Dynamic Programming

- (Weighted) Independent Set in Graphs
- Weighted Independent Sets in Path
- Dynamic Programming Formulation
- Implementation and Backtracking

Imdad ullah Khan

Max weight independent set in path graph

$$\textsc{opt-val}(k) \;=\; \max \begin{cases} w_1 & \text{if } k = 1 \\ \max\{w_1, w_2\} & \text{if } k = 2 \\ \textsc{opt-val}(k-2) + w_k & \text{if } v_k \in \textsc{opt-set}(k) \\ \textsc{opt-val}(k-1) & \text{if } v_k \notin \textsc{opt-set}(k) \end{cases}$$

Algorithm Recursive $\textsc{opt-val}(n)$

function $\textsc{opt-val}(k)$ ▷ implements the above recurrence
  if $k = 1$ **then**
    return w_1
  else if $k = 2$ **then**
    return $\max\{w_1, w_2\}$
  else
    return $\max\{\textsc{opt-val}(k-1), \textsc{opt-val}(k-2) + w_k\}$

▷ Only computes $\textsc{opt-val}(n)$, will extend it to get $\textsc{opt-set}(n)$

Exponential runtime due to unnecessary repeated calls

Max WIS in P_n: Dynamic Programming

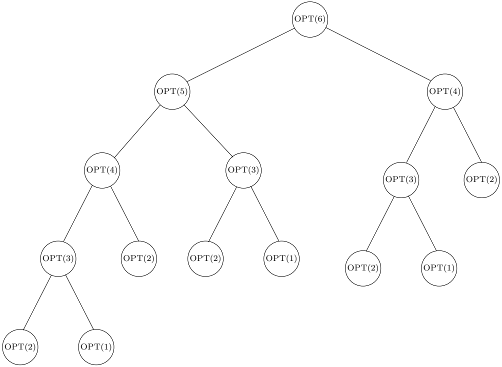
Recall the dynamic programming paradigm

- Express optimal solution in terms of optimal solution to smaller subproblems
- Identify repetition in the above recursion
- Use memoization or bottom-up computation

Max weight independent set in path graph

Algorithm Recursive $\text{OPT-VAL}(n)$

function $\text{OPT-VAL}(k)$
  if $k = 1$ **then**
    return w_1
  else if $k = 2$ **then**
    return $\max\{w_1, w_2\}$
  else
    return $\max\{\text{OPT-VAL}(k-1), \text{OPT-VAL}(k-2) + w_k\}$



Max WIS in P_n: Dynamic Programming

Store $\text{OPT-VAL}(k)$ for small k in memo M and use without recomputing

Algorithm Recursive $\text{OPT-VAL}(n)$ with memoization

```
function OPT-VAL(k)
  if M[k] is empty then
    if k = 1 then
      M[k] ← w_k
    else if k = 2 then
      M[k] ← max{w_1, w_2}
    else
      M[k] ← max{w_k + OPT-VAL(k − 2), OPT-VAL(k − 1)}
  return M[k]
```

- In one call to $\text{OPT-VAL}(\cdot)$ one memo entry $M[\cdot]$ gets filled
- A memo entry is filled only once; total calls to $\text{OPT-VAL}(\cdot)$ is n
- Number of ops in a call to $\text{OPT-VAL}(\cdot)$ is $O(1)$ + some recursive calls

Runtime of $\text{OPT-VAL}(n)$ is $O(n)$

Max WIS in P_n: Dynamic Programming

- Avoid overhead of recursive calls
- Write the code bottom up

$\triangleright$ Unwind the recursion

- Solve smaller problems first and then bigger
- Until we solve the original problem

Algorithm Bottom-Up Computation of OPT-VAL(n)

$M[1] \leftarrow w_1$
$M[2] \leftarrow \max\{w_1, w_2\}$
for $i = 3$ `to` n **do**
    $M[i] \leftarrow \max\{M[i-2] + w[i], M[i-1]\}$
return $M[n]$

Max WIS in P_n: Dynamic Programming

- Both above algorithms give only value of the solution, $\text{OPT-VAL}(n)$
- How to find the solution itself, WIS? $\text{OPT-SET}(n)$
- Like $M[k] = \text{OPT-VAL}(k)$, also maintain $\text{OPT-SET}(k)$
- Just add v_k or not depending on which branch yields larger value
- Wastes a lot of space

$$\text{OPT-VAL}(k) = \max \begin{cases} \text{OPT-VAL}(k-2) + w_k & \text{if } v_k \in \text{OPT-SET}(k) \\ \text{OPT-VAL}(k-1) & \text{if } v_k \notin \text{OPT-SET}(k) \end{cases}$$

- We can only remember the branch which gives higher value
- Backtrack when $\text{OPT-VAL}(n)$ is computed
- Or Scan $M[\cdot]$ again and find the branch

Max WIS in P_n: Dynamic Programming

- $\text{OPT-SET}(k)$ either contains v_k or not
- If $\text{OPT-VAL}(k-2) + w_k \geq \text{OPT-VAL}(k-1)$, then $v_k \in \text{OPT-SET}(k)$
- else $v_k \notin \text{OPT-SET}(k)$

Algorithm Max WIS in P_n

```
OPT-VAL(n)                                  ▷ M[i] = OPT-VAL(i)
function OPT-SET(n)
    S ← ∅
    i ← n
    while i ≥ 1 do
        if M[i − 2] + w_i ≥ M[i − 1] then
            S ← S ∪ {v_i}
            i ← i − 1
        else
            i ← i − 1
    return S
```