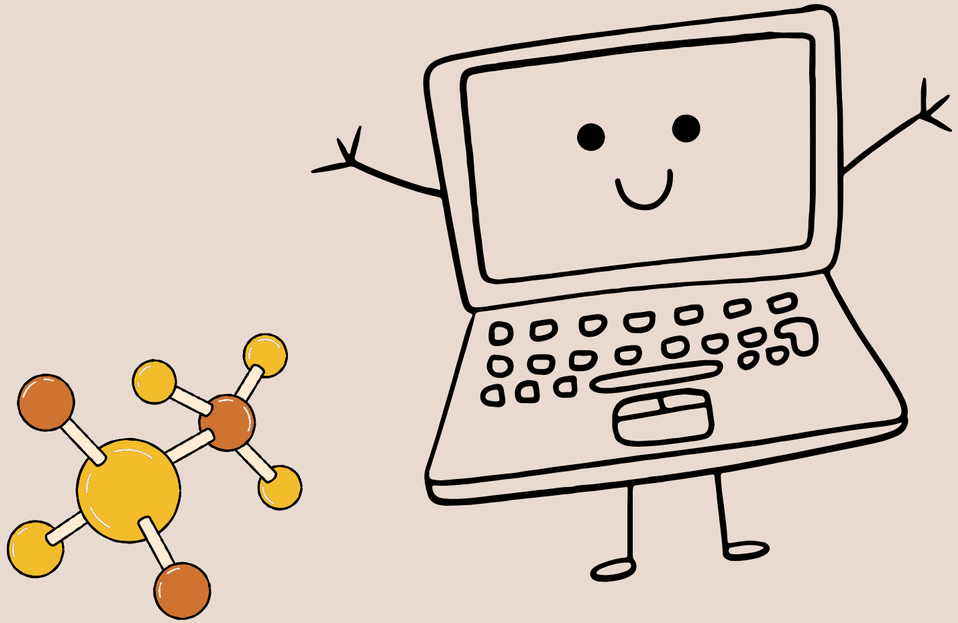
Dynamic Programming : from Fundamentals to Molecular Graph Distances



elicsir
Elevating Italian Computer Science Innovation and Research

Bertinoro 20-22 marzo 2026

SARA GIURIATI - Università degli Studi di Padova

Mentor: Professor **Zeynep Kiziltan**

OUTLINE

Part 1: Introduction to Dynamic Programming

- History & Name Origin
- Definition & Theoretical Bases
- Example: Rod-cutting Problem

Part 2: Floyd-Warshall Algorithm

- What & How
- Comparison with Related Algorithms

Part 3: Application of FW in Cheminformatics

- Distance Computation in Molecular Graphs
- A Use Case

Timeline

World War II

Pierre Massé: hydroelectric dams

+

John von Neumann and **Oskar Morgenstern:** games with 2 players

+

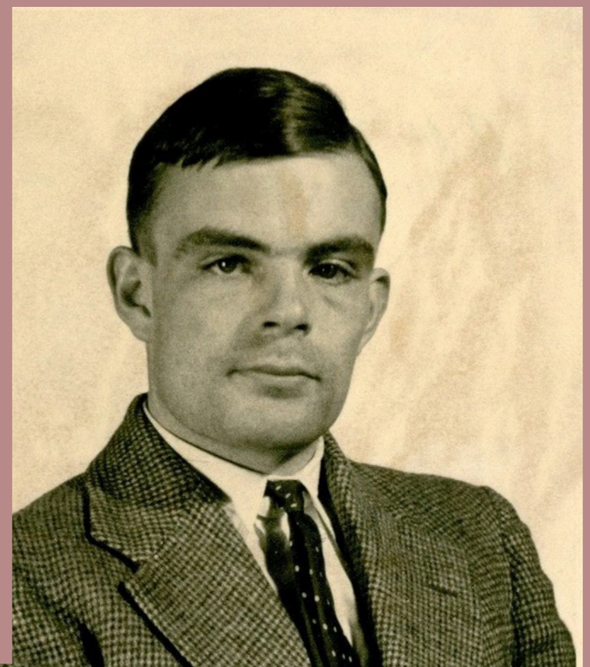
Turing “Banburismus”

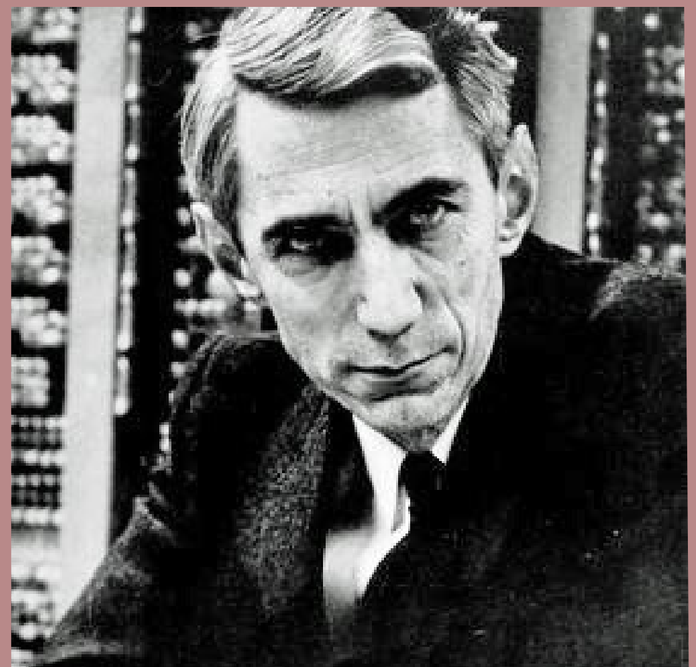
1950

Memoization was used by **Claude Shannon**’s maze-solving robot “Theseus”

1957

“Dynamic Programming” by **Richard Bellman**





Name Origin

"We had a very interesting gentleman in ***Washington*** *named* ***Wilson.*** *He was* ***secretary of Defense,*** *and he actually had a* ***pathological fear and hatred of the word "research".*** *I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term "research" in his presence. You can imagine how he felt, then, about the term* ***"mathematical". . . .I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation.*** *What title, what name, could I choose?"*

Eye of the Hurriance, by Richard Bellman (1984)

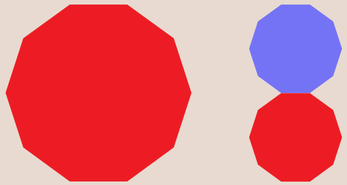
Programming = planning, scheduling

Dynamic = Futuristic Can-Do Zeitgeist of post WW II America

What Is Dynamic Programming?

- **Technique** used in **designing** and analyzing **efficient** algorithms

- Typically applied to **optimization** problems in which a set **choices** are made to arrive at **an optimal solution**



- **Each** choice generates **subproblems**

- Subproblems are **repeated => store** their solution not to have to recompute them!

Rod Cutting Problem

Given a **rod** of **length n** inches and a **table of prices** for i = 1, 2, ..., n, determine the **maximum revenue rn** obtainable by **cutting** the rod and selling the pieces

p1	1
p2	5
...	
pn	20

Note: If the **price pn** for a rod is **large enough**, an optimal solution might require **no cutting** at all

Steps to Develop a DP Algorithm

1. **Structure** the **optimal solution**

2. **Recursively** define the value of an optimal solution

3. **Compute** the value of an optimal solution, typically in a **bottom-up** fashion

4. **Construct** an optimal solution **from computed information**

Example

Length i	1	2	3	4	5	6	7	8	9	10
Price pi	1	5	8	9	10	17	17	20	14	30

Optimal decomposition: $n = i_1 + i_2 + \ldots + i_k \qquad 1 \leqslant k \leqslant n$

Corresponding **revenue:** $r_n = p_{i1} + p_{i2} + \ldots + p_{ik}$

Example

Length i	1	2	3	4	5	6	7	8	9	10
Price pi	1	5	8	9	10	17	17	20	14	30

CANNOT BE CUT

1) NO CUTS = 5 2) + = 1 + 1 = 2 < 5

Subproblems

Example

Length i	1	2	3	4	5	6	7	8	9	10
Price pi	1	5	8	9	10	17	17	20	14	30

1) NO CUTS = 8

2) + + = 1 + 1 + 1 = 3 < 8

3) + = 5 + 1 = 6 < 8

4) + = 1 + 5 = 6 < 8

Repeated subproblems

Mathematical Generalization

$$r_1 = p_1$$

$$r_2 = max\{p_2, r_1 + r_1\}$$

$$r_3 = max\{p_3, r_1 + r_2, r_2 + r_1\} =$$

$$= max\{p_3, r_1 + max\{p_2, r_1 + r_1\}\}$$

Commutative property of the sum

Mathematical Generalization

In **general**

$$r_n = max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\}$$

$$r_n = max\{p_i, r_i + r_{n-i} : 1 \leqslant i \leqslant n\}$$

Dynamic Programming vs Divide and Conquer

They **both** solve problems by **combining** the **solutions** to **subproblems**

Divide-and-Conquer when subproblems are **disjoint**

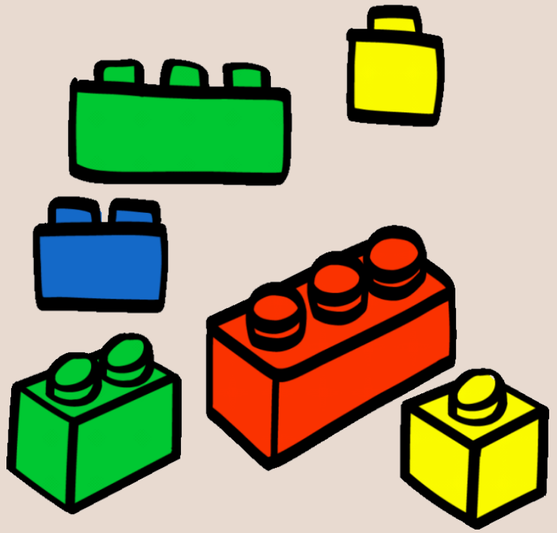
Dynamic-Programming when subproblems **share** subsubproblems

Bottom-up Approach

Solving any particular subproblem

depends only on **solving “smaller”**

subproblems



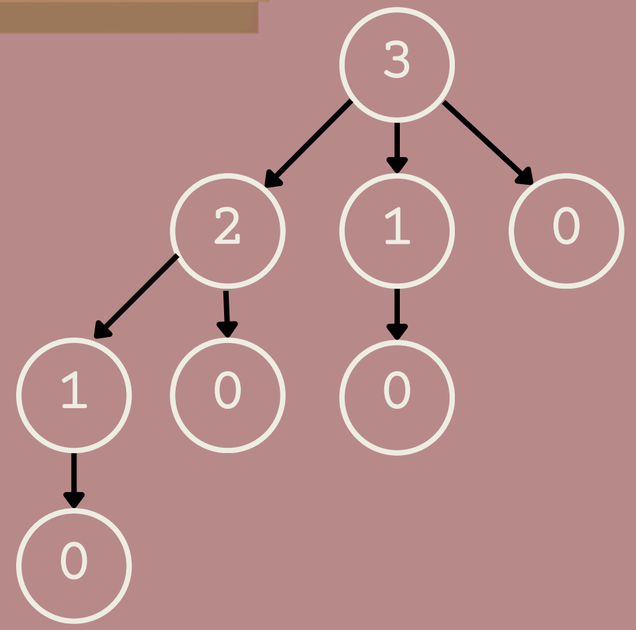
Bottom-up Approach

1. **Solve** the **subproblems** in **size order**, smallest first
2. **Store** the solution to each subproblem when it is first solved
3. Use the **stored data** to compute the **next value**

Rod Cutting Problem

Divide-and-conquer

```
RodCutRec(p,n)
if n == 0
   return 0
q = −∞
for i = 1 to n
   q = max{q,p[i]-RodCutRec(p,n-i)}
return q
```



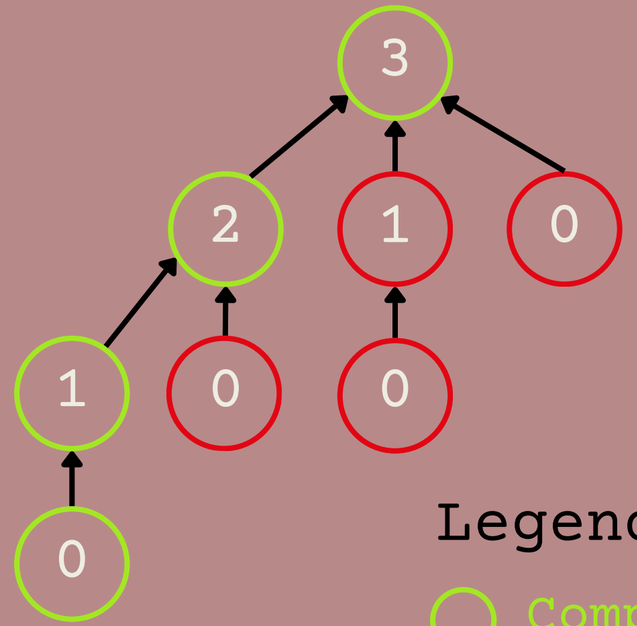
Legend:

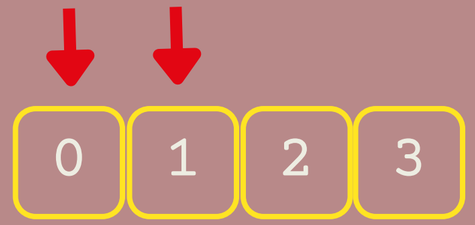
Compute

Rod Cutting Problem

Dynamic Programming: bottom-up

```
BottomUpCutRod(p,n)
let r[0:n] be a new array
r[0] = 0
for j = 1 to n
   q = −∞
   for i = 1 to j
      q = max{q, p[i]+r[j-i]}
   r[j] = q
return r[n]
```





Legend:

Compute + store

Reuse

Memory

Access

Top-down Approach + Memoization

Store the results already **computed**

in order **not** to have to **recompute**

them when they are **repeated**

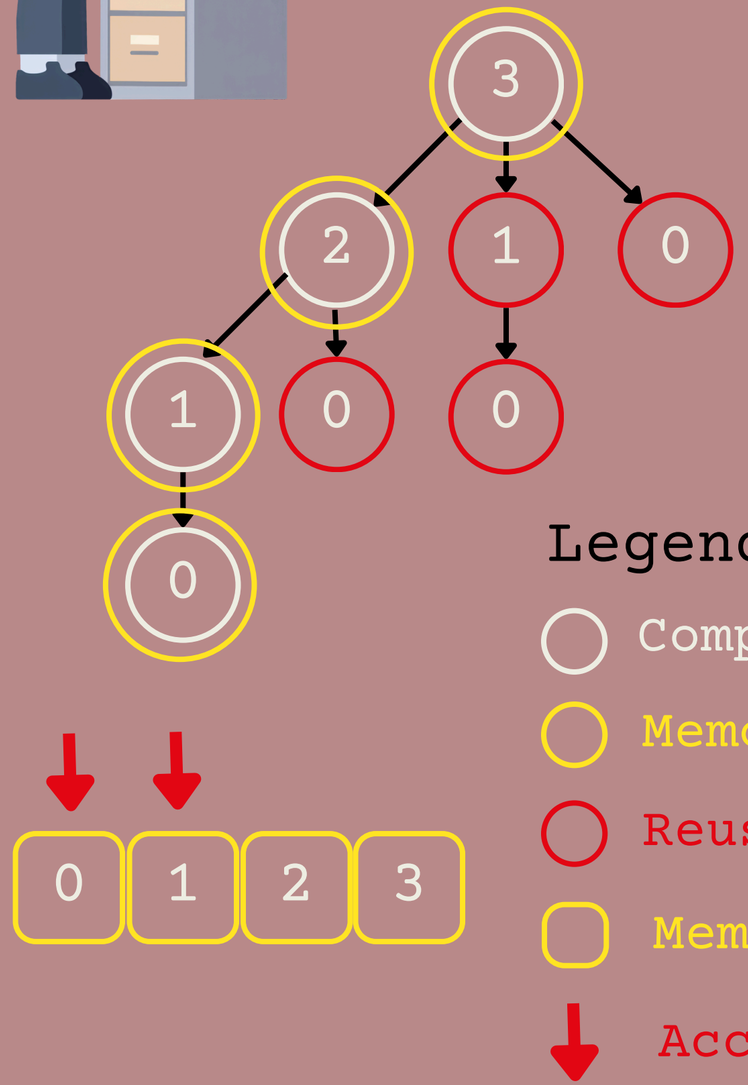
Top-down Approach + Memoization

1. **Write** the procedure **recursively** in a natural manner
2. **Modify** the algorithm to **save** the **result** of each **subproblem** (usually in an array or hash table)

Rod Cutting Problem

Dynamic Programming: top-down

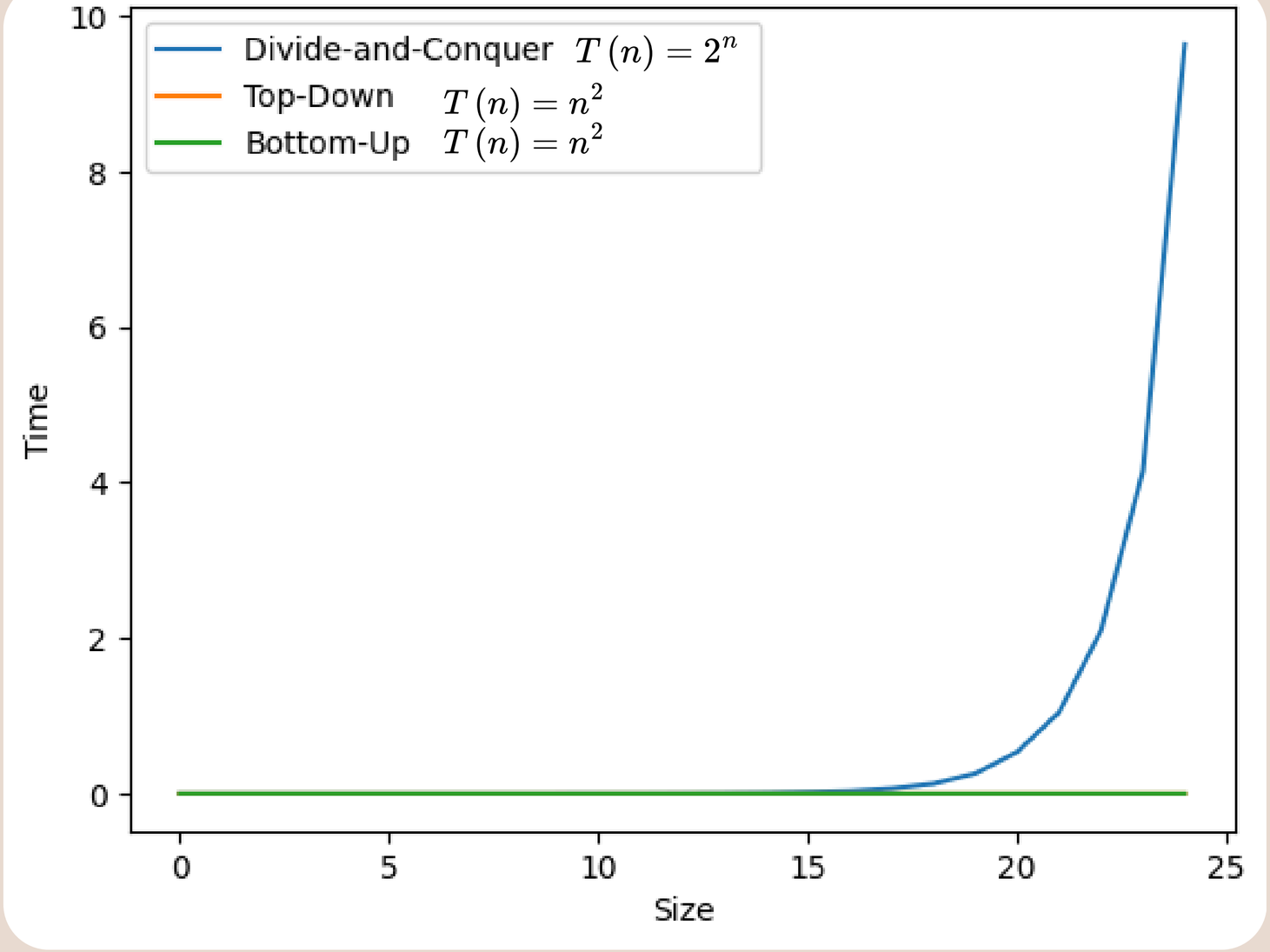
```
MemoizedCutRod(p,n)
let r[0:n] be a new array
for i = 0 to n
  r[i]=−∞
return MemoizedCutRodAux(p,n,r)

MemoizedCutRodAux(p,n,r)
if r[n] ⩾ 0
  return r[n]
if n == 0
  q = 0
else q = −∞
     for i = 0 to n
         q = max{q, p[i] + MemoizedCutRodAux(p,n-i,r)}
r[n] = q
return q
```



Time Complexity

Space

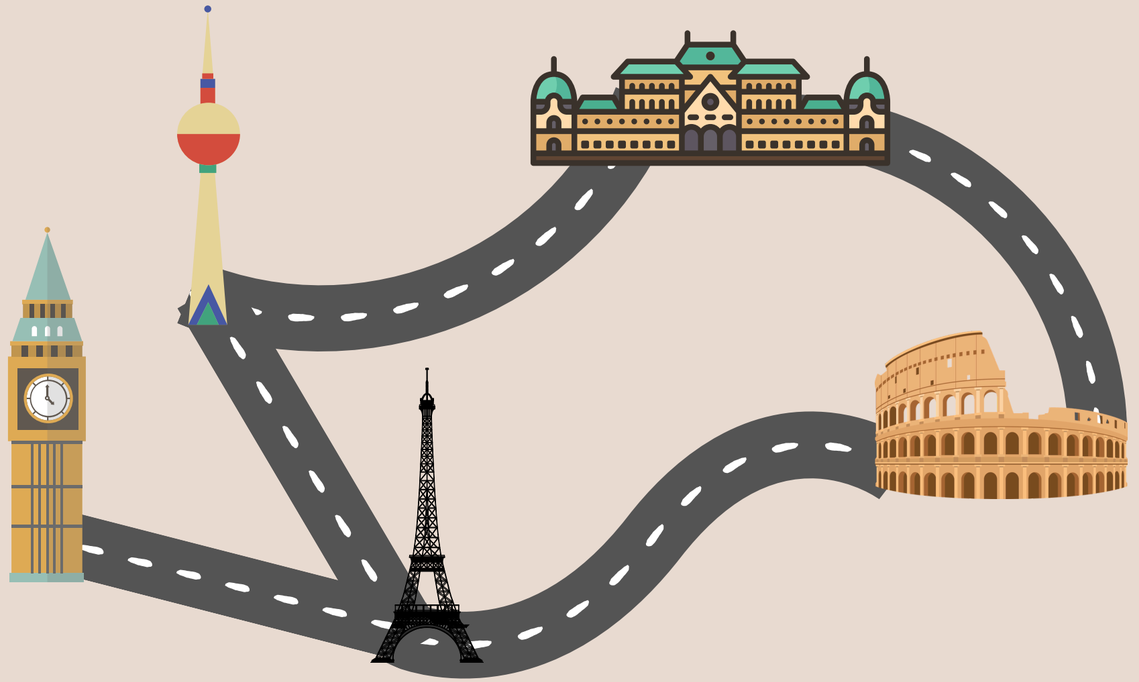
Time



demo for curious minds!

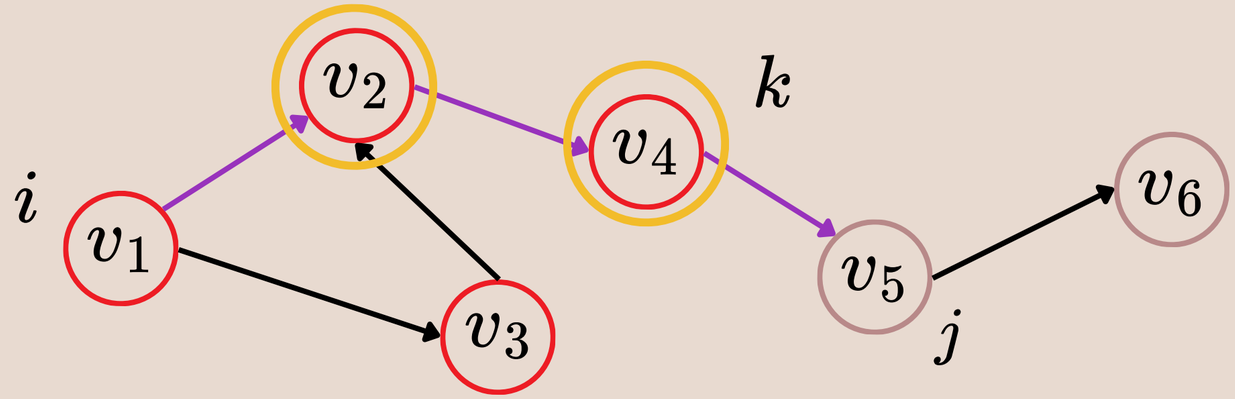
Floyd-Warshall Algorithm
1962

All-Pairs Shortest Paths



Given a **weighted, directed graph** G(V,E) and a **weight function** w: E → **R,** the goal is to find, **for every pair of vertices u,v** ∈ V, a **shortest** (least-weight) path from u to v, where the weight of the path is the sum of the weights of its constituent edges

Shortest Path

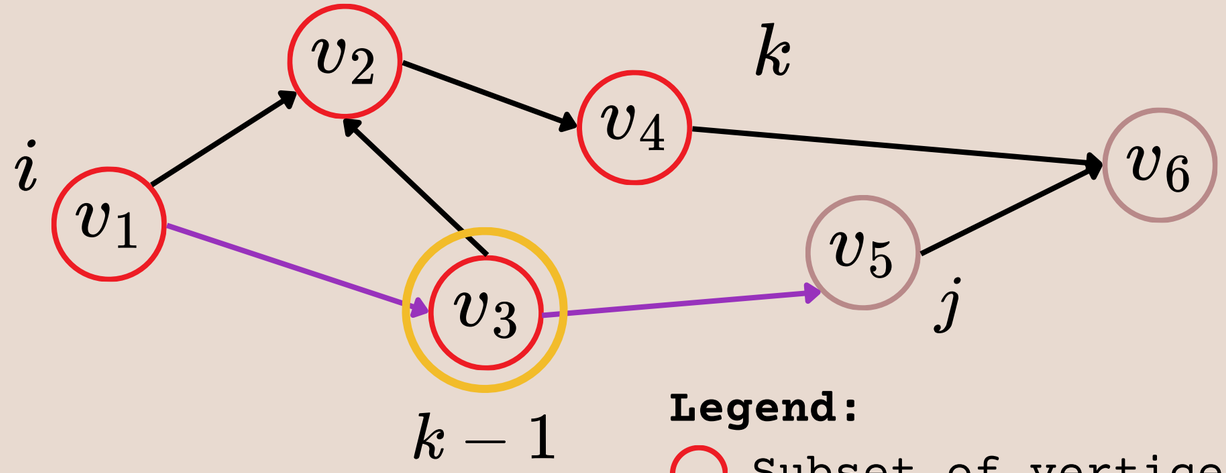


Legend:

- Subset of vertices
- Shortest path **p**
- Intermediate vertex

- Let **V = {1,2,...n}**
- Take a **V'** $\subseteq$ **V = {1,2,...,k}** for some $1 \leqslant k \leqslant n$
- For **any pair** $i,j \in$ **V**, consider all paths from i to j whose **intermediate** vertices comes from **V'** and let **p** be a **shortest path**

k is NOT an intermediate vertex



Legend:

- Subset of vertices
- Shortest path **p**
- Intermediate vertices

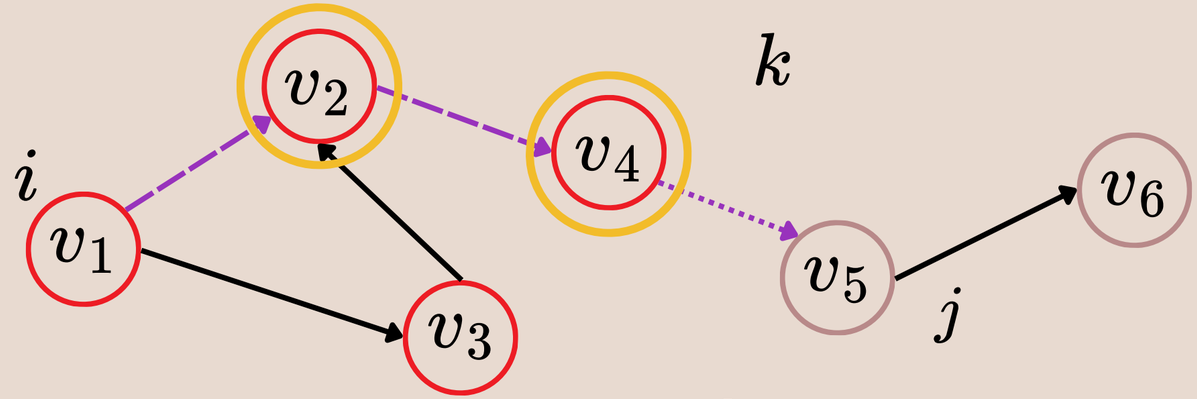
Observation

If k is **not an intermediate** vertex:

=> all intermediate vertices of path **p** ∈ **{1,2,...,k-1}**

=> a **shortest** path with all intermediate vertices ∈ **{1,2,...,k-1}** is **also** a **shortest** path with all intermediate vertices ∈ **{1,2,...,k}**

k IS an intermediate vertex



Legend:

- Subset of vertices
- Intermediate vertices
- Shortest path **p1** from 1 to 4
- Shortest path **p2** from 4 to 5

Observation

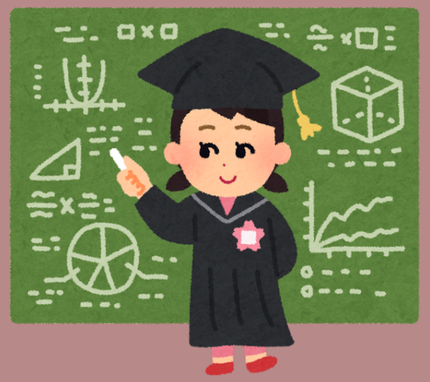
If k **is an intermediate** vertex of path p

=> decompose p into i → k → j

=> p1 = i → k is a **shortest** path from i to k with all intermediate vertices $\in \{1,2,...,k-1\}$

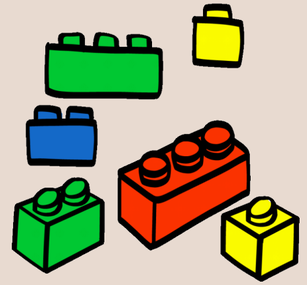
=> p2 = k ⇢ j is a **shortest** path from k to j with all intermediate vertices $\in \{k+1,...,j\}$

Mathematical Definition



Distances Matrix $D^{(k)}$

Predecessors Matrix $\Pi^{(k)}$

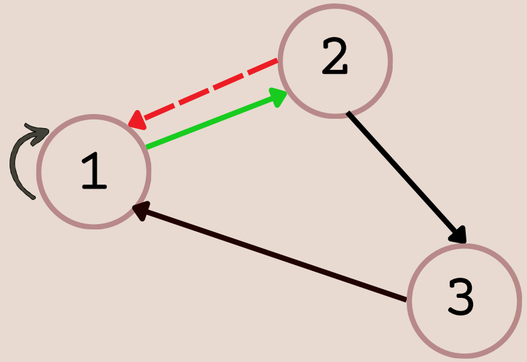


$$d_{ij}^{(k)} = \begin{cases} w_{ij} & if \;\; k = 0 \\ min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & if \;\; k \geqslant 1 \end{cases}$$

$$\pi_{ij}^{(0)} = \begin{cases} NIL & if \;\; i = j \vee w_{ij} = \infty \\ i & if \;\; i \neq j \wedge w_{ij} < \infty \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{kj}^{(k-1)} & if \;\; d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{ij}^{(k-1)} & if \;\; d_{ij}^{(k-1)} < d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

How It Works



When k = 0

$$d_{ij}^{(0)} = \begin{cases} 0 & if \quad i = j \\ w_{ij} & if \quad i \neq j \quad \text{AND edge from } i \text{ to } j \\ \infty & if \quad i \neq j \quad \text{AND no edge from } i \text{ to } j \end{cases}$$

$$\pi_{ij}^{(0)} = \begin{cases} NIL & if \quad i = j \ \vee w_{ij} = \infty \\ i & if \quad i \neq j \ \wedge w_{ij} < \infty \end{cases}$$

How It Works

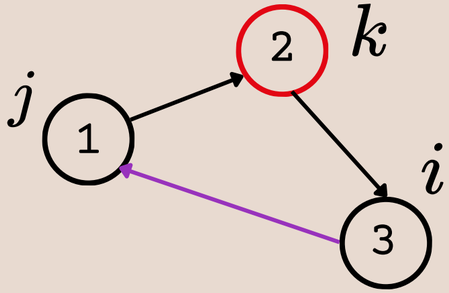
When **k > 0**

$$d_{ij}^{(k)} = min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

$d_{ij}^{(k-1)}$

k is **not** an intermediate node

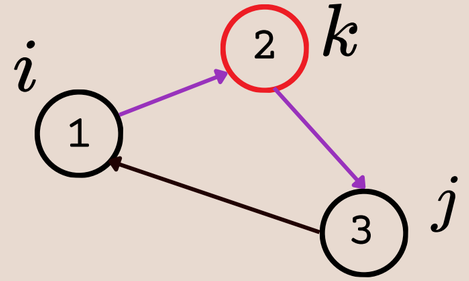
$$\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$$



$d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

k **is** an intermediate node

$$\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$$



Pseudocode

Time complexity:

$$\Theta\left(n^3\right)$$

Floyd-Warshall(W,n)

$D^{(0)} = \mathtt{W}$

foreach d_{ij} **in** $D^{(0)}$

  if $(d_{ij} \neq \infty \wedge d_{ij} \neq 0)$

    $\pi_{ij}^{(0)} = i$

for k = 1 **to** n **do**

  D(k) = $\left(d_{ij}^{(k)}\right)$ be a new n × n

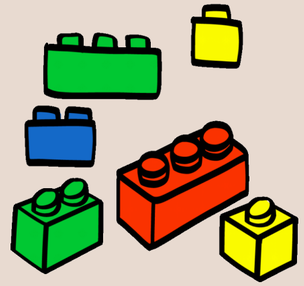
  for i = 1 **to** n **do**

    for j = 1 **to** n **do**

      $d_{ij}^{(k)} = min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$

      $\pi_{ij}^{(0)} = \pi_{kj}^{(k-1)}$ if $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

return $D^{(n)}, \pi^{(n)}$

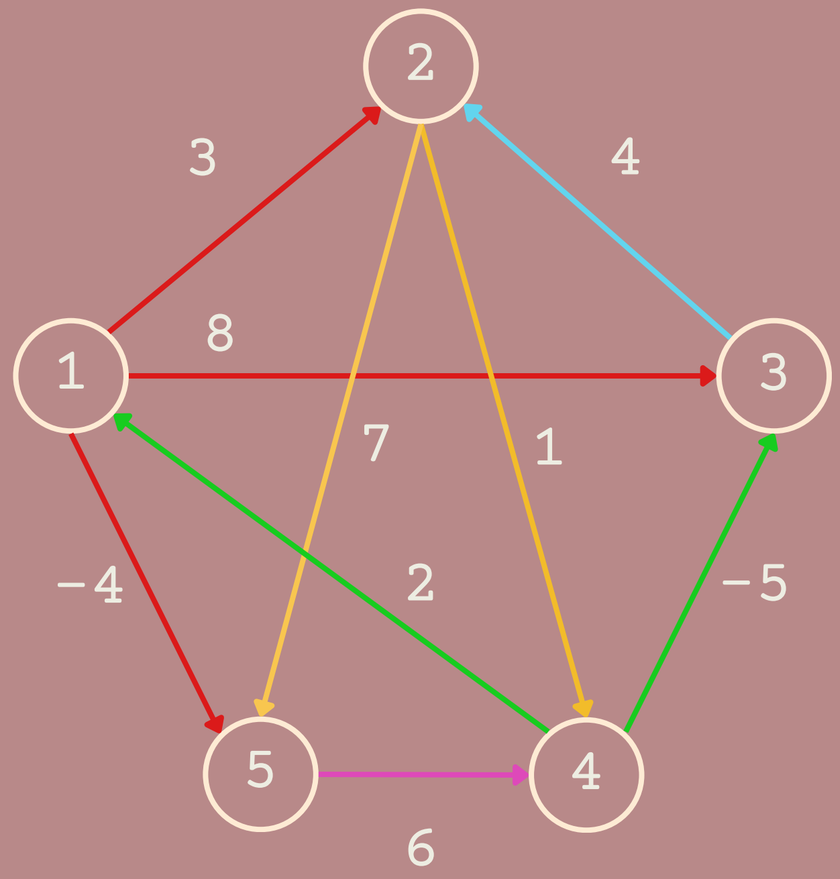


Example

$$D^{(0)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$\Pi^{(0)} = \begin{bmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 0

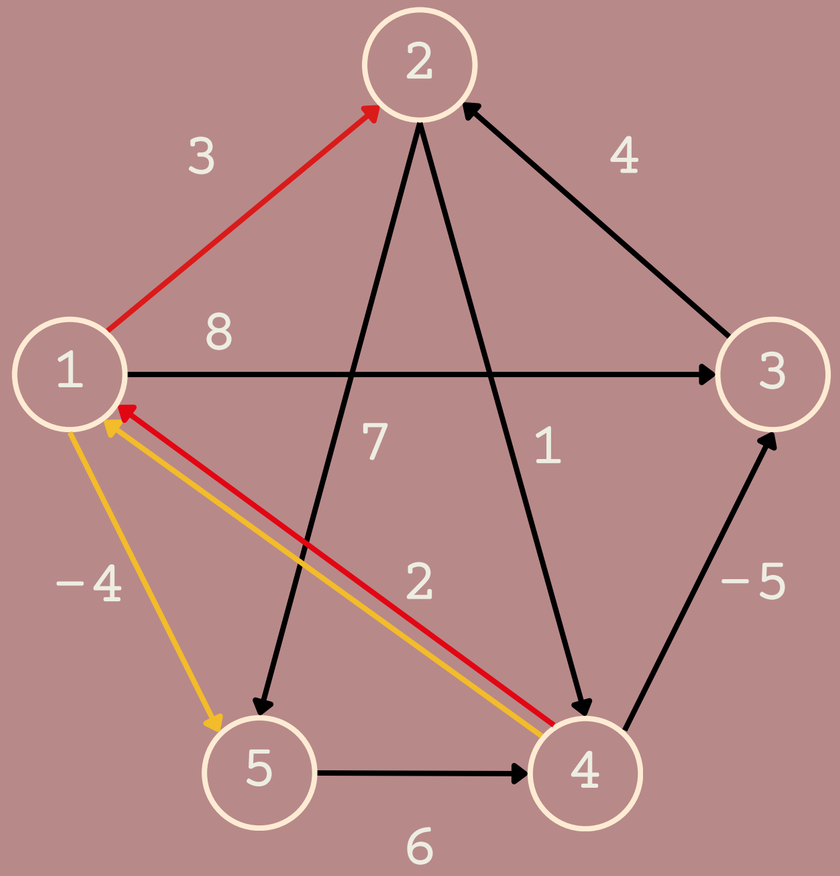


Example

$$D^{(1)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$\Pi^{(1)} = \begin{bmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 1

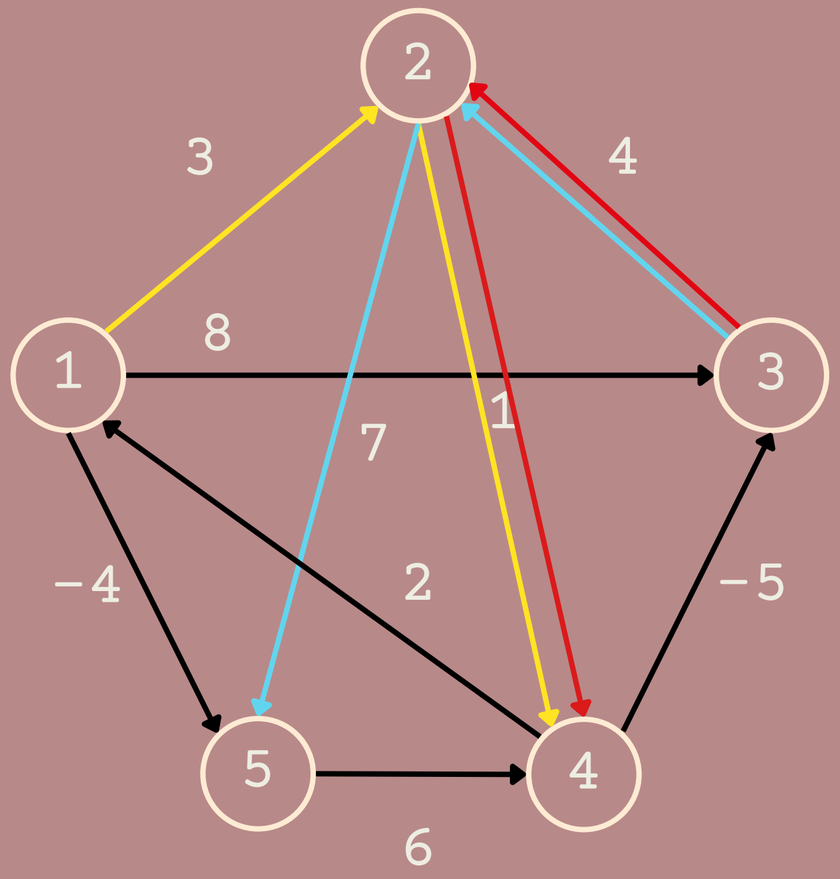


Example

$$D^{(2)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$\Pi^{(2)} = \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 2

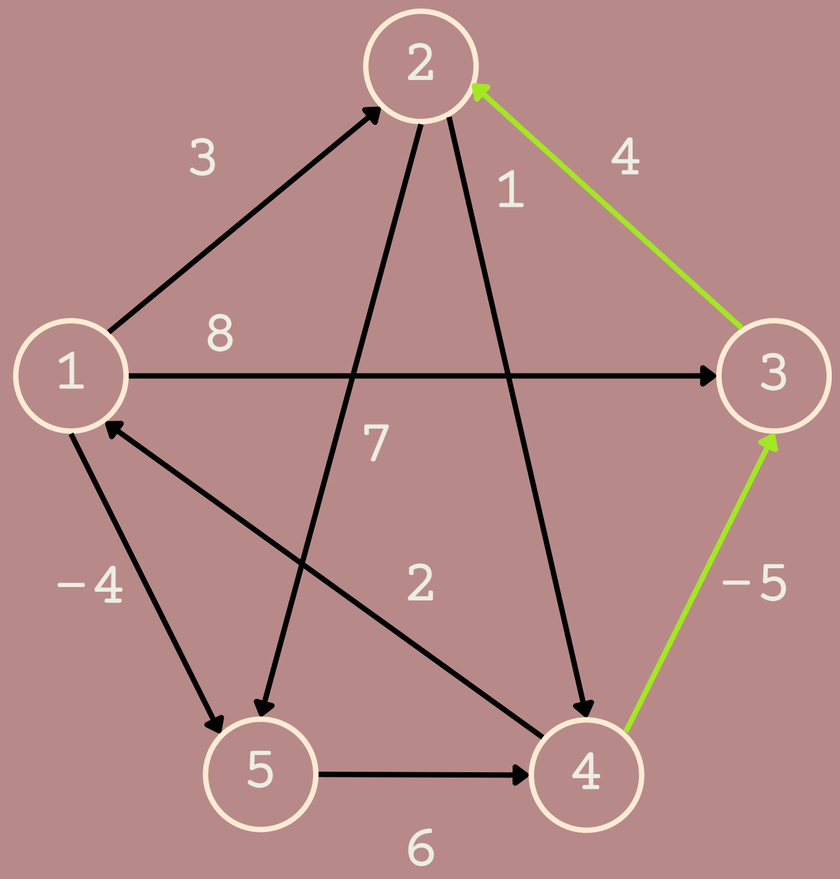


Example

$$D^{(3)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$\pi^{(3)} = \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 3

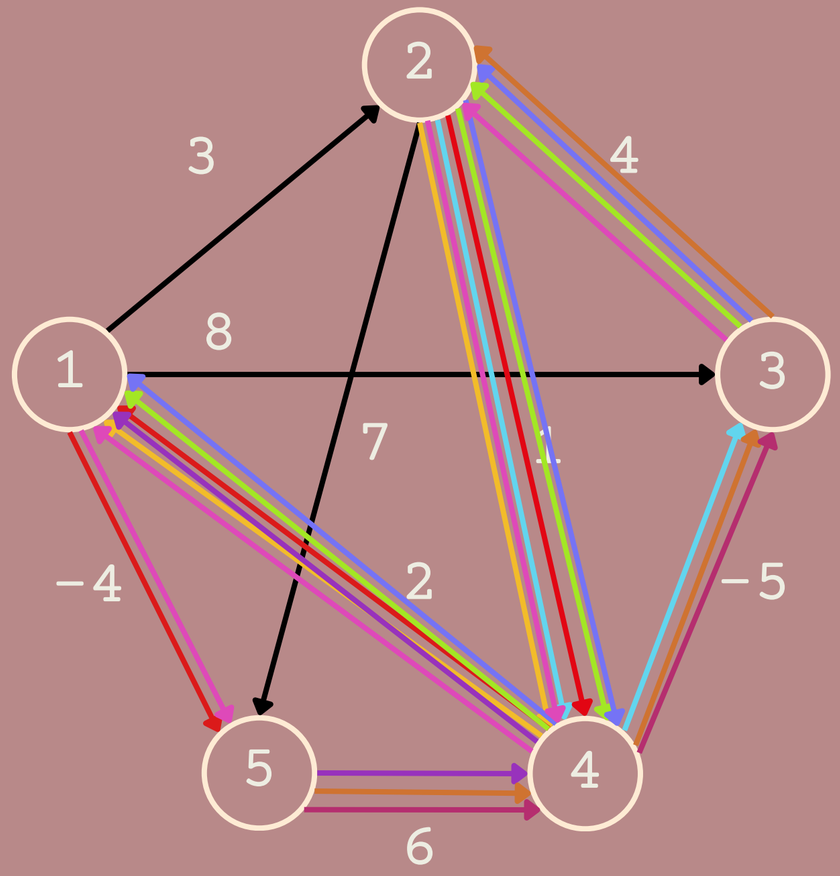


Example

$$D^{(4)} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$\pi^{(4)} = \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 4

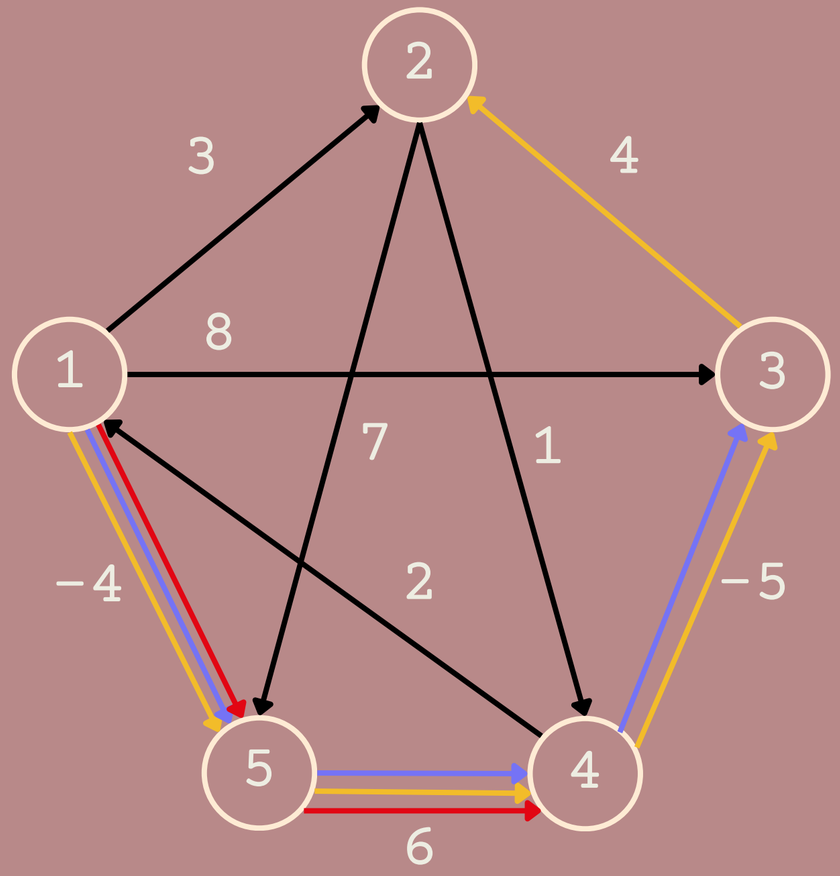


Example

$$D^{(5)} = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$\Pi^{(5)} = \begin{bmatrix} \text{NIL} & 3 & 1 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix}$$

Iteration k = 5



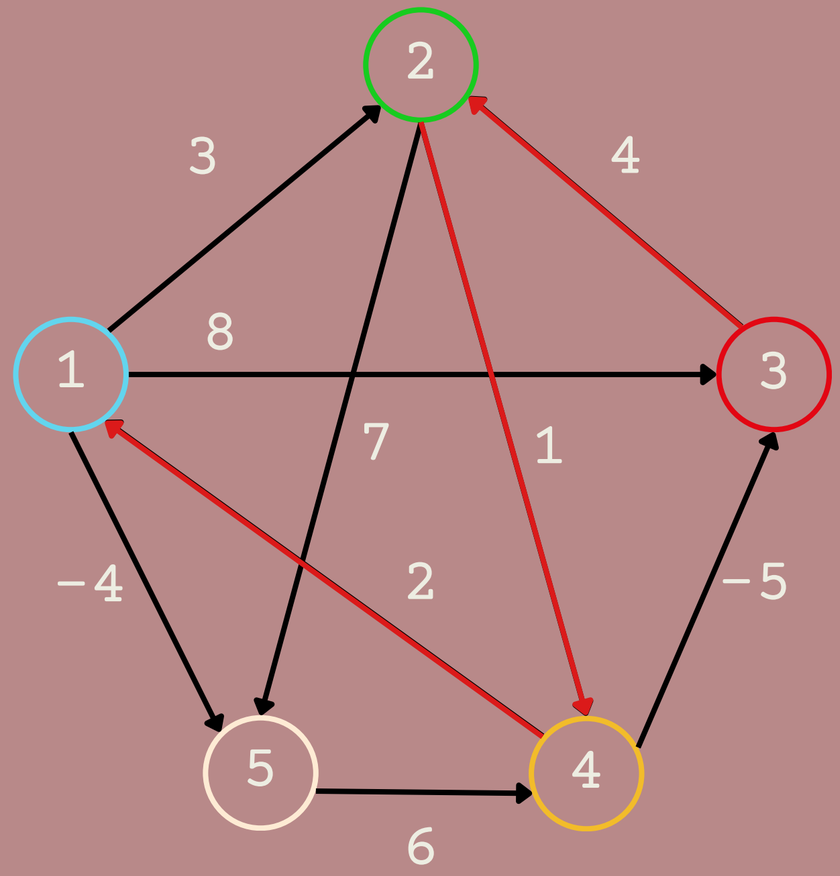
Shortest-path Reconstruction

Which is the shortest path

from 3 to 1?

$$\Pi^{(5)} =$$

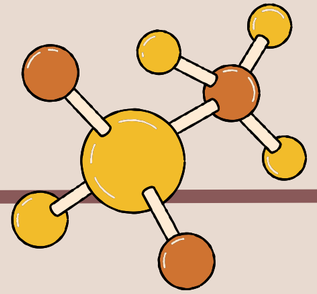
1	2	3	4	5	
NIL	3	1	2	1	1
4	NIL	4	2	1	2
4	3	NIL	2	1	3
4	3	4	NIL	1	4
4	3	4	5	NIL	5



Time Complexity

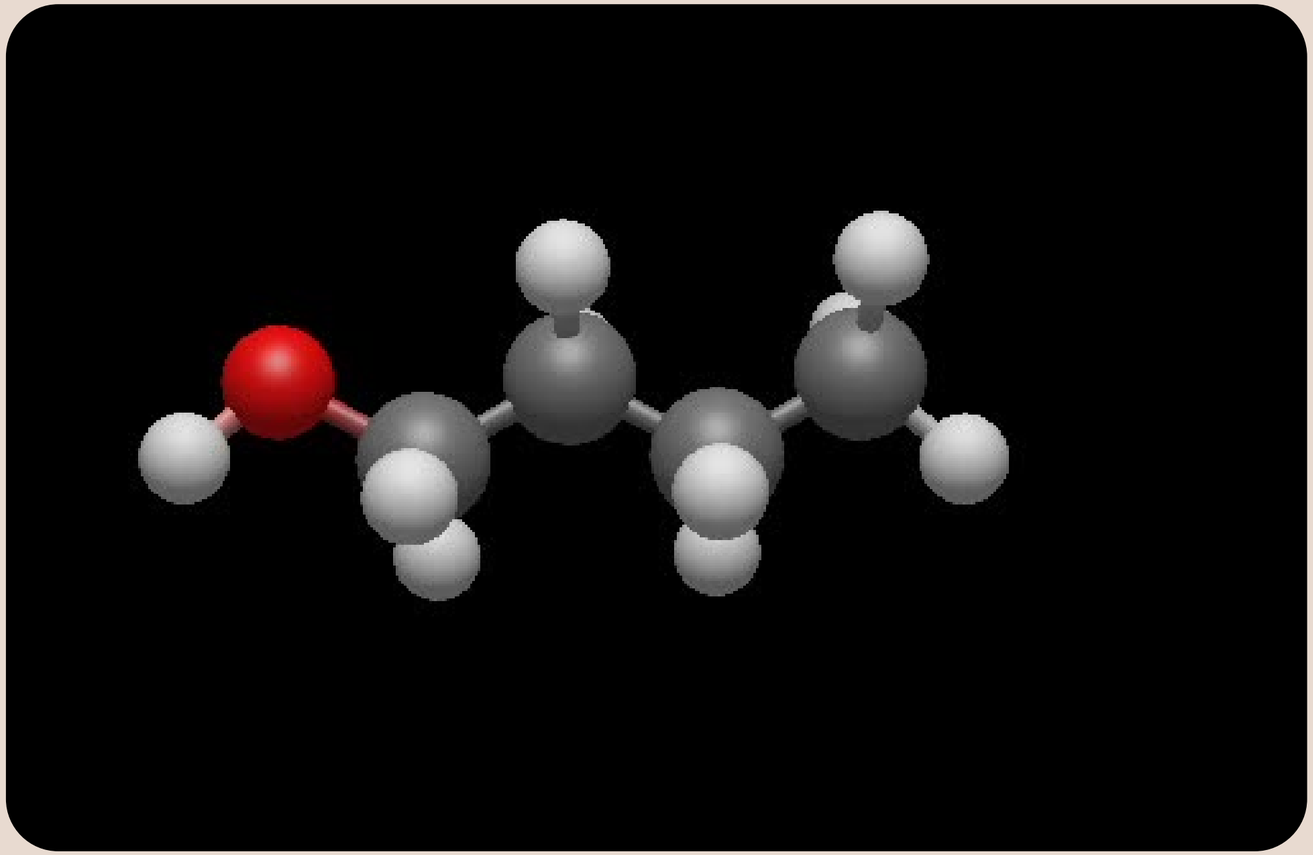
Algorithms & Complexity	Negative weights (no negative cycles)	Dense graphs
Floyd-Warshall $O(V^3)$	✓	✓
Johnson $O(V^2 \lg V + VE)$	✓	✗
Dijkstra $O(VE \lg V)$	✗	✗
Bellman-Ford $O(V^2 E)$	✓	✗

Floyd-Warshall Algorithm in Cheminformatics

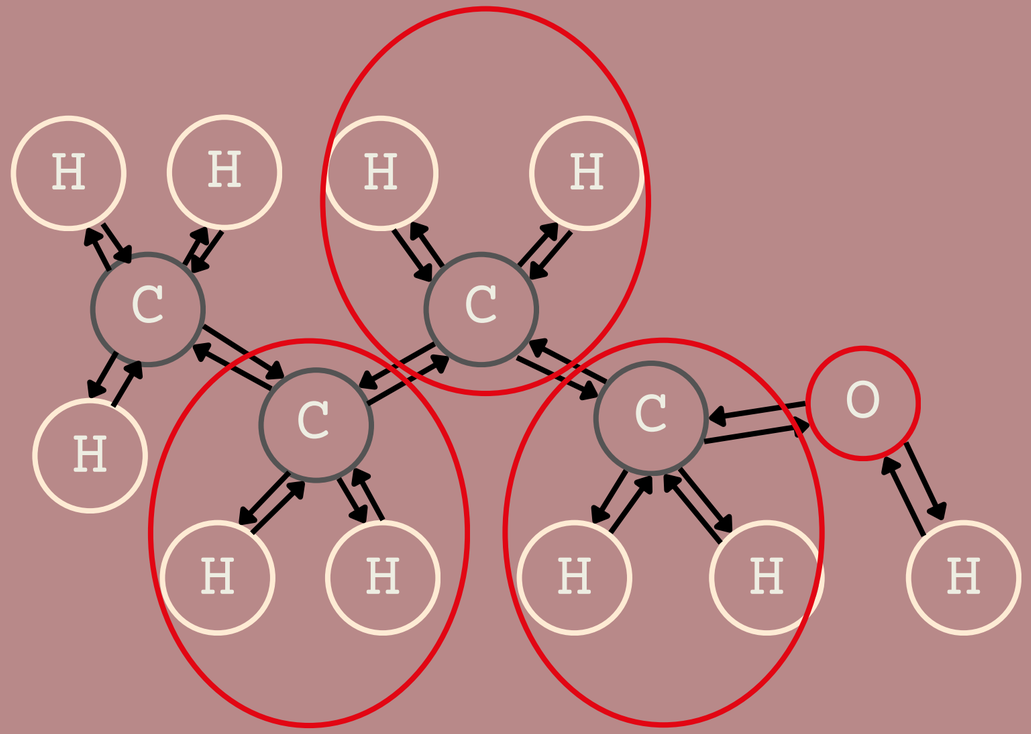


Molecular Graph

Butanol: $C_4H_{10}O$



Intuitively



FW for Molecular Distance

Butanol: $C_4H_{10}O$

	Elemento	Valence	Carica formale	MMFF94 Partial Charge	X (Å)	Y (Å)	Z (Å)
Atomo 1	O	2	0	-0,680	4,6099	-0,6638	0,0000
Atomo 2	C	4	0	0,000	2,3050	-0,6638	0,0000
Atomo 3	C	4	0	0,280	3,4575	0,0000	0,0000
Atomo 4	C	4	0	0,000	1,1525	0,0000	0,0000
Atomo 5	C	4	0	0,000	0,0000	-0,6638	0,0000
Atomo 6	H	1	0	0,400	5,3314	-0,0457	0,0000
Atomo 7	H	1	0	0,000	2,3050	-1,2816	-0,8737
Atomo 8	H	1	0	0,000	2,3050	-1,1540	0,9511
Atomo 9	H	1	0	0,000	3,4575	0,6178	0,8737
Atomo 10	H	1	0	0,000	3,4575	0,4903	-0,9511
Atomo 11	H	1	0	0,000	1,1525	0,6178	0,8737
Atomo 12	H	1	0	0,000	1,1525	0,4902	-0,9511
Atomo 13	H	1	0	0,000	-0,1037	-1,1983	0,9211
Atomo 14	H	1	0	0,000	-0,8099	0,0277	-0,1043
Atomo 15	H	1	0	0,000	-0,0136	-1,3548	-0,8168

In practice

1. Compute the squared distance between every pair of atoms given their 3D coordinates

$$d = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

FW for Molecular Distance

Atomic radius (pm)

The reported atomic radius values are estimated relative to the distance between the nuclei of two adjacent atoms in a molecule or in a crystal. For the noble gases, the reference is the distance between atoms bound by van der Waals interactions, rather than by a covalent bond.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	H 30																	He 93
2	Li 155	Be 112											B 98	C 91	N 92	O 66	F 64	Ne 160
3	Na 190	Mg 160											Al 143	Si 132	P 128	S 104	Cl 99	Ar 191
4	K 235	Ca 197	Sc 160	Ti 146	V 134	Cr 127	Mn 126	Fe 126	Co 125	Ni 124	Cu 128	Zn 133	Ga 141	Ge 137	As 139	Se 140	Br 114	Kr 200
5	Rb 248	Sr 215	Y 179	Zr 160	Nb 146	Mo 139	Tc 136	Ru 133	Rh 134	Pd 138	Ag 144	Cd 154	In 166	Sn 162	Sb 159	Te 160	I 133	Xe 220
6	Cs 267	Ba 222	La 187	Hf 158	Ta 146	W 139	Re 137	Os 134	Ir 136	Pt 138	Au 144	Hg 157	Tl 171	Pb 175	Bi 170	Po 176	At 140	Rn 250
7	Fr 272	Ra 220	Ac 188	Rf /	Db /	Sg /	Bh /	Hs /	Mt /	Ds /	Rg /	Cn /	Nh /	Fl /	Mc /	Lv /	Ts /	Og /

	Ce 161	Pr 182	Nd 182	Pm /	Sm 166	Eu 204	Gd 179	Tb 177	Dy 177	Ho 176	Er 175	Tm 174	Yb 192	Lu 158
	Th /	Pa 161	U /	Np /	Pu /	Am /	Cm /	Bk /	Cf /	Es /	Fm /	Md /	No /	Lr /

TAVOLA PERIODICA DEGLI ELEMENTI

ZANICHELLI

In practice

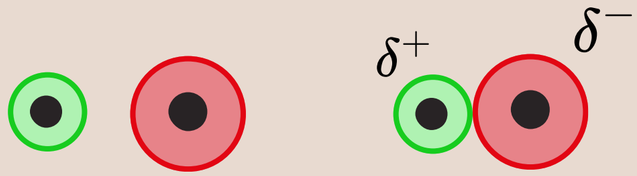
2 Compute the threshold that identifies the presence of a bond between two atoms

$$t = 1.2 \times (r_1 + r_2)^2$$

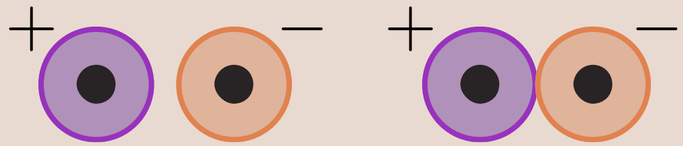
FW for Molecular Distance

Pure covalent bond

Polar covalent bond



Ionic bond



In practice

3 **Compare the distance and the threshold: there is a bond if**

$$d < t$$

FW for Molecular Distance

Butanol: $C_4H_{10}O$

```
distances
[[ 0. inf  1. inf inf  1. inf inf inf inf inf inf inf inf inf]
 [inf  0.  1.  1. inf inf  1.  1. inf inf inf inf inf inf inf]
 [ 1.  1.  0. inf inf inf inf inf  1.  1. inf inf inf inf inf]
 [inf  1. inf  0.  1. inf inf inf inf inf  1.  1. inf inf inf]
 [inf inf inf  1.  0. inf inf inf inf inf inf inf  1.  1.  1.]
 [ 1. inf inf inf inf  0. inf inf inf inf inf inf inf inf inf]
 [inf  1. inf inf inf inf  0. inf inf inf inf inf inf inf inf]
 [inf  1. inf inf inf inf inf  0. inf inf inf inf inf inf inf]
 [inf inf  1. inf inf inf inf inf  0. inf inf inf inf inf inf]
 [inf inf  1. inf inf inf inf inf inf  0. inf inf inf inf inf]
 [inf inf inf  1. inf inf inf inf inf inf  0. inf inf inf inf]
 [inf inf inf  1. inf inf inf inf inf inf inf  0. inf inf inf]
 [inf inf inf inf  1. inf inf inf inf inf inf inf  0. inf inf]
 [inf inf inf inf  1. inf inf inf inf inf inf inf inf  0. inf]
 [inf inf inf inf  1. inf inf inf inf inf inf inf inf inf  0.]]
```

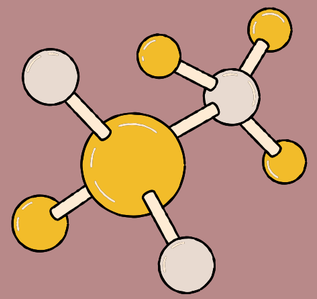
In practice

4

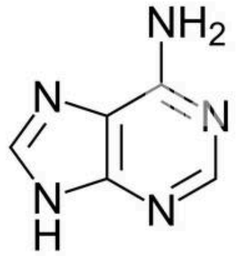
- Assign to each bond weight 1
- Apply the Floyd-Warshall algorithm

Demo for curious minds!

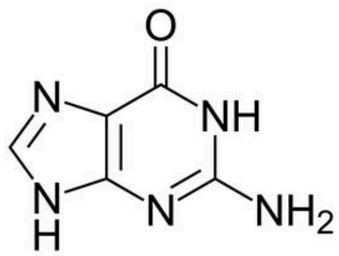
A Use Case



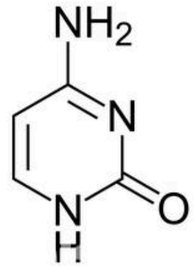
Smallest Set of Smallest Rings (SSSR):
fundamental task in chemical parsing



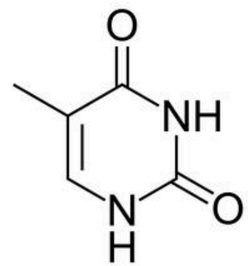
A
Adenine
$C_5H_5N_5$



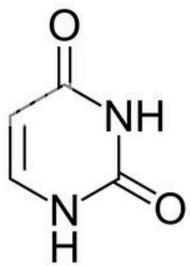
G
Guanine
$C_5H_5N_5O$



C
Cytosine
$C_4H_5N_3O$



T
Thymine
$C_5H_6N_2O_2$



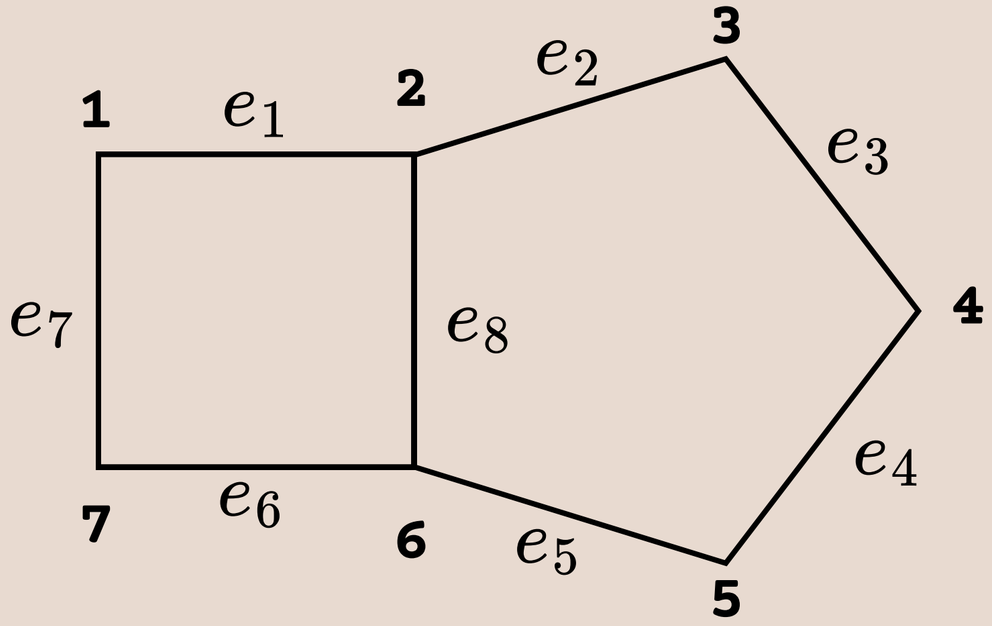
U
Uracil
$C_4H_4N_2O_2$

Smallest Set of Smallest Rings

$G = (V, E)$ connected graph with:

- $V(G) = \{v_1, v_2, \ldots, v_n\}$
- $E(G) = \{e_1, e_2, \ldots, e_m\}$

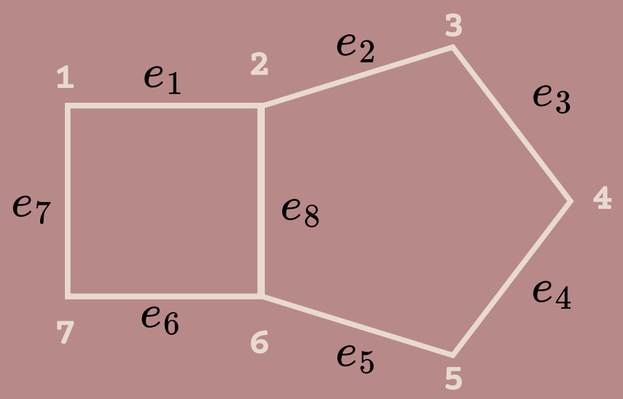
Example



$n = 7$

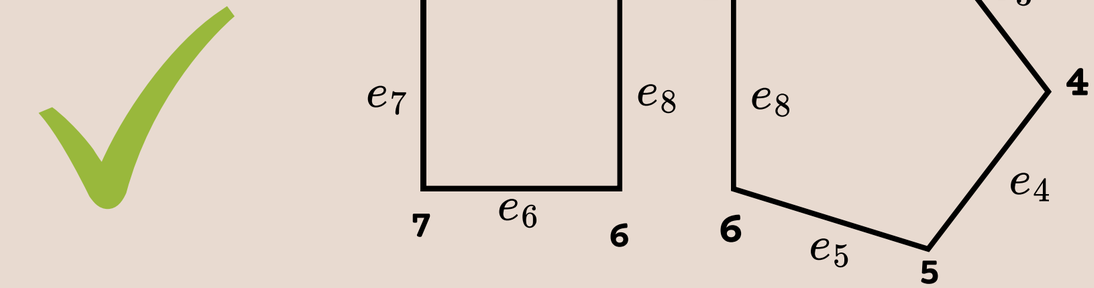
$m = 8$

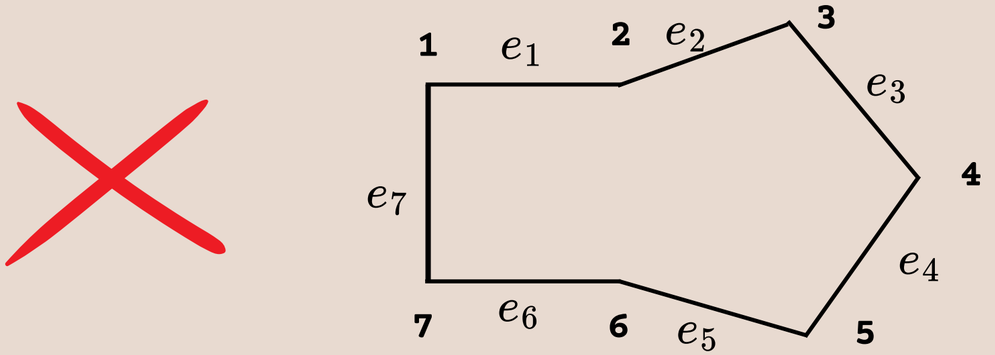
Mathematical Definition of SSSR



Cycle in a graph = non empty sequence of adjacent edges in which only the first and the last nodes repeat

Cycle space:





$\varnothing$

The **theoretical** number of SSSR is given by **Euler rule:** $n_{SSSR} = m - n + 1 = 8 - 7 + 1 = 2$

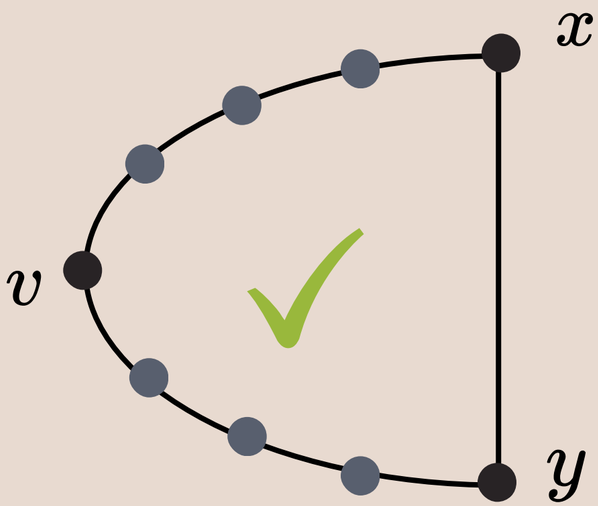
SSSR & Shortest Path

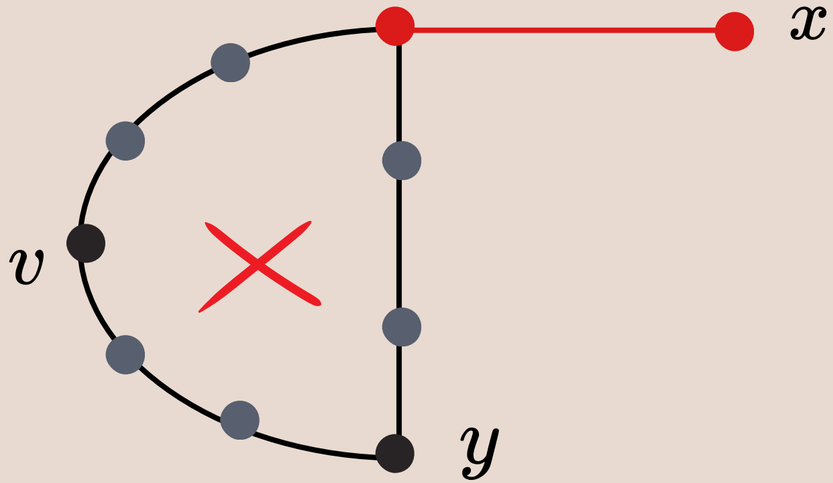
HORTON'S METHOD

We define a **ring** as:

$$C(v,x,y) = p(v,x) + p(v,y) + p(x,y)$$

The three points v, x and y are members of a ring if the **edges** of the three **shortest** paths $p(v,y)$, $p(v,x)$ and $p(x,y)$ are **not shared** among them



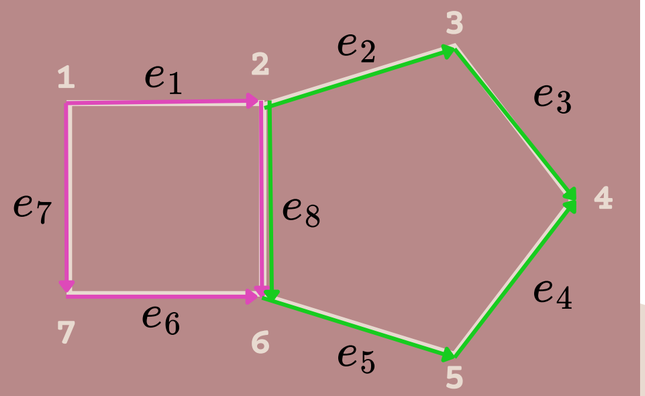


Horton's Method Steps and Warnings

1. Run the **Floyd-Warshall** algorithm to obtain the **shortest distance matrix**
2. Obtain the **shortest paths** from the predecessor matrix: TIME CONSUMING
3. Consider **all possible combinations** of one point and two points connected by a edge: TIME CONSUMING

Overall complexity: $O\left(m^3 n\right)$

Path-Included Distance Matrices



Example

$$\mathbf{P}_e = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{pmatrix} 0 & e_1 & e_1e_2 & e_1e_2e_3 & e_7e_6e_5, e_1e_8e_5 & e_1e_8, e_7e_6 & e_7 \\ e_1 & 0 & e_2 & e_2e_3 & e_8e_5 & e_8 & e_1e_7, e_8e_6 \\ e_2e_1 & e_2 & 0 & e_3 & e_3e_4 & e_2e_8 & e_2e_1e_7, e_2e_8e_6 \\ e_3e_2e_1 & e_3e_2 & e_3 & 0 & e_4 & e_4e_5 & e_4e_5e_6 \\ e_5e_6e_7, e_5e_8e_1 & e_5e_8 & e_4e_3 & e_4 & 0 & e_5 & e_5e_6 \\ e_6e_7, e_8e_1 & e_8 & e_8e_2 & e_5e_4 & e_5 & 0 & e_6 \\ e_7 & e_6e_8, e_7e_1 & e_6e_8e_2, e_7e_1e_2 & e_6e_5e_4 & e_6e_5 & e_6 & 0 \end{pmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

$$\mathbf{P}'_e = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & e_1e_8e_5e_4, e_7e_6e_5e_4 & e_1e_2e_3e_4 & 0 & 0 \\ 0 & 0 & 0 & e_8e_5e_4 & e_2e_3e_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & e_2e_8e_5 & e_3e_4e_5 & e_3e_4e_5e_6 \\ e_4e_5e_6e_7, e_4e_5e_8e_1 & e_4e_5e_8 & 0 & 0 & 0 & e_3e_2e_8 & e_6e_8e_2e_3, e_7e_1e_2e_3 \\ e_4e_3e_2e_1 & e_4e_3e_2 & e_5e_8e_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & e_5e_4e_3 & e_8e_2e_3 & 0 & 0 & 0 \\ 0 & 0 & e_6e_5e_4e_3 & e_3e_2e_1e_7, e_3e_2e_8e_6 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix}$$

Note that:

1. $\{e_1, e_8\} \cap \{e_7, e_6\} = \varnothing \wedge \{e_2, e_3\} \cap \{e_4, e_5, e_8\} = \varnothing$
2. both shortest paths => even ring,otherwise odd

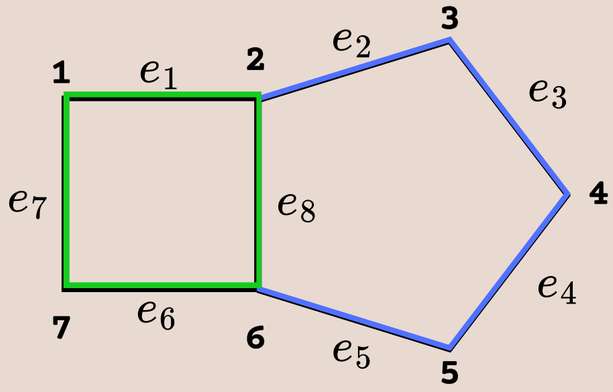
Procedure P

1. Identify the **set of all cycles C**
2. **Order** the members of C by **ascending** edge count
3. Create an **empty set that will contain the SSSR**
4. **Add** the **first** member C to C_{SSSR}
5. **For each remaining** cycle in C,**add** it only if it is **not the sum of cycles already added to** C_{SSSR} **(XOR)**

Example

$$C = \{\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\}\ \{e_7,e_6,e_5,e_4,e_3,e_2,e_1\}\ \{e_1,e_8,e_6,e_7\}\ \{e_2,e_3,e_4,e_5,e_8\}\ \{e_8,e_5,e_4,e_3,e_2\}\ \{e_1,e_7,e_6,e_8\}\ \{e_3,e_4,e_5,e_8,e_2\}\ \{e_2,e_8,e_3,e_4,e_5\}\ \{e_2,e_1,e_7,e_6,e_5,e_4,e_3\}\ \{e_3,e_2,e_1,e_7,e_6,e_5,e_4\}\ \{e_3,e_2,e_8,e_5,e_4\}\ \{e_4,e_5,e_8,e_2,e_3\}\ \{e_4,e_5,e_6,e_3,e_2,e_1,e_7\}\ \{e_5,e_6,e_7,e_1,e_2,e_3,e_4\}\ \{e_5,e_8,e_2,e_3,e_4\}\ \{e_4,e_3,e_2,e_8,e_5\}\ \{e_6,e_7,e_1,e_8\}\ \{e_8,e_2,e_3,e_4,e_5\}\ \{e_5,e_4,e_3,e_2,e_8\}\ \{e_6,e_8,e_7,e_1\}\ \{e_7,e_1,e_2,e_3,e_4,e_5,e_6\}\ \{e_6,e_5,e_4,e_7,e_1,e_2,e_3\}\}$$

$$C_{SSSR} = \varnothing$$



PID Method vs Horton's Method

Space

Time

PID method		Horton's method	
Distance matrix calculation	$O(n^3)$	Distance matrix calculation	$O(n^3)$
Ring candidate search	$O(n^2)$	Ring search	$O(mn^2)$
sort $(\{C\})$	$O(a' \log a')$	sort $(\{C\})$	$O(a \log a)$
Finding SSSR	$O(a'k)$	Finding MCB (SSSR)	$O(m^3 n)$

Observation:

If: $m \geqslant n$

$$O(m^3 n) > O(n^4) > O(n^3)$$

Legend:

n = number of vertices
m = number of edges
a = number of cycles
a' = ring candidates
k = number of min paths from i to j

“Those who cannot remember the past are condemned to repeat it”

Jorge Agustín Nicolás Ruiz de Santayana y Borrás, The Life of Reason (1905)

Bibliography and Websites

[1] Introduction to Algorithms, by T.H. Cormen, C. e. Leiserson, R.L. Rivest, C. Stein. Chapters 4,14,22,23. MIT Press (4th ed.), 2022.

[2] Algorithms, by J. Erickson. Chapter 3. Available online, 2019.

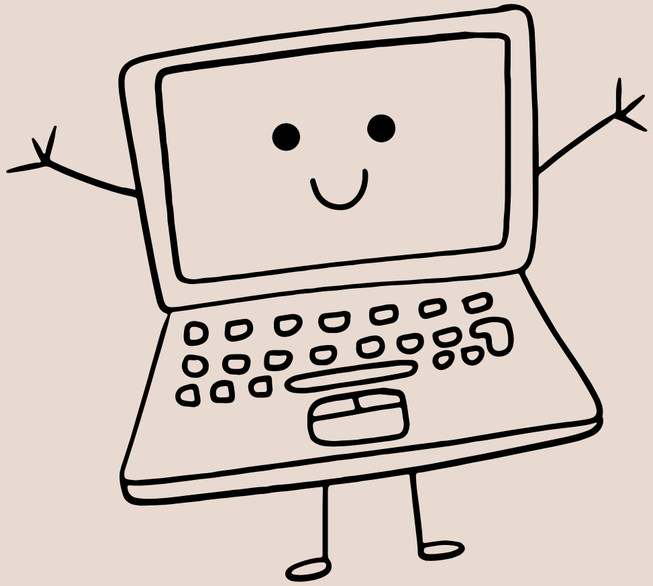
[3] A Robust Method for Searching the Smallest Set of Smallest Rings with a Path-included Distance Matrix, by C. J. Lee, Y.-M. Kang, K.-H. Cho, and K. T. No. Proc. Natl. Acad. Sci. U.S.A. vol 106 (41), 2009.

[4] The Floyd-Warshall Algorithm From Graph Theory, Applied to Parsing Molecular Structures, by Luciano Abriata. Blogpost in Towards Data Science, August 2024.

[5] A Smallest Set of Smallest Rings, by Richard L. Apodaca. Blogpost in Depth-First, August 2020.

[6] ChemSpider(Chemical Structure Database), Royal Society of Chemistry.

THANK YOU FOR YOUR ATTENTION!



Sara Giuriati