COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 23

DYNAMIC PROGRAMMING II :

WEIGHTED INDEPEDENT SET (CONTD.) & KNAPSACK

SEPT 24, 2024 | ROHIT VAISH

MAX WEIGHT INDEPENDENT SET ON PATHS

MAX WEIGHT INDEPENDENT SET ON PATHS

input: a path graph $G = (V, E)$ with nonnegative weights on vertices $\{w_v\}_{v \in V}$

MAX WEIGHT INDEPENDENT SET ON PATHS

input: a path graph $G=(V,E)$ with nonnegative weights on vertices $\{w_v\}_{v\in V}$

output: an independent set $S \subseteq V$ of G with maximum $\sum_{v\in S} w_v$

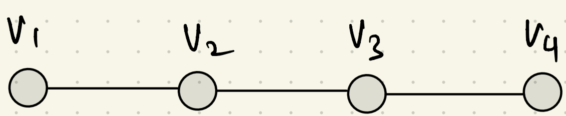
subset of non-adjacent vertices

MAX WEIGHT INDEPENDENT SET ON PATHS

input: a path graph $G = (V, E)$ with nonnegative weights on vertices $\{w_v\}_{v \in V}$

output: an independent set $S \subseteq V$ of G with maximum $\sum_{v \in S} w_v$

subset of non-adjacent vertices



Brute force

Greedy

Divide and conquer

OPTIMAL SUBSTRUCTURE

Optimal solution of overall problem built up from optimal solution of subproblems in a prescribed way.

OPTIMAL SUBSTRUCTURE

$$G = v_1 — v_2 — \dots — v_{n-2} — v_{n-1} — v_n$$

S : max wt independent set of graph G

OPTIMAL SUBSTRUCTURE

$$G = v_1 — v_2 — \dots — v_{n-2} — v_{n-1} — v_n$$

S : max wt independent set of graph G

OPTIMAL SUBSTRUCTURE

$$G = v_1 — v_2 — \dots — v_{n-2} — v_{n-1} — v_n$$

S : max wt independent set of graph G

if $v_n \notin S$

$\Rightarrow S$ must be a max wt ind. set of $G' := G \setminus \{v_n\}$.

OPTIMAL SUBSTRUCTURE

$$G = v_1 - v_2 - \dots - v_{n-2} - v_{n-1} - v_n$$

S : max wt independent set of graph G

if $v_n \notin S$

$\Rightarrow S$ must be a max wt ind. set of $G' := G \setminus \{v_n\}$.

if $v_n \in S$

$\Rightarrow S \setminus \{v_n\}$ must be a max wt ind. set of $G'' := G \setminus \{v_{n-1}, v_n\}$.

A RECURSIVE ALGORITHM

A RECURSIVE ALGORITHM

input: a path graph G with vertices $v_1, v_2, \ldots, v_n$, nonnegative weights $\{w_i\}$

output: a maximum-weight independent set of G

A RECURSIVE ALGORITHM

input: a path graph G with vertices $v_1, v_2, \ldots, v_n$, nonnegative weights $\{w_i\}$

output: a maximum-weight independent set of G

if $n = 1$

return $\{v_1\}$

A RECURSIVE ALGORITHM

input: a path graph G with vertices $v_1, v_2, \ldots, v_n$, nonnegative weights $\{w_i\}$

output: a maximum-weight independent set of G

if $n = 1$

return $\{v_1\}$

// recursion when $n \geq 2$

A RECURSIVE ALGORITHM

input: a path graph G with vertices $v_1, v_2, \dots, v_n$, nonnegative weights $\{w_i\}$

output: a maximum-weight independent set of G

if $n = 1$

    return $\{v_1\}$

// recursion when $n \geq 2$

S_1 := recursively compute an MWIS of $G \setminus \{v_n\}$

S_2 := recursively compute an MWIS of $G \setminus \{v_{n-1}, v_n\}$

A RECURSIVE ALGORITHM

input: a path graph G with vertices $v_1, v_2, \ldots, v_n$, nonnegative weights $\{w_i\}$

output: a maximum-weight independent set of G

if $n = 1$

return $\{v_1\}$

// recursion when $n \geq 2$

$S_1 :=$ recursively compute an MWIS of $G \setminus \{v_n\}$

$S_2 :=$ recursively compute an MWIS of $G \setminus \{v_{n-1}, v_n\}$

return S_1 or $S_2 \cup \{v_n\}$, whichever has higher weight

A RECURSIVE ALGORITHM

Claim: The recursive algorithm for MWIS is correct.

A RECURSIVE ALGORITHM

Claim: The recursive algorithm for MWIS is correct.

Proof: Exercise, by induction on number of vertices n.

A RECURSIVE ALGORITHM

Claim : The recursive algorithm for MWIS is correct.

Proof : Exercise, by induction on number of vertices n.

Claim : The recursive algorithm for MWIS takes exponential time.

A RECURSIVE ALGORITHM

Claim: The recursive algorithm for MWIS is correct.

Proof: Exercise, by induction on number of vertices n.

Claim: The recursive algorithm for MWIS takes exponential time.

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

A RECURSIVE ALGORITHM

Claim: The recursive algorithm for MWIS is correct.

Proof: Exercise, by induction on number of vertices n.

Claim: The recursive algorithm for MWIS takes exponential time.

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

branching factor = 2

... but very little progress

A RECURSIVE ALGORITHM

Claim: The recursive algorithm for MWIS is correct.

Proof: Exercise, by induction on number of vertices n.

Claim: The recursive algorithm for MWIS takes exponential time.

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

branching factor = 2 ... but very little progress

$T(n) \geq \text{Fibonacci}(n) \sim \text{const.}^n$

A RECURSIVE ALGORITHM

Claim : The recursive algorithm for MWIS is correct.

Proof : Exercise , by induction on number of vertices n.

Claim : The recursive algorithm for MWIS takes exponential time.

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

branching factor = 2 ... but very little progress

$T(n) \geq \text{Fibonacci}(n) \sim \text{const.}^n$ No better than brute force ☹

Among the exponentially-many recursive calls, how many distinct subproblems are considered?

Among the exponentially-many recursive calls, how many distinct subproblems are considered?

$\Theta(n)$

Among the exponentially-many recursive calls, how many distinct subproblems are considered?

$\Theta(n)$

The prefixes of graph G $v_1 — v_2 — v_3 \cdots v_i — v_{i+1} \cdots v_n$

ELIMINATING REDUNDANCY

The first time we solve a subproblem, cache its solution in a global table for $O(1)$ time lookup later on.

ELIMINATING REDUNDANCY

The first time we solve a subproblem, cache its solution in a global table for $O(1)$ time lookup later on.

(i.e., memoization)

BLENDING CACHE INTO PSEUDOCODE

BLENDING CACHE INTO PSEUDOCODE

Let $G_i :=$ first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

// main loop

for $i = 2, 3, \ldots, n$:

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

// main loop

for $i = 2, 3, \ldots, n$:

$$A[i] := \max\{A[i-1],\ A[i-2] + w_i\}.$$

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

Running time: ?

// main loop

for $i = 2, 3, \ldots, n$:

$$A[i] := \max\{A[i-1], A[i-2] + w_i\}.$$

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

Running time: $O(n)$

// main loop

for $i = 2, 3, \ldots, n$:

$$A[i] := \max\{A[i-1],\ A[i-2] + w_i\}.$$

BLENDING CACHE INTO PSEUDOCODE

Let G_i := first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan : populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

// main loop

for $i = 2, 3, \ldots, n$:

$$A[i] := \max\{A[i-1],\ A[i-2] + w_i\}.$$

Running time : $O(n)$

Correctness : same as recursive algorithm

BLENDING CACHE INTO PSEUDOCODE

Let $G_i :=$ first i vertices of graph G and corresponding edges

$$v_1 — v_2 — \cdots — v_{i-1} — v_i$$

Plan: populate array A left to right with $A[i]$ = value of MWIS of G_i.

// initialization

$A[0] = 0$

$A[1] = w_1$

☹️

Computes the weight of Ind Set, but not the set itself

// main loop

for $i = 2, 3, \ldots, n$:

$$A[i] := \max\{A[i-1], A[i-2] + w_i\}.$$

Reconstruction Algorithm

RECONSTRUCTION ALGORITHM

Idea: Trace back through array A to reconstruct optimal solution.

RECONSTRUCTION ALGORITHM

Idea: Trace back through array A to reconstruct optimal solution.

Key point:

Vertex v_i belongs to MWIS of G_i $\iff$ w_i + MWIS of G_{i-2} $\geq$ MWIS of G_{i-1}

RECONSTRUCTION ALGORITHM

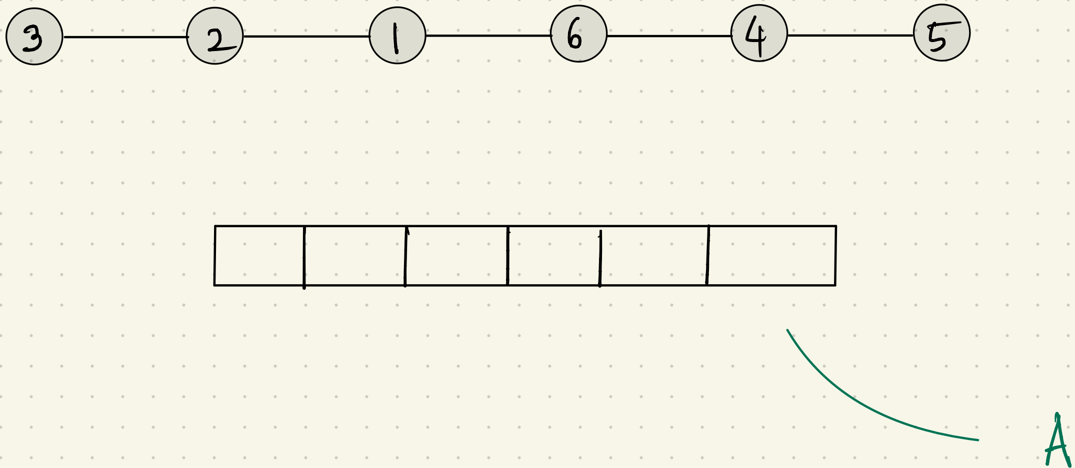
Idea: Trace back through array A to reconstruct optimal solution.

Key point:

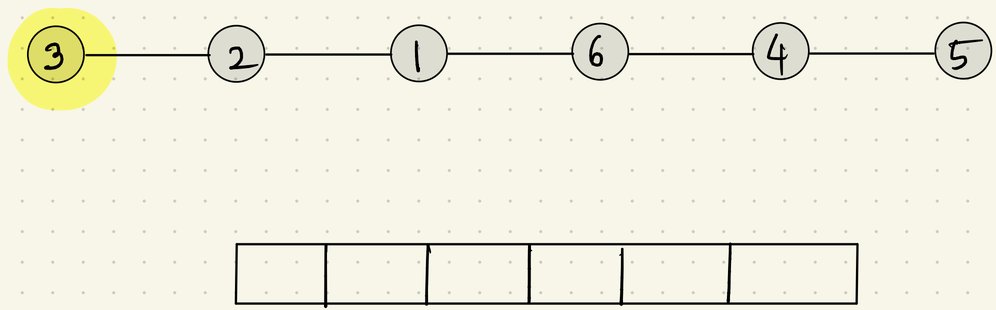
vertex v_i belongs to MWIS of G_i $\Longleftrightarrow$ w_i + MWIS of G_{i-2} $\geq$ MWIS of G_{i-1}

```
S := ∅
while i ≥ 1                          // scan from right to left
    if A[i-1] ≥ A[i-2] + w_i         // v_i ∉ MWIS
        decrease i by 1
    else                             // v_i ∈ MWIS
        add v_i to S, decrease i by 2
return S
```

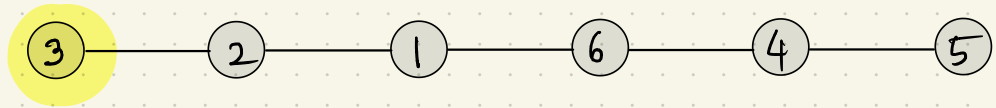
EXAMPLE



EXAMPLE



EXAMPLE

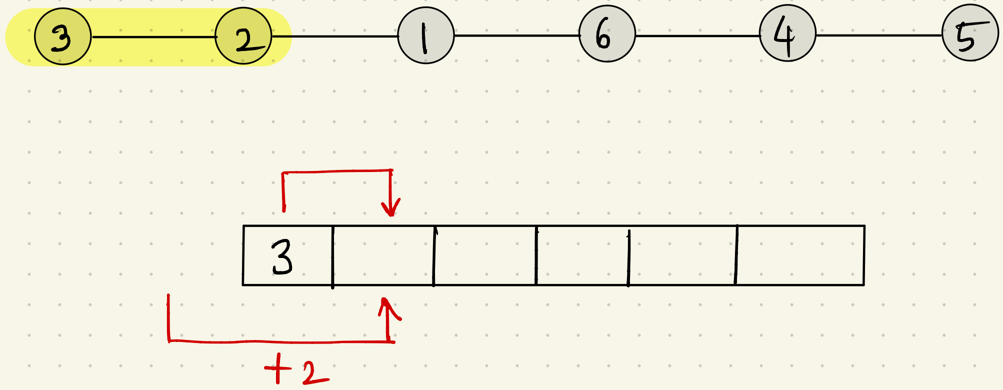


3					

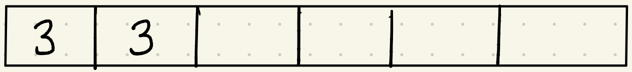
EXAMPLE

3					

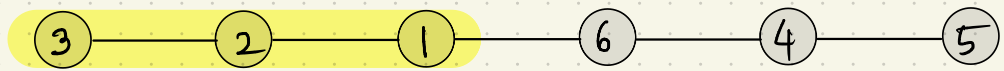
EXAMPLE



EXAMPLE

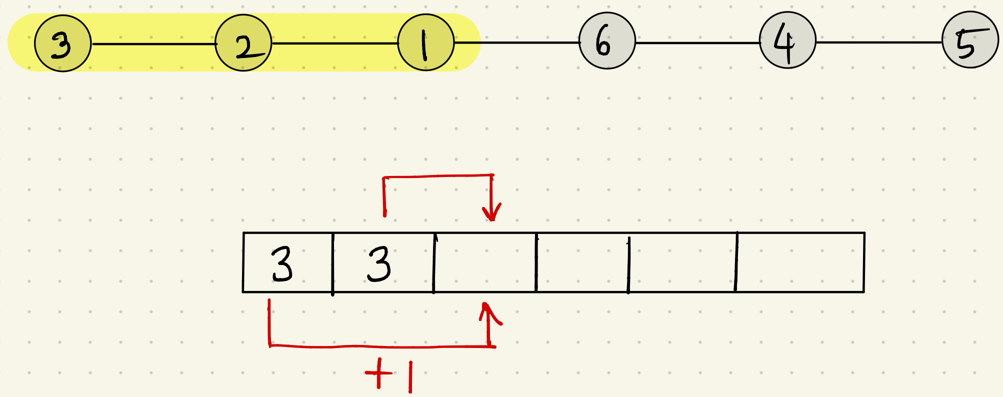


EXAMPLE

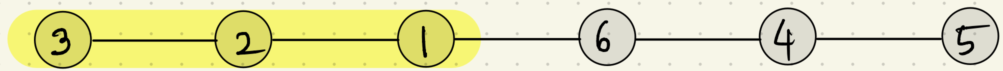


3	3				

Example

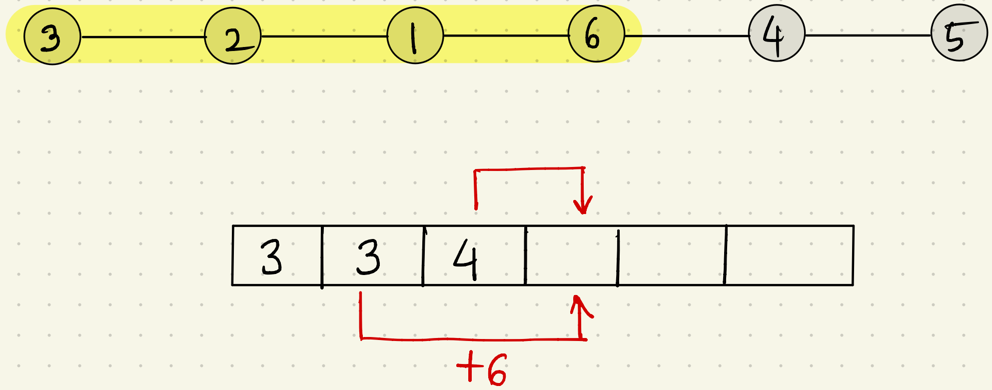


Example

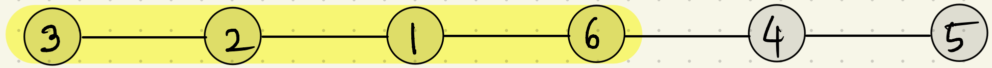


3	3	4			

EXAMPLE

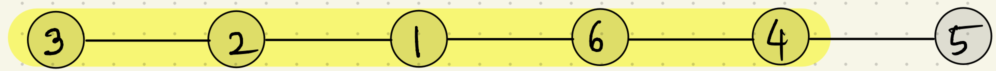


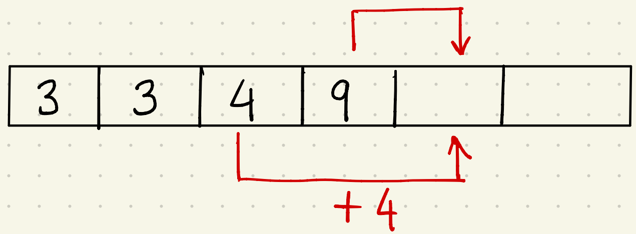
EXAMPLE



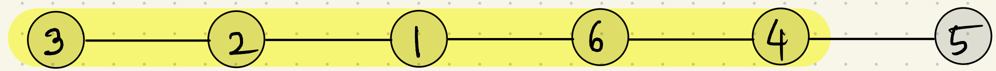
3	3	4	9		

EXAMPLE



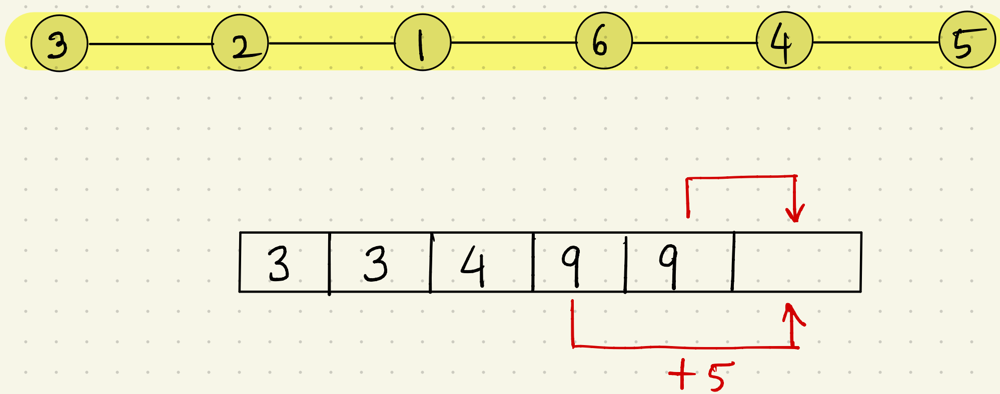


Example

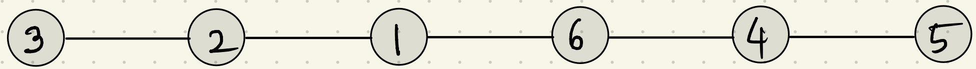


3	3	4	9	9	

EXAMPLE

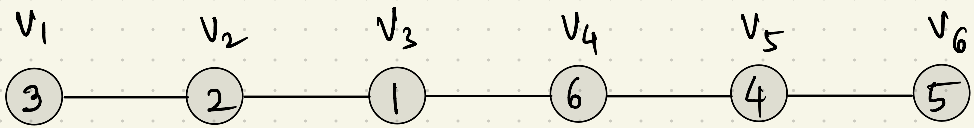


EXAMPLE



3	3	4	9	9	14

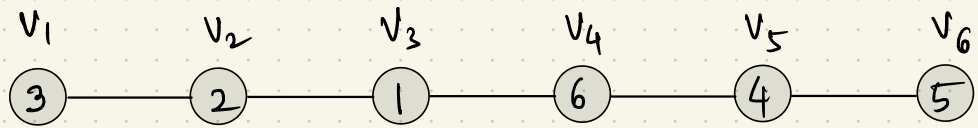
EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ \quad \}$

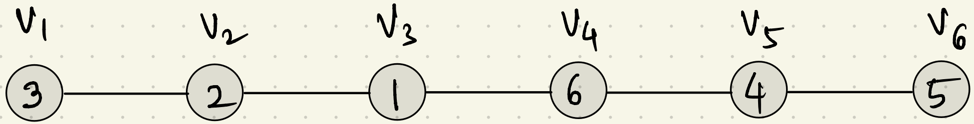
EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ \quad \}$

EXAMPLE



exclude v_6

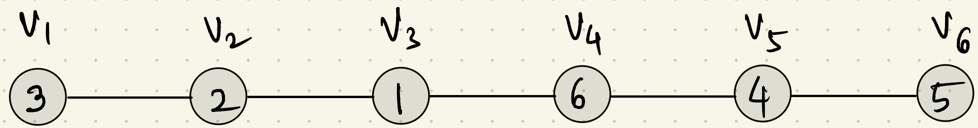
3	3	4	9	9	14

+5

include v_6

Reconstruction $S = \{ \qquad \}$

EXAMPLE



exclude v_6

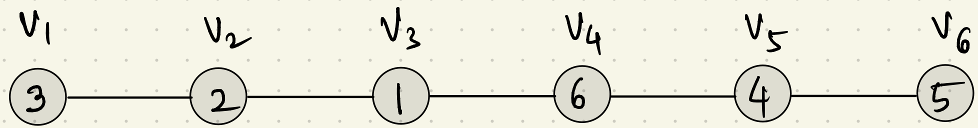
3	3	4	9	9	14

+5

include v_6

Reconstruction $S = \{ \quad v_6 \}$

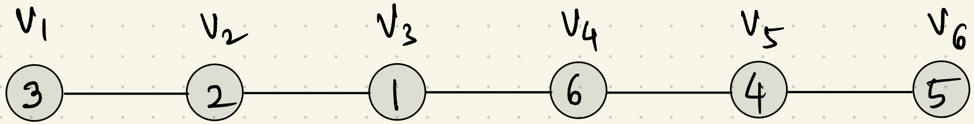
EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ \quad v_6 \}$

EXAMPLE



exclude v_4

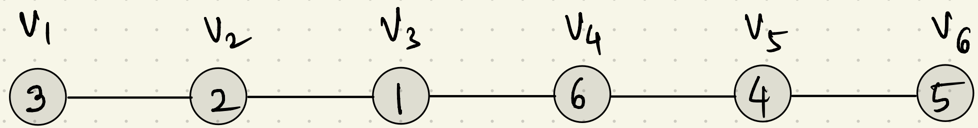
3	3	4	9	9	14

+6

include v_4

Reconstruction $S = \{ \quad v_6 \}$

EXAMPLE



exclude v_4

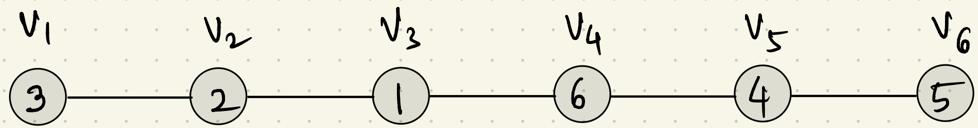
3	3	4	9	9	14

+6

include v_4

Reconstruction $S = \{ \quad v_4 \quad v_6 \}$

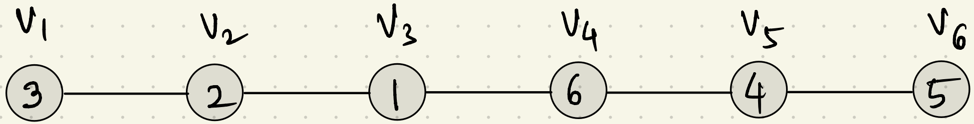
EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ \quad v_4 \quad v_6 \}$

EXAMPLE



exclude v_2

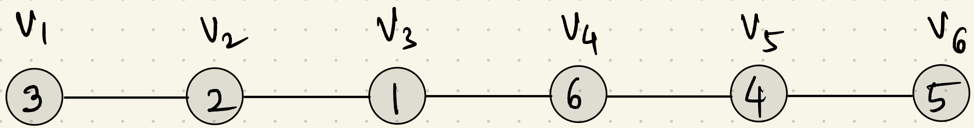
3	3	4	9	9	14

include v_2

+2

Reconstruction $S = \{ \quad v_4 \quad v_6 \}$

EXAMPLE



exclude v_2

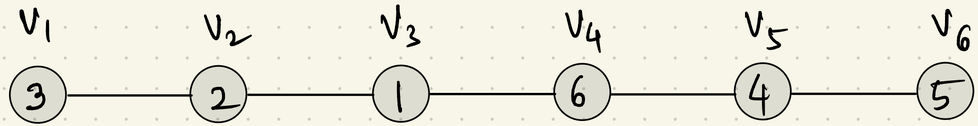
3	3	4	9	9	14

$+2$

include v_2

Reconstruction $S = \{ \cancel{v_2} \; v_4 \; v_6 \}$

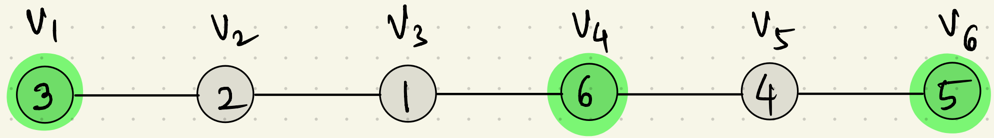
EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ v_1 \ v_4 \ v_6 \}$

EXAMPLE



3	3	4	9	9	14

Reconstruction $S = \{ v_1 \ v_4 \ v_6 \}$

A DAG VIEW OF DYNAMIC PROGRAMMING

1.

2.

3.

A DAG VIEW OF DYNAMIC PROGRAMMING

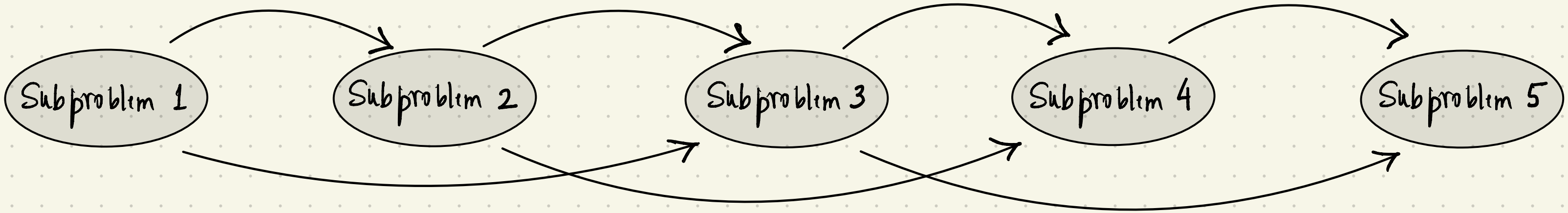
Subproblem 1   Subproblem 2   Subproblem 3   Subproblem 4   Subproblem 5

1. Identify a small number of subproblems

2.

3.

A DAG VIEW OF DYNAMIC PROGRAMMING

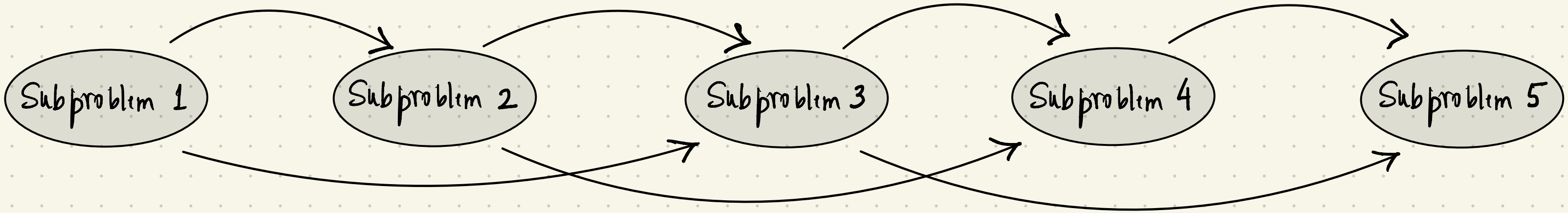


1. Identify a small number of subproblems

2. Show how to quickly and correctly solve "large" subproblems given solution of "smaller" ones.

3.

A DAG VIEW OF DYNAMIC PROGRAMMING

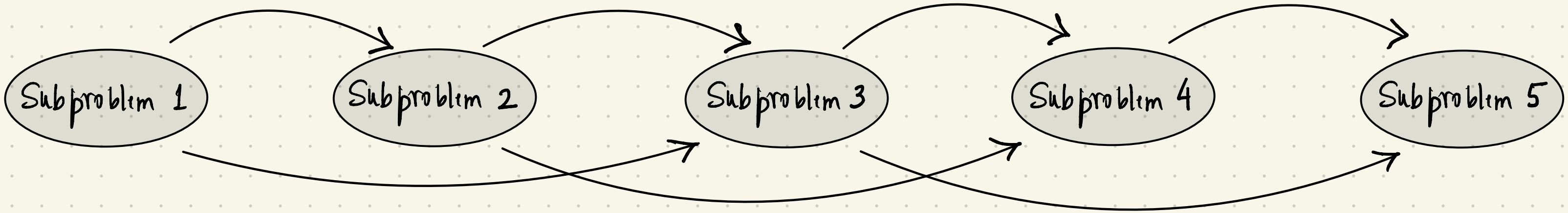


1. Identify a small number of subproblems

2. Show how to quickly and correctly solve "large" subproblems given solution of "smaller" ones.

3. " " " " " final problem given solutions of all other subproblems.

A DAG VIEW OF DYNAMIC PROGRAMMING



1. Identify a small number of subproblems

   e.g., max wt independent set of G_i for $i \in \{0, 1, 2, \ldots, n\}$

2. Show how to quickly and correctly solve "large" subproblems given solution of "smaller" ones.

   e.g., recurrence $A[i] = \max\{A[i-1], A[i-2] + w_i\}$.

3. " " " " " final problem given solutions of all other subproblems.

   e.g., return $A[n]$

KNAPSACK

KNAPSACK

input : 1

2

KNAPSACK

input : ① n items, each having a

- value v_i (non negative)
- size s_i (non negative and integral)

②

KNAPSACK

input : ① n items, each having a
- value v_i (non negative)
- size s_i (non negative and integral)

② capacity C (non negative and integral)

KNAPSACK

input : ① n items, each having a

- value v_i (non negative)
- size s_i (non negative and integral)

② capacity C (non negative and integral)

output : a subset $S \subseteq \{1, 2, \dots, n\}$

that maximizes $\sum_{i \in S} v_i$

KNAPSACK

input : (1) n items , each having a

- value v_i (non negative)
- size s_i (non negative and integral)

(2) capacity C (non negative and integral)

output : a subset $S \subseteq \{1, 2, \dots, n\}$

that maximizes $\sum_{i \in S} v_i$

subject to $\sum_{i \in S} s_i \leq C$

EXAMPLE

item	value	size
1	3	4
2	2	3
3	4	2
4	4	3

capacity = 6

EXAMPLE

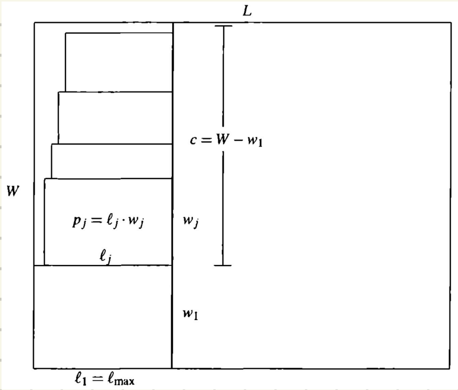
item	value	size
1	3	4
2	2	3
3	4	2
4	4	3

capacity = 6

APPLICATIONS

APPLICATIONS

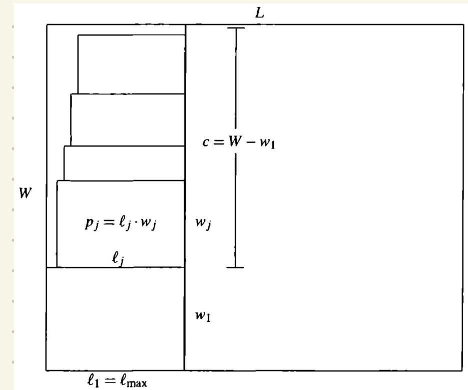
Scheduling advertisements



Profit - maximizing cutting

APPLICATIONS

Scheduling advertisements



Profit - maximizing cutting

Other applications : Credit assignment, Cryptosystems, ...

OPTIMAL SUBSTRUCTURE

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

Case I : item $n \notin S$

Case $\underline{\text{II}}$: item $n \in S$

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

Case I : item $n \notin S$

$\Rightarrow$ S must be optimal for the first $(n-1)$ items

Case II : item $n \in S$

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

Case I : item $n \notin S$

$\Rightarrow$ S must be optimal for the first $(n-1)$ items and residual capacity C.

Case II : item $n \in S$

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

Case I : item $n \notin S$

$\Rightarrow$ S must be optimal for the first $(n-1)$ items and residual capacity C.

Case II : item $n \in S$

$\Rightarrow$ $S \setminus \{n\}$ must be optimal for the first $(n-1)$ items and residual capacity $C - s_n$.

OPTIMAL SUBSTRUCTURE

Let $S :=$ a max value solution of given knapsack instance

Case I : item $n \notin S$

$\Rightarrow$ S must be optimal for the first $(n-1)$ items and residual capacity C.

Case II : item $n \in S$

$\Rightarrow$ $S \setminus \{n\}$ must be optimal for the first $(n-1)$ items and residual capacity $C - s_n$.

buffer for item n