Dynamic programming

Athasit Surarerks

THEORETICAL MODEL

March 28, 2005

Introduction

- Dynamic Programming is a general algorithm design paradigm.
- Dynamic Programming is a technique for solving problems “bottom-up”:
- first, solve small problems, and then use the solutions to solve larger problems.
- What kind of problems can Dynamic Programming solve efficiently?

Introduction

- **Optimal substructure: The optimal solution contains optimal solutions to sub-problems.**
- **Overlapping sub-problems: the number of different sub-problems is small, and a recursive algorithm might solve the same sub-problem a few times.**

Optimization problems

- **Optimization problem is an important and practical class of computational problems. For most of these, the best known algorithm runs in exponential time.**

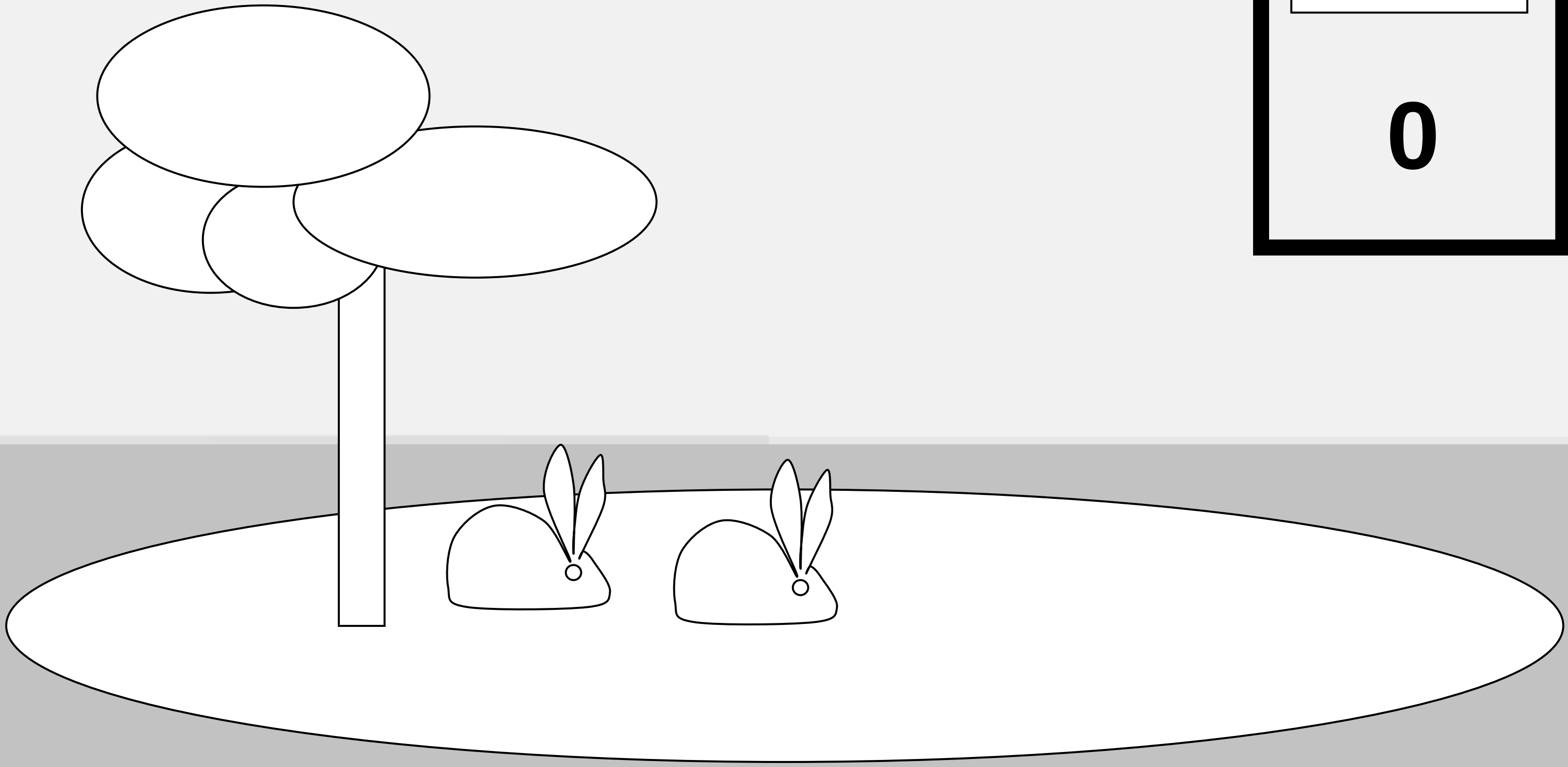
Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.

month

0



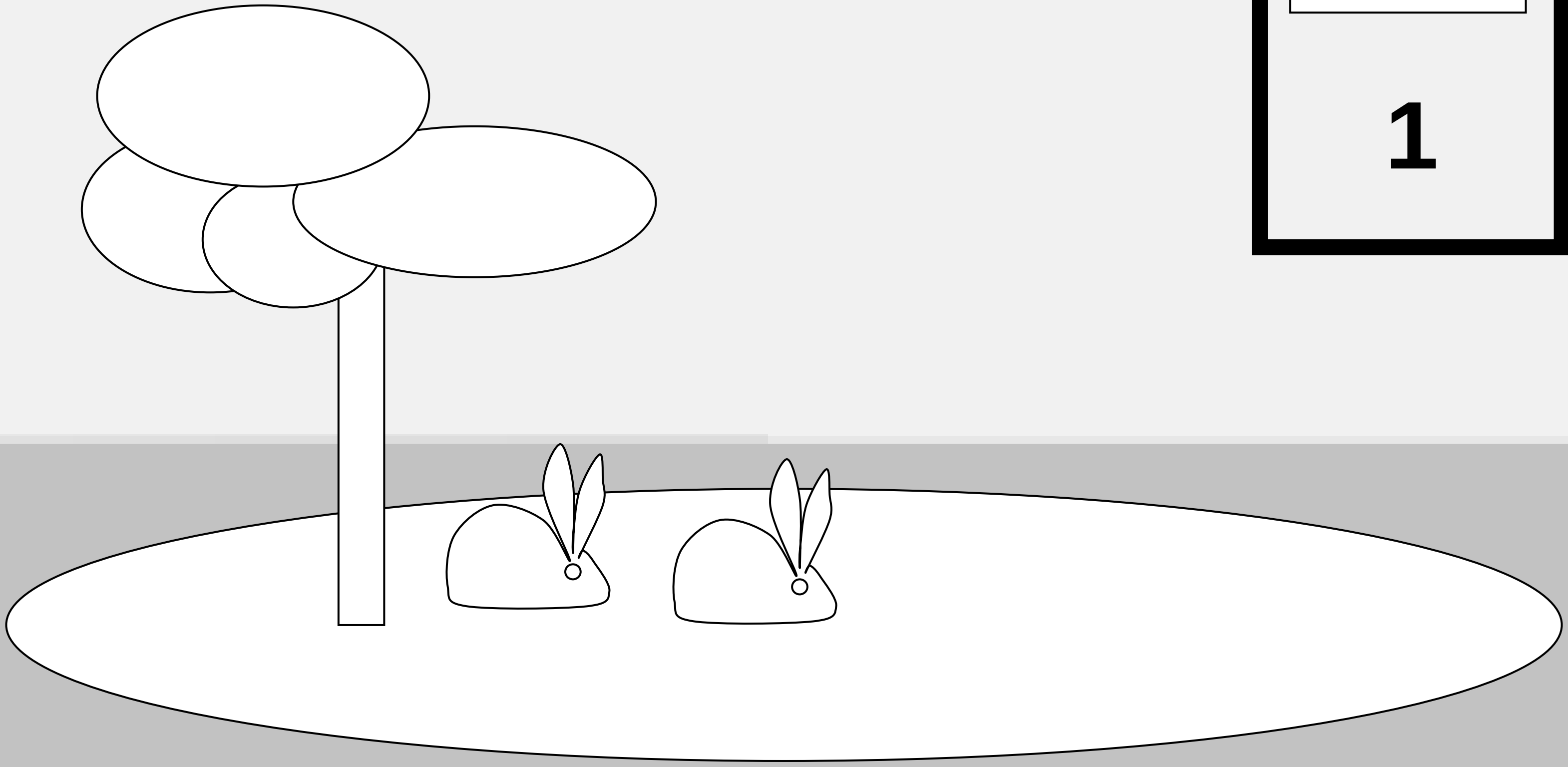
Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.

month

1



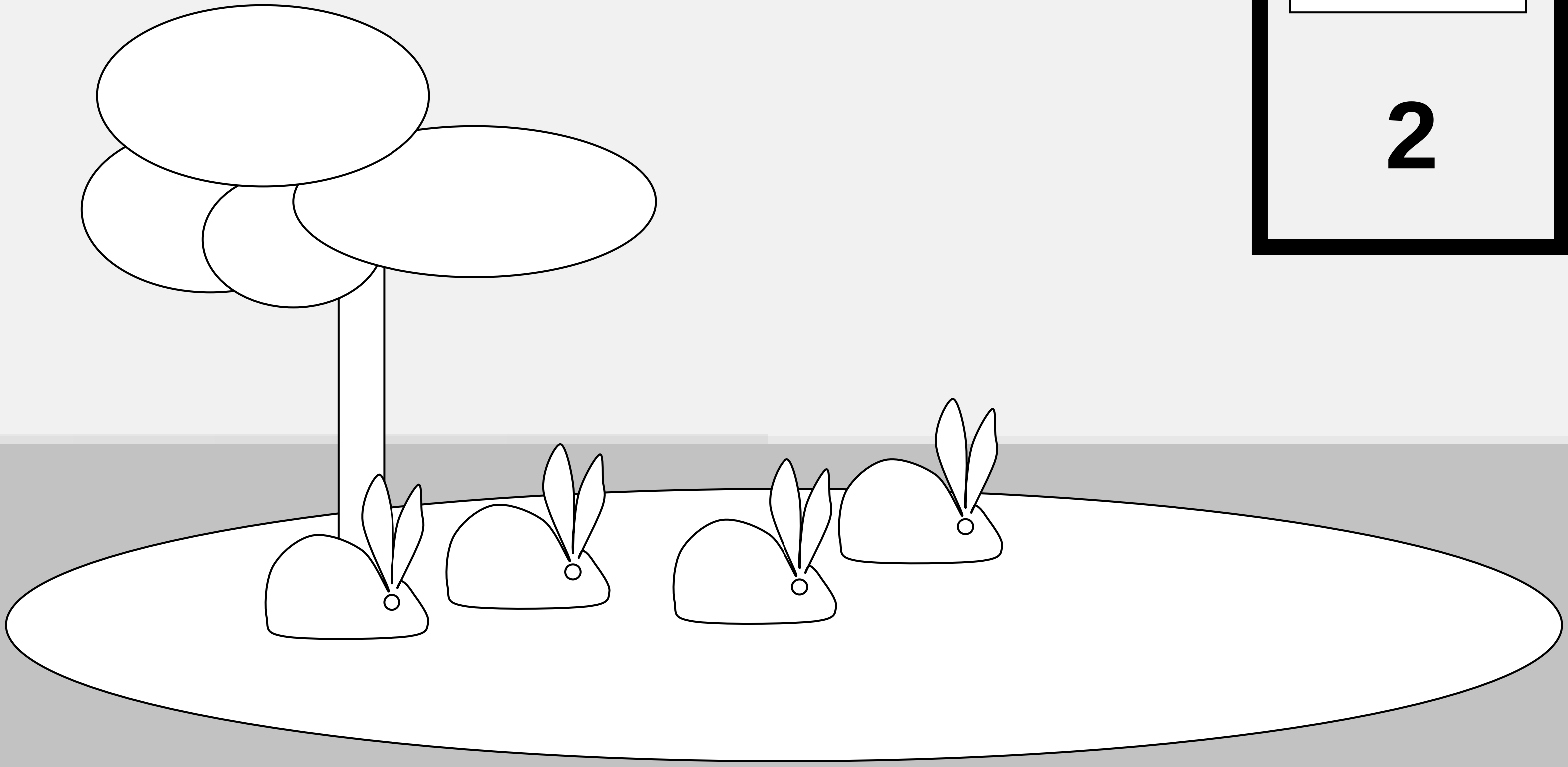
Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.

month

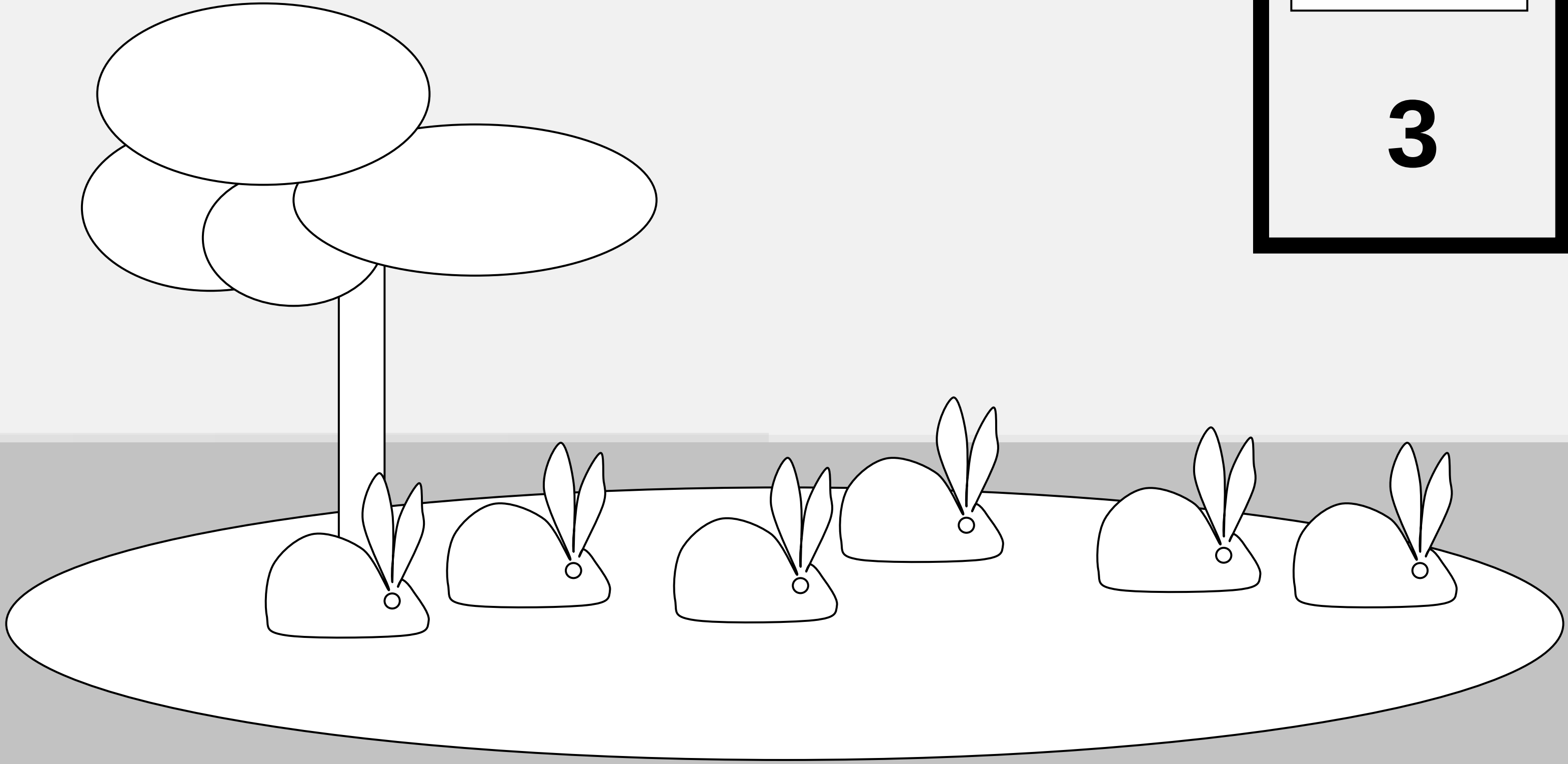
2



Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.



month

3

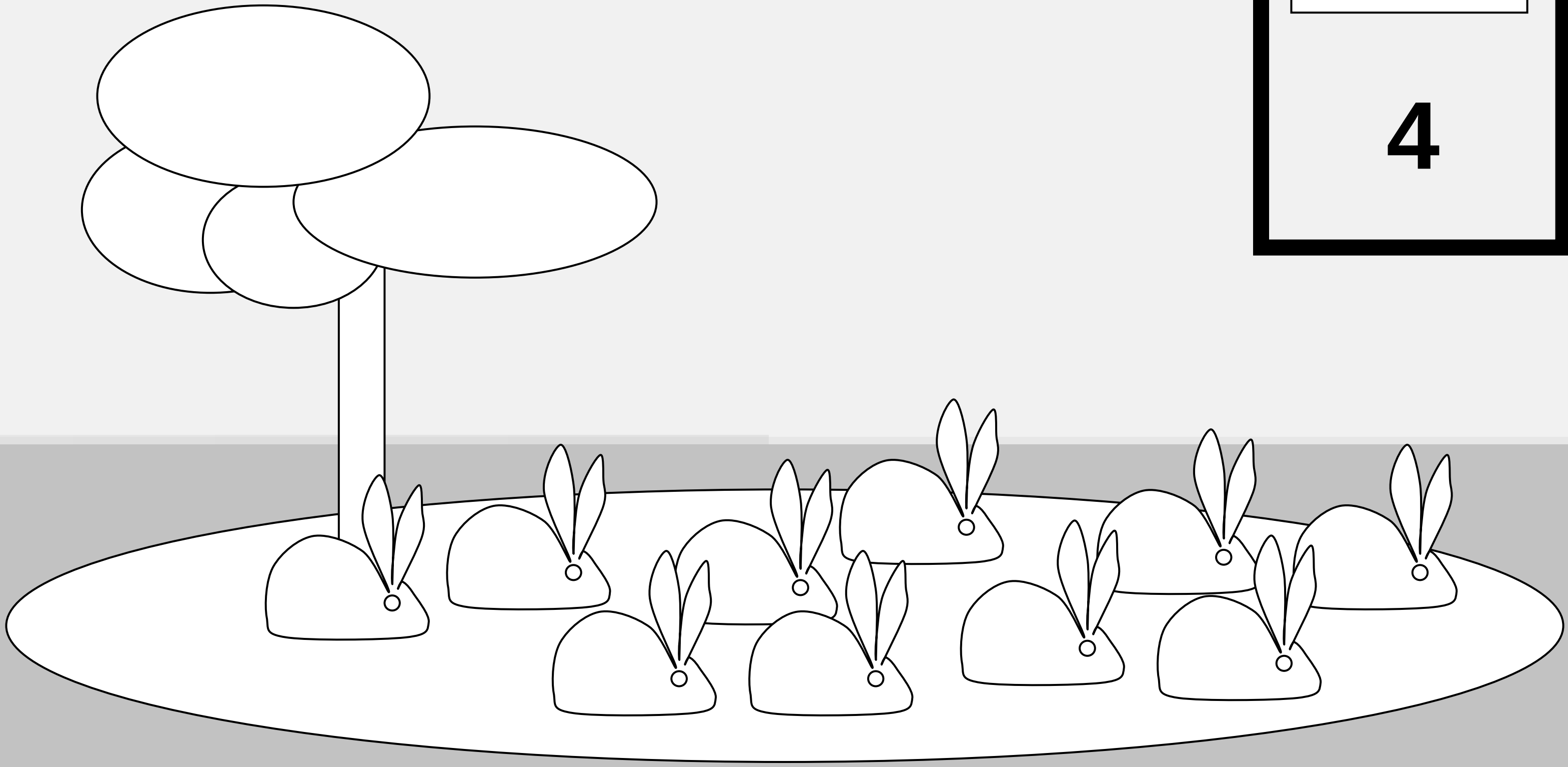
Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.

month

4



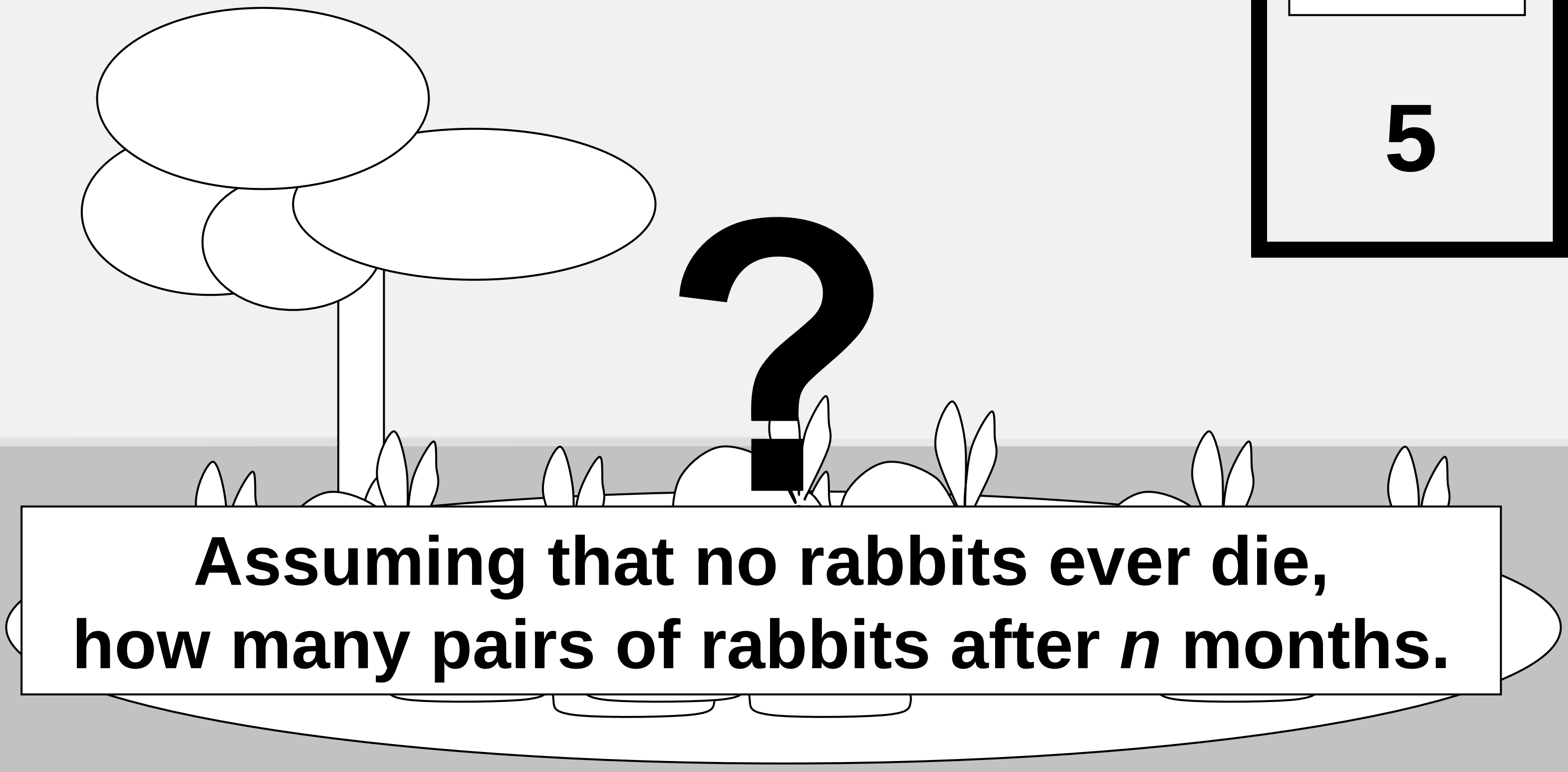
Rabbits on an island

By Leonardo di Pisa, 13th century

A pair of rabbits does not breed until they are two months old, then each pair produces another pair each month.

month

5



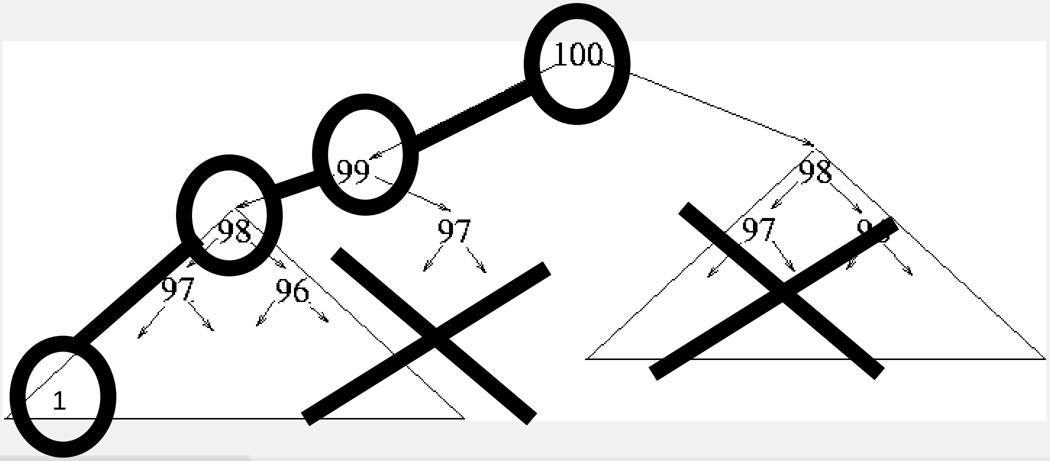
Assuming that no rabbits ever die,
how many pairs of rabbits after *n* months.

Rabbits on an island

By Leonardo di Pisa, 13th century

Fibonacci Number

$F_0 = 1$
$F_1 = 1$
$F_2 = F_0 + F_1 = 2$
$F_3 = F_1 + F_2 = 3$
$F_4 = F_2 + F_3 = 5$
…
$F_n = F_{n\text{-}2} + F_{n\text{-}1}$

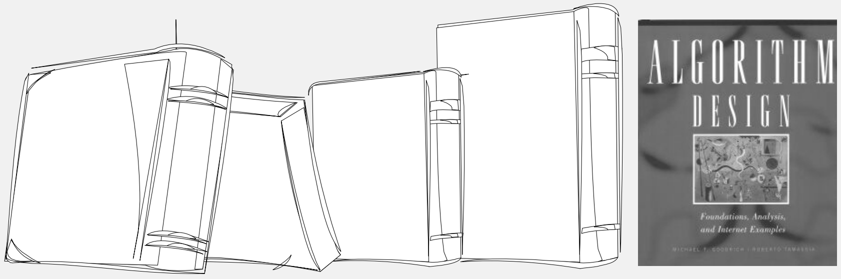


Exponential time → Linear time

OVERLAPPING SUBPROBLEMS

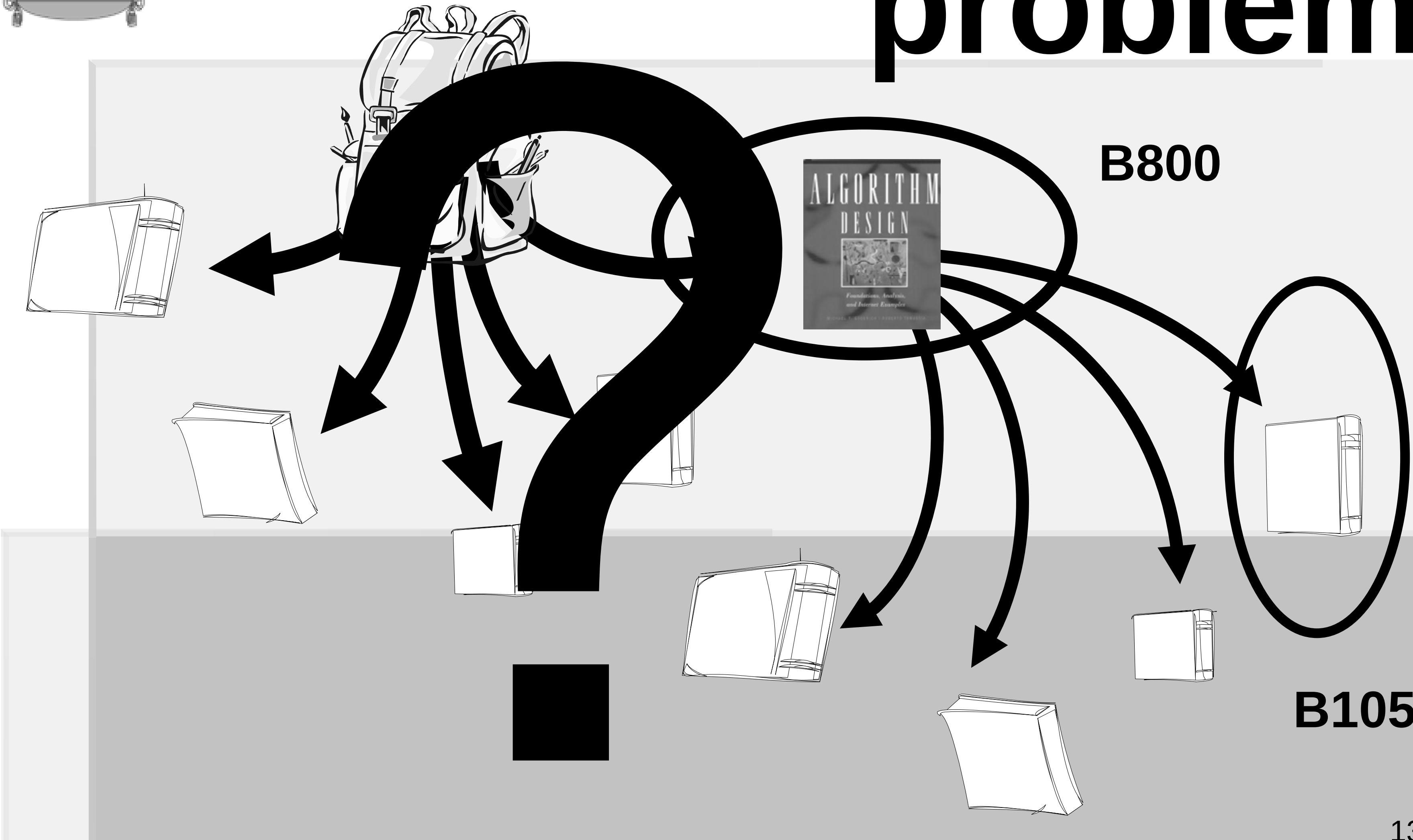
0/1 Knapsack problem

Choose items with maximum total benefit but with some limitation.

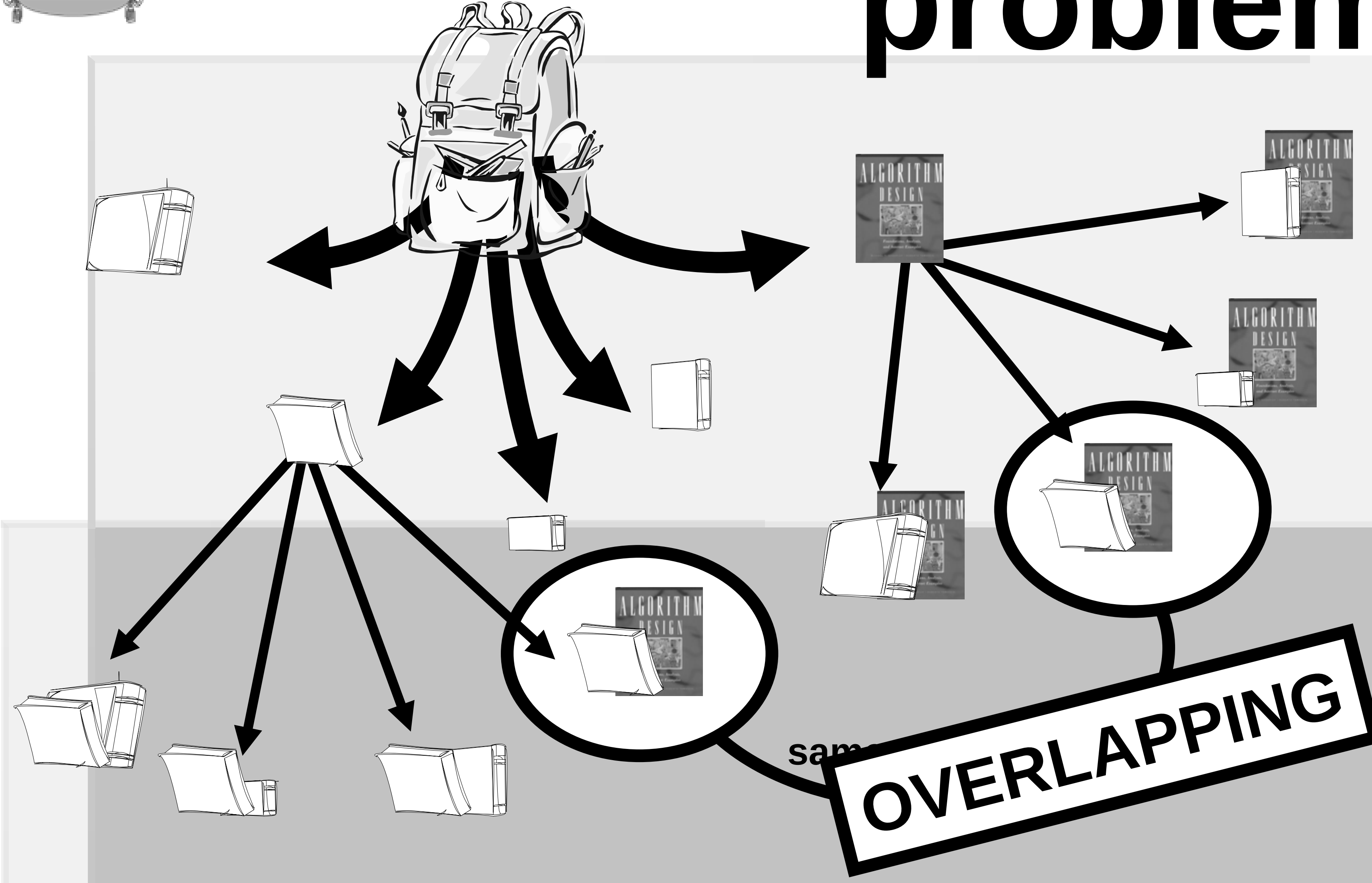


weight	4 kgs	2 kgs	2 kgs	6 kgs	2 kgs
value	B200	B30	B60	B250	B800

0/1 Knapsack problem



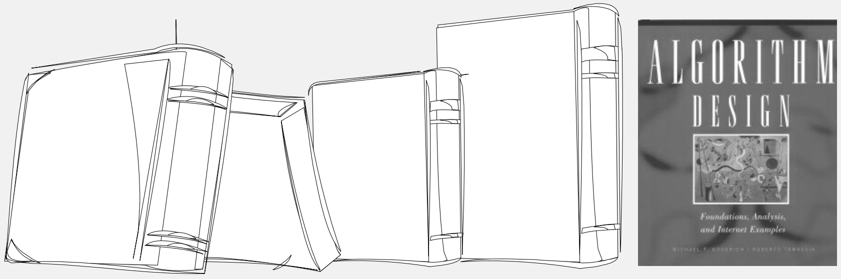
0/1 Knapsack problem



0/1 Knapsack problem

Choose items with maximum total benefit but with some limitation.

Max 9 kgs


ALGORITHM
DESIGN
Foundations, Analysis, and Internet Examples
MICHAEL T. GOODRICH • ROBERTO TAMASSIA

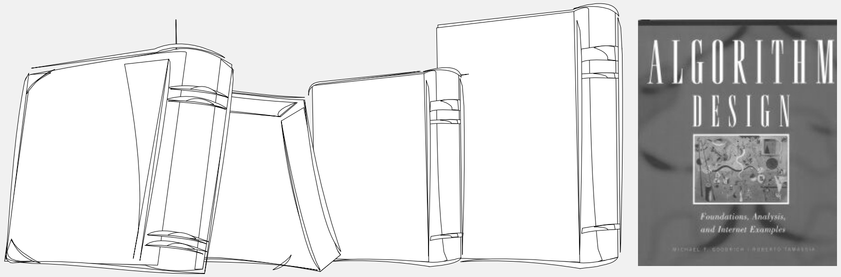
weight	**4 kgs**	**2 kgs**	**2 kgs**	**6 kgs**	**2 kgs**
value	**B200**	**B30**	**B60**	**B250**	**B800**

Number of solutions : 2^n (n : number of books)

0/1 Knapsack problem

Choose items with maximum total benefit but with some limitation.

Max 9 kgs

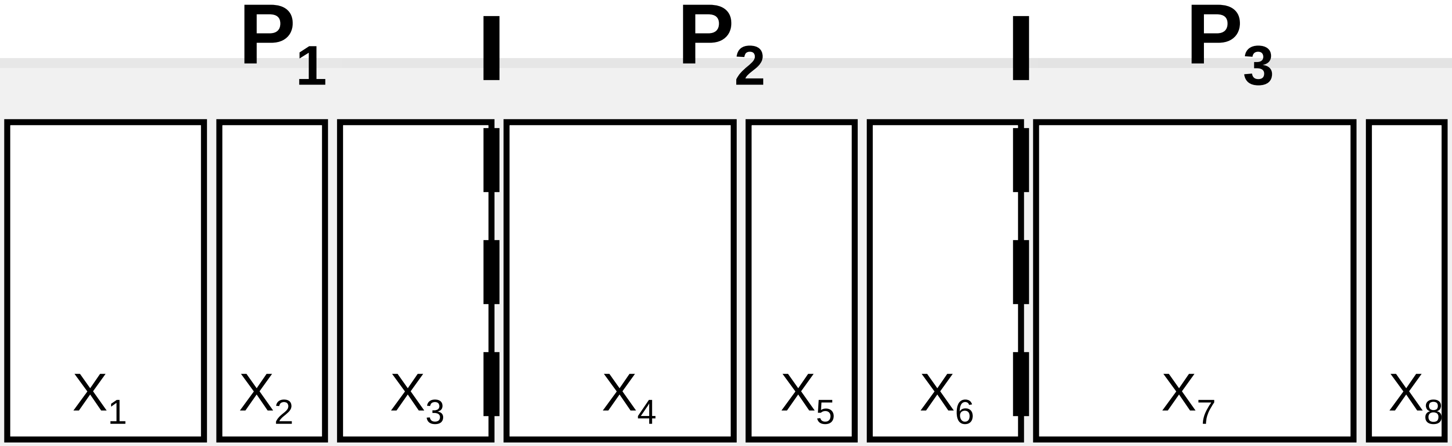

ALGORITHM
DESIGN
Foundations, Analysis, and Internet Examples
MICHAEL T. GOODRICH • ROBERTO TAMASSIA

weight	4 kgs	2 kgs	2 kgs	6 kgs	2 kgs
value	B200	B30	B60	B250	B800

OBJECTIVE FUNCTION: $\max(\sum_{i=1}^{n} x_i v_i)$ $\quad x_i = \begin{cases} 1 & selected \\ 0 & otherwise \end{cases}$

$\sum_{i=1}^{n} x_i w_i \leq W$ **CONSTRAINT**

Linear Partition



Partition all fields into 3 groups with approximately the same size.

OBJECTIVE FUNCTION:

$$\min\left(\max\left(P_j = \sum_{i=s}^{t} x_i\right)\right)$$

NUMBER OF SOLUTIONS $\mathbf{C}_{\mathbf{(n+k-1,k-1)}}$

Printing neatly

Application for word processor

Problem: **English text with *n* words**

word *i* with length w_i (no.of.chars)

Each line contains max *M* chars

Solution: **Close to right justified text**

$(x_1, x_2, x_3, \ldots, x_m)$ x_j : last wordth of line *j*.

Penalty: sum of right blank-end square.

$$S_i = M - \left[\left(x_i - (x_{i-1}+1)\right) + \sum_{k=x_{i-1}+1}^{x_i} w_k \right]$$

blank-end of line *i*

$$\min\left(\sum_{k=1}^{m} S_k^2\right)$$

Objective function

Sequence of matrix multiplication

Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1} \times d_i$ dimension.

Find an algorithm for

$$M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k$$

?

Example: $A_{5\times10} \times B_{10\times20} \times C_{20\times1} \times D_{1\times10}$

Cost of ((AB)C)D) is 1000+100+50 = 1150

Cost of ((AB)(CD) is 10 … 00

Cost of (A(B(C … +500 = 2700

HOW MANY SOLUTIONS ?

Sequence of matrix multiplication

Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1}\times d_i$ dimension.

Find an algorithm for

$$M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k$$

Example: $A_{5\times10} \times B_{10\times20} \times C_{20\times1} \times D_{1\times10}$

Number of solutions = number of sequence of multiplication

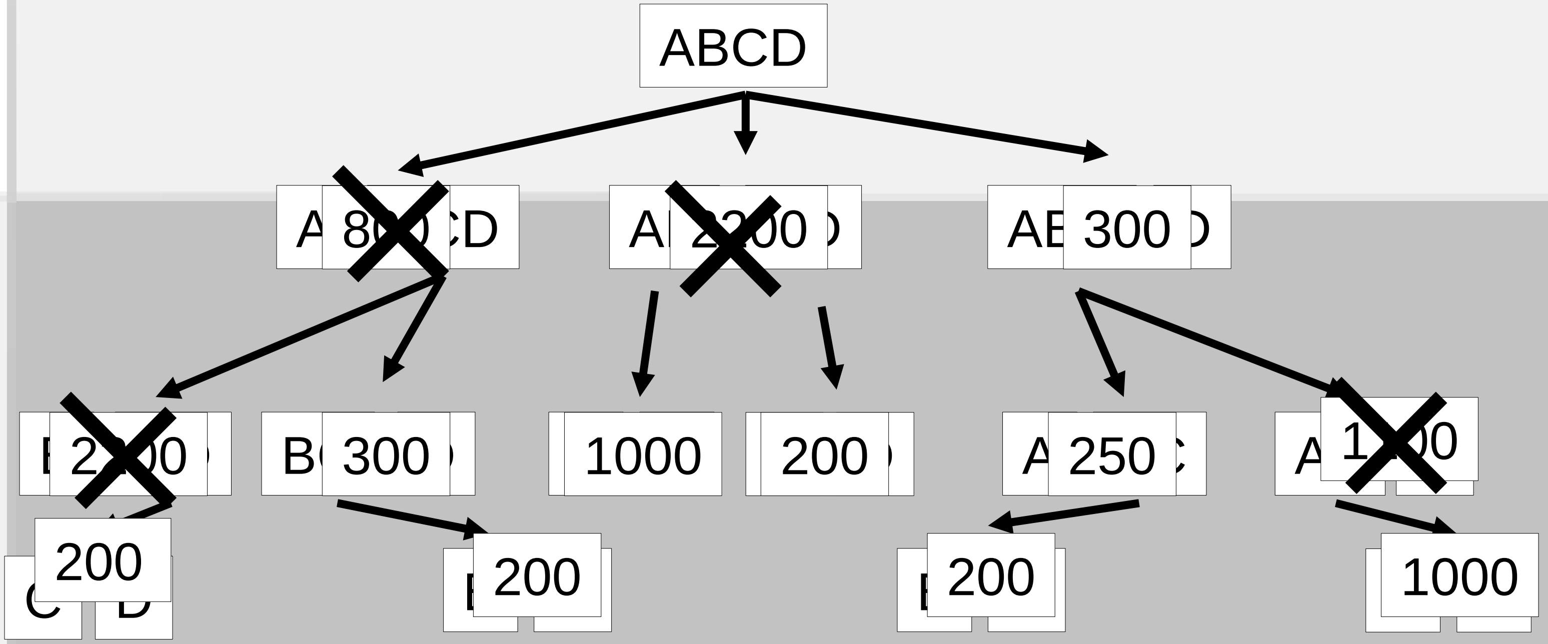
Sequence of matrix multiplication

Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1} \times d_i$ dimension.

Find an algorithm for

$$M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k$$

Example: $A_{5\times10} \times B_{10\times20} \times C_{20\times1} \times D_{1\times10}$



Sequence of matrix multiplication

Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1}\times d_i$ dimension.

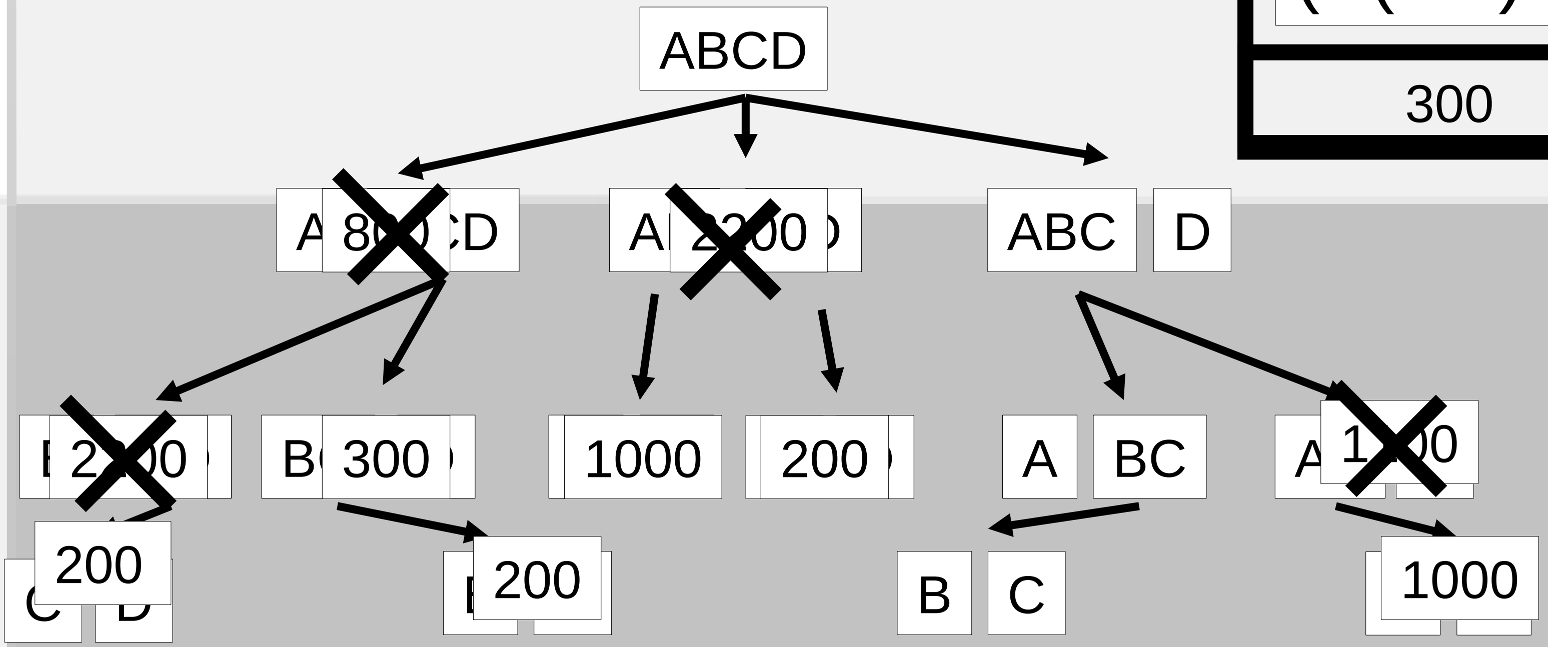
Find an algorithm for

$$\mathbf{M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k}$$

Example: $A_{5\times10} \times B_{10\times20} \times C_{20\times1} \times D_{1\times10}$

(A(BC)D)

300



Sequence of matrix multiplication

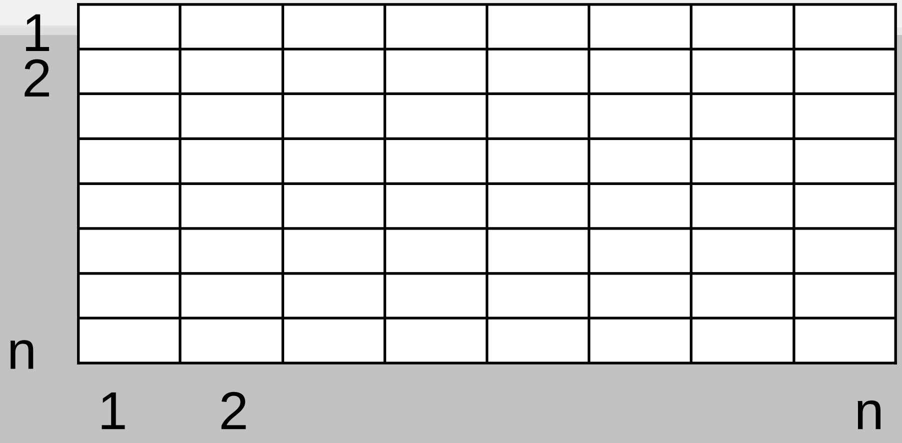
Given M_1 M_2 M_3 M_4 …M_k with d_{i-1}×d_i dimension.

Find an algorithm for

$$\mathbf{M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k}$$

Let $T_{i,j}$ be the cost of multiplication of M_i .. M_j

$$T_{i,j} = \min i \le k \le j - 1\left(T_{i,k} + T_{k+1,j} + d_i \times d_k \times d_j\right)$$



DICTIONARY

Sequence of matrix multiplication

Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1}\times d_i$ dimension.

Find an algorithm for

$$\mathbf{M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k}$$

Let $T_{i,j}$ be the cost of multiplication of M_i .. M_j

$$T_{i,j} = \min i \le k \le j - 1\left(T_{i,k} + T_{k+1,j} + d_i \times d_k \times d_j\right)$$

	1	2						n
1								
2								
n								

DICTIONARY

Sequence of matrix multiplication

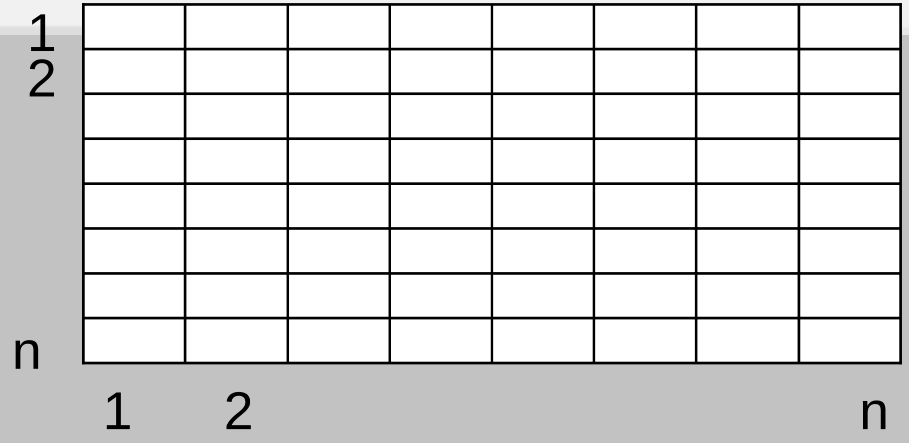
Given M_1 M_2 M_3 M_4 …M_k with $d_{i-1} \times d_i$ dimension.

Find an algorithm for

$$\mathbf{M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k}$$

Let $T_{i,j}$ be the cost of multiplication of M_i .. M_j

$$T_{i,j} = \min i \le k \le j-1 \left(T_{i,k} + T_{k+1,j} + d_i \times d_k \times d_j\right)$$



DICTIONARY

Sequence of matrix multiplication

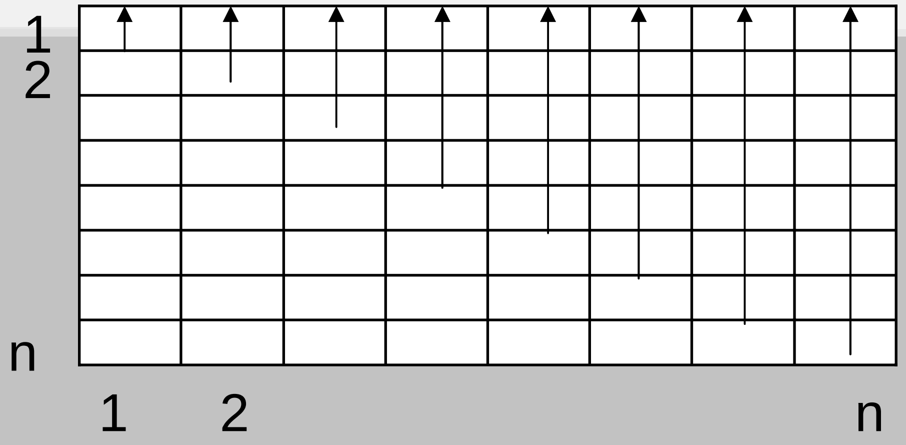
Given M_1 M_2 M_3 M_4 …M_k with d_{i-1}×d_i dimension.

Find an algorithm for

$$M_1 \times M_2 \times M_3 \times M_4 \times \ldots \times M_k$$

Let $T_{i,j}$ be the cost of multiplication of M_i .. M_j

$$T_{i,j} = \min i \le k \le j - 1\left(T_{i,k} + T_{k+1,j} + d_i \times d_k \times d_j\right)$$



DICTIONARY

Optimal binary search tree

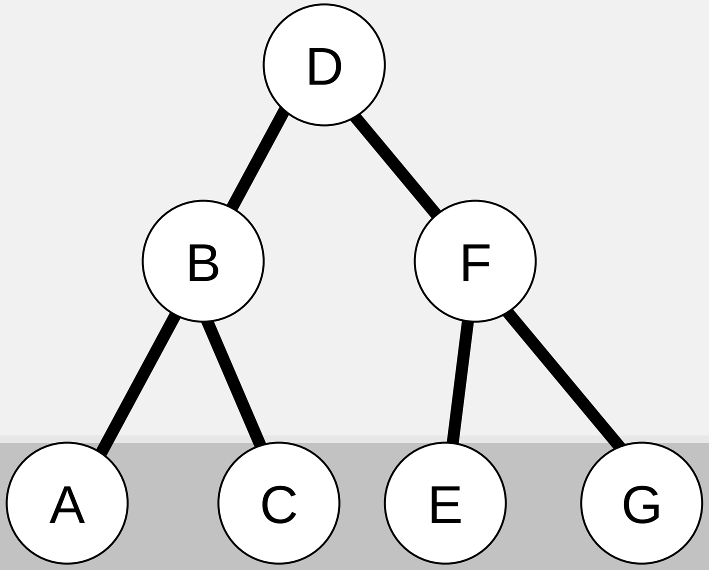
Create an optimal binary search tree

DATA	Probability
A	0.25
B	0.22
C	0.20
D	0.18
E	0.08
F	0.05
G	0.02

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability	TIME
A	0.25	3
B	0.22	2
C	0.20	3
D	0.18	1
E	0.08	3
F	0.05	2
G	0.02	3

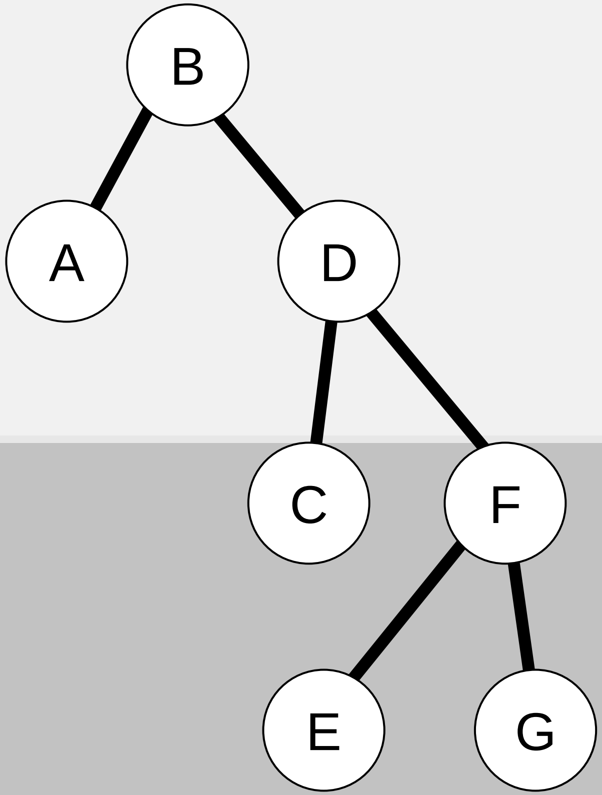


Average-time for searching **2.37**

Optimal binary search tree

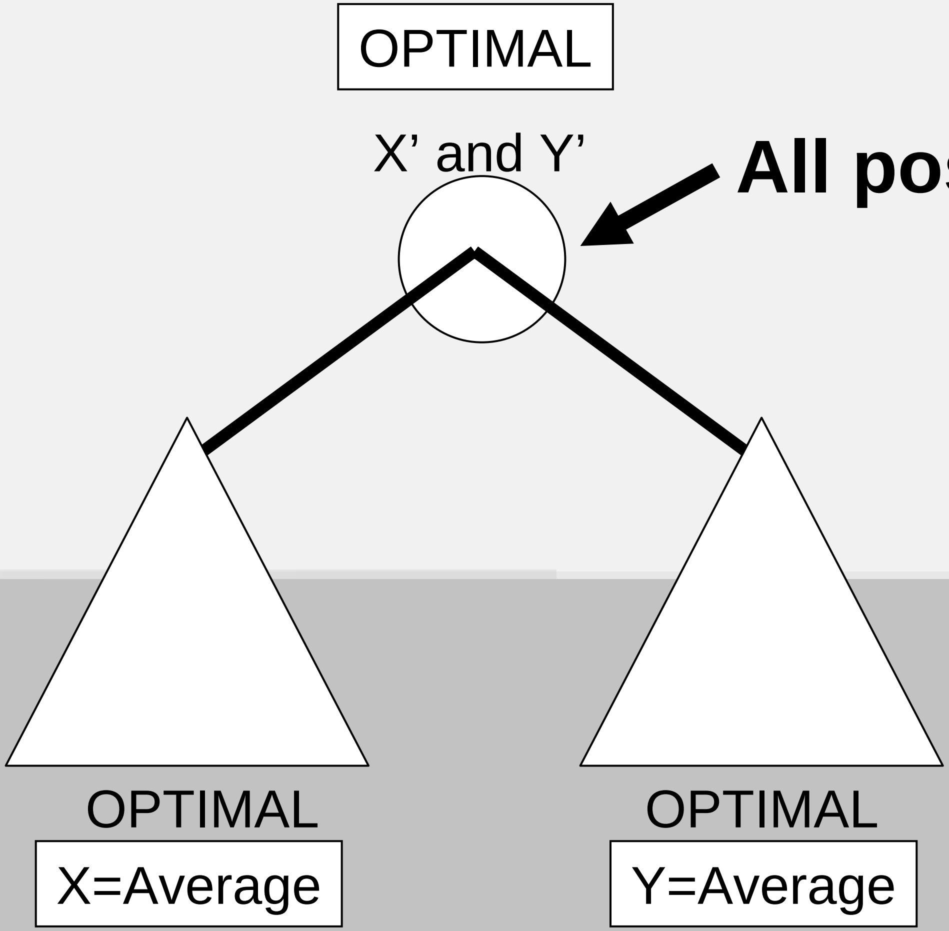
Create an optimal binary search tree

DATA	Probability	TIME
A	0.25	2
B	0.22	1
C	0.20	3
D	0.18	2
E	0.08	4
F	0.05	3
G	0.02	4



Average-time for searching **1.98**

Optimal binary search tree

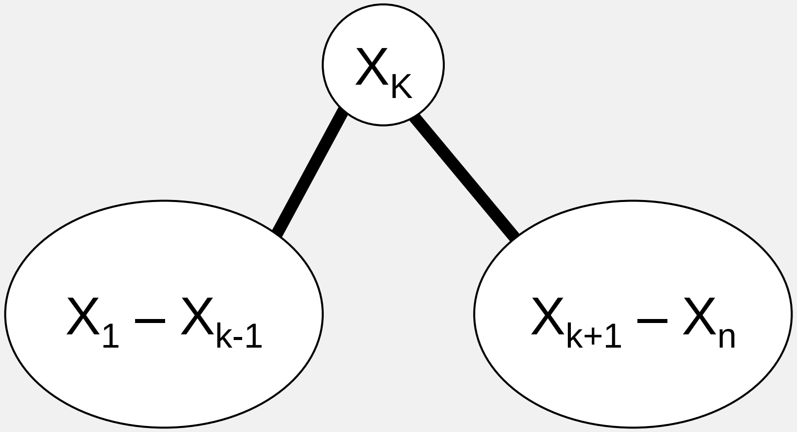


All possible ways

Optimal binary search tree

Create an optimal binary search tree

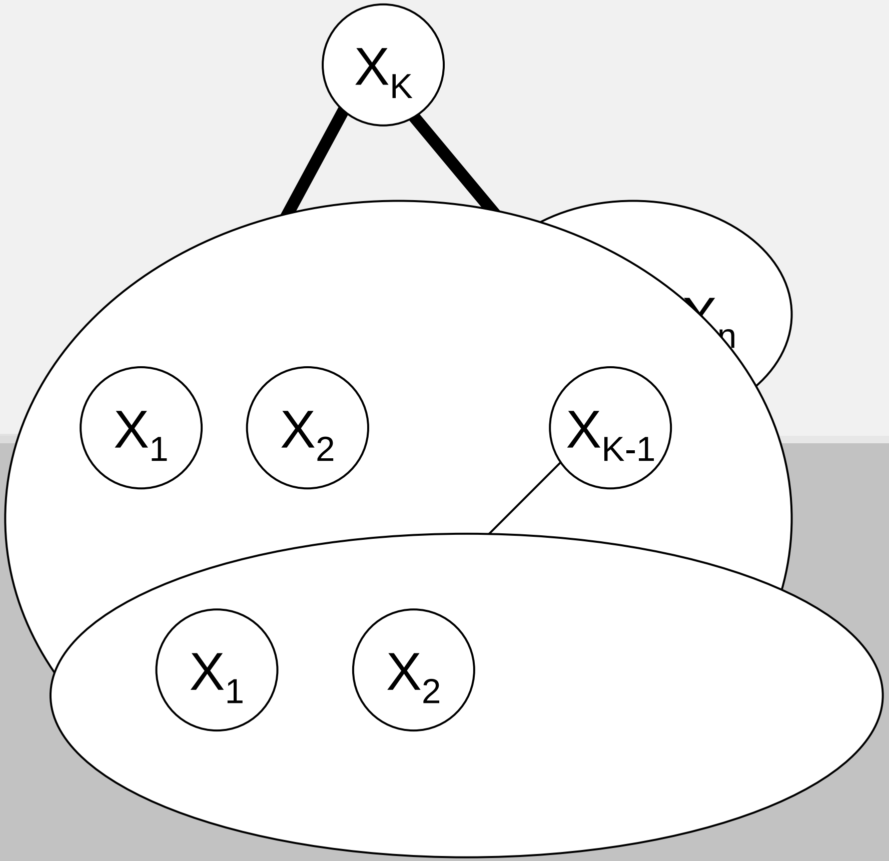
DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7



Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7



Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7

DICTIONARY

1								
2								
n								
	1	2						n

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7

DICTIONARY

	1	2						n
1								
2								
n								

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7

DICTIONARY

	1	2						n
1								
2								
n								

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7

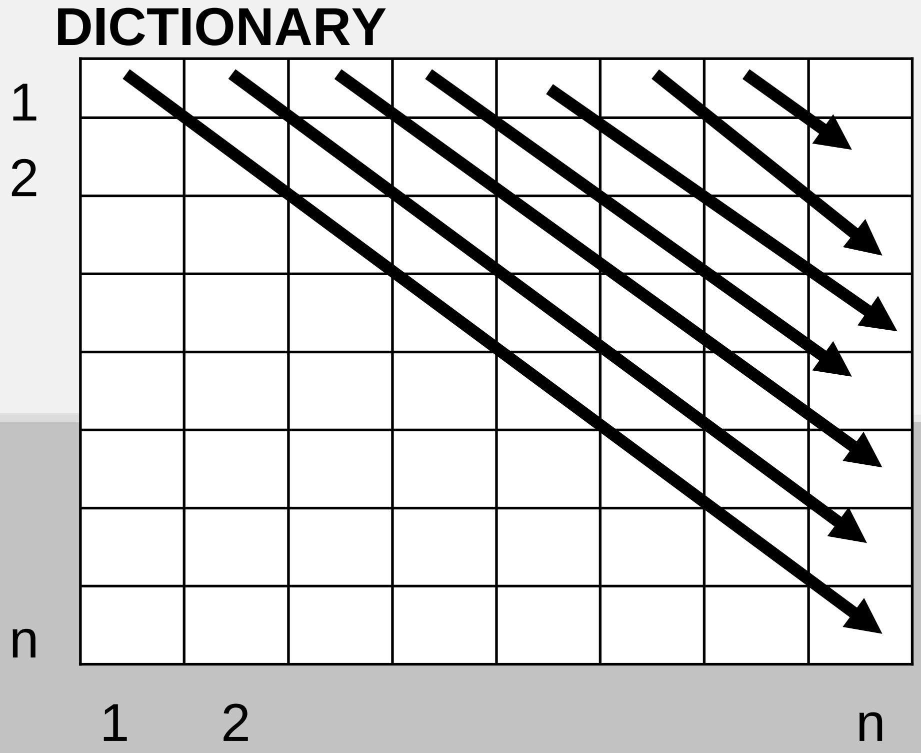
DICTIONARY

1								
2								
n								
	1	2						n

Optimal binary search tree

Create an optimal binary search tree

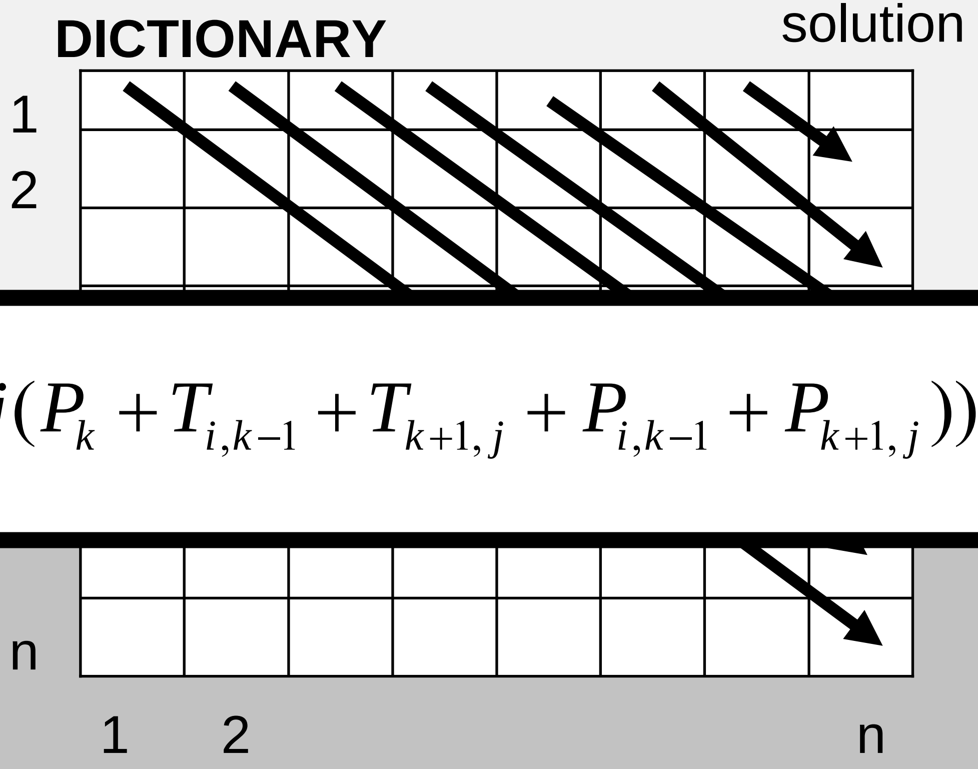
DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4
X_5	P_5
X_6	P_6
x_7	p_7



Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
x_7	p_7

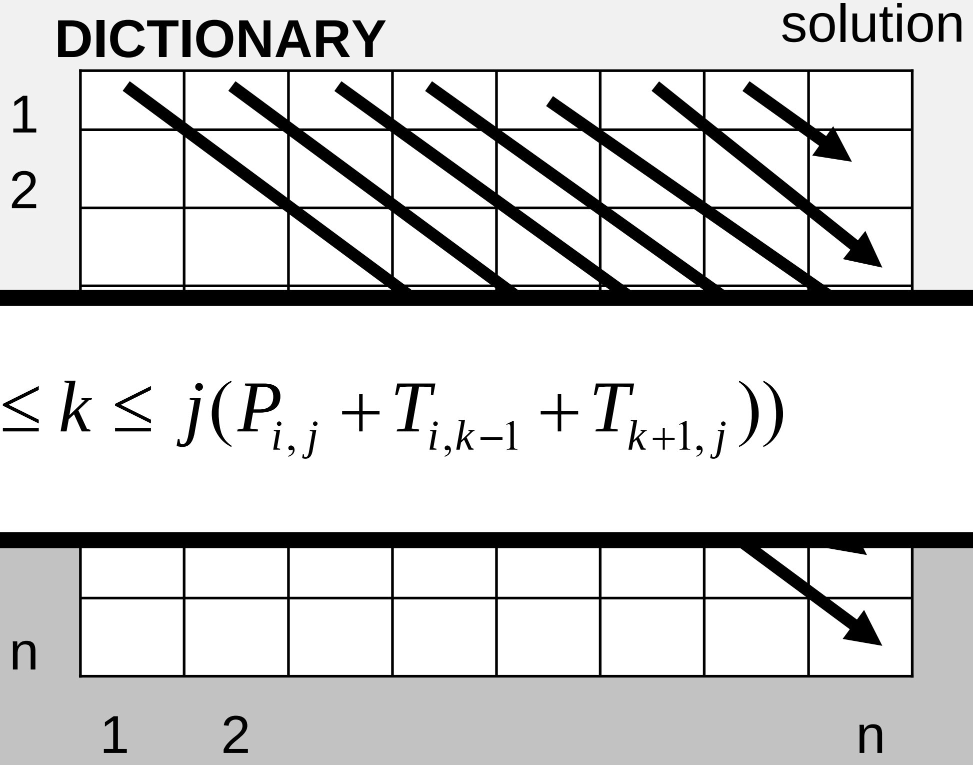


$$T_{i,j}(\min i \le k \le j(P_k + T_{i,k-1} + T_{k+1,j} + P_{i,k-1} + P_{k+1,j}))$$

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
x_7	p_7



$$T_{i,j}(\min i \le k \le j(P_{i,j} + T_{i,k-1} + T_{k+1,j}))$$

Optimal binary search tree

Create an optimal binary search tree

DATA	Probability

Min

$T_{i+1,j}$

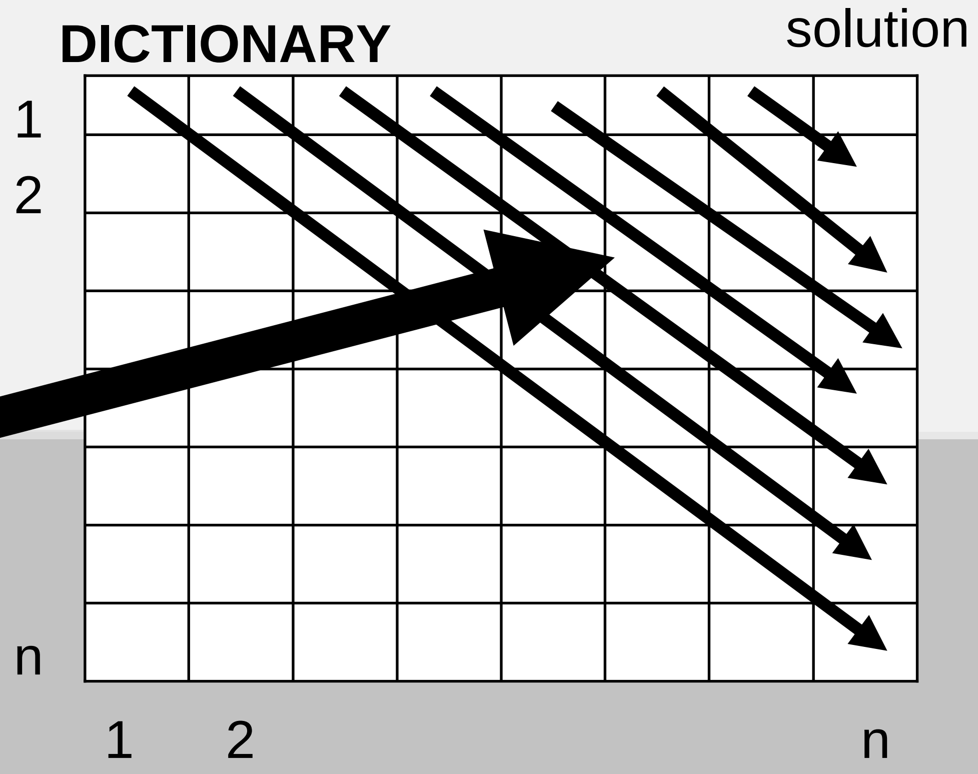
$T_{i,i}+T_{i+2,j}$

$T_{i,i+1}+T_{i+3,j}$

$T_{i,i+2}+T_{i+4,j}$

…

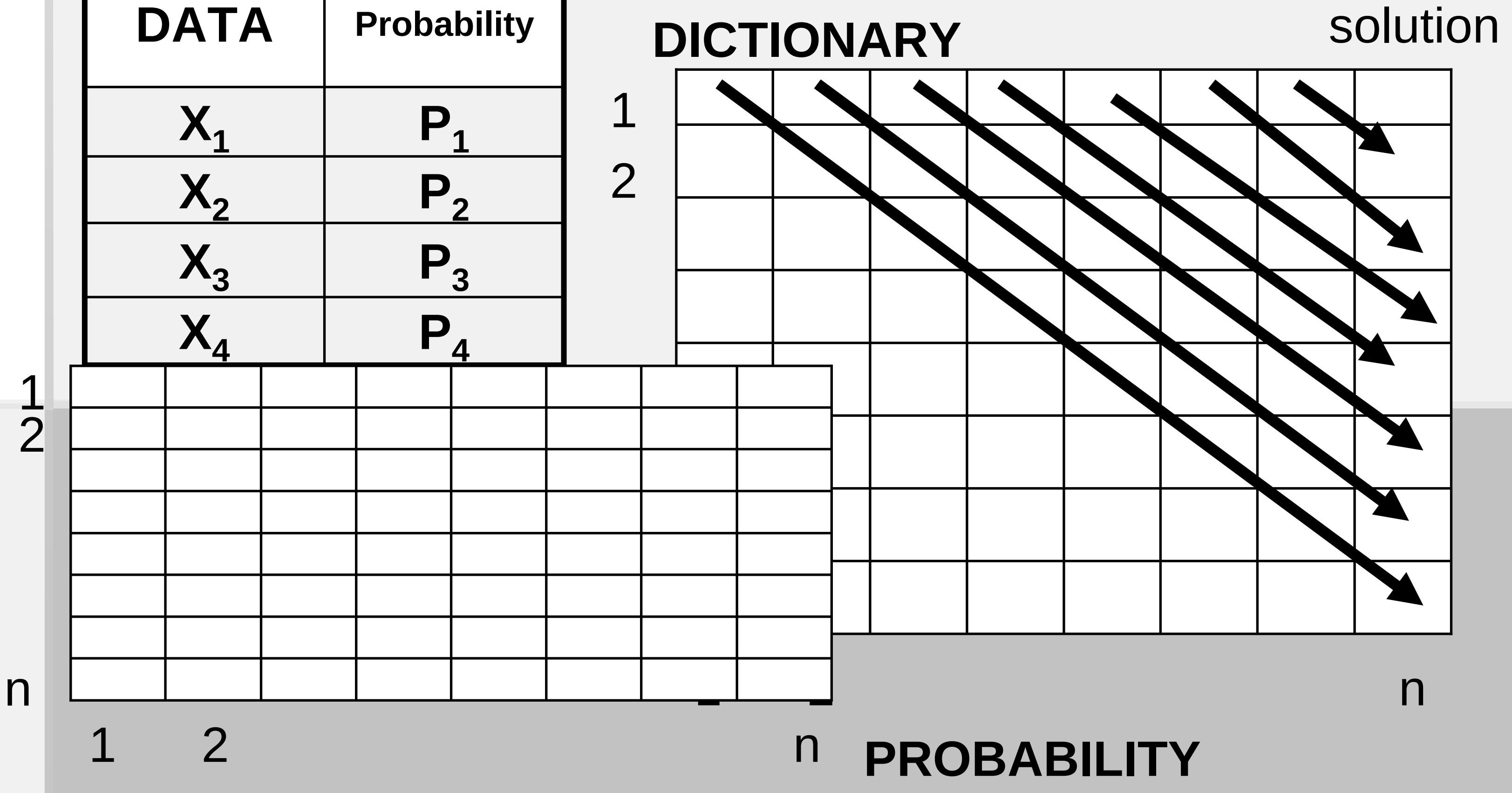
$T_{i,j-1}$



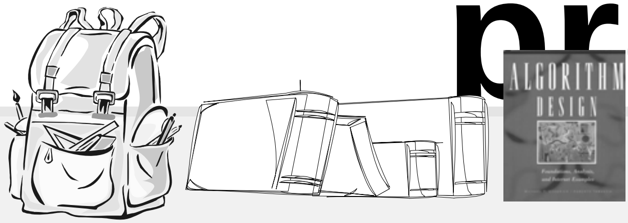
Optimal binary search tree

Create an optimal binary search tree

DATA	Probability
X_1	P_1
X_2	P_2
X_3	P_3
X_4	P_4



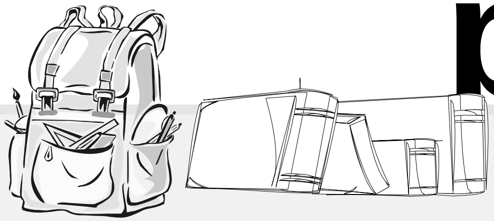
0/1 Knapsack problem



- S_k: Set of items numbered 1 to k.
- Define $B[k,w]$ = best selection from S_k with weight exactly equal to w
- Best subset of S_k with weight exactly w is either:
    - - the best subset of S_{k-1} weight w
    - - the best subset of S_{k-1} weight w-w_k plus item k

$$B[k,w] = \begin{cases} B[k-1,w] & \text{if } w_k > w \\ \max\{B[k-1,w],\, B[k-1,w-w_k]+b_k\} & \text{otherwise} \end{cases}$$

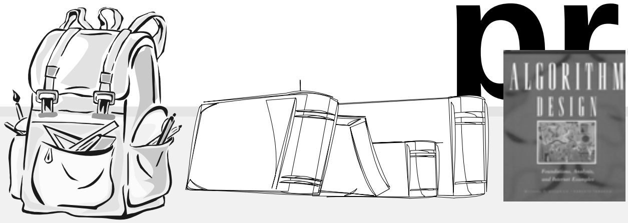
0/1 Knapsack problem



- **Since *B*[*k,w*] is defined in terms of *B*[*k*-1,*], we can reuse the same array**

```
Algorithm 0-1Knapsack(S, W):
    Input: set S of items with benefit $b_i$
    and weight $w_i$; max. weight W
    Output: value of best subset with weight $\leq W$
    for $w \leftarrow 0$ to $W$ do
        $B[0,w] \leftarrow 0$
    for $k \leftarrow 1$ to $n$ do
        for $w \leftarrow W$ down to $w_k$ do
            $B[k,w] \leftarrow \max(B[k-1,w],$
                                  $B[k-1,w-w_k]+b_k)$
```

0/1 Knapsack problem



- **Since *B*[*k,w*] is defined in terms of *B*[*k*-1,*], we can reuse the same array**

Algorithm *0-1Knapsack*(*S, W*):

```
Input: set $S$ of items with benefit $b_i$
and weight $w_i$; max. weight $W$
Output: value of best subset with weight $\leq W$
for $w \leftarrow 0$ to $W$ do
    $B[0,w] \leftarrow 0$
for $k \leftarrow 1$ to $n$ do
    for $w \leftarrow W$ downto $w_k$ do
        $B[k,w] \leftarrow \max(B[k-1,w],$
                          $B[k-1,w-w_k]+b_k)$
```

Running time: O(*nW*).

All shortest paths

Floyd-Warshall Algorithm

Given a directed weighted graph G = (V,E),
find all shortest paths between any two vertices in G.

- If we already know the all shortest paths whose intermediate vertices belong to the set {1,…,*k*-1}, how can we find all shortest paths with intermediate vertices {1,…,*k*}?
- Consider the shortest path *p* between (*i, j*), whose intermediate vertices belong to {1,…*k*}
- If *k* is not an intermediate vertex in *p*, then *p* is the path found in the previous iteration.
- If *k* is in *p*, then we can write *p* as *i*~> *k* ~> *j,* where the intermediate vertices in *i*~> *k* and *k*~> *j* belong to {1,…,*k*-1}.

$$\boxed{d_{i,j}(k) = \min(d_{i,j}\ (k\text{-}1),\ d_{i,k}(k\text{-}1) + d_{k,j}(k\text{-}1))}$$

All shortest paths

Floyd-Warshall Algorithm

Given a directed weighted graph G = (V,E),
find all shortest paths between any two vertices in G.

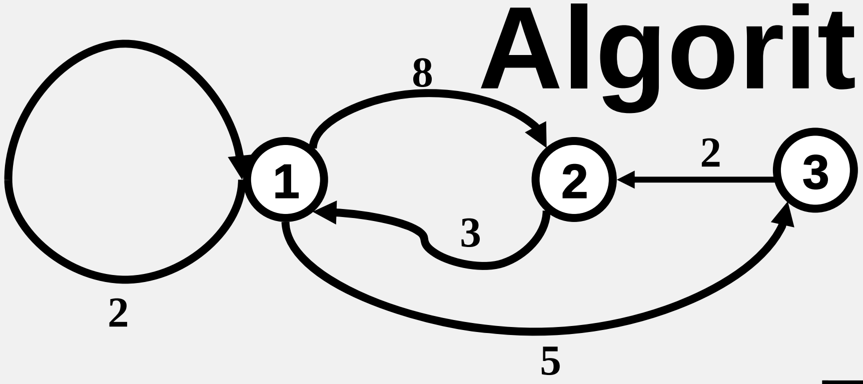
The algorithm:

```
–Initialize: $D_{(0)} = W$
–For k = 1…|V|
    – Fo
        or j = 1…|V|
            »$d_{i,j}(k) = \min(d_{i,j}(k-1), d_{i,k}(k-1)+d_{k,j}(k-1))$
```

Time complexity = $O(|V|^3)$

All shortest paths

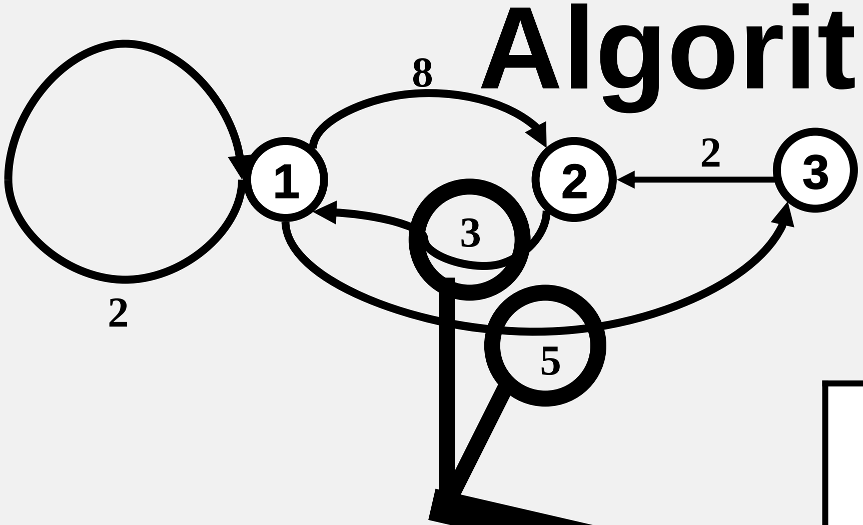
Floyd-Warshall Algorithm



D_0	1	2	3
1	**0**	**8**	**5**
2	**3**	**0**	∞
3	∞	**2**	**0**

All shortest paths

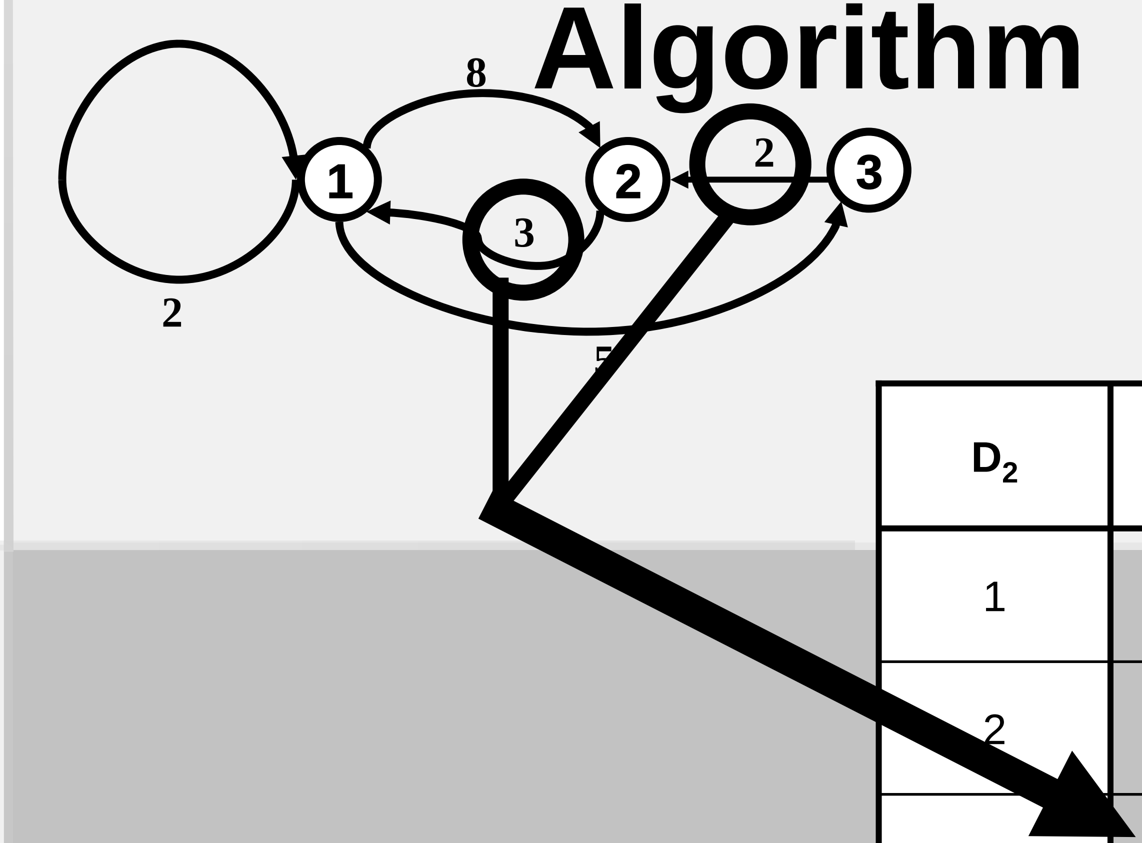
Floyd-Warshall Algorithm



D_1	1	2	3
1	**0**	**8**	**5**
2	**3**	**0**	**8**
3	∞	**2**	**0**

All shortest paths

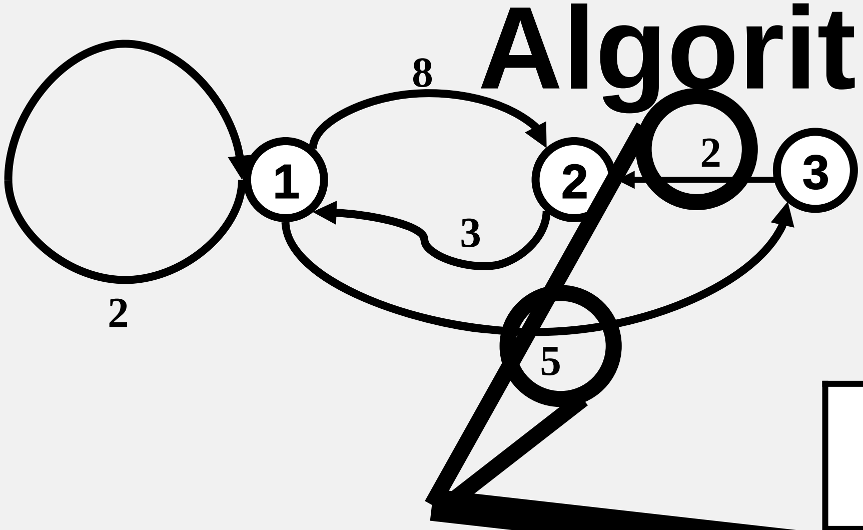
Floyd-Warshall Algorithm



D_2	1	2	3
1	**0**	**8**	**5**
2	**3**	**0**	**8**
3	**5**	**2**	**0**

All shortest paths

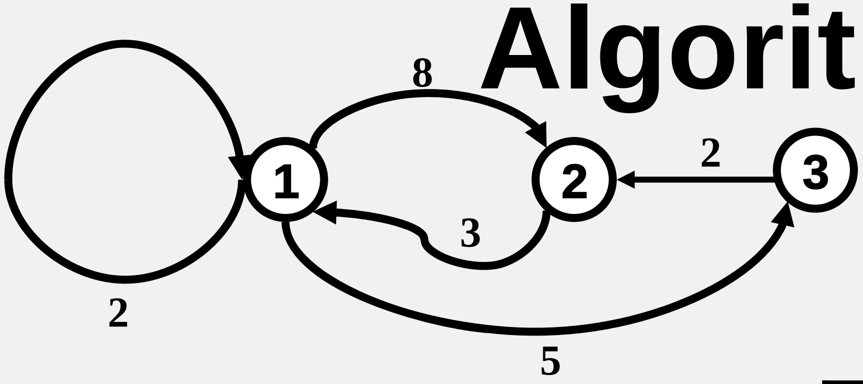
Floyd-Warshall Algorithm



D$_3$	1	2	3
1	**0**	**7**	**5**
2	**3**	**0**	**8**
3	**5**	**2**	**0**

All shortest paths

Floyd-Warshall Algorithm



D_3	1	2	3
1	**0**	**7**	**5**
2	**3**	**0**	**8**
3	**5**	**2**	**0**

Longest common subsequence

Given two sequences

- X = ABCB
- Y = BDCAB

Longest common subsequence

Find the longest common subsequence of two sequences

- **X = ABCB**
- **Y = BDCAB**

Brute force algorithm would compare each subsequence of X with the symbols in Y.

If |X| = m, |Y| = n, then there are 2^m subsequences of x; we must compare each with Y (n comparisons).

So the running time of the brute-force algorithm is $O(n\ 2^m)$.

Notice that the LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.

Longest common subsequence

Find the longest common subsequence of two sequences

- **X = ABCB**
- **Y = BDCAB**

Define X_i, Y_j to be the prefixes of X and Y of length i and j respectively
Define c[i,j] to be the length of LCS of X_i and Y_j
Then the length of LCS of X and Y will be c[m,n]

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**				
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	C	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**			
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	+1	
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	+1				
3	**C**	**0**					
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	+1
3	C	0					
4	B	0					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0					
4	B	0					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	+1		
4	**B**	**0**					

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	+1				

Longest common subsequence

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	+1

Longest common subsequence

```
_    B    D    C    A    B
     |         |         |
A    B    _    C    _    B
```

TIME COMPLEXITY = O(mn)

i							
0							
1							
	B						
4	B	0	1	1	2	2	3

Natural language

Given a sentence (sequence of words)

John called Mary from Denver.

Find a grammar tree matched to *X*.

Given GRAMMAR

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$NP \rightarrow NP\ PP$

$VP \rightarrow VP\ PP$

$PP \rightarrow P\ NP$

John	NP
called	V
Mary	NP
from	P
Denver	NP

Natural language

John called Mary from Denver.

Given GRAMMAR

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

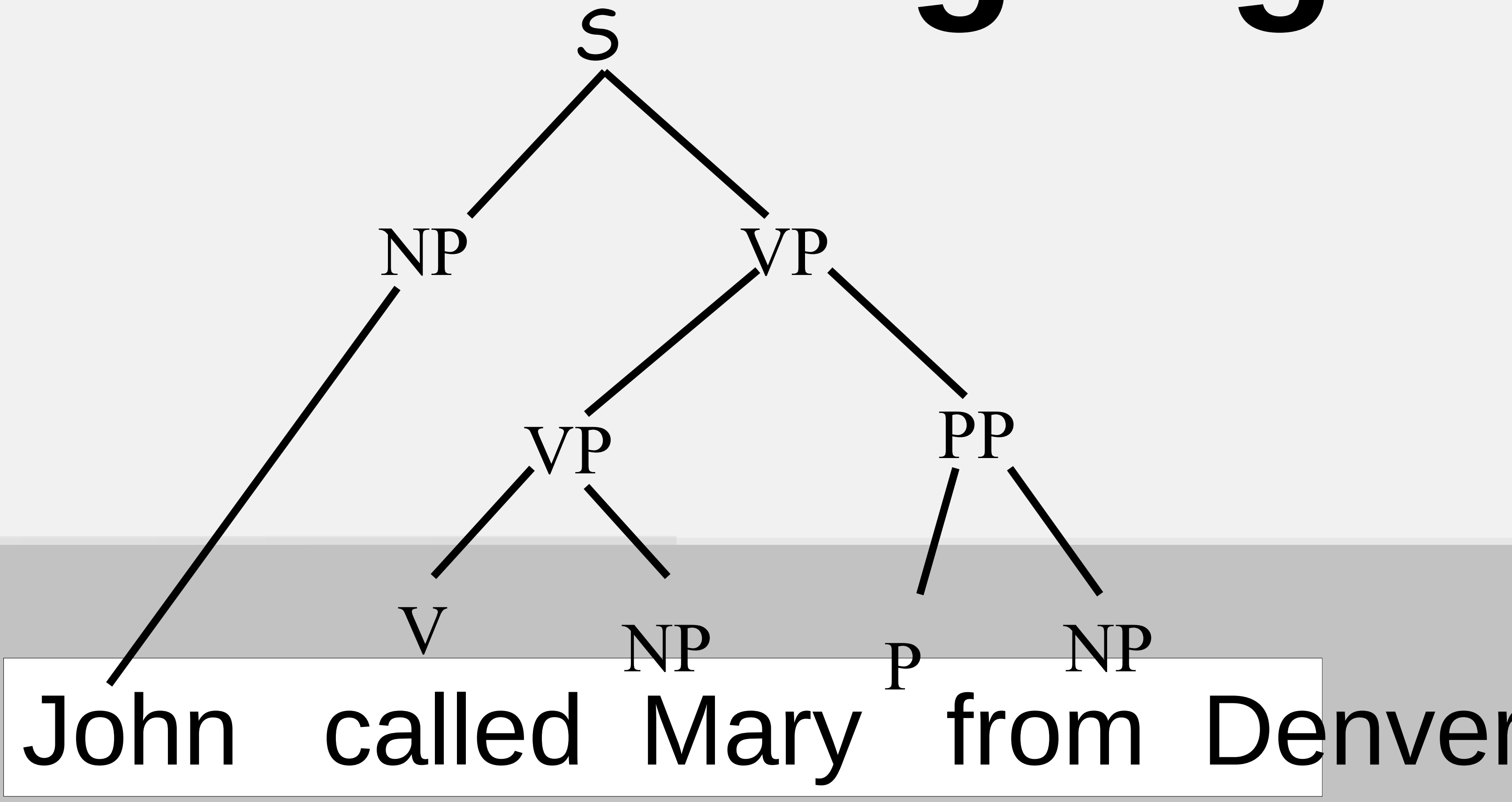
$NP \rightarrow NP\ PP$

$VP \rightarrow VP\ PP$

$PP \rightarrow P\ NP$

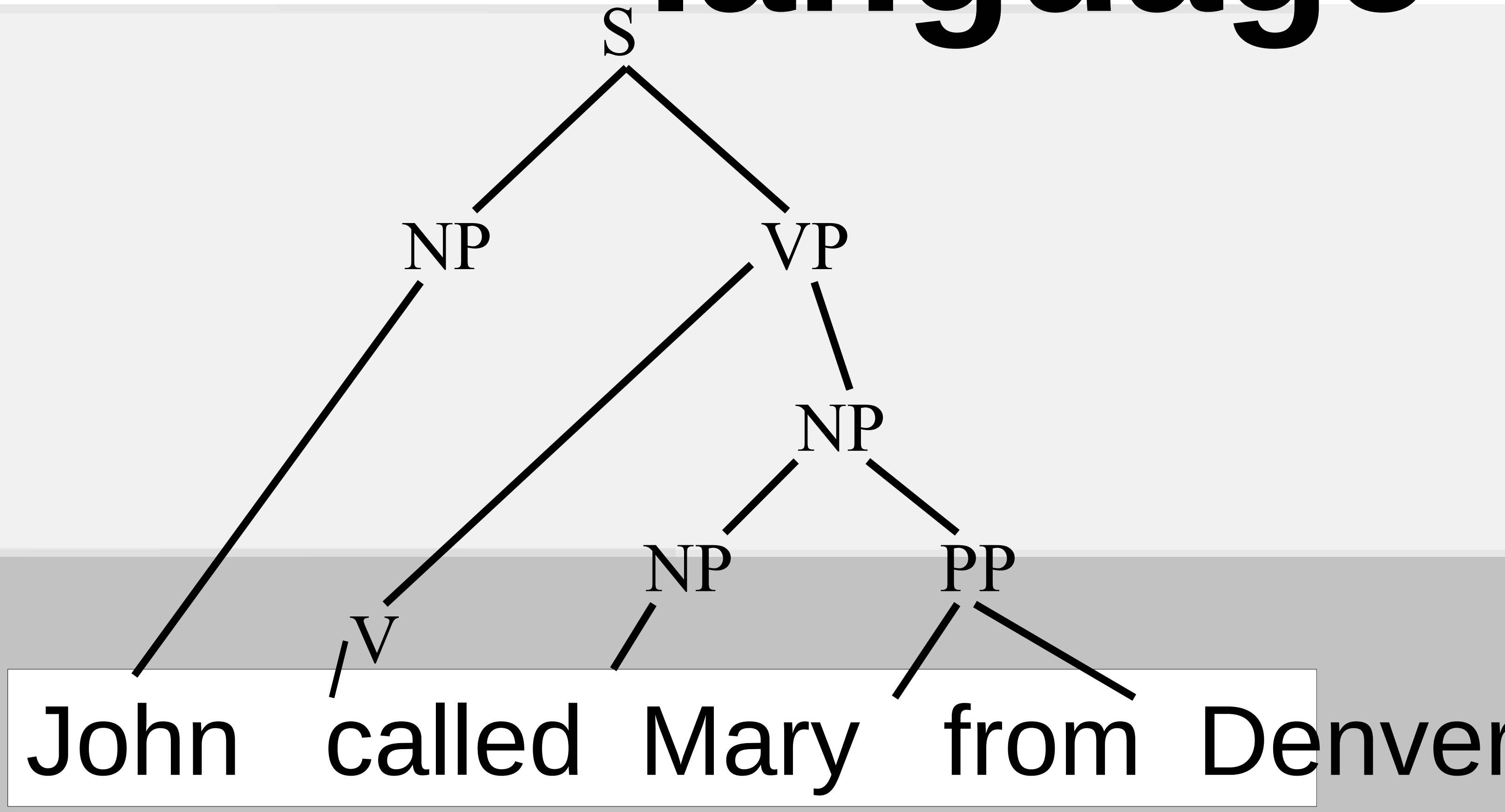
John	NP
called	V
Mary	NP
from	P
Denver	NP

Natural language



John called Mary from Denver

Natural language



Natural language

John	**called**	**Mary**	**from**	**Denver**

Natural language

				NP
			P	Denver
		NP	from	
	V	Mary		
NP	called			
John				

Natural language

				NP
			P	Denver
		NP	from	
X	V	Mary		
NP	called			
John				

Natural language

				NP
			P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

				NP
		X	P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

			PP	NP
		X	P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

			PP	NP
		X	P	Denver
S	VP	NP	from	
	V	Mary		
NP	called			
John				

Natural language

			PP	NP
	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

		NP	PP	NP
	X		P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

		NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

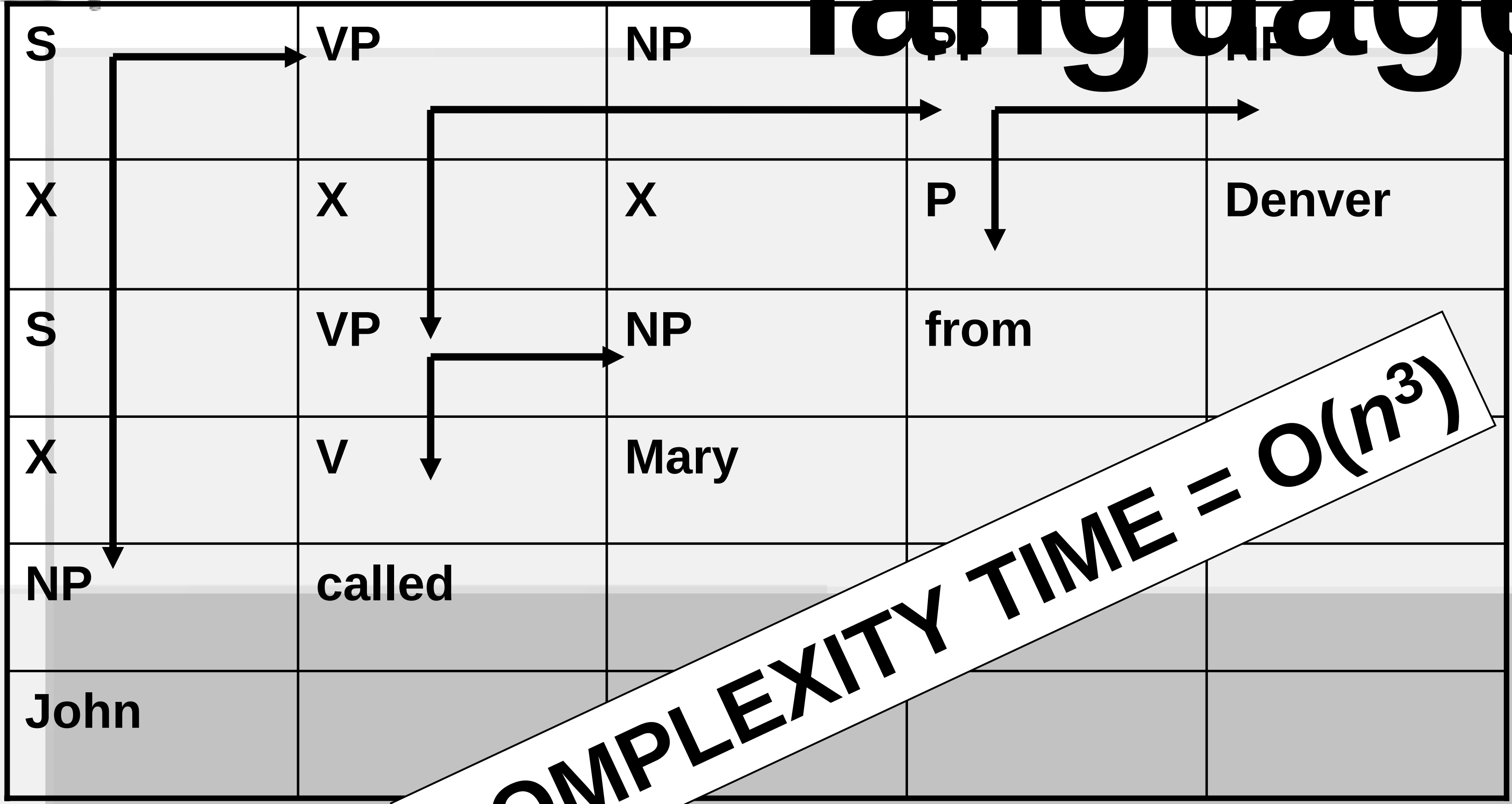
Natural language

	VP_1 VP_2	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Natural language

S	VP_1 VP_2	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

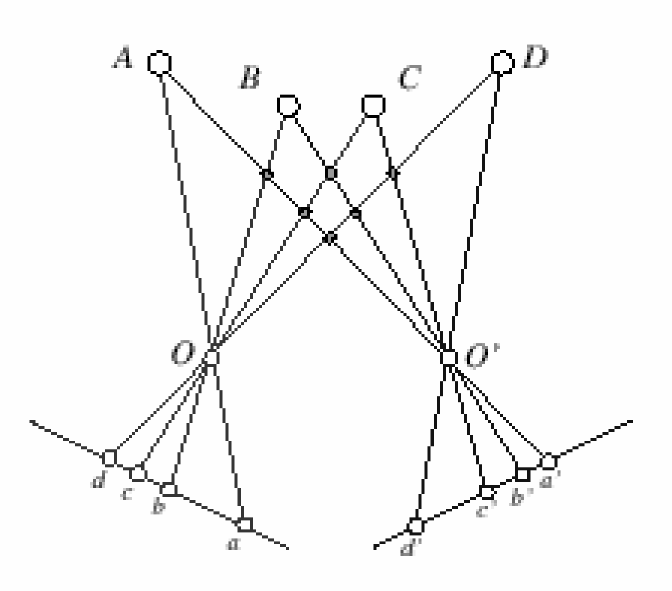
Natural language



S	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

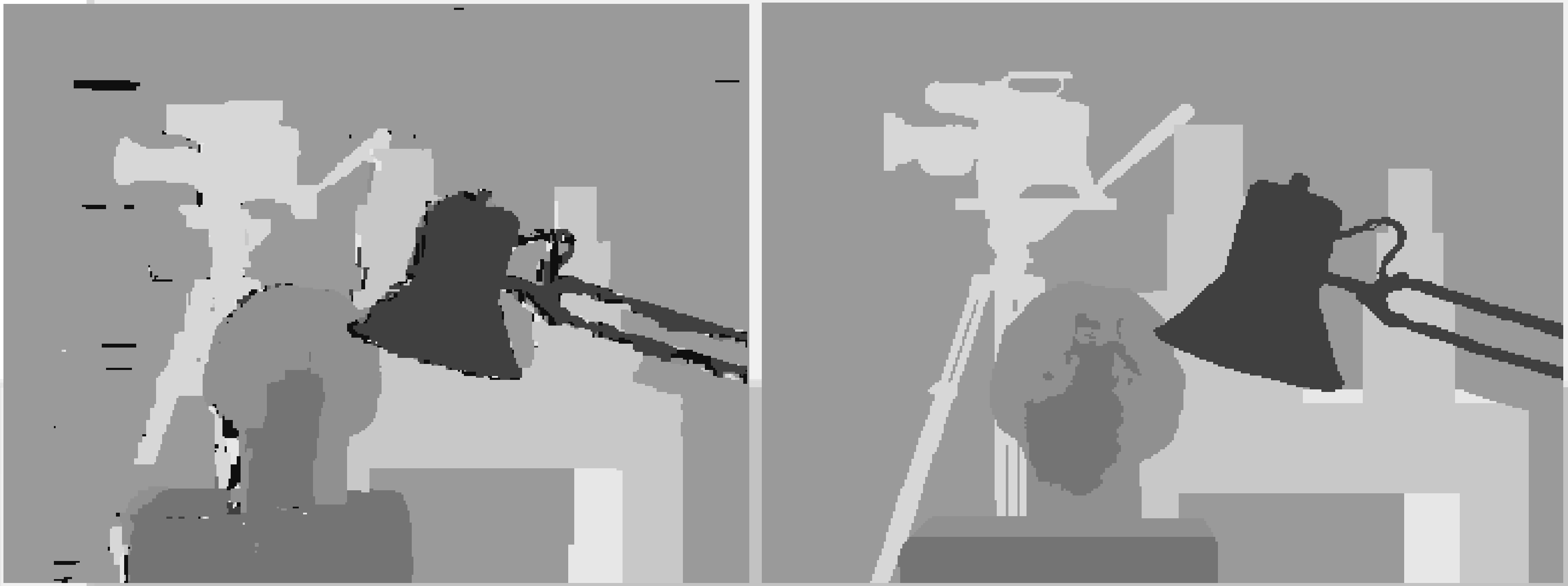
COMPLEXITY TIME = O(n^3)

Stereo vision

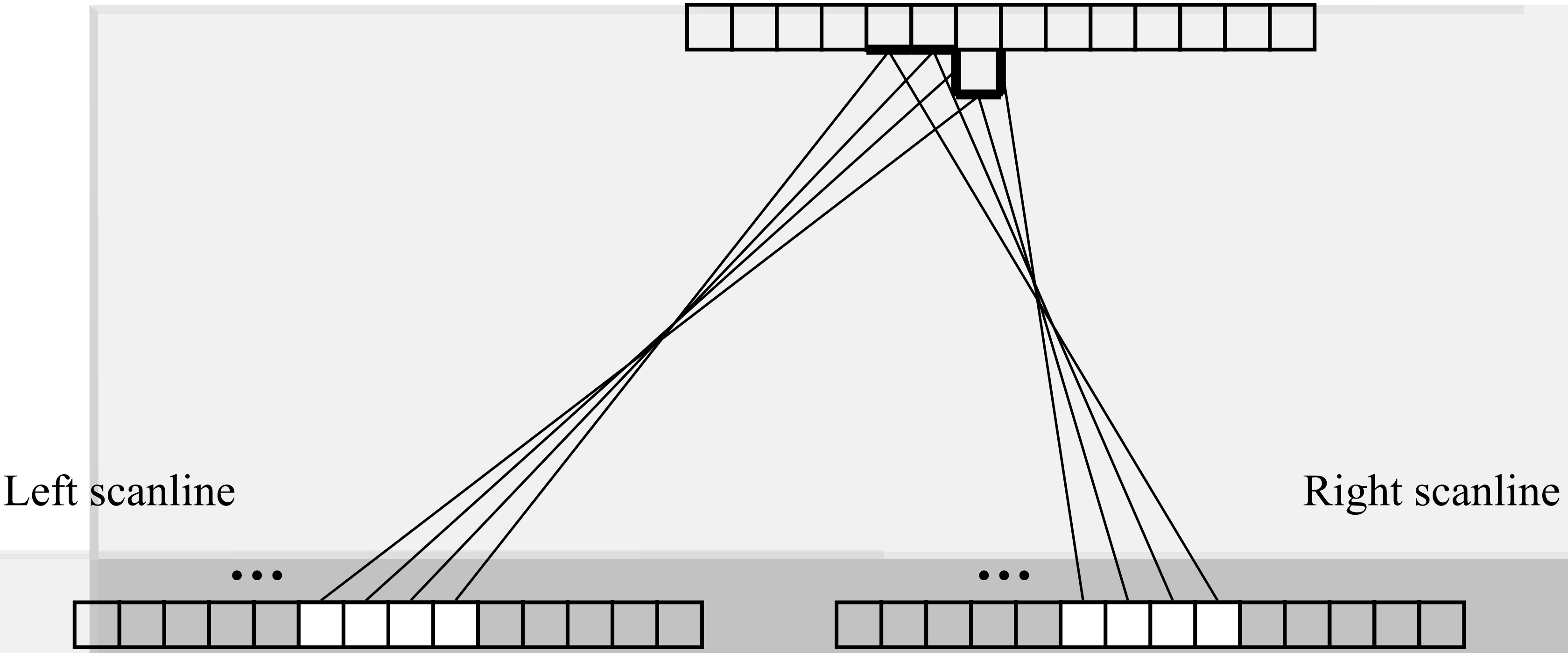


Ordering constraint…

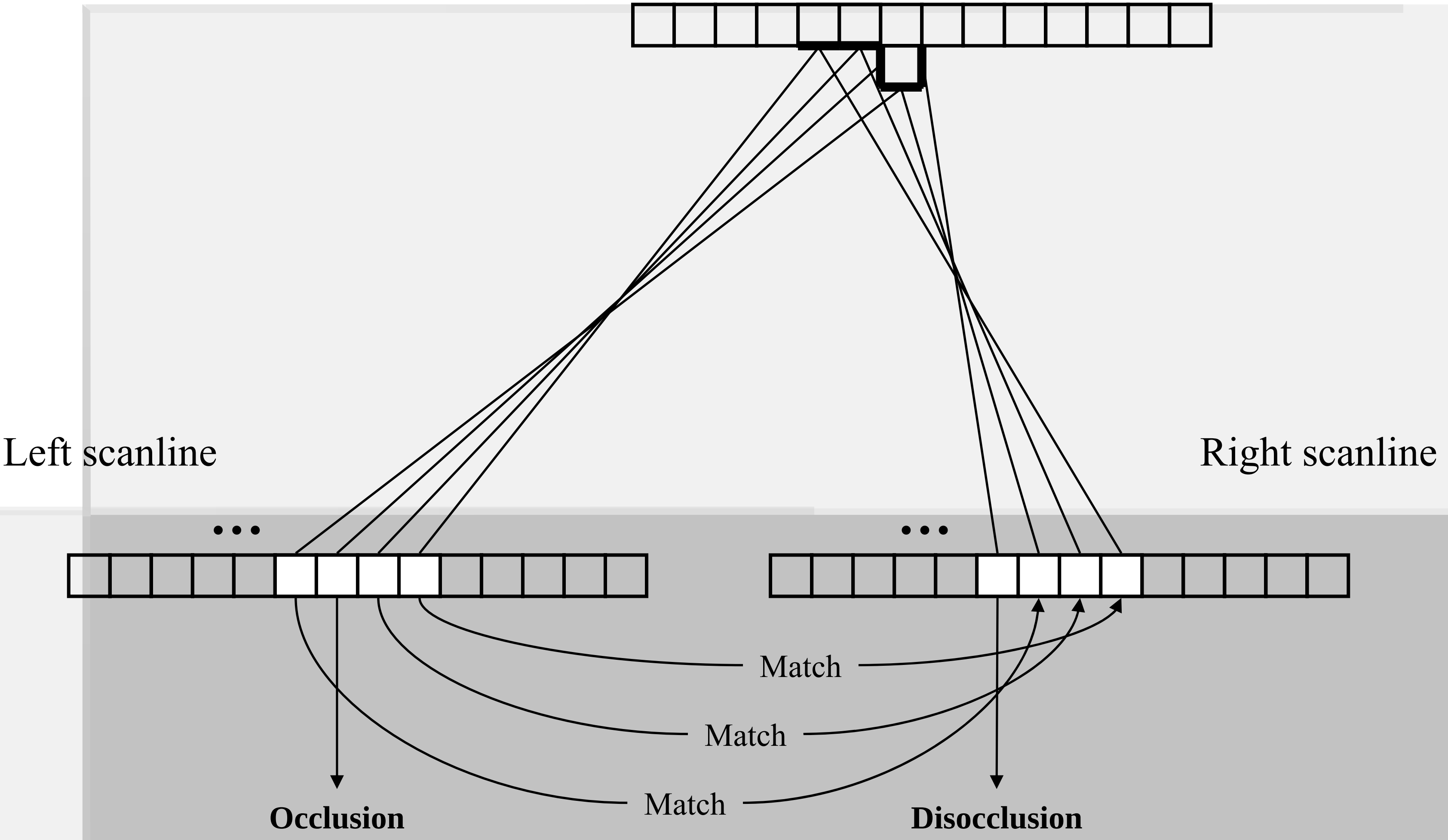
Stereo vision



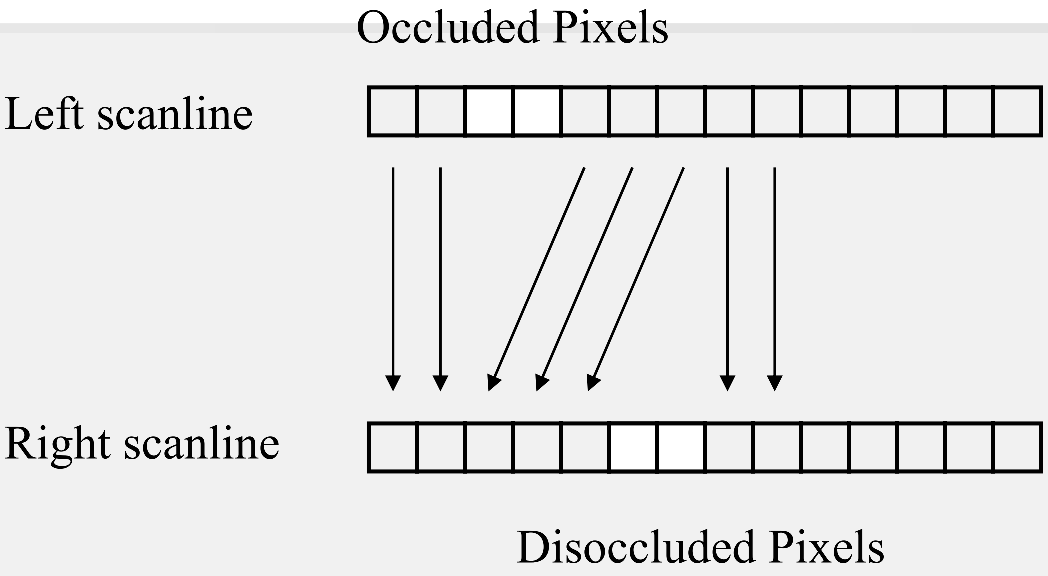
Stereo vision



Stereo vision



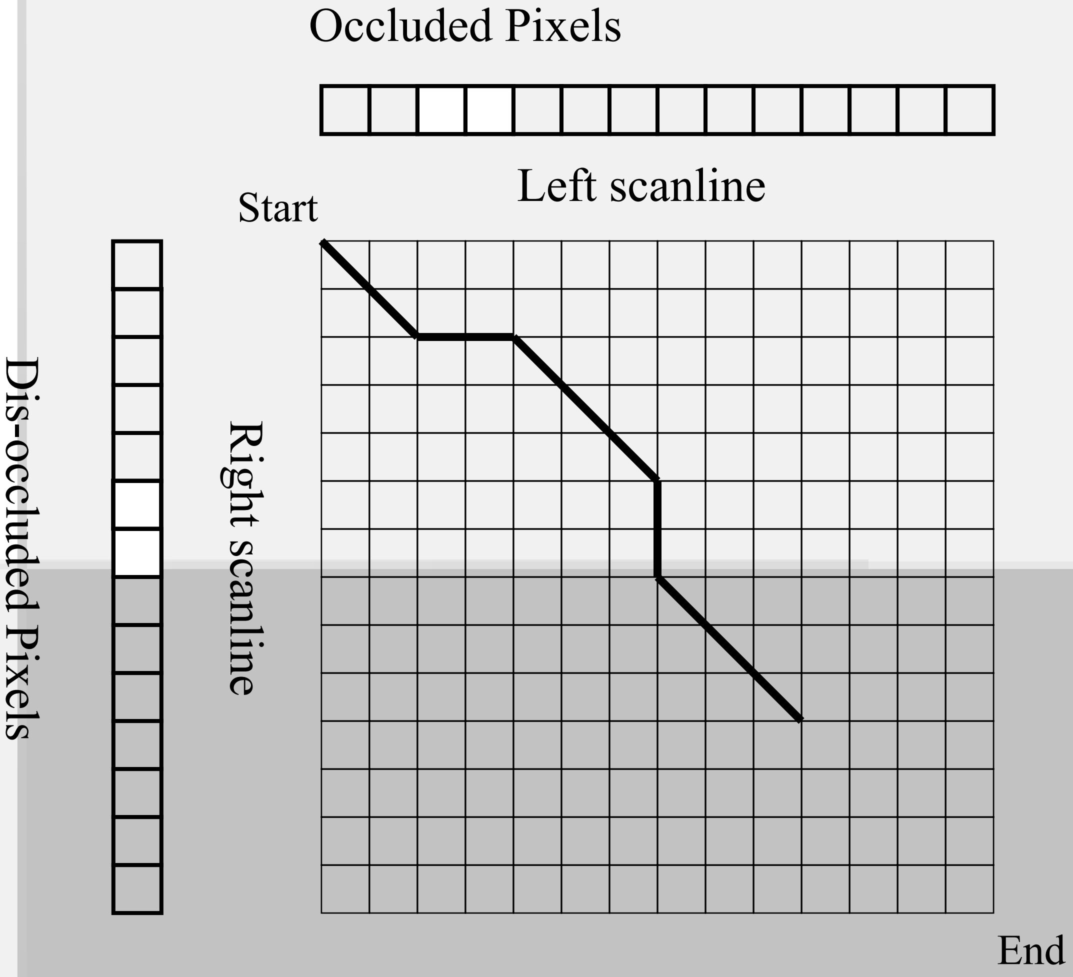
Stereo vision



Three cases:

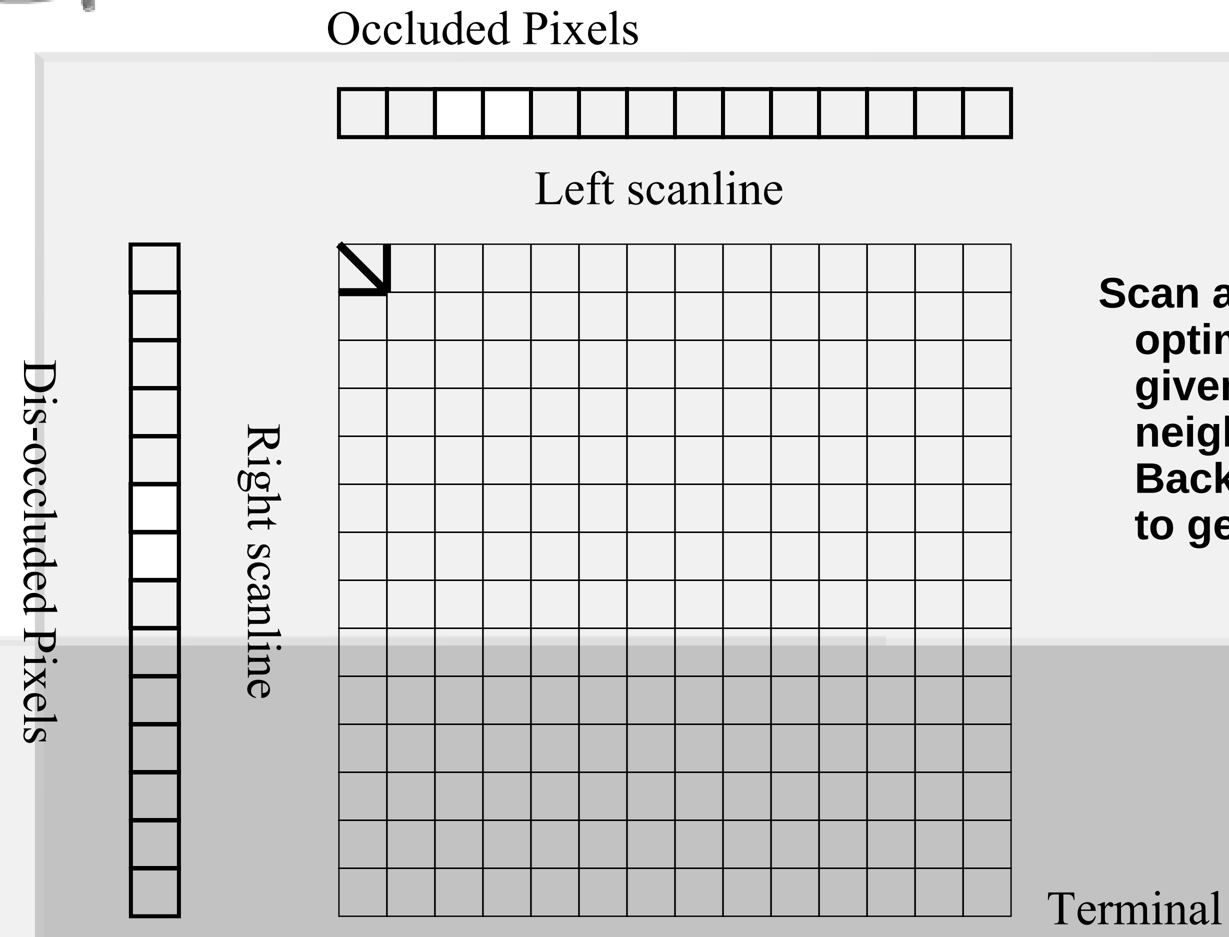
- Sequential – add cost of match (small if intensities agree)
- Occluded – add cost of no match (large cost)
- Disoccluded – add cost of no match (large cost)

Stereo vision



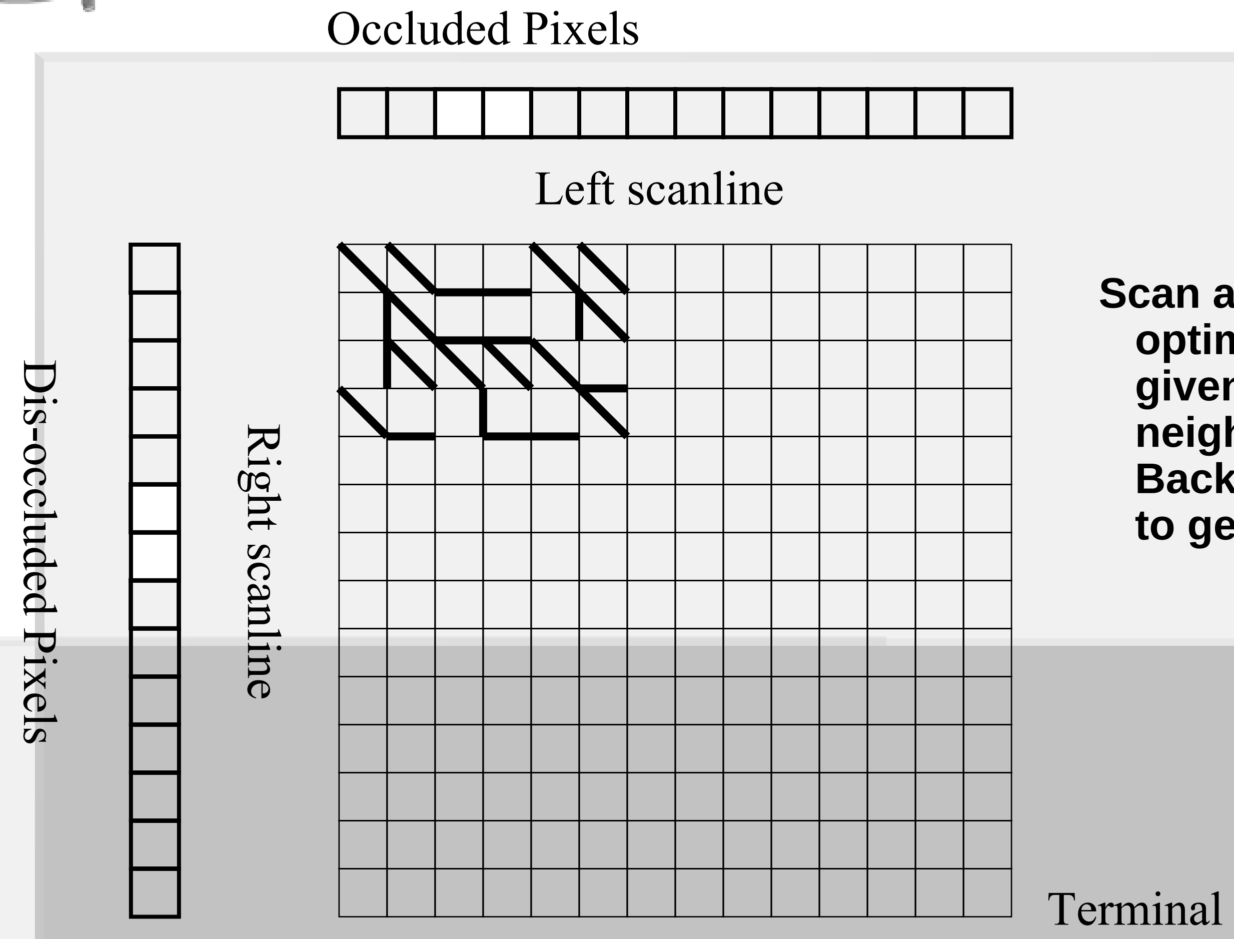
Dynamic programming yields the optimal path through grid. This is the best set of matches that satisfy the ordering constraint

Stereo vision



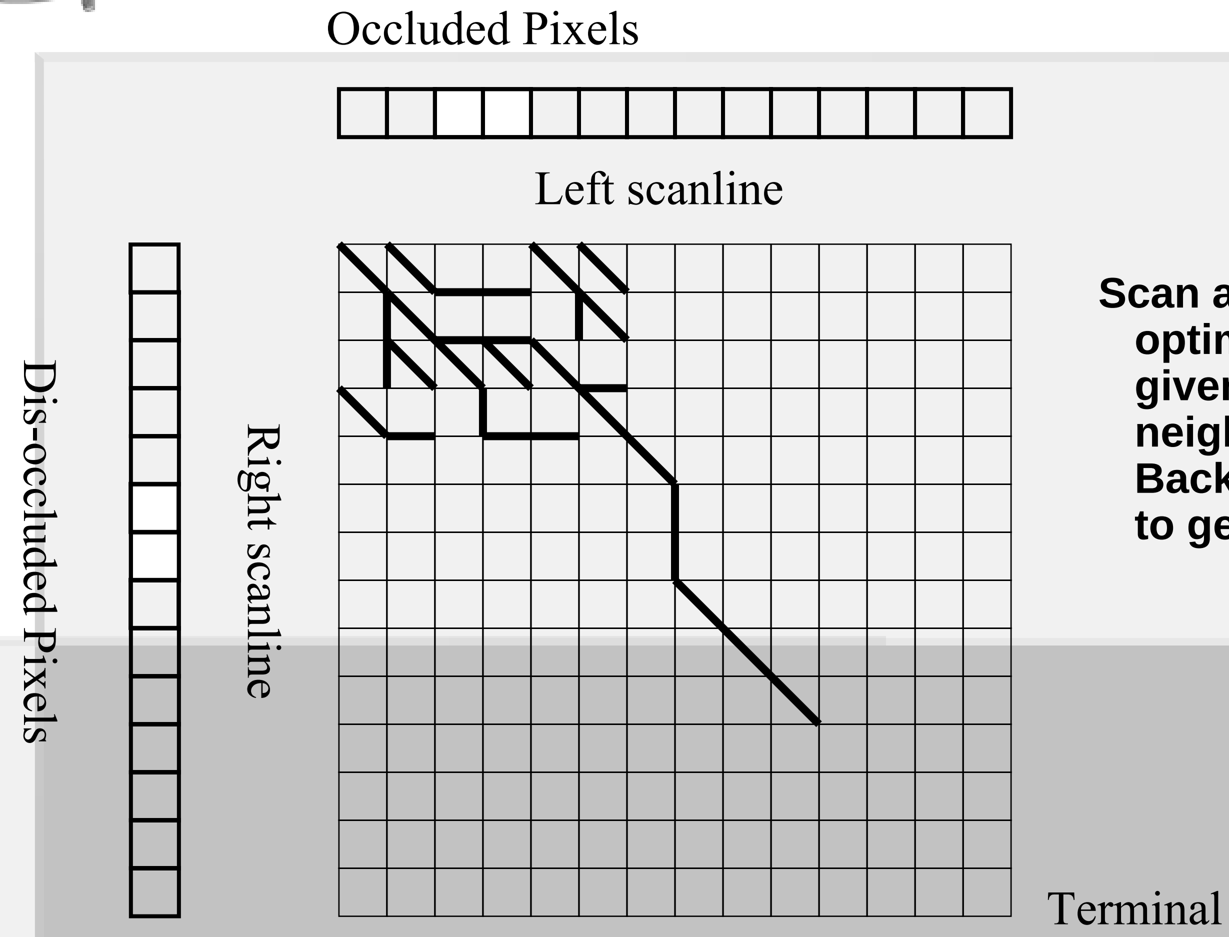
Scan across grid computing optimal cost for each node given its upper-left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo vision



Scan across grid computing optimal cost for each node given its upper-left neighbors. Backtrack from the terminal to get the optimal path.

Stereo vision

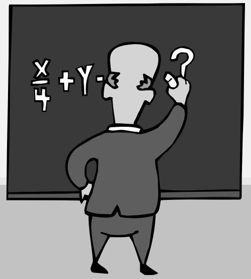


Scan across grid computing optimal cost for each node given its upper-left neighbors. Backtrack from the terminal to get the optimal path.

Problems

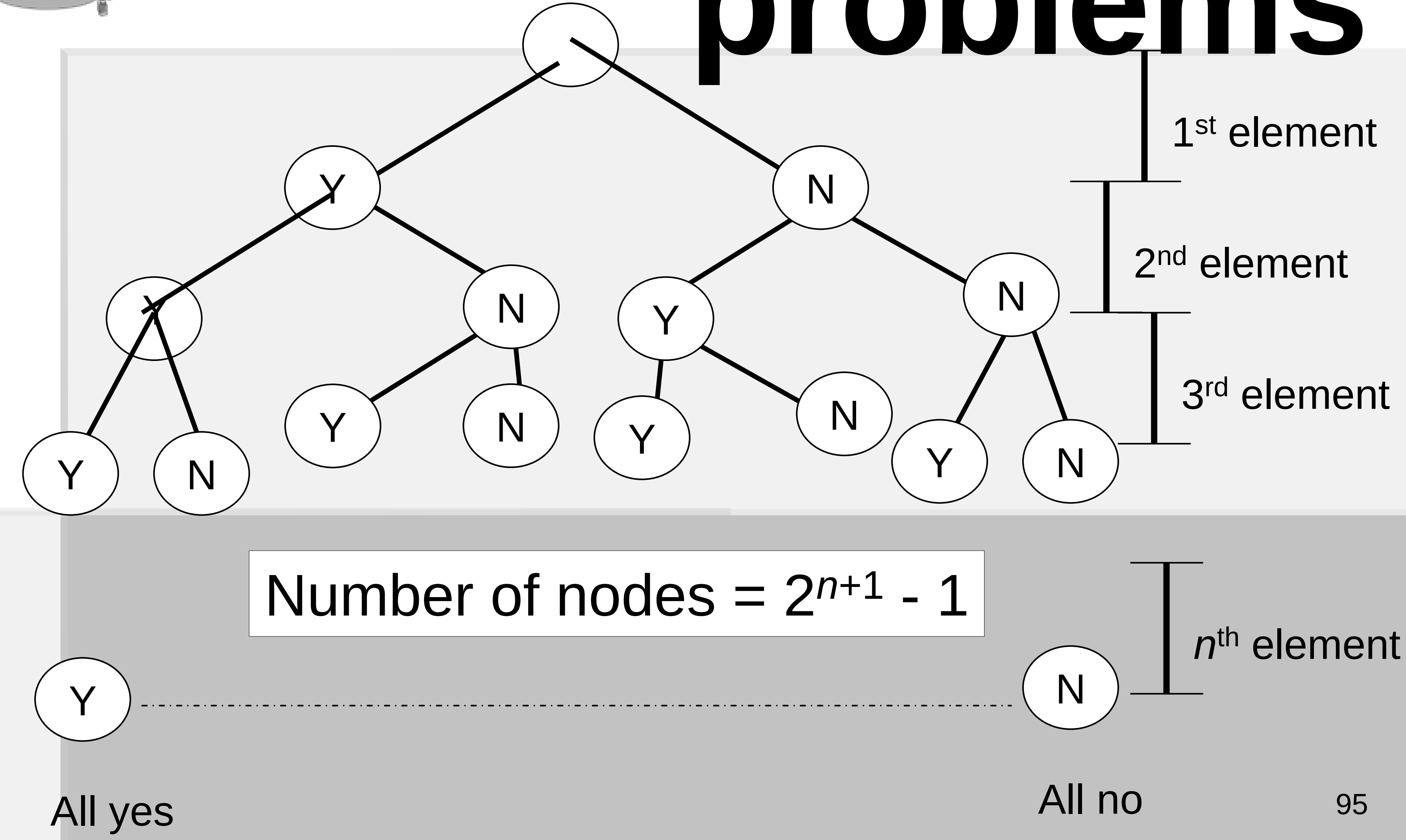
Three groups of complexity problems

- **Subset problems (2^n)**
- **Permutation problems (n!)**
- **Partition problems (n!)**



State-space graph / tree

Subset problems

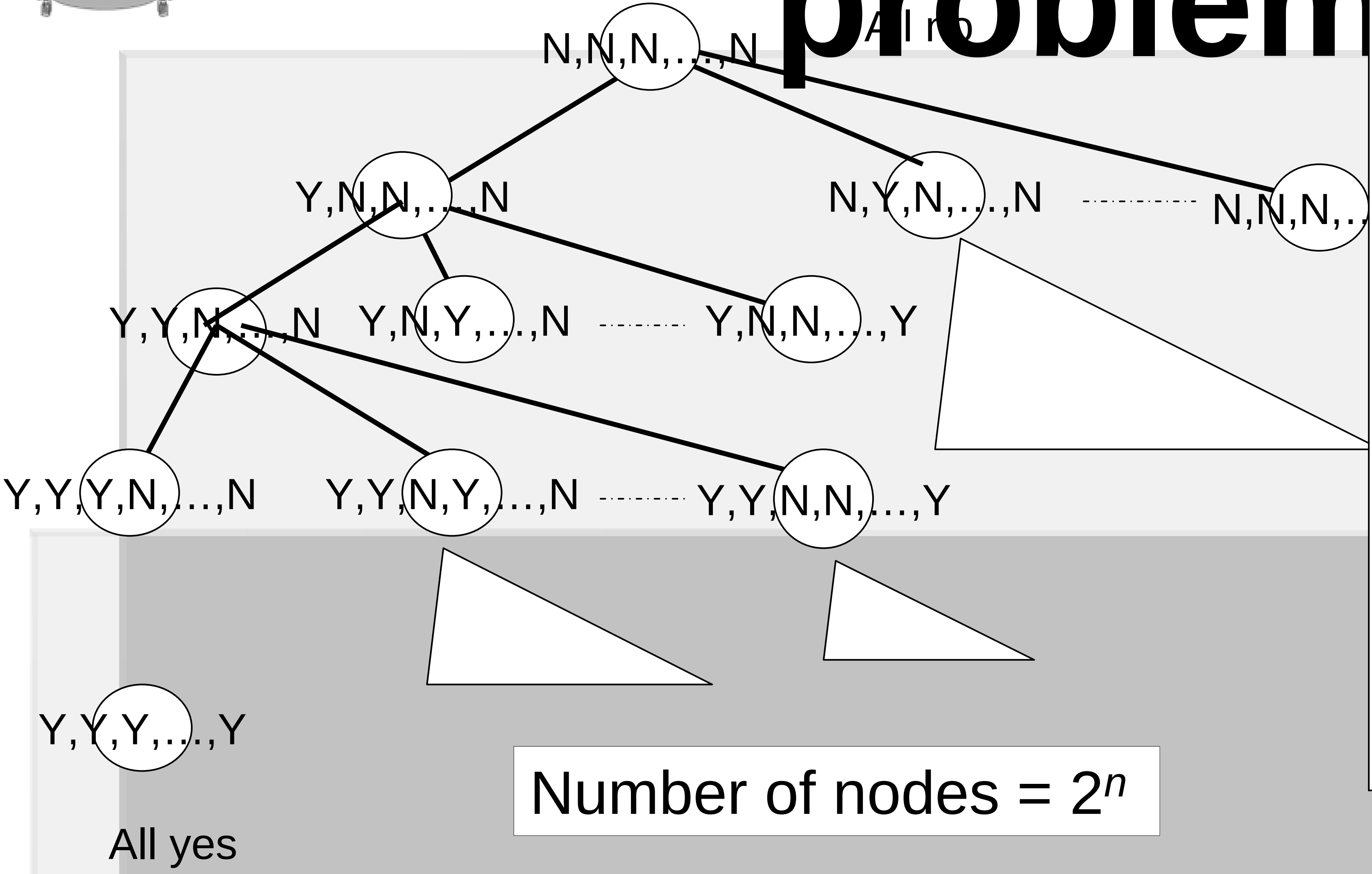


Number of nodes = $2^{n+1} - 1$

All yes

All no

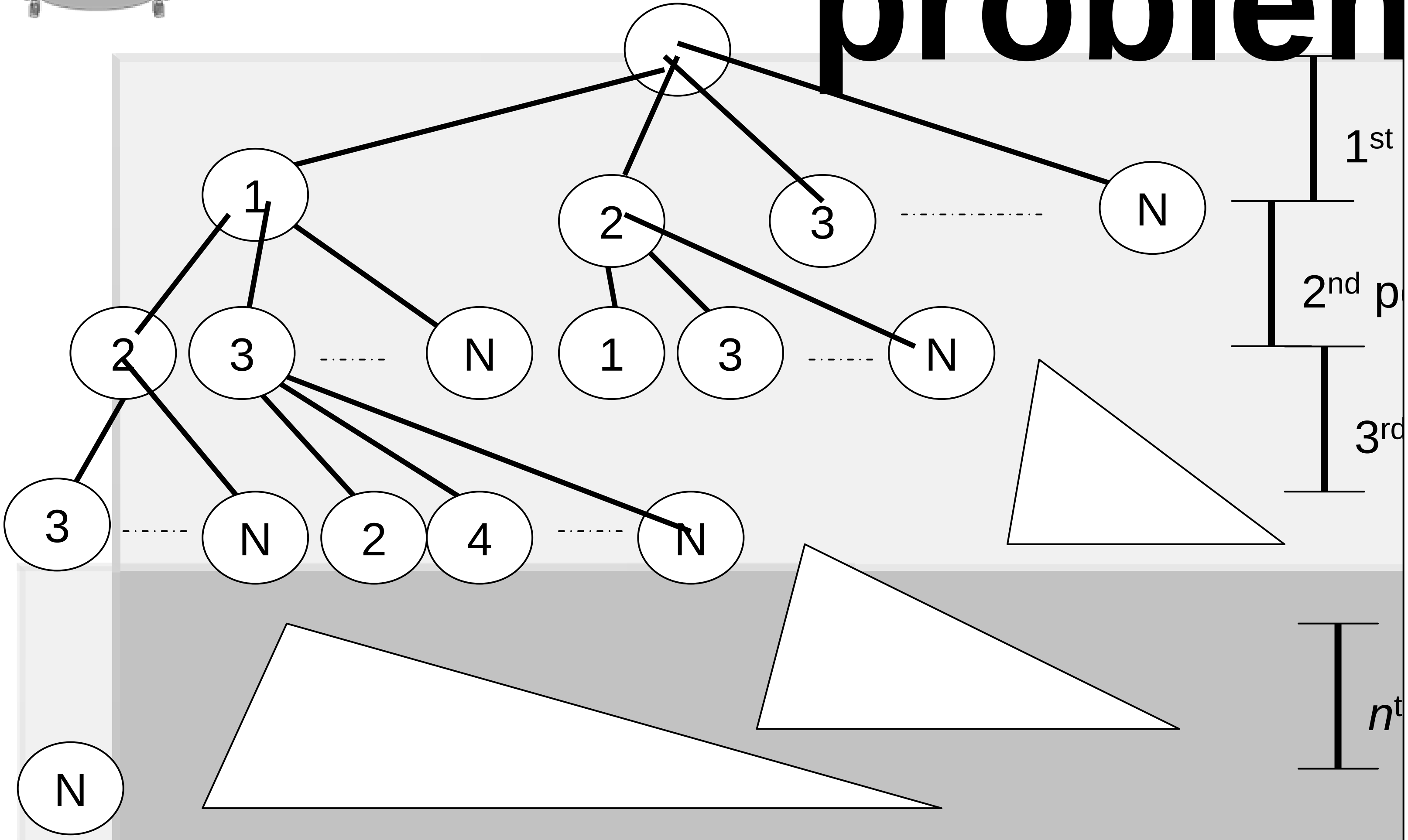
Subse
problem



0000
1000
1100
1110
1111
1101
1010
1011
1001
0100
0110
0111
0101
0010
0011
0001

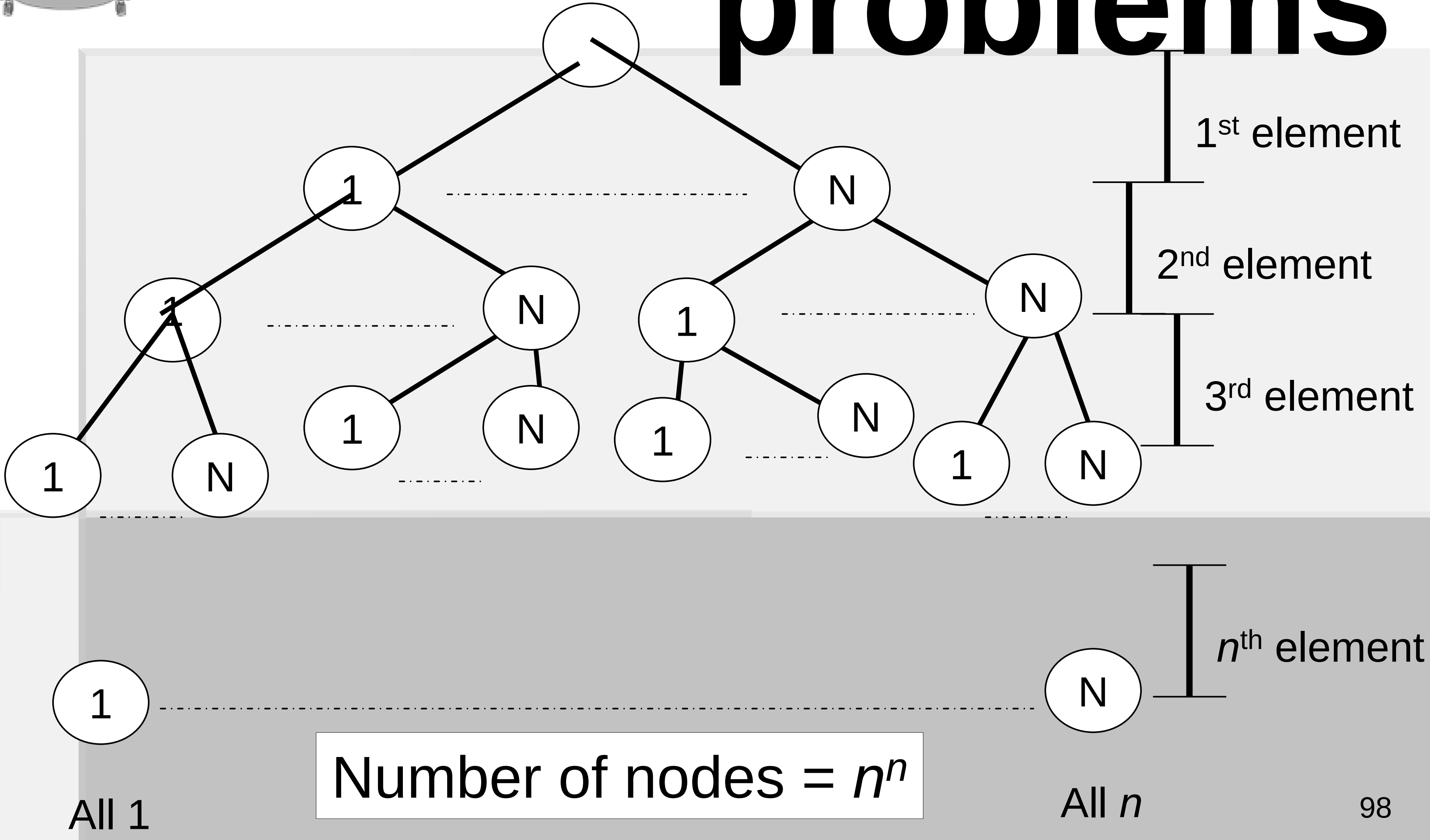
Number of nodes = 2^n

Permutatio problen



1234
1243
1324
1342
1423
1432
2134
2143
2314
2341
2413
2431
3124
3142
…
4312
4321

Partition problems



Number of nodes = n^n

Traversal

Three possible ways

- Depth-first technique
- Breadth-first technique
- Best-first technique

Back tracking

Technique

- Depth-first technique
- Keep track and return back when it cannot be branched.

Branch & bound

	1	2	3	4
A	10	7	13	15
B	12	5	16	12
C	14	9	14	20
D	11	7	14	13