ALG 11

Dynamic programming

- Knapsack problem unlimited
- Knapsack problem 0/1

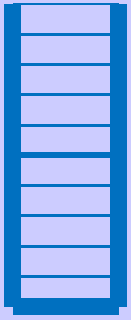
Knapsack problem

There are N items,
an item weight (volume) is Vi and its cost is Ci, (i = 1, 2, ..., N)
and there is a knapsack (=container) of (weight) capacity K.
The knapsack is to be loaded with items in such way that:

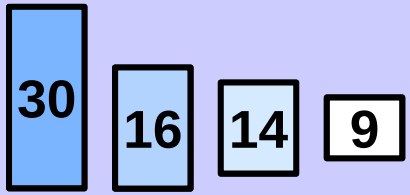
A. Capacity K is not exceeded.
B. The knapsack contents cost is maximized.

Unlimited variant -- Any item might be used more times.
0/1 variant -- Any item might be used at most once.

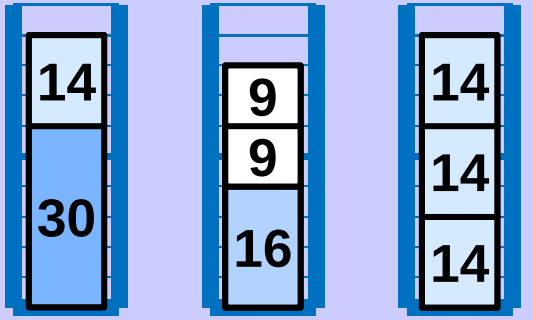
Schematic knapsack of capacity 10



Items and their costs,
height ~ weight



A few possible loads



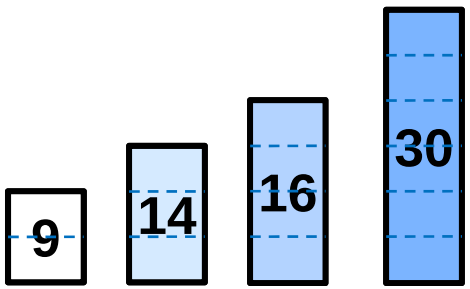
Unlimited Knapsack problem

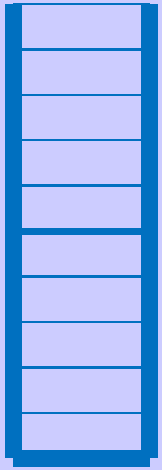
Knapsack capacity = 10

Example

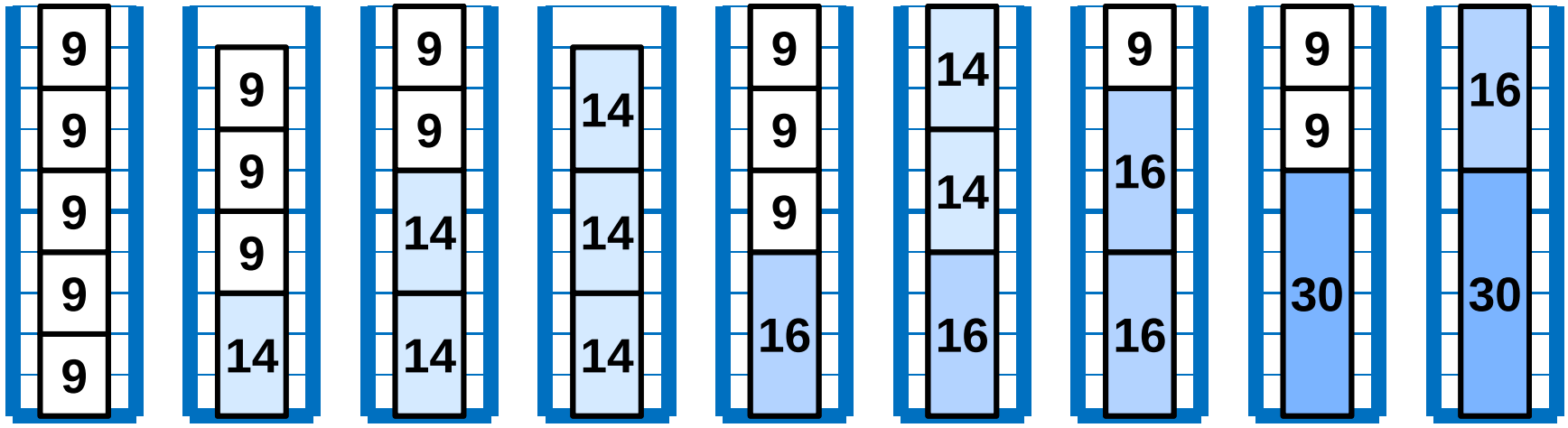
N = 4

Weight	2	3	4	6
Cost	9	14	16	30





Some possible loads and corresponding costs:



Unlimited Knapsack problem

Utilise K+1 knapsacks, with capacities 0, 1, 2, 3, ..., K.

The value of the optimum load of a knapsack of capacity K is equal to the maximum of values

-- (optimum load of knapsack of capacity K - V1) + C1,
-- (optimum load of knapsack of capacity K - V2) + C2,
...
-- (optimum load of knapsack of capacity K - VN) + CN.

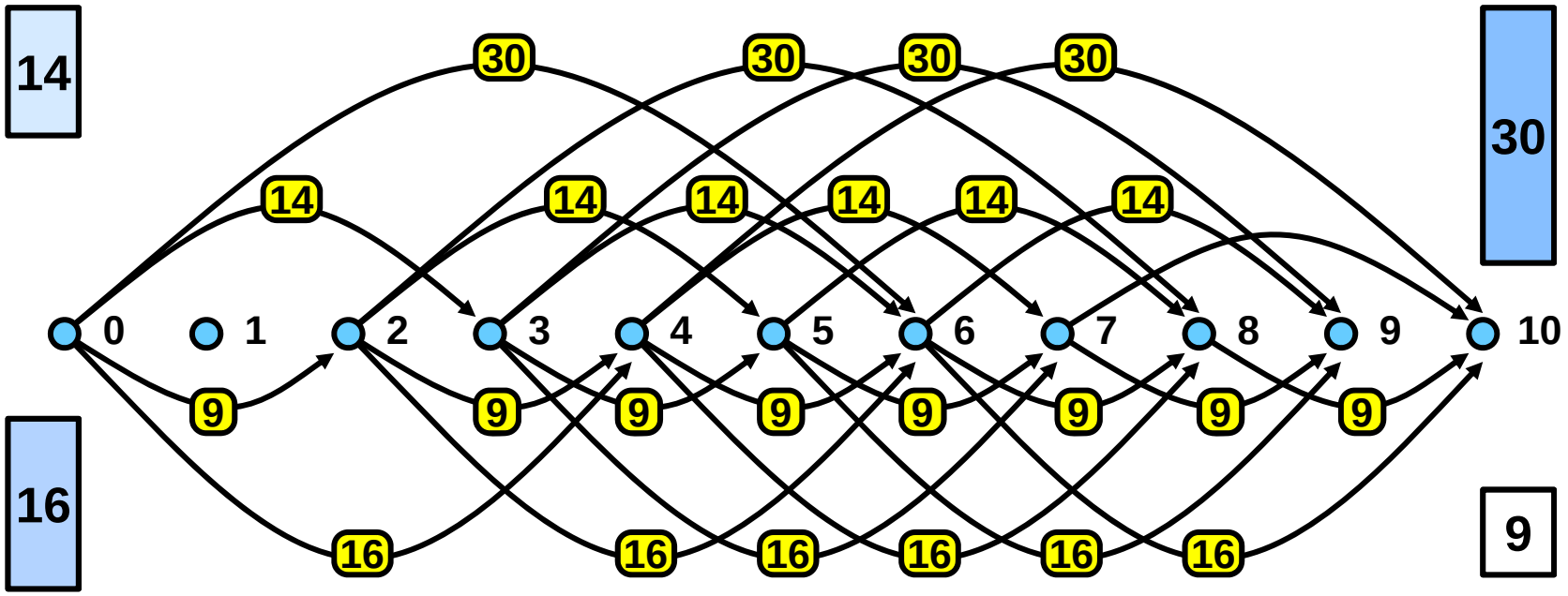
Optimum load of knapsack of capacity K - Vi (i = 1..N) is the same problem as the original one, only the data (capacity) are smaller. The solutions can be precomputed by the standard DP approach and stored in a 1D table.

⇨ Unlimited Knapsack problem can be viewed as a problem of finding longest path in a DAG. The solution methods are identical.

Unlimited Knapsack problem -- transformed to DAG

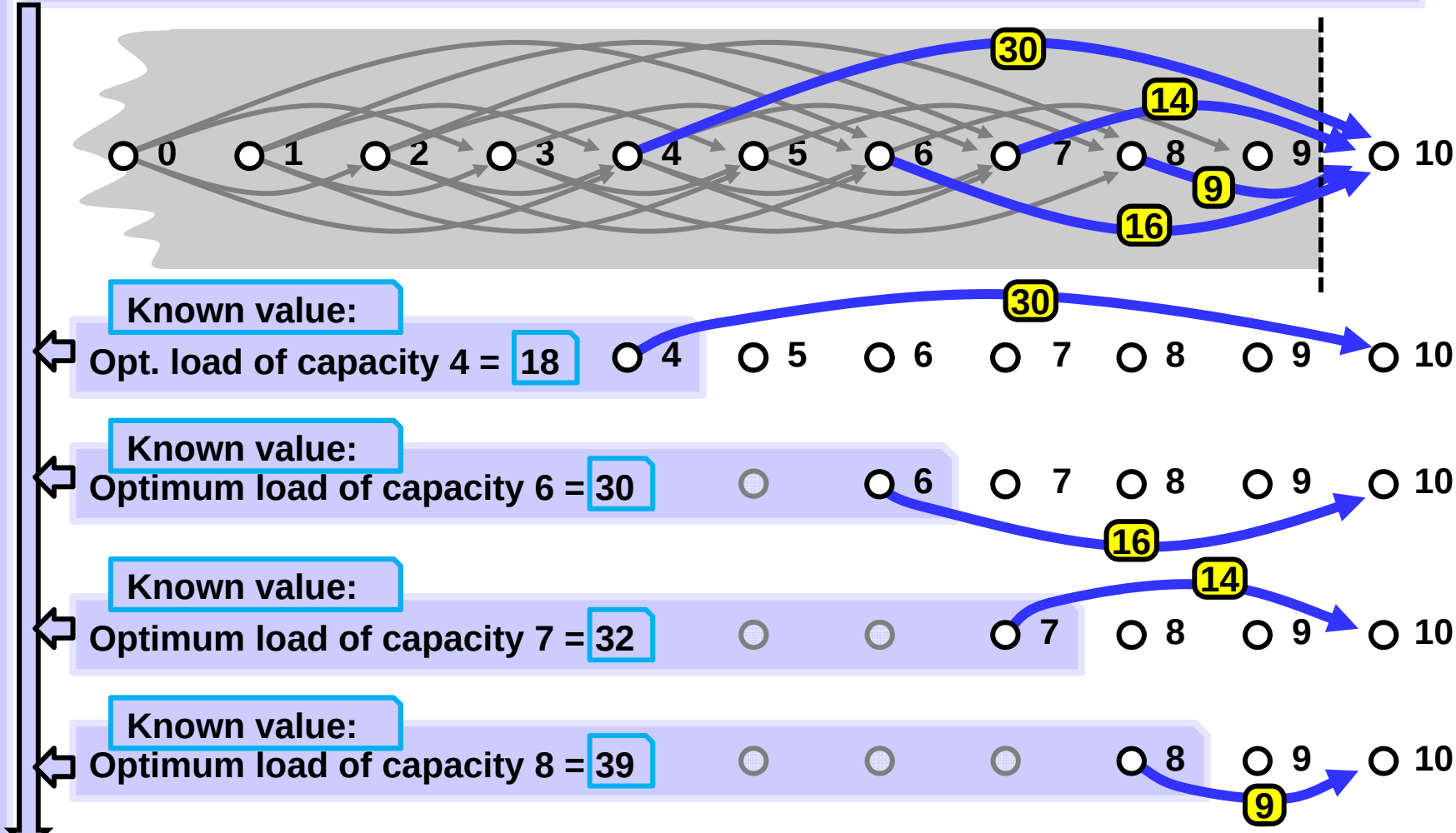
DAG:
Nodes: Capacities 0, 1, 2, 3, ..., K.
Edges: From any node X to nodes of capac. X+V1, X+V2, ..., V+VN, the costs of these edges are C1, C2, ..., CN, respectively

Example **K = 10, N = 4, Vi = (2, 3, 4, 6), Ci = (9, 14, 16, 30), i = 1..4.**



Unlimited Knapsack problem -- transformed to DAG

Optimum load of capacity 10 = ??

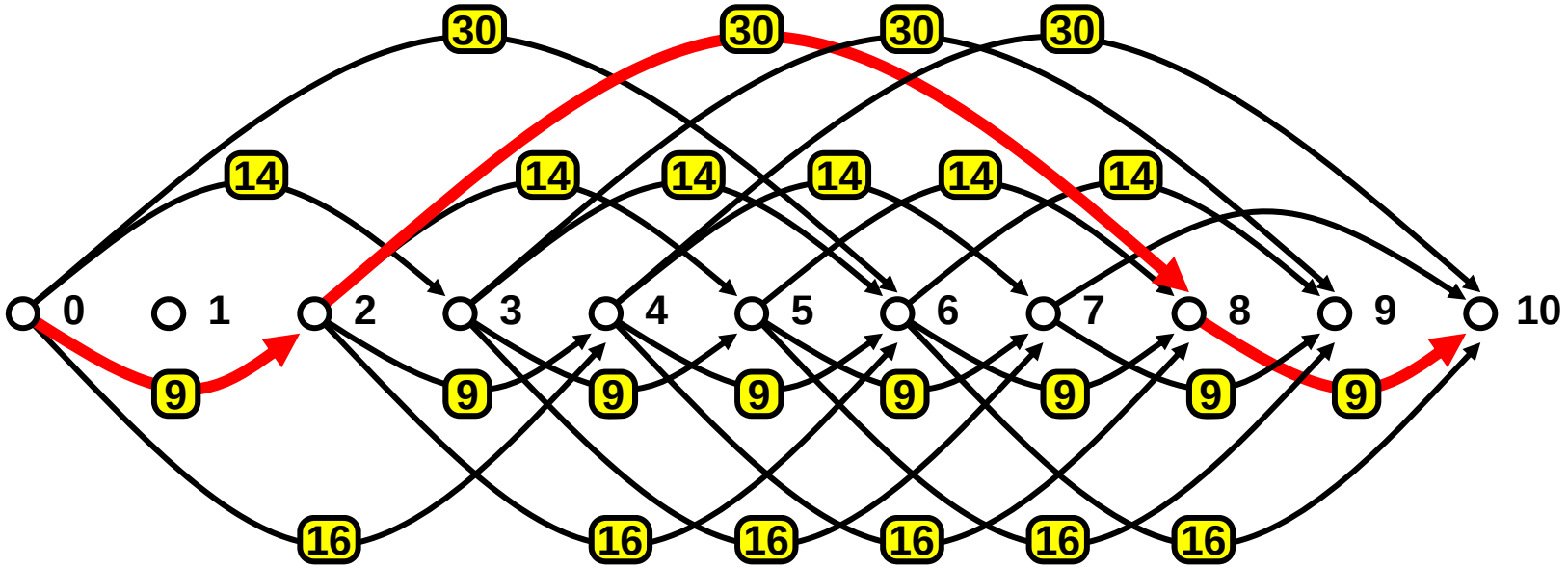


Optimum load of capacity 10 = max(18 + 30, 30 + 16, 32 + 14, 39 + 9) = 48

Unlimited Knapsack problem

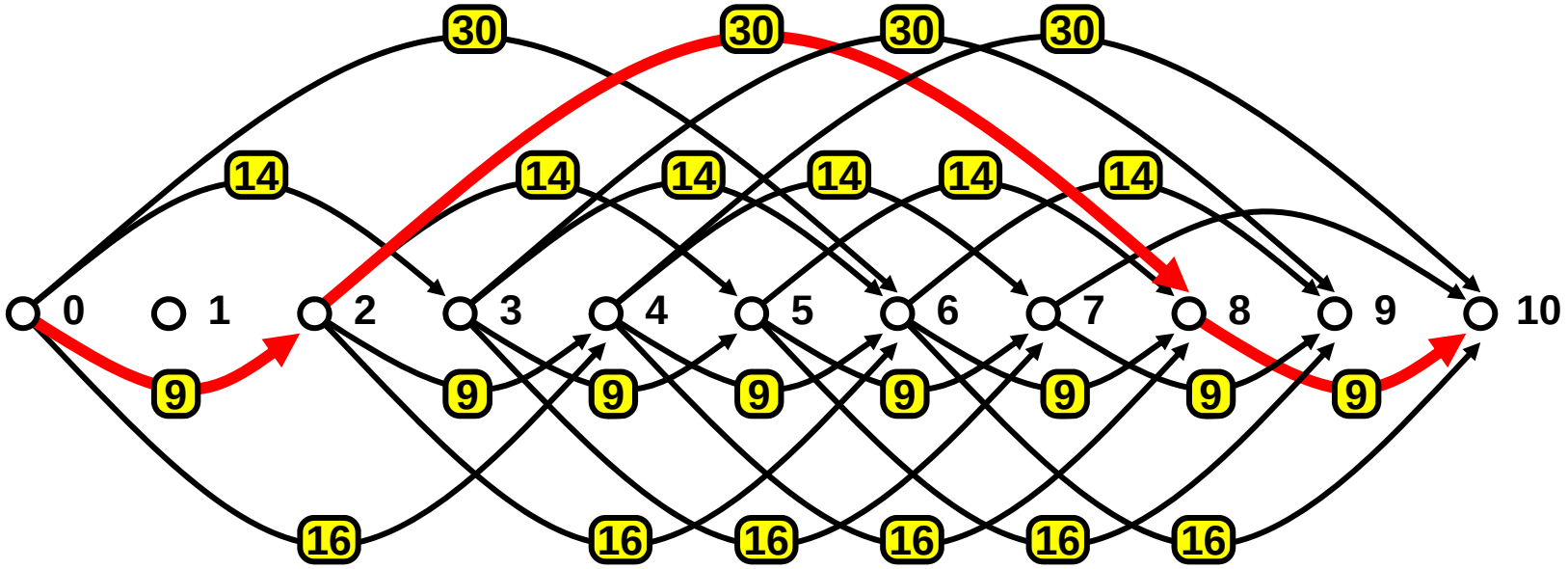
The longest path corresponds to the optimum knapsack load. Two edges of cost 9 and one of cost 30, total cost = 48.

The knapsack is optimally loaded with two items of weight 2 and cost 9 and one item of weight 6 and cost 30.



Unlimited Knapsack problem -- asymptotic complexity

DAG contains K+1 nodes and (slightly) less that K*N edges.
Thus, there are V = Θ(K) nodes and E = O(K*N) edges.
Asymptotic complexity of finding the longest path is Θ(V+E), therefore, Unlimited Knapsack problem complexity is O(K + K*N) = O(K*N).



Unlimited Knapsack problem-- Asymptotic complexity

Apparent discrepancy?

1. Literature: NP-hard problem, no effective algorithm is known.
2. ALG OI: DP solves the problem effectively in time O(N*K).

The time of DP solution depends linearly on capacity K.

Example
Big capacity, e.g. 2^{64} can be specified by a very short string:
Capacity = 18446744073709551616.
Let N = 3, items (weight, cost): (2, 345), (3, 456), (5, 678).
Input data fits to cca 100 bits < 16 Bytes = "two long ints"
The DP method will run over 584 years provided that it fills 10^9 table values in 1 second.

The time of DP solution depends exponentially on the length of the string which definines capacity K.

0/1 Knapsack problem

Any item can be used at most once.

A suitable subset of available items has to be chosen.
Each possible subset is specified by a characteristic 0/1-vector of length N. The position in the vector corresponds to an item, tha values 0 and 1 correspond to the item not being or being present in the subset, respectively.

There are 2^N 0/1-vectors of length N. Therefore, a systematic checking of all subsets would take exponentially long time with respect to N, it would be too slow.

DP method offers (for relatively small values) a significantly more efficient solution method.

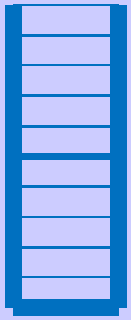
0/1 Knapsack problem

Example

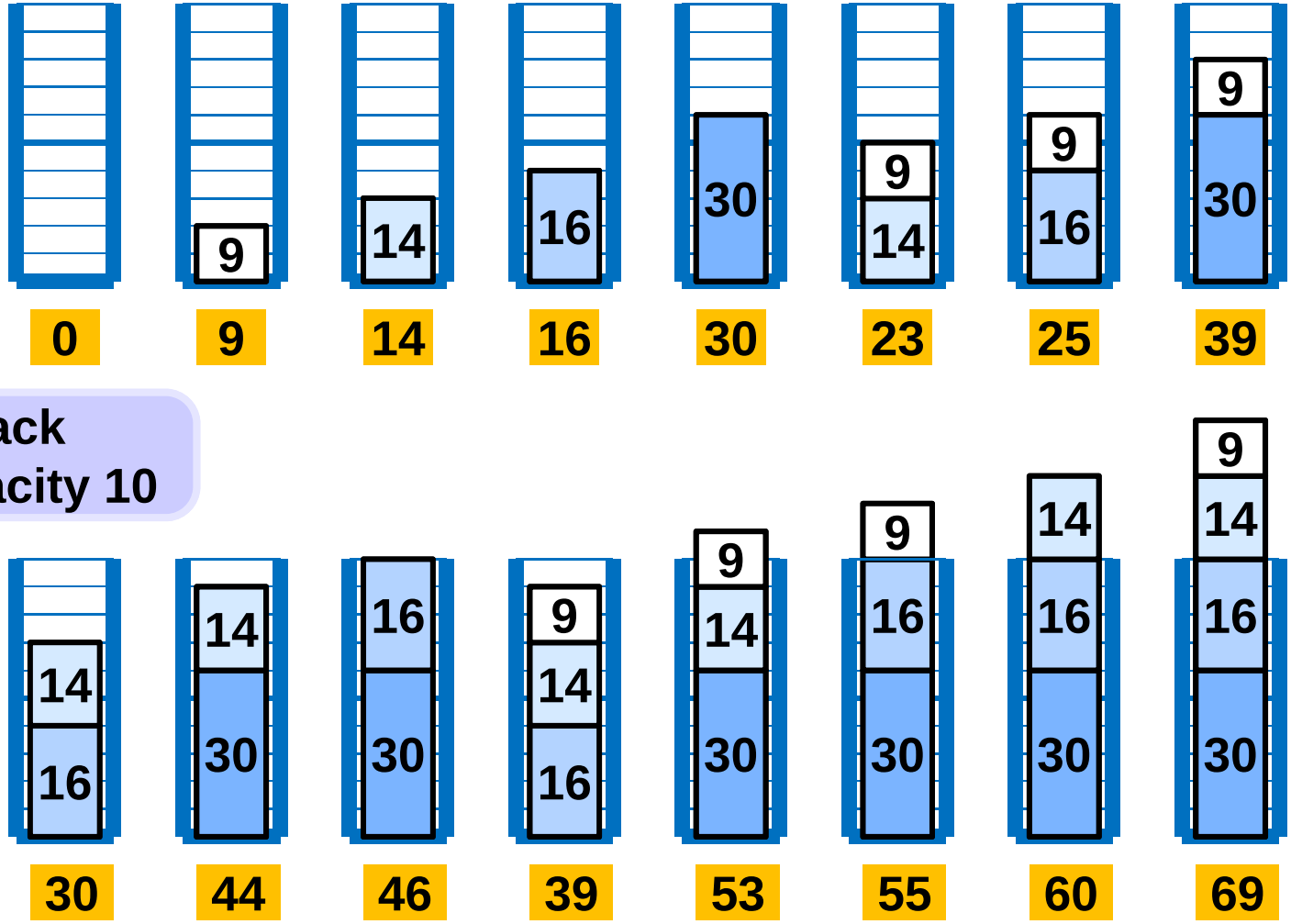
N = 4

Weight	Cost
2	9
3	14
4	16
6	30

Knapsack of capacity 10



All $2^4 = 16$ subsets of 4 items and their costs:



0/1 Knapsack -- solution

Use K+1 knapsacks, of capacities 0, 1, 2, 3, ..., K.
Consider N+1 sets of given items.

Set 0 contains no item.
Set 1 contains item 1.
Set 2 contains items 1 and 2.
Set 3 contains items 1, 2, 3.
...
Set N contains items 1, 2, 3, ..., N.

The order of the items is arbitrary, but it is fixed.

For each capacity 0...K and for each set 0...N:
Define the same DP problem as the original one and solve
all those problems in ascending order of their sizes and capacities.

0/1 Knapsack problem -- solution

Denote by symbol P(x, y) the problem with
the set of items 1, 2, ..., x and with knapsack capacity y.
Denote by symbol Opt(x, y) the optimal solution of P(x, y) .
To solve P(x,y), use optimal solutions of P(x-1, _):

There are just two possibilities for the item x:
Either it is included in Opt(x, y) or it is not.
If it is included then the current cost of knapsack load is equal to the solution which considers items 1..x-1 and which knapsack capacity is smaller by the weight of item x, plus the cost of item x.
If it is not included then the current cost of knapsack load is equal to the solution which considers items 1..x-1 and which knapsack capacity is equal to the current one.
The better of the costs is the current optimal solution:

$$Opt(x, y) = \max(Opt(x-1, y-Vx) + Cx, Opt(x-1, y)).$$

Obvious base cases: Opt(0, y) = Opt(x, 0) = 0,for x = 0..N, y = 0..K.

0/1 Knapsack problem -- solution

Opt(0, y) = Opt(x, 0) = Opt(0, 0) = 0.

For x = 1..N, y = 1..K:
Opt(x, y) = max(Opt(x-1, y-Vx) + Cx, Opt(x-1, y)).

if y-Vx < 0, set Opt(x, y-Vx) := $-\infty$ (do not store it in the table).

Values Opt(x,y) are stored in 2D table of size (N+1)×(K+1)
with row index x (items in any order)
and column index y (knapsacks capacities in increasing order).

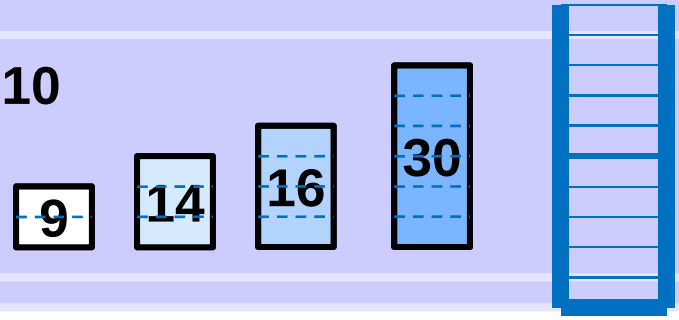
A table of predecessors (previous item is or is not among the items in the optimal solution) is the size as the main table and enables the reconstruction of the optimal solution:
The item predecessor is in the previous row and its position is either y (item x was not added) or y-Vx (item x was added) .

0/1 Knapsack problem -- solution

Example

N = 4 Capacity = 10

Weight	2	3	4	6
Cost	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

16 0 30 0

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 Knapsack problem

Expressed as optimum path in DAG

DAG nodes are values Opt(x, y), x = 0..N, y = 0..K,
there are (N+1)*(K+1) nodes in DAG.

Edges: There are two in-edges in node Opt(x, y) (x > 0, K > 0):

-- Opt(x-1, y) --> Opt(x, y)
with cost 0 (item x not added to the knapsack),

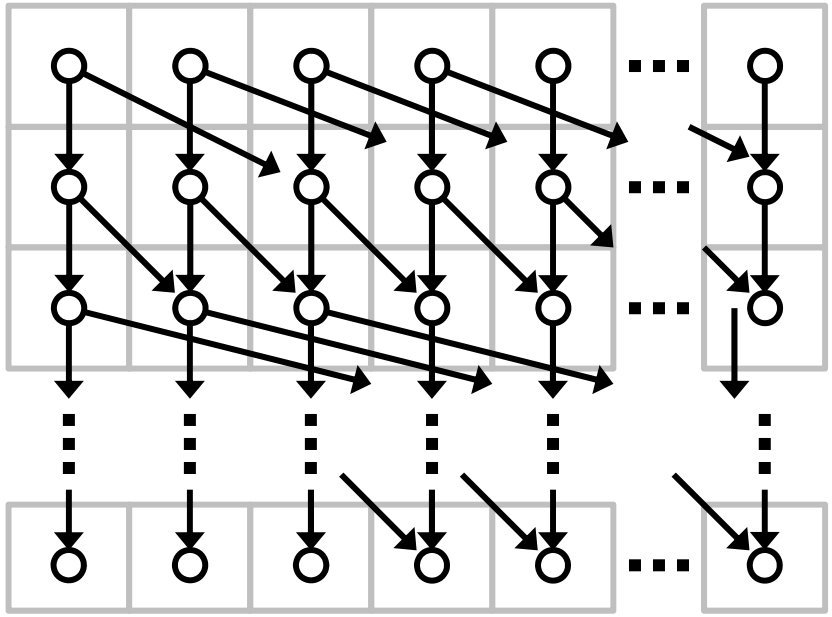
-- Opt(x-1, y-Vx) --> Opt(x, y) (only if y-Vx $\geq$ 0)
with cost Cx (= the cost of added item x).

In this DAG we search for the longest (=most costly) path
using standard DP procedure.

What is the topological order of this DAG?

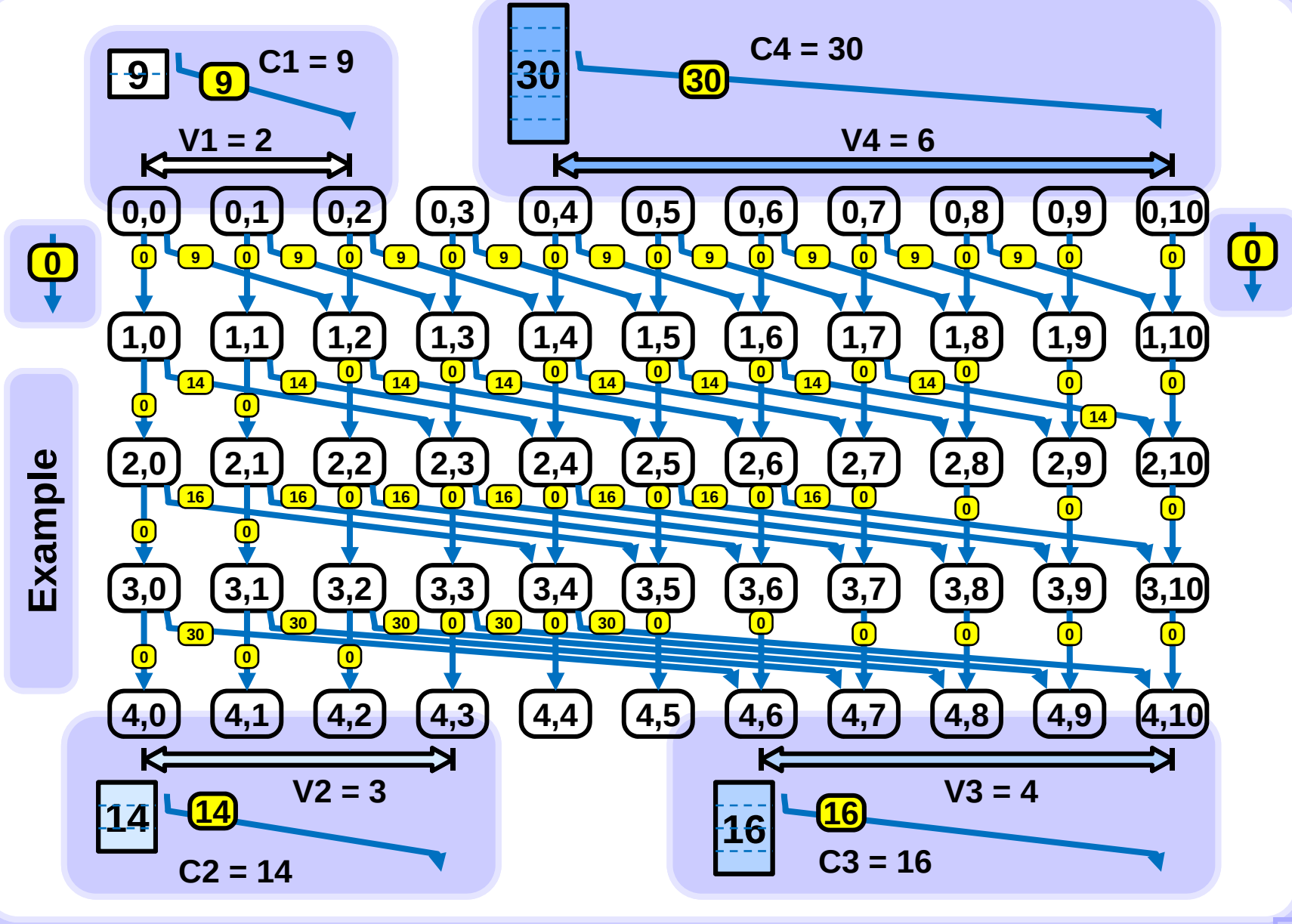
0/1 Knapsack problem - DAG topological order

Informally, imagine DAG drawn in the table, with node Opt(x, y) in the middle of the cell with indexes x a y. Then, each edge connects a cell in one row with a cell in the immediately next row. Traversing this DAG in the row-by-row order (and left-to-right order in each row), which is the same order in which the DP table contents is calculated, respects the topological order of the DAG. A node is processed only after both its parents were processed.



In this case, it is not necessary to put all nodes in topological order on a single line. The "table-like" (partial) order is much more easy to view/analyse.

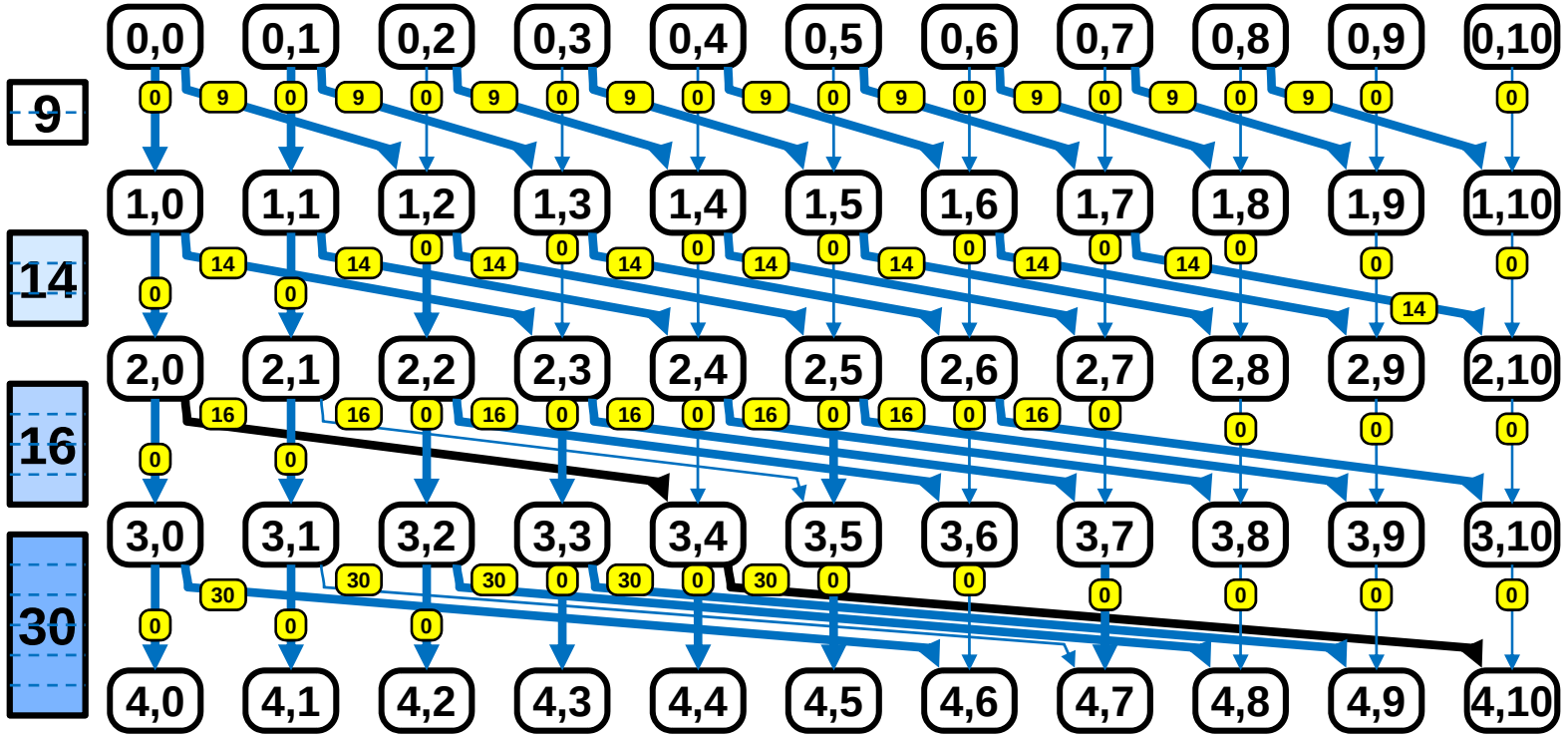
0/1 Knapsack Problem -- DAG



0/1 Knapsack problem -- reconstruction of optimal solution using predecessor table

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4



0/1 Knapsack Problem

Asymptotic complexity

Table ... Size ... $(N+1)*(K+1) \in \Theta(N*K)$.
Fill one table field ... $\Theta(1)$.
Fill the entire table ... $\Theta(N*K*1) = \Theta(N*K)$.
Optimum solution reconstruction ... $\Theta(N)$.
Total ... $\Theta(N*K + N) = \Theta(N*K)$.

DAG .... Nodes... $(N+1)*(K+1) \in \Theta(N*K)$.
Edges ... at most $2*(N+1)*(K+1) \in O(N*K)$.
Finding optimum path ... $\Theta(|nodes|+|edges|) = \Theta(N*K)$.

The asymptotic complexity of both variants of Knapsack problem (0/1 and unbounded) is $\Theta(N*K)$.

Simultaneously, it holds:

The asymptotic complexity of DP solution is exponencial with respect to the length of the string defining the knapsack capacity K.