CSC373

ALGORITHMS

Max-WEIGHTED INTERVAL SCHEDULING (RECAP)

Given : n intervals $[s_i, f_i]$ $\quad s_i < f_i$
weights $wt(i)$

$f_1 \leq \ldots \leq f_{p(j)} < s_j \leq f_{p(j)+1} \leq \ldots \leq f_n$

Output : A subset I of non-overlapping intervals with maximum total wt $\sum_{i \in I} wt(i)$

PREPROCESS : SORT BY FINISH TIMES

$f_1 \leq f_2 \leq \ldots \leq f_n$

COMPUTE $p(j)$ = max-index $i < j$ s.t interval i does not overlap with j

j does not overlap w $\{1, \ldots, p(j)\}$ & overlaps with $\{p(j)+1, \ldots, j-1\}$

RECURSION PROVED :

$$OPT(\{1, \ldots, j\}) = \max \{ OPT(\{1, \ldots, j-1\}), wt(j) + OPT(\{1, \ldots, p(j)\}) \}$$

BASE CASE :

$$OPT(\{\}) = 0$$

$OPT(\{1, \ldots, j\})$ = weight of non-overlapping subset of $\{1, \ldots, j\}$ with max wt

MAX-WT-IS

1. Sort intervals by finishing time
2. Compute $p(j)$ for all j by Binary Search
3. RETURN R-OPT(n)

```
R-OPT(j)
    if j == 0
        RETURN 0
    ELSE
        RETURN MAX( R-OPT(j-1),
                    wt(j) + R-OPT(p(j)) )
```

$OPT(j) \equiv OPT(\{1, \ldots, j\})$

$\forall j \; p(j) = j-1$

$T(n) = 2T(n-1) + O(1)$

$T(n) = \Omega(2^n)$

EXPONENTIAL

SUBPROBLEMS TO SOLVE:

$OPT(\{1, \ldots j\})$ $\quad j = 0, 1, \ldots, n$

$n+1$ subproblems

MAX-WT-IS (MEMOIZATION)

1. Sort intervals by finishing time $O(n \log n)$
2. Compute $p(j)$ for all j by Binary Search $O(n \log n)$

$M[j] \leftarrow OPT(\{1, \ldots j\})$

$j = 0 \equiv OPT(\{\})$

$M^- \leftarrow [-1] * (n+1)$

$M[0] \leftarrow 0$

$O(n)$

RETURN M-OPT(n)

$O(n)$

$O(n \log n)$

M-OPT(j)

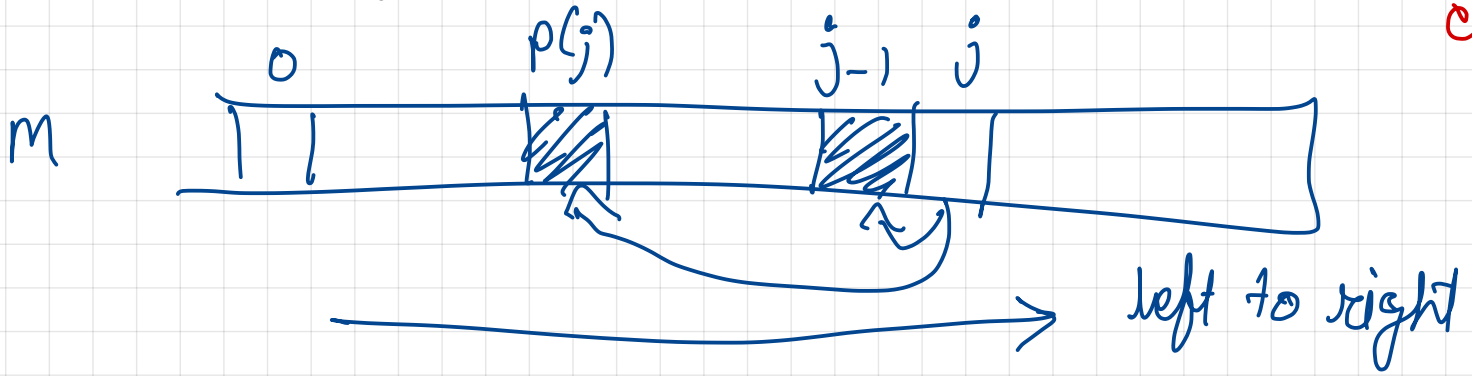
IF $M[j] == -1$

$M[j] = \max(M\text{-OPT}(j-1),$

$wt(j) + M\text{-OPT}(p(j)))$

RETURN $M[j]$

Recursive step executes exactly once for every j



MAX-WT-IS (NON-RECURSIVE)

1. Sort intervals by finishing time
2. Compute $p(j)$ for all j by binary search

$O(n \log n)$

```
M ← [-1] * (n+1)
M[0] ← 0

FOR j in RANGE(1, n+1):
    M[j] = max( M[j-1],
                wt(j) + M[p(j)] )

RETURN M[n]
```

excluded

$O(n)$ time

$O(n \log n)$ time

$O(n)$ space

NON-RECURSIVE. Advantages:

- Avoid recursion overhead
- Avoids recursion depth issues

RECOVERING THE OPTIMAL SOLUTION

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

$OPT(j)$ = wt of non-overlapping subset of $\{1, \ldots, j\}$ with max wt

$$S(j) = \begin{cases} \{\} & \text{if } j = 0 \\ S(j-1) & \text{if } OPT(j) = OPT(j-1) \\ \{j\} \cup S(p(j)) & \text{o/w} \end{cases}$$


```
j ⇐ n
subset ← []
while j > 0
        if OPT[j] == OPT[j-1]
                j ← j-1
        else
                subset.append(j)
                j ← p(j)
```

OPT ⇐ M

KNAPSACK PROBLEM

GIVEN : n items with weight w_i, cost c_i; budget B

GOAL : Find a subset I of items with total cost $\sum_{i \in I} c_i \leq B$ and maximum possible weight $\sum_{i \in I} w_i$

ASSUME : All c_i and B are integers

$OPT(B, \{1,\dots,n\})$ = max possible wt of a subset I of items $\{1,\dots,n\}$ while satisfying budget $\sum_{i \in I} c_i \le B$

$OPT(b, \{1,\dots,j\})$ = max possible wt of subset I of items $\{1,\dots,j\}$ satisfying $\sum_{i \in I} c_i \le b$ (remaining budget)

don't pick j^{th} item

$OPT(b, \{1,\dots,j-1\})$

pick j^{th} item if $b \ge c_j$

$w_j + OPT(b - c_j, \{1,\dots,j-1\})$

$$OPT(b, \{1, \dots, j\}) = \begin{cases} 0 & j = 0 \\ OPT(b, \{1, \dots, j-1\}) & j > 1,\ b < c_j \\ \max\{ OPT(b, \{1, \dots, j-1\}), & \text{o/w} \\ \quad w_j + OPT(b - c_j, \{1, \dots, j-1\}) \} & \end{cases}$$

$j = 0 \iff \{\}$

$OPT(b, \{\}) = 0$

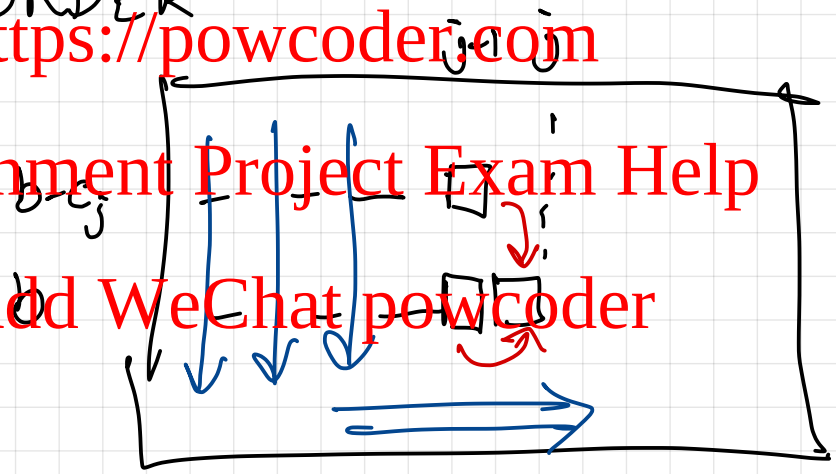
IDENTIFY SUBPROBLEMS:

$OPT(b, \{1, \dots, j\})$ $\quad 0 \le b \le B$, $\quad 0 \le j \le n$

MEMOIZATION DATASTRUCTURE

$M[b][j] \Leftarrow OPT(b, \{1, \dots, j\})$

EVALUATION ORDER



M [B] x [n]

column-wise
left to right

KNAPSACK

M is a [B] x [n] array

FOR b in RANGE (0, B+1)

excluded

M[b][0] ← 0

FOR j in RANGE (1, n+1)

FOR b in RANGE (0, B+1):

IF $b \geq c_j$:

$M[b][j] = MAX\{M[b][j-1], w_j + M[b-c_j][j-1]\}$

ELSE

$M[b][j] = M[b][j-1]$

RETURN M[B][n]

} O(nB) time / space

RECOVER-SUBSET → Returns best subset of items that achieves max. total cut while having total cost ≤ B

```
j ← n
b ← B
subset ← []
while j > 0 :
    If M[b][j] == M[b][j-1]
        j ← j - 1
    ELSE
        subset.append(j)
        j ← j - 1
        b ← b - c_j
RETURN subset
```