Dynamic Programming

Weighted Interval Scheduling

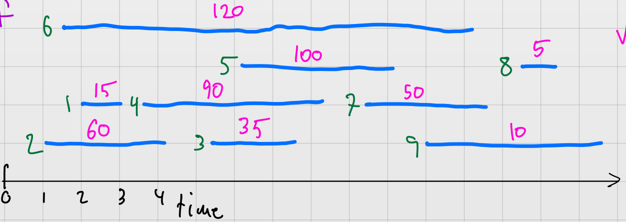
Subset Sum

Weighted Interval Scheduling

Input: Jobs $J_1, J_2, \ldots, J_n$ $\quad J_i = (s_i, f_i, v_i)$

value/weight

A subset of jobs are compatible if no two overlap



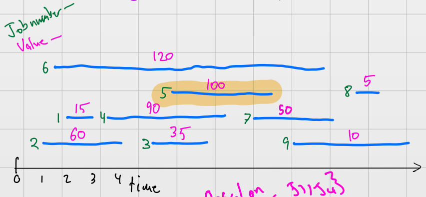
Output: The compatible subset with greatest value

Weighted Interval Scheduling

```
M-Opt-Val(j):
   if j = 0
      return 0
   else if M[j] != -1
      return M[j]
   else
      M[j] = max(M-Opt-Val(p(j)) + v_j, M-Opt-Val(j -1))
      Return M[j]
```

Weighted Interval Scheduling

Assume jobs are ordered by EFT. $(f_1 \le f_2 \le f_3 \le \dots)$



Opt-val on $\{J_1, J_2, J_3, J_4\}$

Opt-val(9)

val1

Opt-val(5)

val1 val2

Opt-val(2) = 60 Opt-val(4) Opt-val(3)

val1 = 0 + 60 val2 Opt-val(1) = 15 Opt-val = 15 + 90 = 105

val1 = 0 val2 = 0 + 15

M: 0 1 2 3 4 5 6 7 8 9

Iteration vs Recursion

Iterative

```
Iteration-M-Opt-Val(j):
    M[0] = 0
    for i = 1; i <= n; i++
        M[I] = max(M[p(i) + v_i, M[i-1])
    return M[n]
```

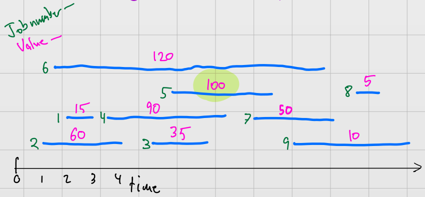
Recursive

```
M-Opt-Val(j):
    if j = 0
        return 0
    else if M[j] != -1
        return M[j]
    else
        M[j] = max(M-Opt-Val(p(j)) + v_j, M-Opt-Val(j -1))
        Return M[j]
```

Example

```
Iteration-M-Opt-Val(j):
    M[0] = 0
    for i = 1; i <= n; i++
        M[I] = max(M[p(i)] + v_i, M[i-1])
    return M[n]
```

Assume jobs are ordered by EFT. ($f_1 \le f_2 \le f_3 \le \dots$)



	M				
0	0				
1	15	{1}			
2	60		$p(2)=0$	60	15
3	95		$p(3)=2$	95	60
4	105		$p(4)=1$	105	95
5	160		$p(5)=2$	160	105
6					
7					
8					
9					

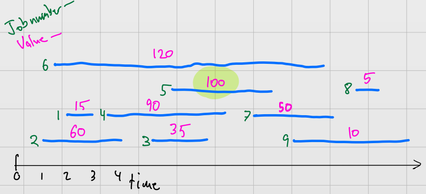
$M[p(5)] + v_5$

$60 + 100 = 160$

Example (Subset)

```
Iteration-M-Opt-Val(j):
    M[0] = 0
    for i = 1; i <= n; i++
        M[I] = max(M[p(i)] + v_i, M[i-1])
    return M[n]
```

Assume jobs are ordered by EFT. $(f_1 \le f_2 \le f_3 \le \dots)$



	M	
0	0	$\emptyset$
1	15	{1}
2	60	{2}
3	95	{2, 3}
4	105	{1, 4}
5	160	{2, 5}
6	120	{6}
7	155	{1, 4, 6}
8		
9		

Runtime can be $O(n^2)$

subsets can get long $\to O(n)$

Return Subset (not value)

Another way:

At most n recursions

$\Rightarrow O(n)$

	M
0	0
1	15
2	60
3	95
4	105
5	160
6	120
7	155
8	
9	X

$M[p(j)] + v_j$ OR $M[j-1]$

```
Opt-Subset(j):
   if j = 0
      L = empty list
      return L
   if M[p(j)] + v_j >= M[j-1]
      return Opt-Subset(p(j)).append(j)
   else
      return Opt-Subset(j-1)
```

where did X come from?

Subset Sum

Input: $\{w_i\}$ $1 \leq i \leq n$ threshold T

Output: $S \subseteq \{1, \dots, n\}$ such that

$$\sum_{i \in S} w_i \leq T$$

and $\sum_{i \in S} w_i$ is maximal

Subset Sum Greedy

1. Highest weight

$T = 100 \quad \{51, 50, 50\}$

$\xrightarrow{HW} \{51\}$

$\rightarrow \{50, 50\}$

2. Lowest weight

$T = 100 \quad \{1, 50, 50\}$

$\xrightarrow{LW} \{1, 50\}$

$\rightarrow \{50, 50\}$

Subset Sum Dynamic Programming

Input: $\{w_1, \dots, w_n\}$ T

Let's imagine an optimal subset Opt. Is $n \in \text{Opt}$

Case 1 $n \notin \text{Opt}$ $\Rightarrow \text{Opt} \subseteq \{1, \dots, n-1\}$

$\Rightarrow$ Opt is an optimal subset on the same problem w/ $\{w_1, \dots, w_{n-1}\}$ and threshold T

Case 2 $n \in \text{Opt}$ What does $\text{Opt} - \{n\}$ look like?

It's weight is at most $T - w_n$.

$\text{Opt} - \{n\} \subseteq \{1, \dots, n-1\}$

$\Rightarrow$ same problem w/ $\{w_1, \dots, w_{n-1}\}$ and $T - w_n$

Subset Sum Dynamic Programming

Opt-Weight(j, V) is the optimal weight for the problem with $\{w_1, \ldots, w_j\}$ and threshold V

$$\left.\begin{array}{l}\text{Case 1}: \mathrm{Opt}(j-1, V) \\ \text{Case 2}: \mathrm{Opt}(j-1, V-w_j) + w_j\end{array}\right\} \text{Want the bigger one}$$

At the top level

$\mathrm{Opt}(n-1, T)$

$\mathrm{Opt}(n-1, T-w_n) + w_n$

Subset Sum Example

$$\begin{array}{cccc} w_1 & w_2 & w_3 & w_4 \\ \shortparallel & \shortparallel & \shortparallel & \shortparallel \\ \end{array}$$
$$\{2, 3, 5, 7\} \qquad T = 10$$

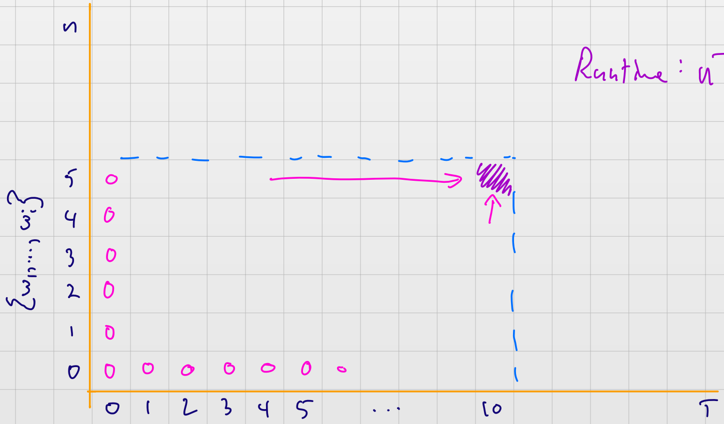
$Opt(4, 10)$

Case 1 : $Opt(3, 10)$
Case 2 : $Opt(3, 3) + 7$

$Opt(3, 10)$ $\quad Opt(2, 10)$
$\quad Opt(2, 5) + 5$

$Opt(2, 10)$ $\quad Opt(1, 10)$
$\quad Opt(1, 7) + 3$

```
SubsetSum({w_1, ..., w_n}, T):   (T, w_i is an integer > 0 ∀ i)
    Array M{0...n, 0...T}
    Set M{i,j} = 0 ∀ i,j
    for i = 1, 2, ..., n
        for t = 0, 1, ..., T:
            if (t - w_i ≥ 0)
                Set M(i,t) = max(M{i-1, t}, M{i-1, t-w_i} + w_i)
            else
                Set M(i,t) = M{i-1, t}
    return M{n, T}
```



Runtime: nT