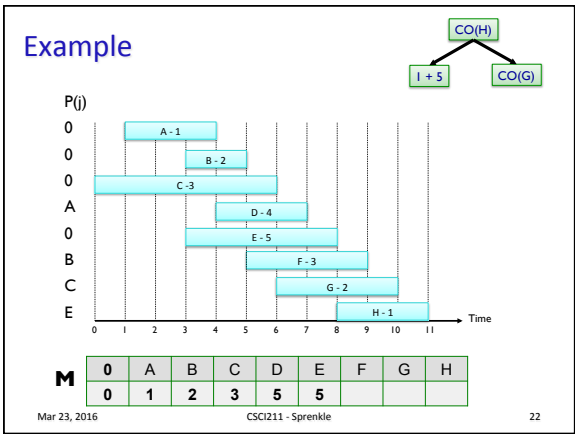

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+I
CO(C)
3
CO(B)
2
1
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1
Mar 23, 2016
CSCI211 - Sprenkle
17

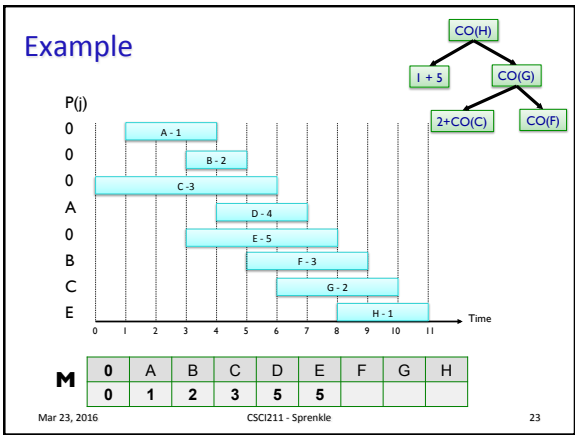

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+I
CO(C)
3
CO(B)
2
1
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2
Mar 23, 2016
CSCI211 - Sprenkle
18

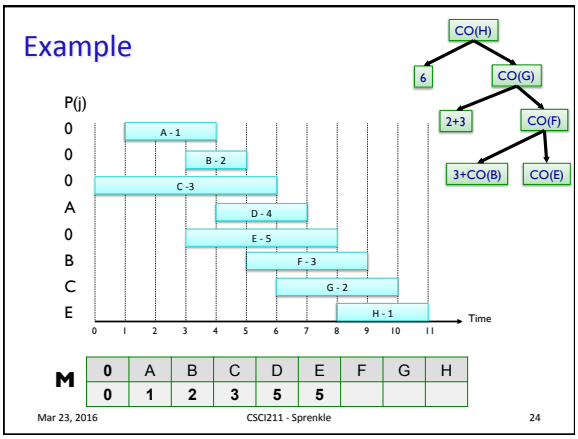

Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
4+I
CO(C)
3
2
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3
Mar 23, 2016
CSCI211 - Sprenkle
19


Example
CO(H)
I + CO(E)
CO(G)
5
CO(D)
5
3
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3 5
Mar 23, 2016
CSCI211 - Sprenkle
20

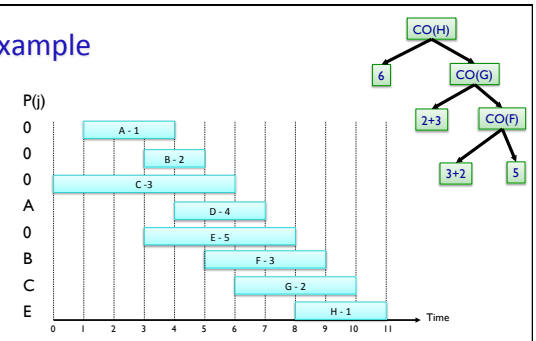

Example
CO(H)
I + CO(E)
CO(G)
5
5
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3 5 5
Mar 23, 2016
CSCI211 - Sprenkle
21


Example
CO(H)
I + 5
CO(G)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3 5 5
Mar 23, 2016
CSCI211 - Sprenkle
22


Example
CO(H)
I + 5
CO(G)
2+CO(C)
CO(F)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3 5 5
Mar 23, 2016
CSCI211 - Sprenkle
23

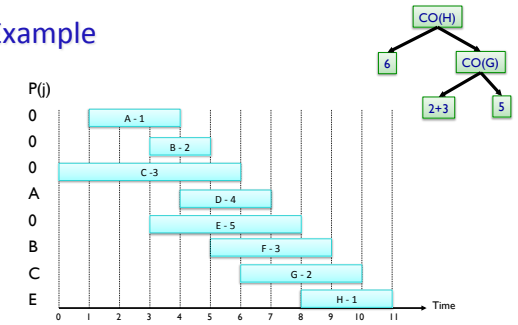

Example
CO(H)
6
CO(G)
2+3
CO(F)
3+CO(B)
CO(E)
P(j)
0
0
0
A
0
B
C
E
A - 1
B - 2
C -3
D - 4
E - 5
F - 3
G - 2
H - 1
Time
0 1 2 3 4 5 6 7 8 9 10 11
M
0 A B C D E F G H
0 1 2 3 5 5
Mar 23, 2016
CSCI211 - Sprenkle
24

Example



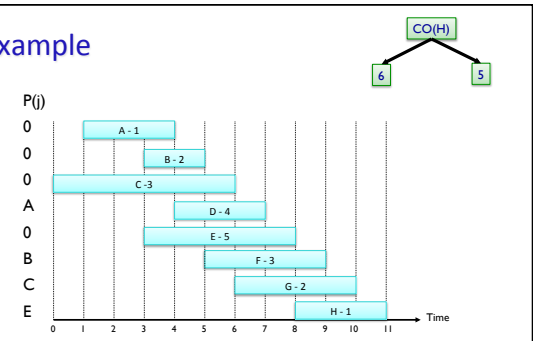
M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5		

Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	

Example



M	0	A	B	C	D	E	F	G	H
	0	1	2	3	5	5	5	5	6

Weighted Interval Scheduling: Memoization Analysis

Costs?

```
Input: n jobs (associated start time s_j, finish time f_j, and value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]

M-Compute-Opt(n)
```


Weighted Interval Scheduling: Memoization Analysis

```
Input: n jobs (associated start time s_j, finish time f_j, and value v_j)

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n   O(n log n)
Compute p(1), p(2), …, p(n)  O(n log n);

for j = 1 to n
   M[j] = empty   O(n)
M[0] = 0

M-Compute-Opt(j):
   if M[j] is empty:
      M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]

M-Compute-Opt(n)  O(n)
```


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 log n)
  - `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
  - Progress measure Φ = # nonempty entries of `M[]`
    - (i) initially $\Phi = 0$, throughout $\Phi \le n$
    - (ii) increases Φ by 1 $\Rightarrow$ at most 2n recursive calls
  - Running time of `M-Compute-Opt(n)` is O(n). ▪
- Remark.
  - O(n) if jobs are *pre-sorted* by start and finish times

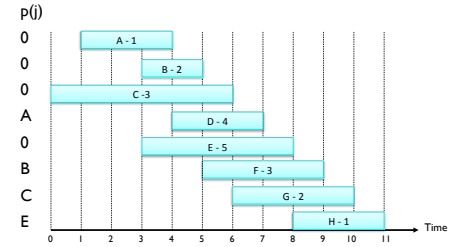
Weighted Interval Scheduling: Finding a Solution

- Dynamic programming algorithms compute ***optimal value***
- What if we want the ***solution*** itself (**not** simply the value)?
- Do some post-processing
  - Looking at M, how do we know which set of intervals were chosen?

M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5	5	5	6

Towards Finding a Solution



M

0	A	B	C	D	E	F	G	H
0	1	2	3	5	5	5	5	6

Weighted Interval Scheduling: Finding a Solution

- Dynamic programming algorithms compute ***optimal value***
- What if we want the ***solution*** itself (**not** simply the value)?
- Do some post-processing

```
M-Compute-Opt(n)
Find-Solution(n)

def Find-Solution(j):
   if j = 0:
      output nothing
   elif v_j + M[p(j)] > M[j-1]:
      print j
      Find-Solution(p(j))
   else:
      Find-Solution(j-1)
```

Runtime?

O(n)

Looking Ahead

- Exam 2 due Friday at 5 p.m.