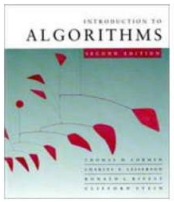


Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$

DAG Shortest Paths

- Bellman-Ford takes O(VE) time.

DAG Shortest Paths

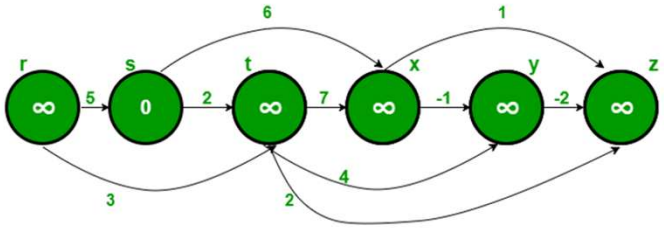
- Bellman-Ford takes O(VE) time.
- Use DAG properties!

```
DAG-Shortest-Paths(G, s)
1.  L = Topological_Sort(G)
2.  for u in V: dist(u) = INF;
3.  dist(s) = 0
4.  for u from s to L.end():
5.      for each vertex v ∈ Adj[u]
6.          if dist(v) > dist(u) + weight(u, v)
7.              dist(v) = dist(u) + weight(u, v)
```

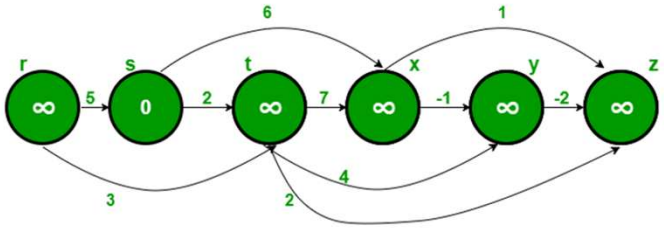
relaxation step

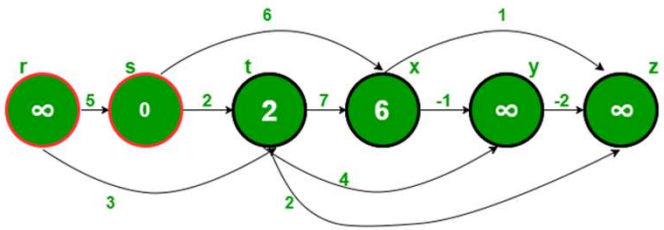
Time: O(V+E)

DAG Shortest Paths

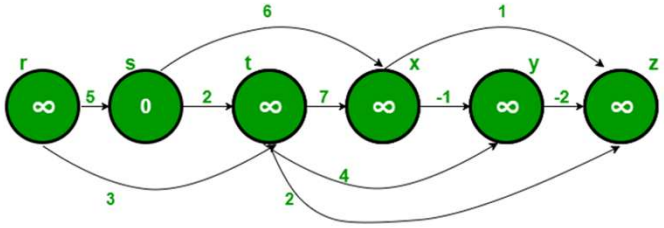


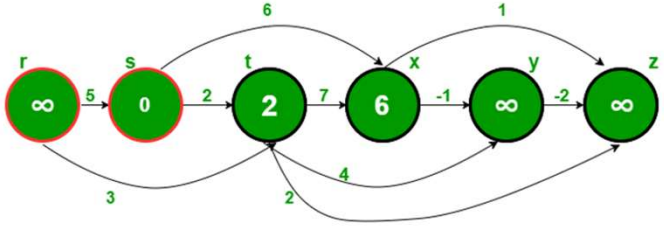
DAG Shortest Paths

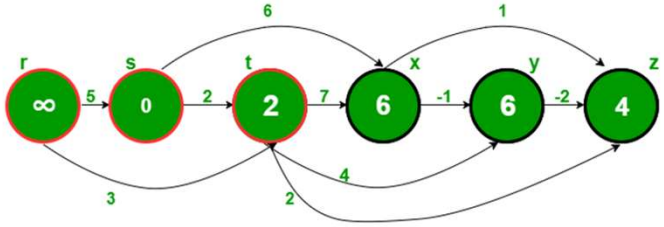




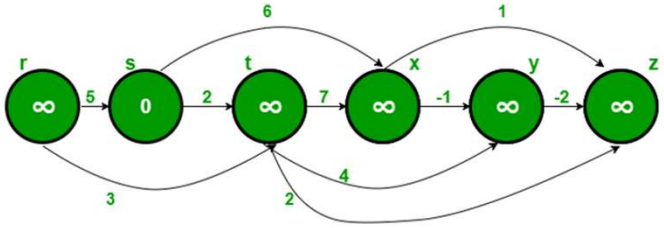
DAG Shortest Paths

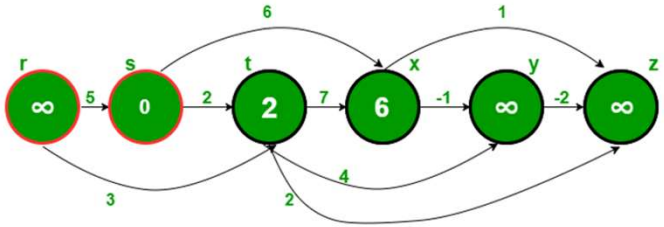


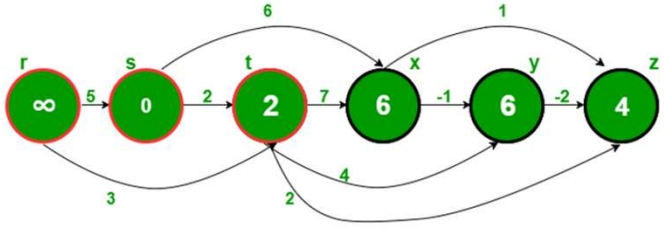


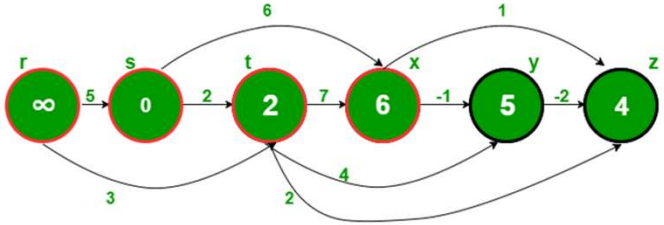


DAG Shortest Paths

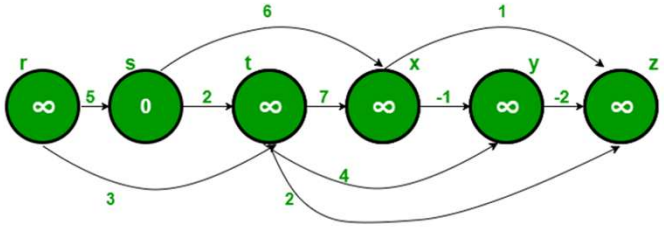


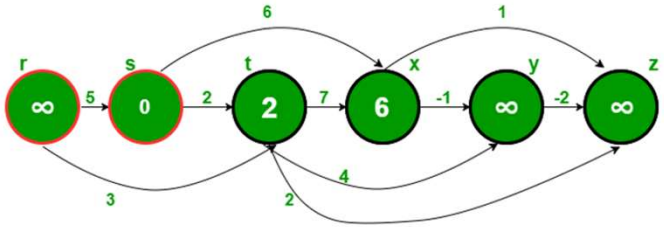


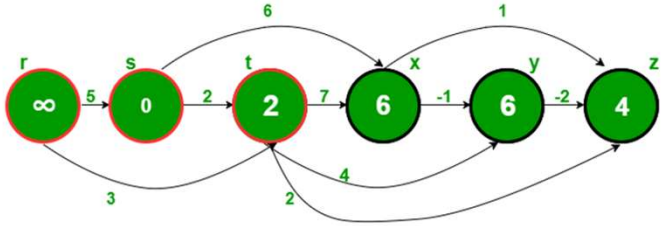


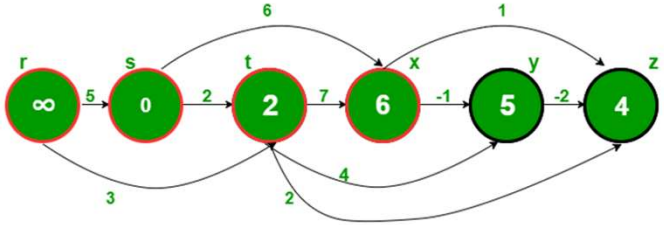


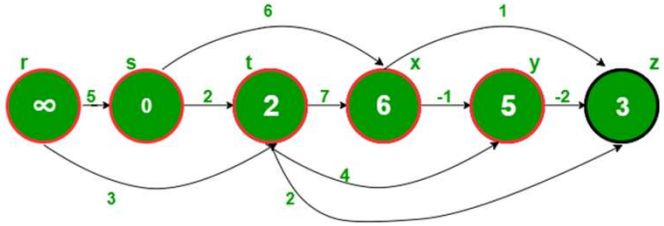
DAG Shortest Paths



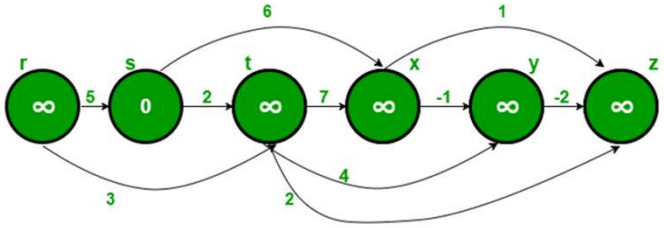


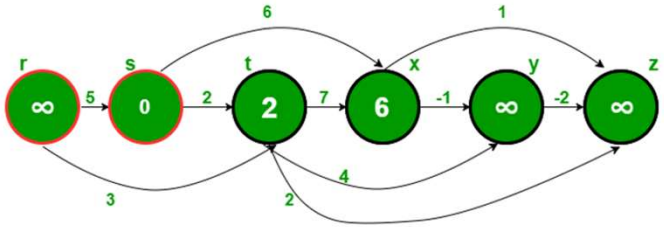


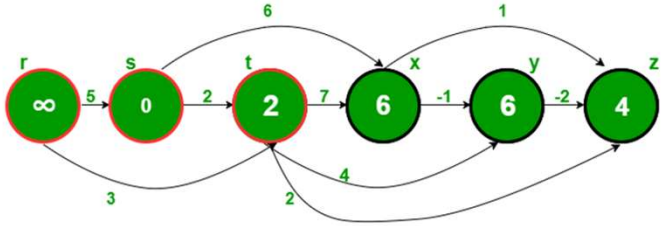


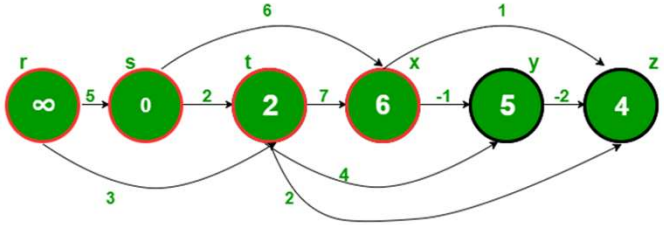


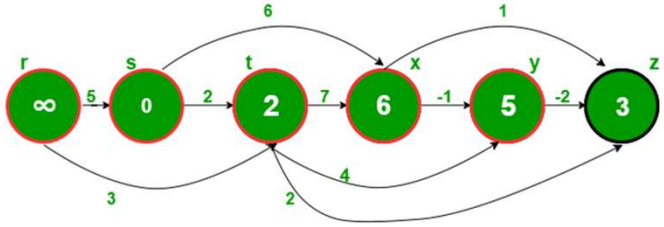
DAG Shortest Paths

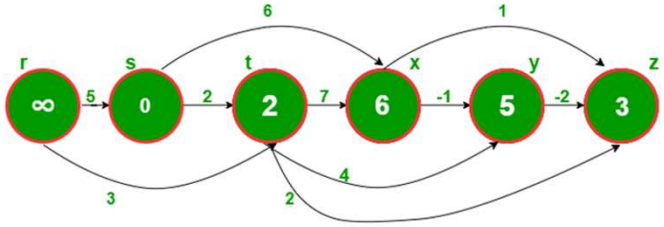


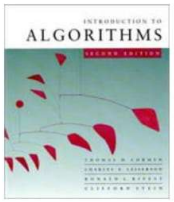








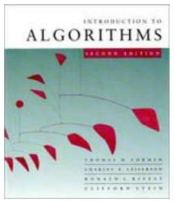




Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

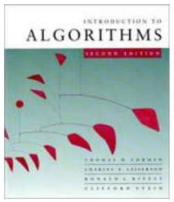


Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

All-pairs shortest paths



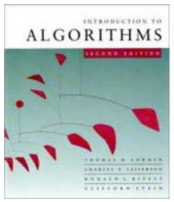
Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times



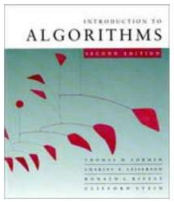
Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(VE + V^2 \lg V)$



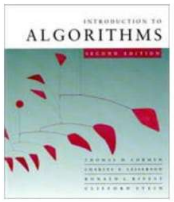
Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

All-pairs shortest paths

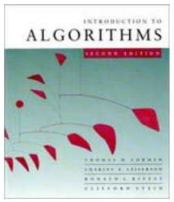
- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(VE + V^2 \lg V)$
- General



All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.



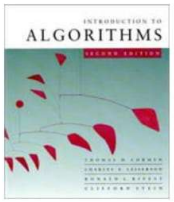
All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

IDEA:

- Run Bellman-Ford once from each vertex.



All-pairs shortest paths

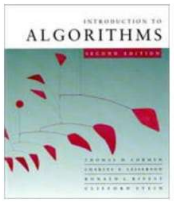
Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

IDEA:

- Run Bellman-Ford once from each vertex.
- Time $= O(V^2E)$.
- Dense graph (n^2 edges) $\Rightarrow \Theta(n^4)$ time in the worst case.

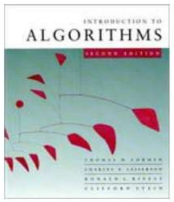
Good first try!



Dynamic programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.



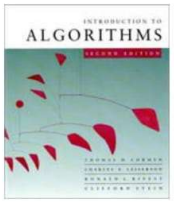
Dynamic programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$



Dynamic programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

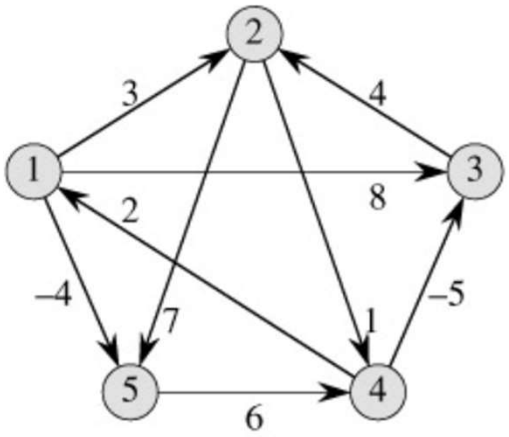
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

and for $m = 1, 2, \ldots, n-1$,

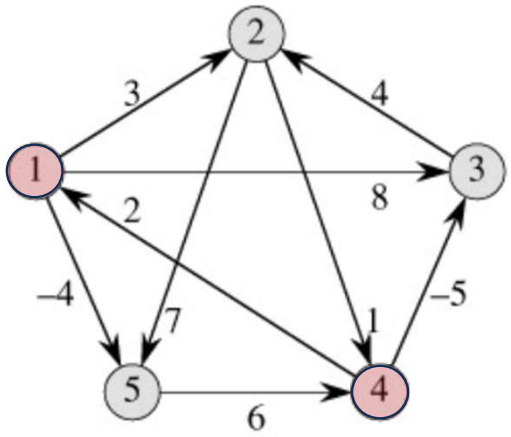
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$



$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

m=1

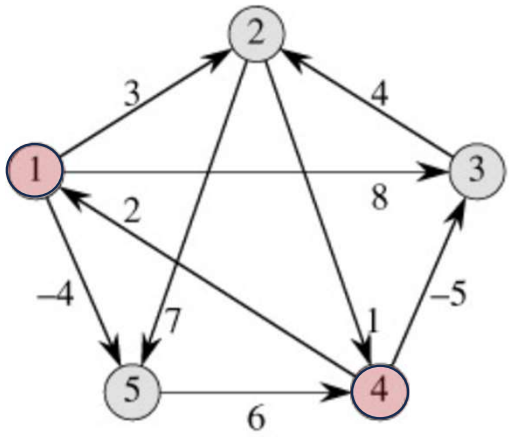
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.



$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

m=2

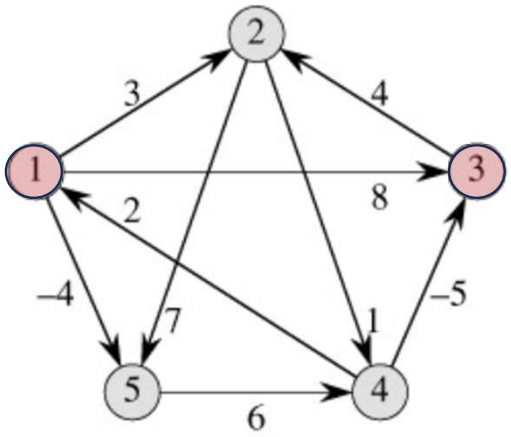
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.



$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

m=2

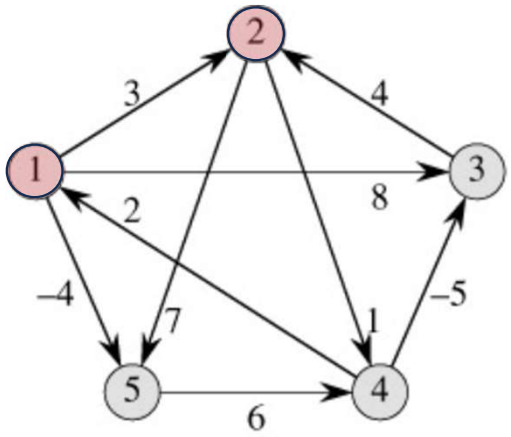
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.



$$\begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

m=3

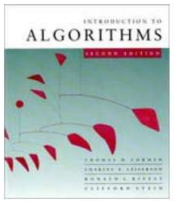
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.



$$\begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

m=4

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

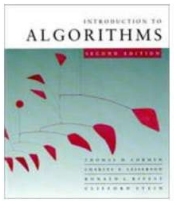


Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \; .$$

Time = $\Theta(n^3)$ using the standard algorithm.



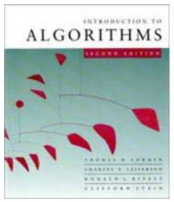
Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \, .$$

Time = $\Theta(n^3)$ using the standard algorithm.

What if we map "+" $\rightarrow$ "min" and "$\cdot$" $\rightarrow$ "+"?



Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

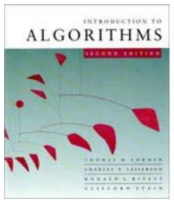
Time $= \Theta(n^3)$ using the standard algorithm.

What if we map "+" $\rightarrow$ "min" and "$\cdot$" $\rightarrow$ "+"?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}.$$

Thus, $D^{(m)} = D^{(m-1)}$ "$\times$" A.

Identity matrix $= \mathrm{I} = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)})$.



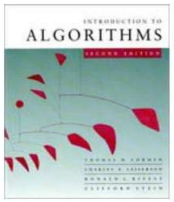
Matrix multiplication (continued)

The (min, +) multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

Consequently, we can compute

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot A &&= A^1 \\ D^{(2)} &= D^{(1)} \cdot A &&= A^2 \\ &\vdots && \vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot A &&= A^{n-1}\,, \end{aligned}$$

yielding $D^{(n-1)} = (\delta(i,j))$.



Matrix multiplication (continued)

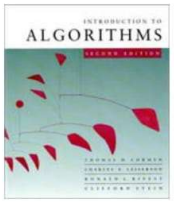
The (min, +) multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

Consequently, we can compute

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot A &&= A^1 \\ D^{(2)} &= D^{(1)} \cdot A &&= A^2 \\ &\vdots && \vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot A &&= A^{n-1}\,, \end{aligned}$$

yielding $D^{(n-1)} = (\delta(i,j))$.

Time $= \Theta(n \cdot n^3) = \Theta(n^4)$. No better than $n \times$ B-F.



Improved matrix multiplication algorithm

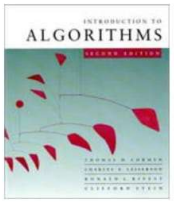
Repeated squaring: $A^{2k} = A^k \times A^k$.

Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n) \text{ squarings}}$.

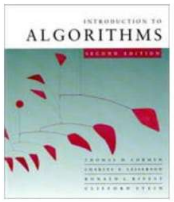
Note: $A^{n-1} = A^n = A^{n+1} = \cdots$.

Time $= \Theta(n^3 \lg n)$.

To detect negative-weight cycles, check the diagonal for negative values in $O(n)$ additional time.



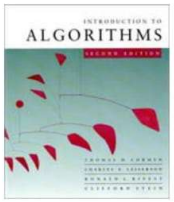
Pseudocode for Floyd-Warshall



Pseudocode for Floyd-Warshall

for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do if $c_{ij} > c_{ik} + c_{kj}$
        then $c_{ij} \leftarrow c_{ik} + c_{kj}$

} ***relaxation***

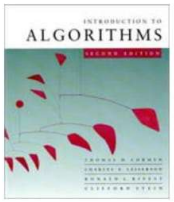


Pseudocode for Floyd-Warshall

for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do if $c_{ij} > c_{ik} + c_{kj}$
        then $c_{ij} \leftarrow c_{ik} + c_{kj}$ } ***relaxation***

Notes:

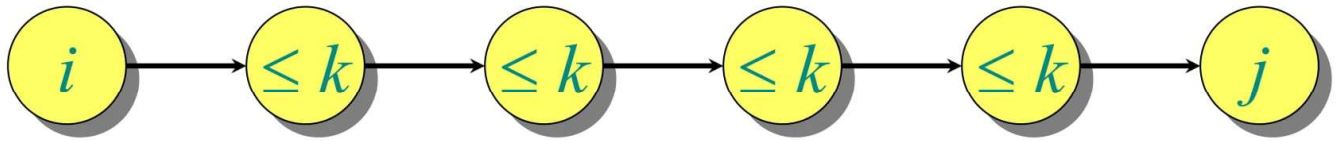
- Okay to omit superscripts, since extra relaxations can't hurt.
- Runs in $\Theta(n^3)$ time.
- Simple to code.
- Efficient in practice.



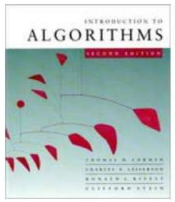
Floyd-Warshall algorithm

Also dynamic programming, but faster!

Define $c_{ij}^{(k)} =$ weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.

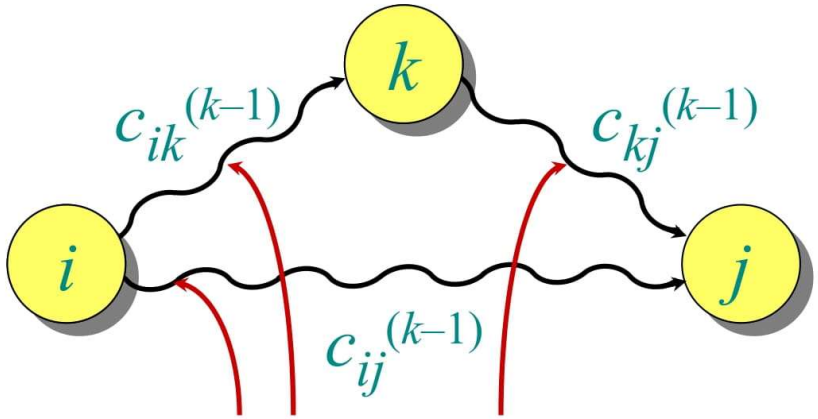


Thus, $\delta(i, j) = c_{ij}^{(n)}$. Also, $c_{ij}^{(0)} = a_{ij}$.

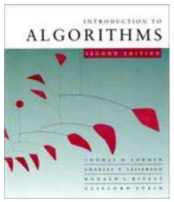


Floyd-Warshall recurrence

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



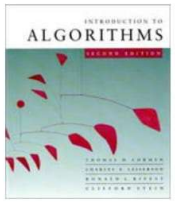
intermediate vertices in $\{1, 2, \ldots, k\}$



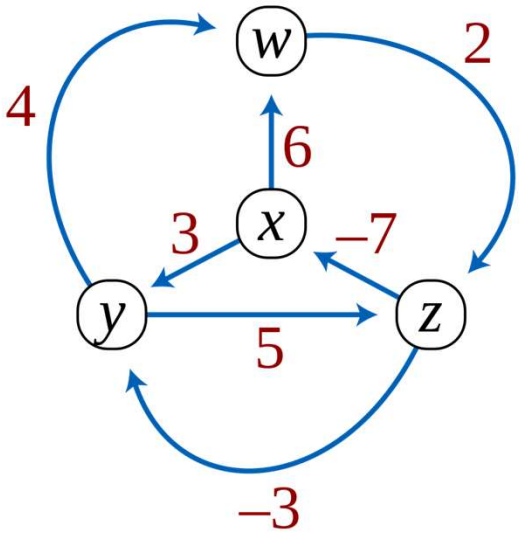
Johnson's algorithm

1. Find a function $h : V \to \mathbb{R}$ such that $w_h(u, v) \geq 0$ for all $(u, v) \in E$ by using Bellman-Ford to solve the difference constraints $h(v) - h(u) \leq w(u, v)$, or determine that a negative-weight cycle exists.
   - Time = $O(VE)$.
2. Run Dijkstra's algorithm using w_h from each vertex $u \in V$ to compute $\delta_h(u, v)$ for all $v \in V$.
   - Time = $O(VE + V^2 \lg V)$.
3. For each $(u, v) \in V \times V$, compute
   $$\delta(u, v) = \delta_h(u, v) - h(u) + h(v) .$$
   - Time = $O(V^2)$.

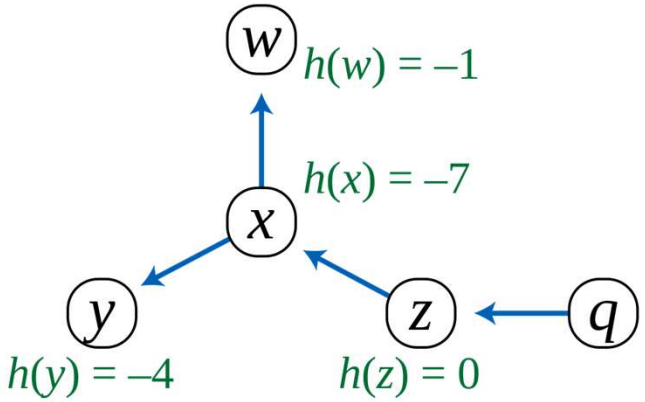
Total time = $O(VE + V^2 \lg V)$.



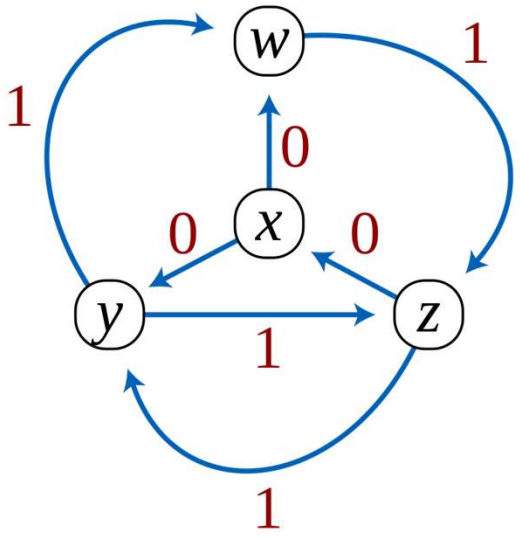
Johnson's algorithm



original graph
with negative edges



shortest path tree
found by Bellman-Ford



reweighted graph with
no negative edges

אלגוריתם קירוב

אלגוריתם קירוב - אינטואיציה

אלגוריתם שמוצא פתרון שאינו בהכרח פתרון אופטימלי לבעיה נתונה, אלא פתרון שקרוב לפתרון אופטימלי.

אלגוריתם קירוב - אינטואיציה

אלגוריתם שמוצא פתרון שאינו בהכרח פתרון אופטימלי
לבעיה נתונה, אלא פתרון שקרוב לפתרון אופטימלי.

למה להסתפק רק בקירוב ??

אלגוריתם קירוב - אינטואיציה

אלגוריתם שמוצא פתרון שאינו בהכרח פתרון אופטימלי לבעיה נתונה, אלא פתרון שקרוב לפתרון אופטימלי.

מציאת פתרון אופטימלי היא בלתי מעשית

NP

אלגוריתם קירוב

אלגוריתם שמוצא פתרון שאינו בהכרח פתרון אופטימלי לבעיה נתונה, אלא פתרון שקרוב לפתרון אופטימלי.

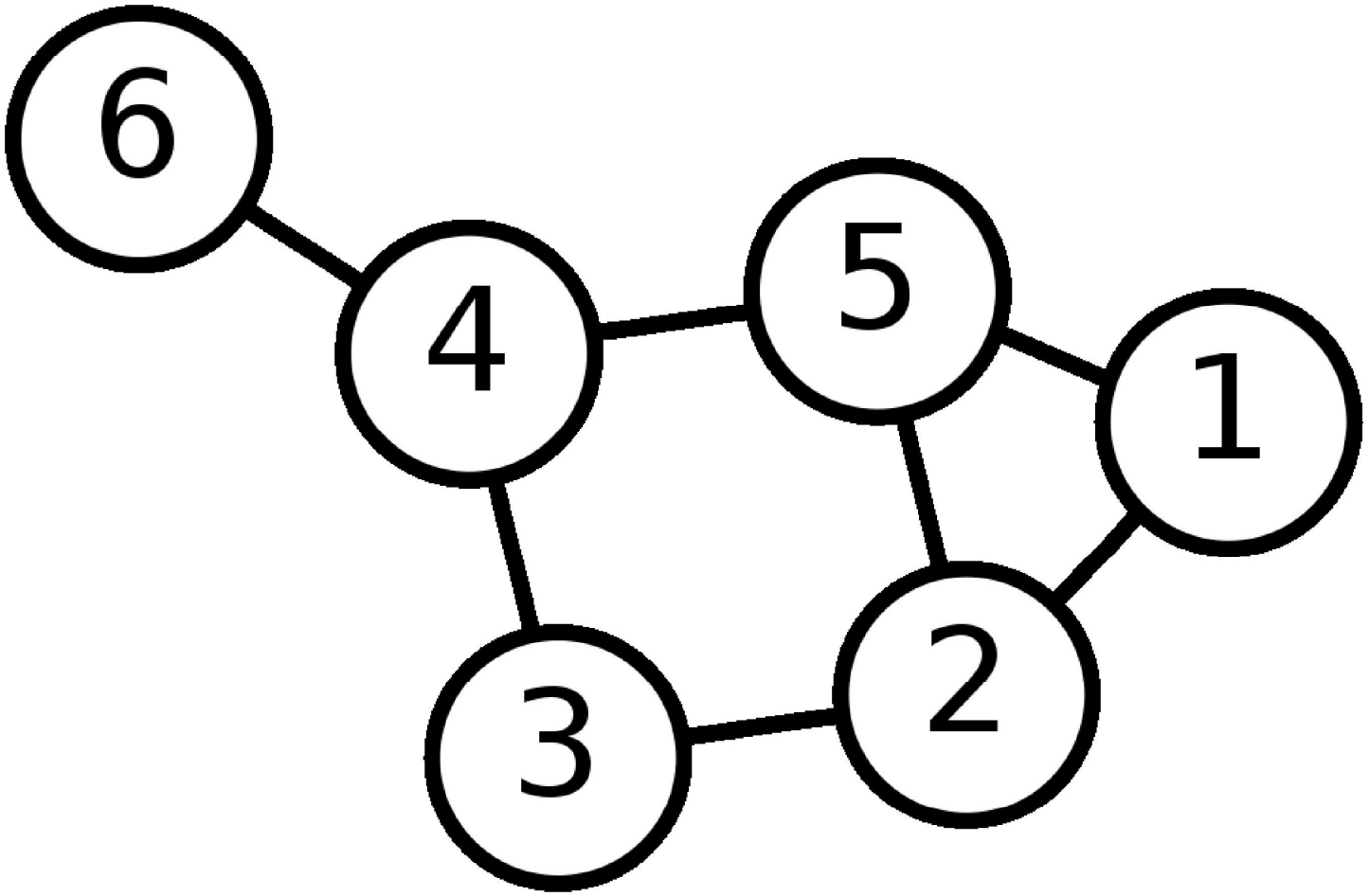
מודדים את הקירוב של אלגוריתם בהתאם ליחס בין הפתרון שנמצא על ידי האלגוריתם לבין הפתרון האופטימלי.

אלגוריתם קירוב – בעיות אופטימיזציה

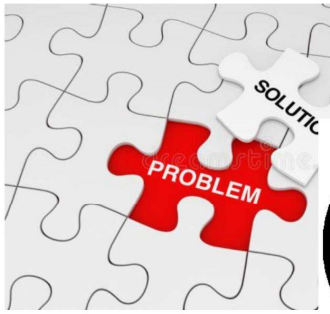
אלגוריתם קירוב **A** הוא בעל יחס קירוב **ρ(n)**
אם עבור כל קלט בגודל **n**,
היחס בין הגודל **C** של הפתרון שמפיק **A**
לגודל הפתרון האופטימלי **C***
אינו עולה על **ρ(n)**

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

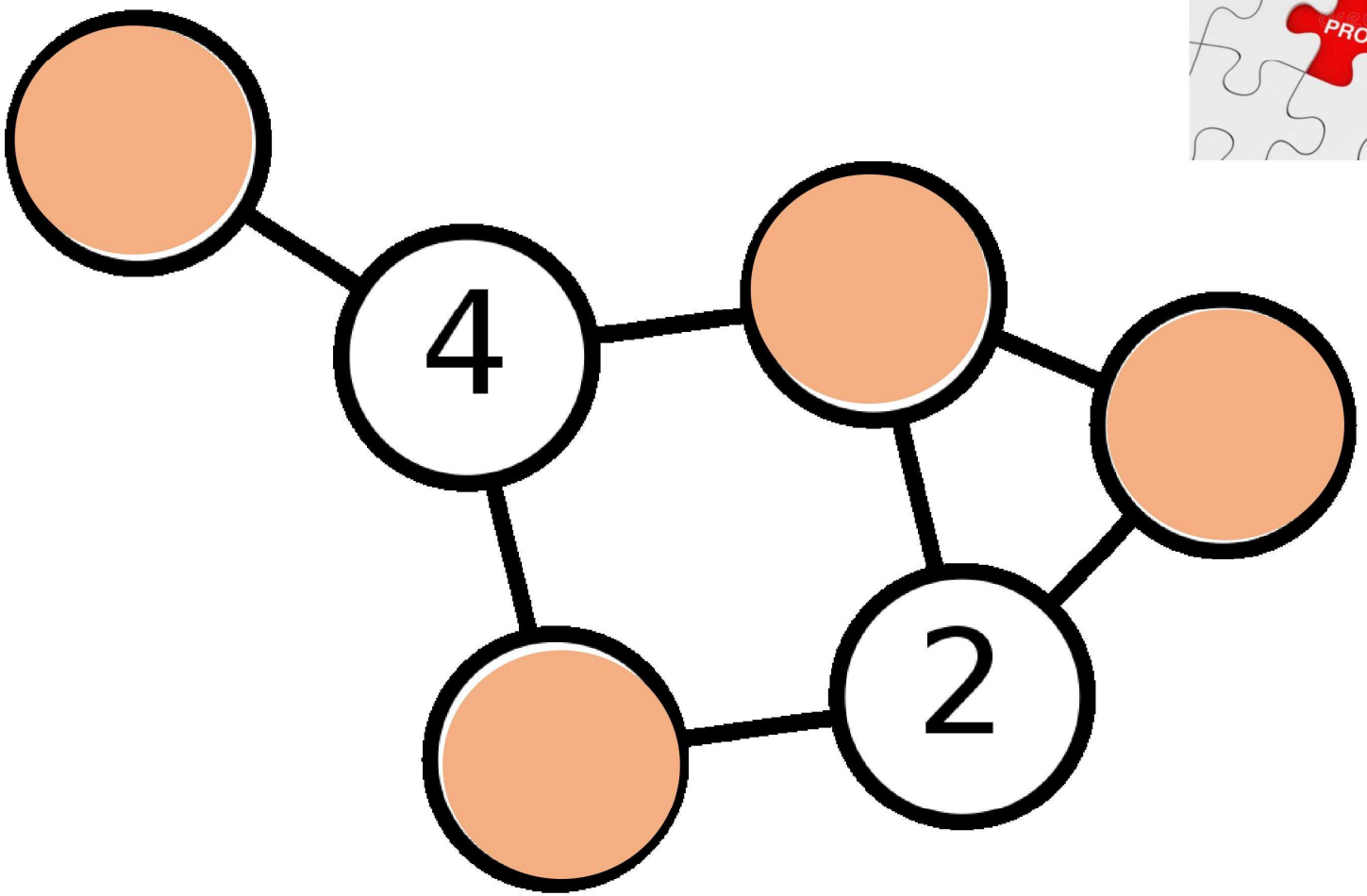
ריחוי הודהודייח



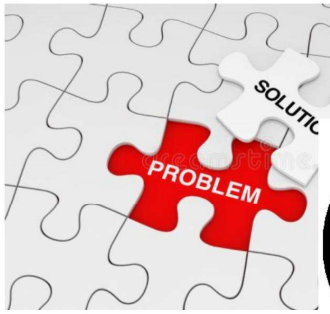
ריחוי הודהודיות



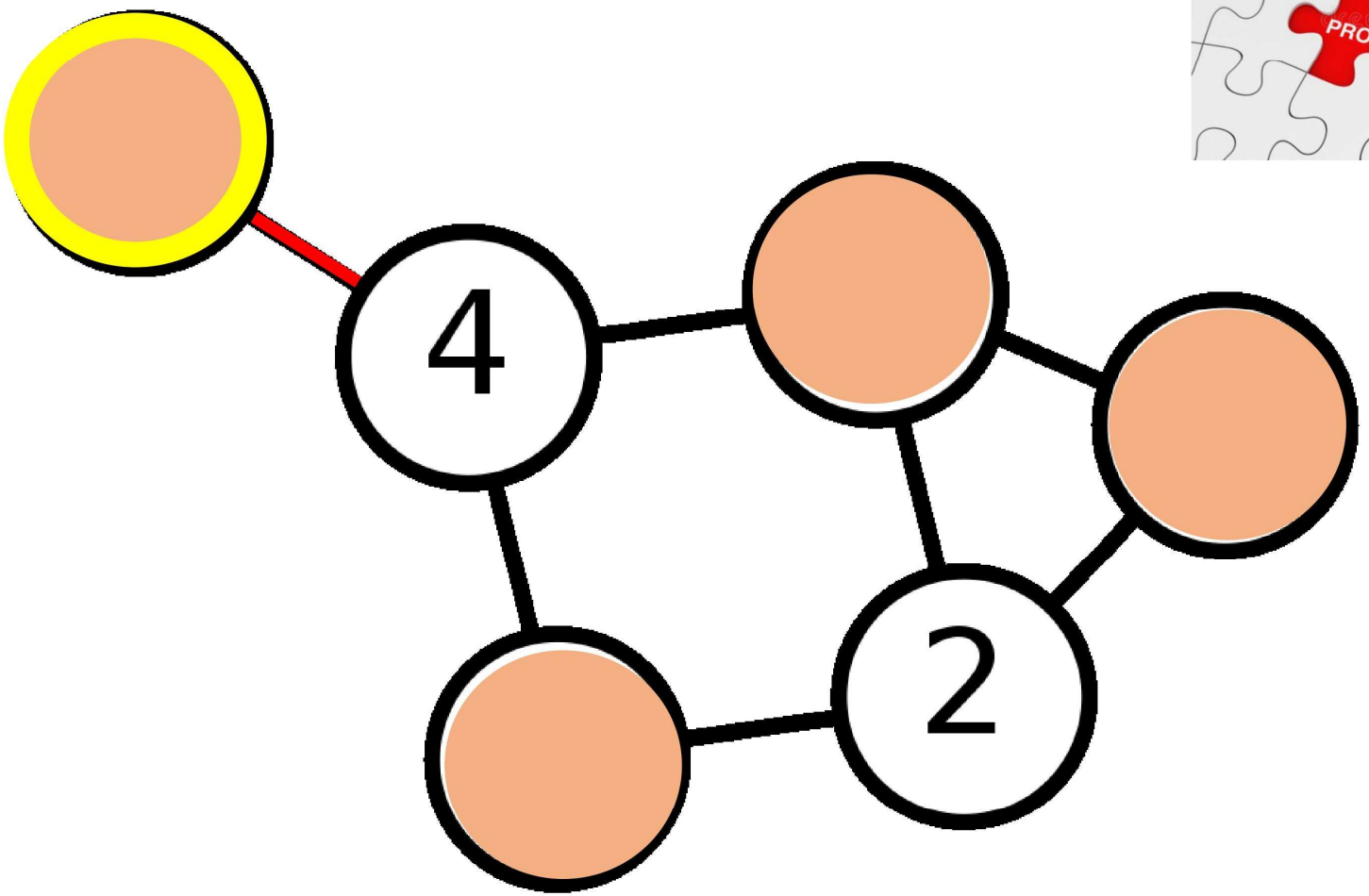
1



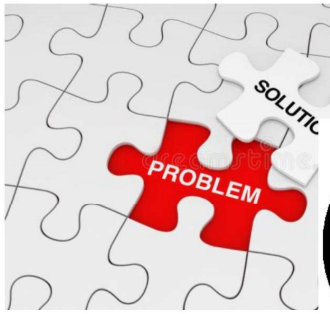
ריחוי הודהודיות



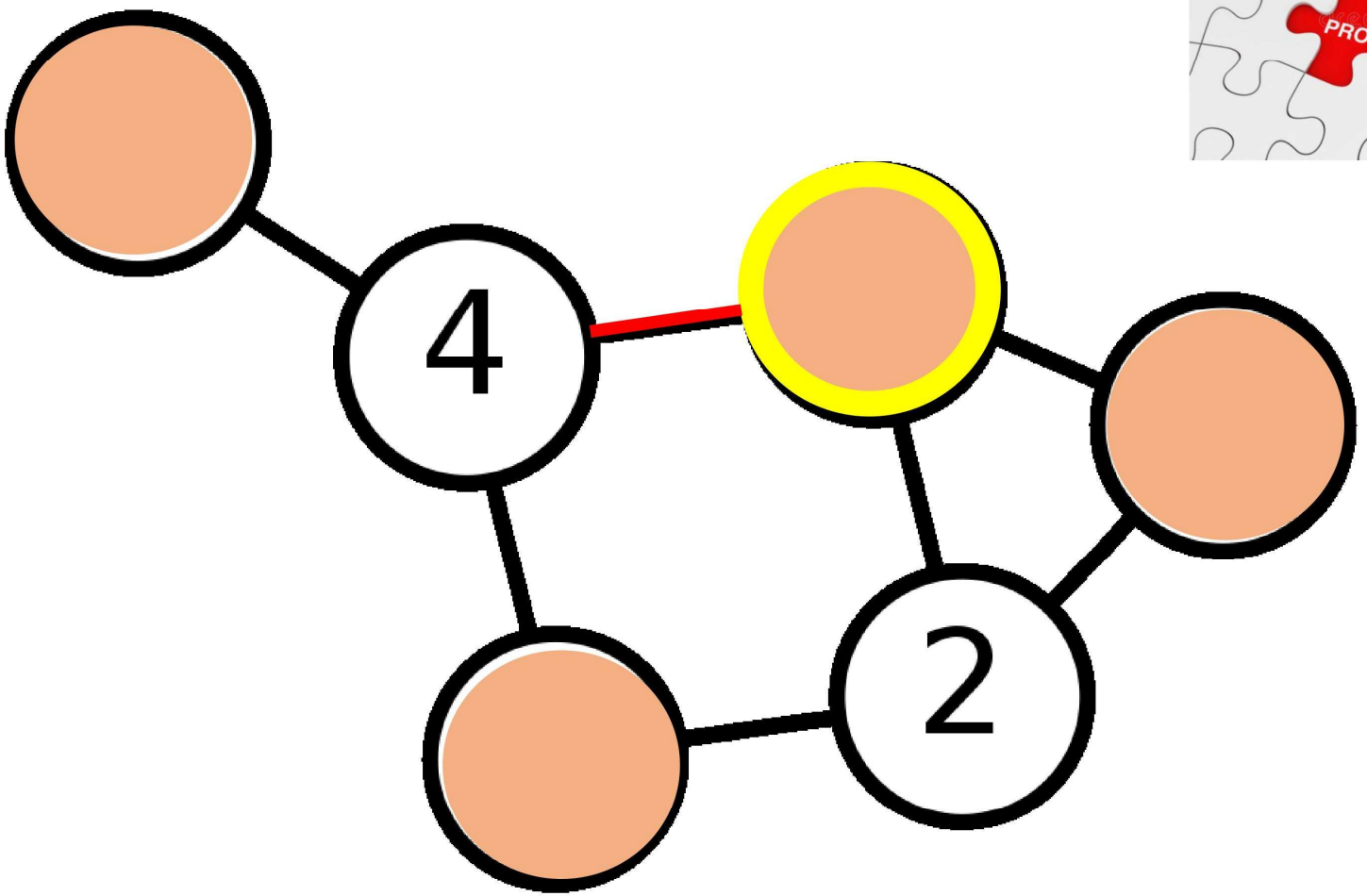
1



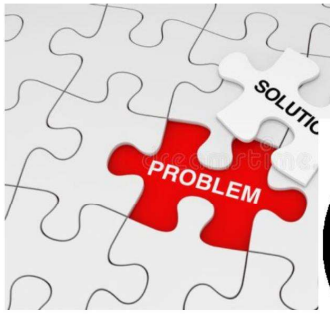
ריחוי הדדי



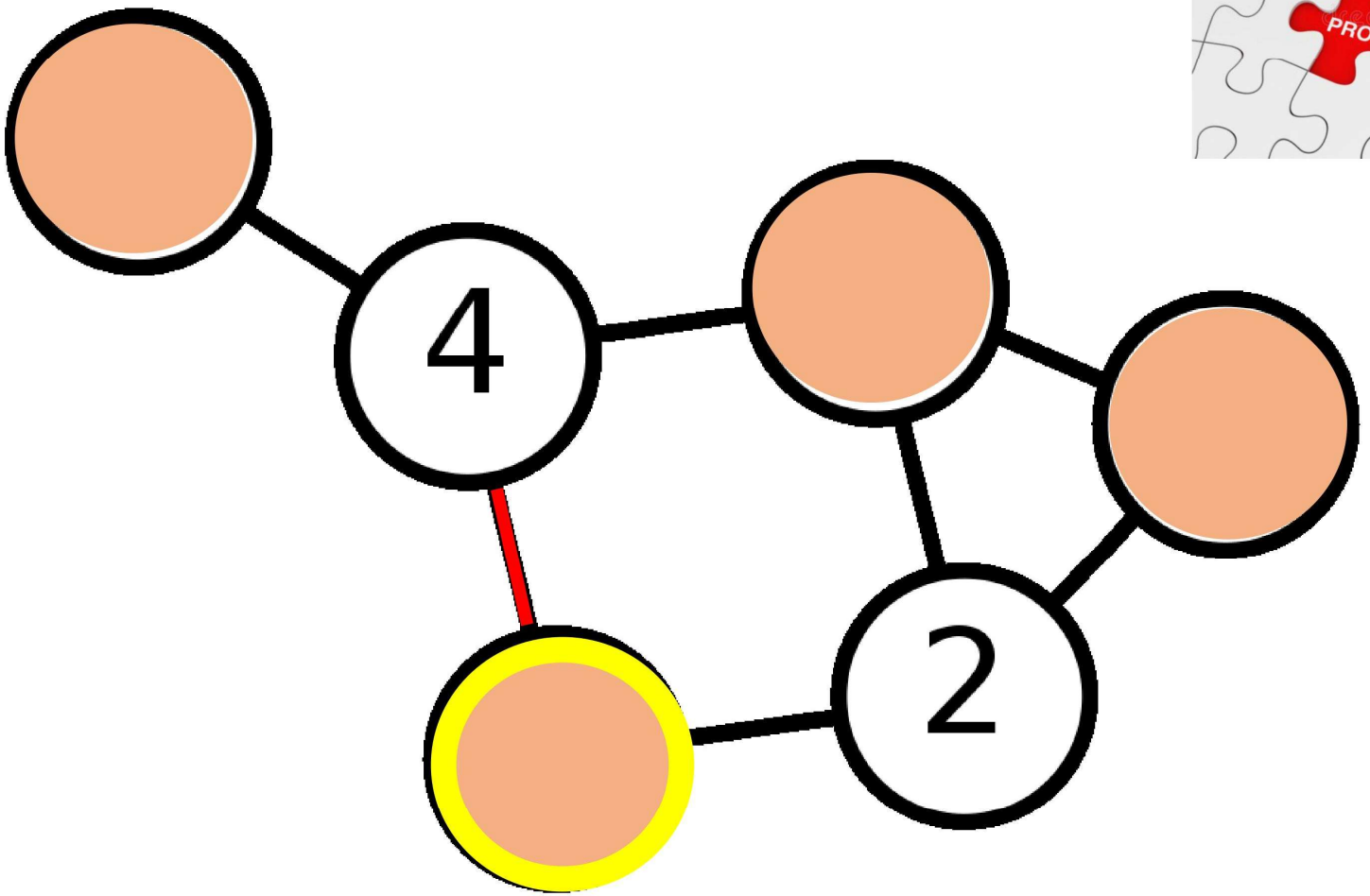
1



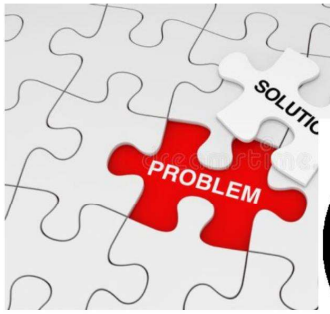
ריחוי הדדי



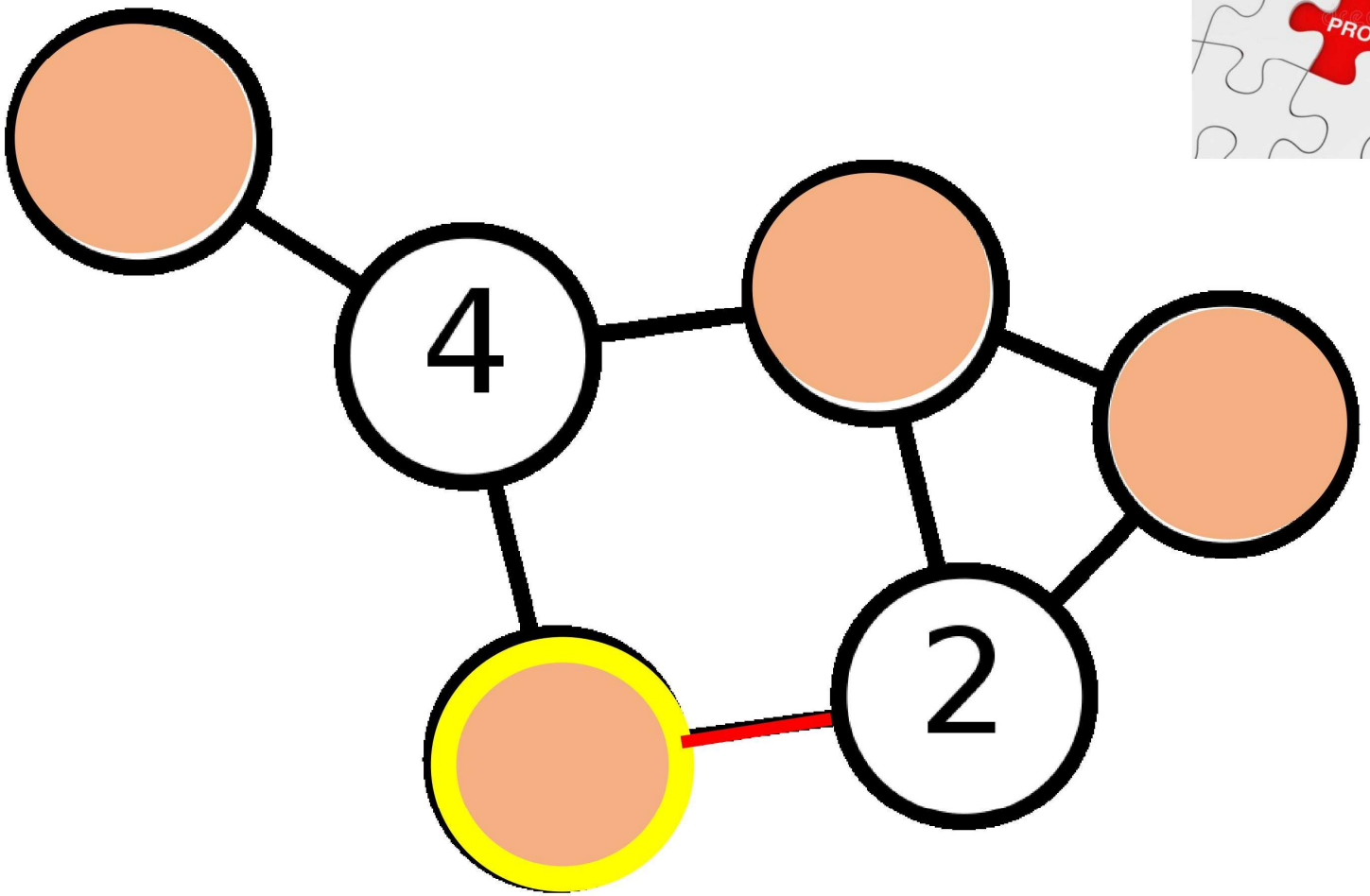
1



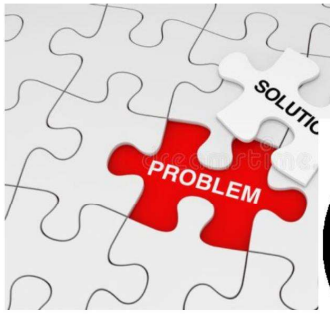
ריחוי הדדי



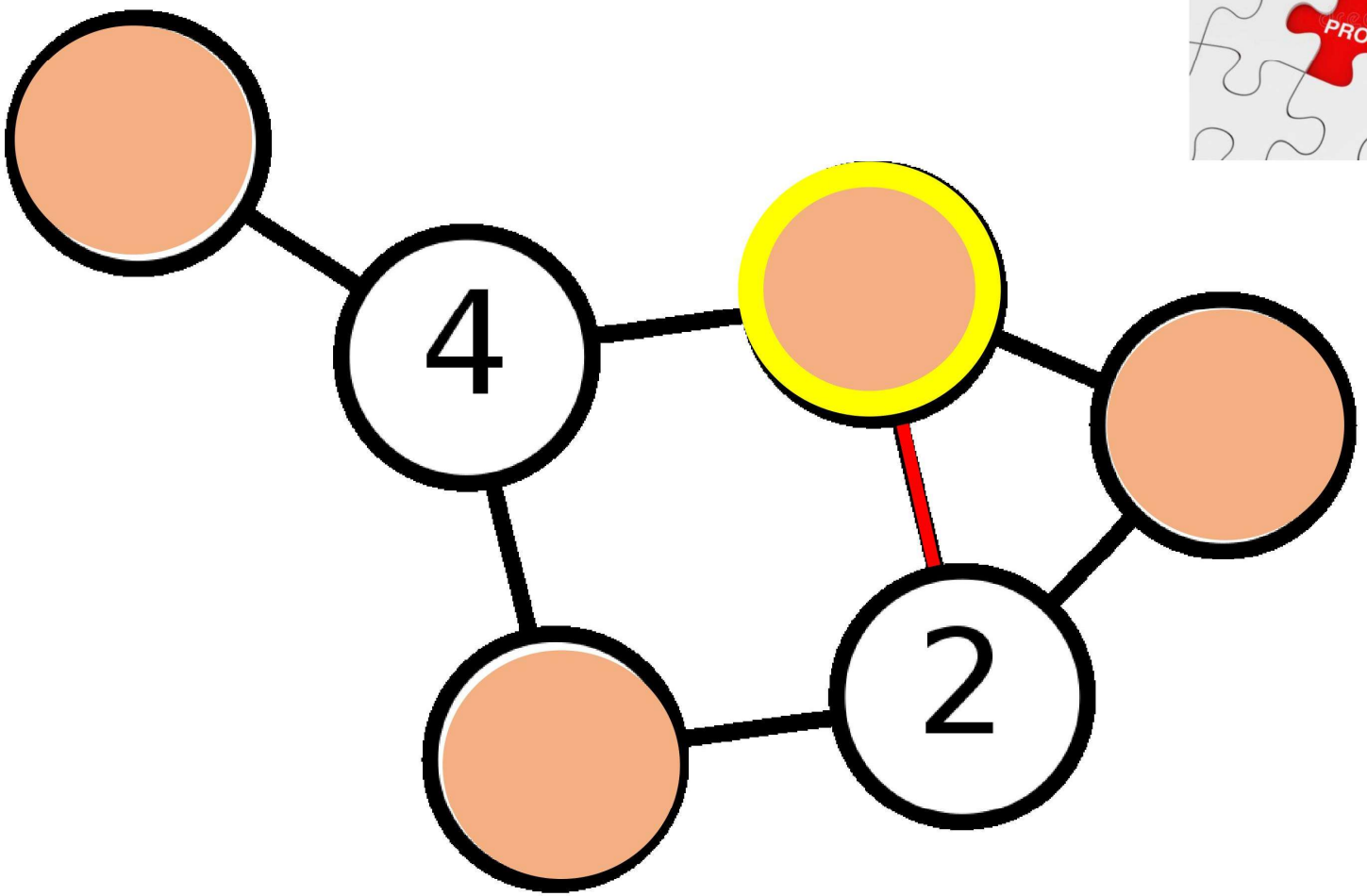
1



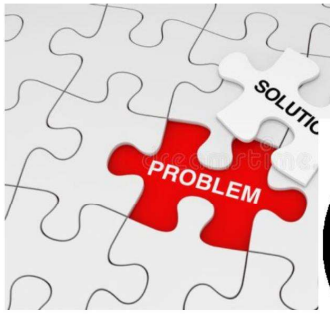
ריחוי הדדי



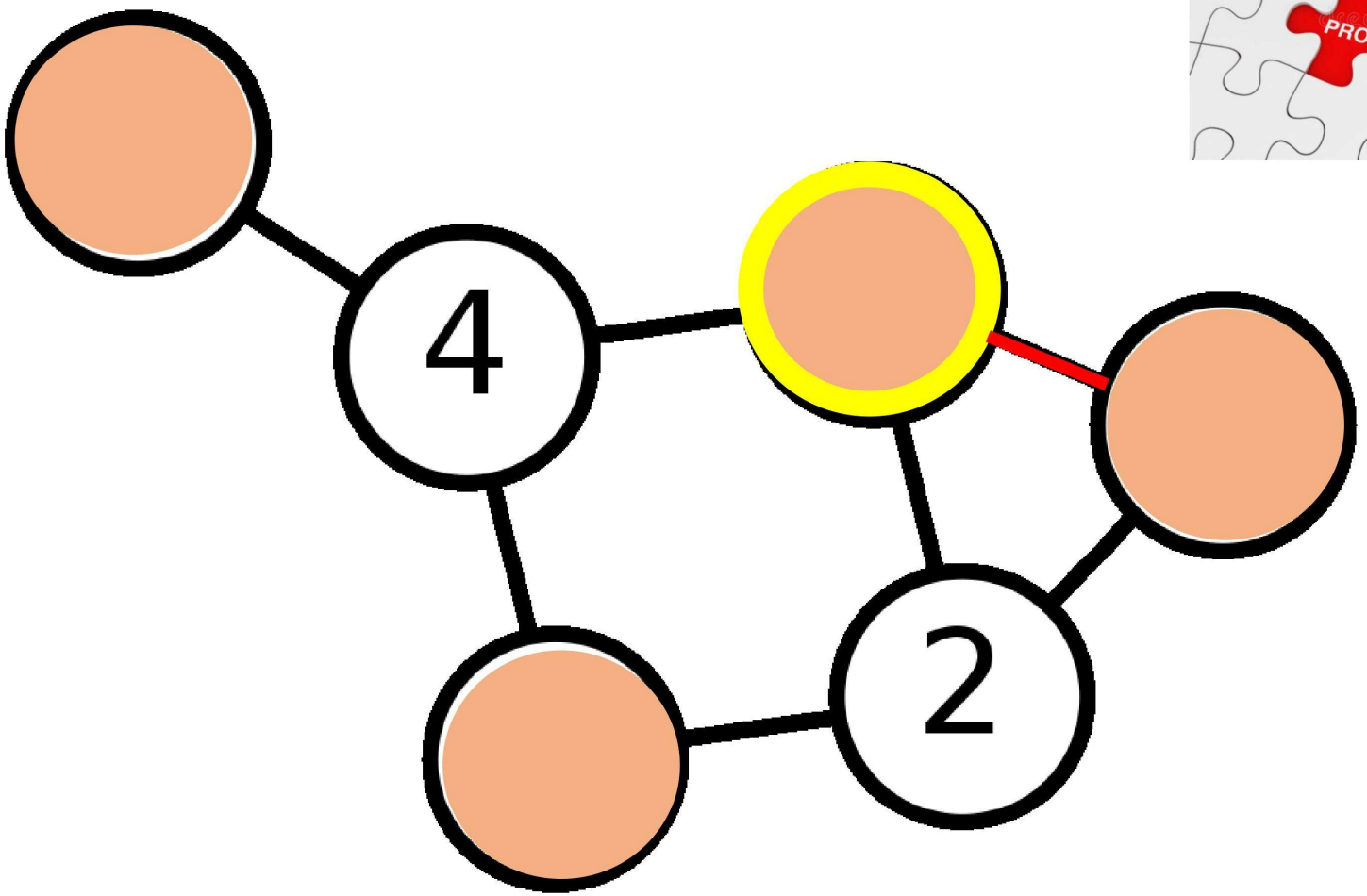
1

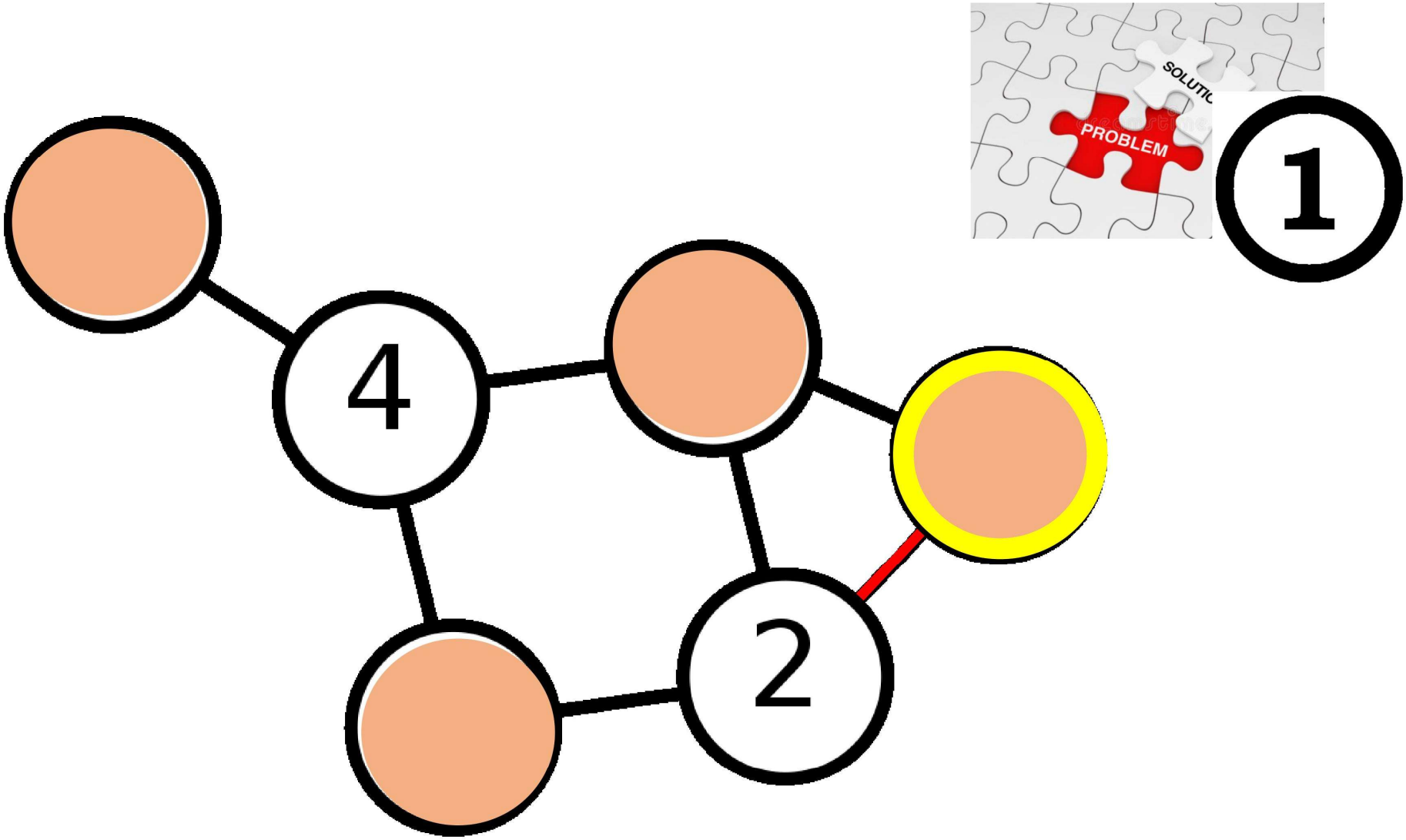


ריחוי הודהודיות



1




PROBLEM
1
4
2

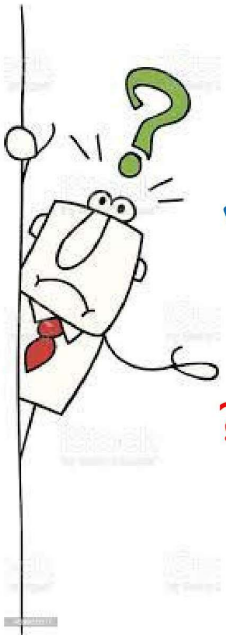
כיסוי קודקודיים

כיסוי על ידי קדקודים **(vertex cover)** של גרף בלתי מכוון $G=(V,E)$ הוא

תת קבוצה $V' \subseteq V$ כך שאם $(u,v) \in E$, אזי $u \in V'$ או $v \in V'$ (או שניהם).

כיסוי קודקודיים

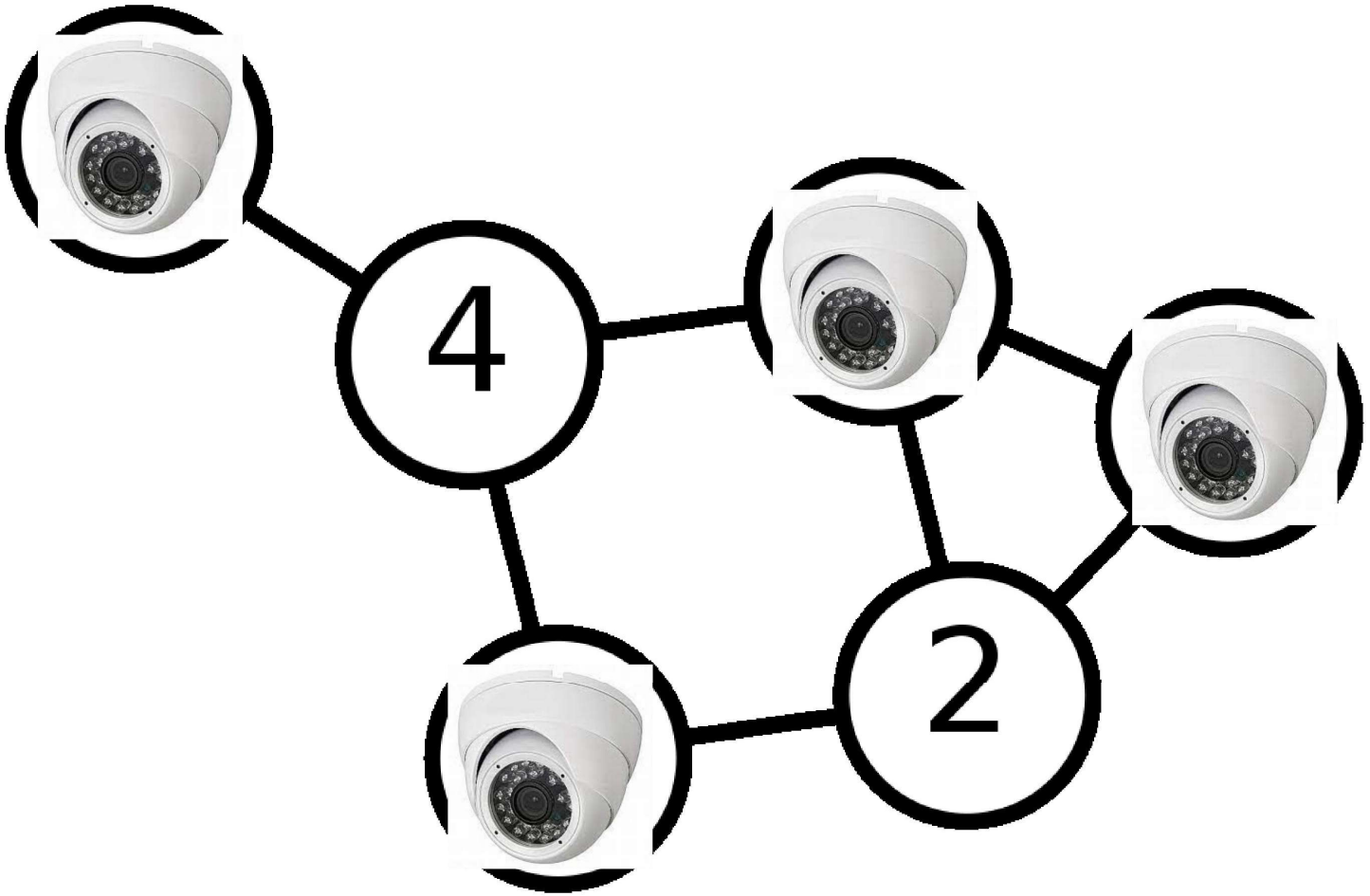
כיסוי על ידי קדקודים (vertex cover) של גרף בלתי מכוון $G=(V,E)$ הוא



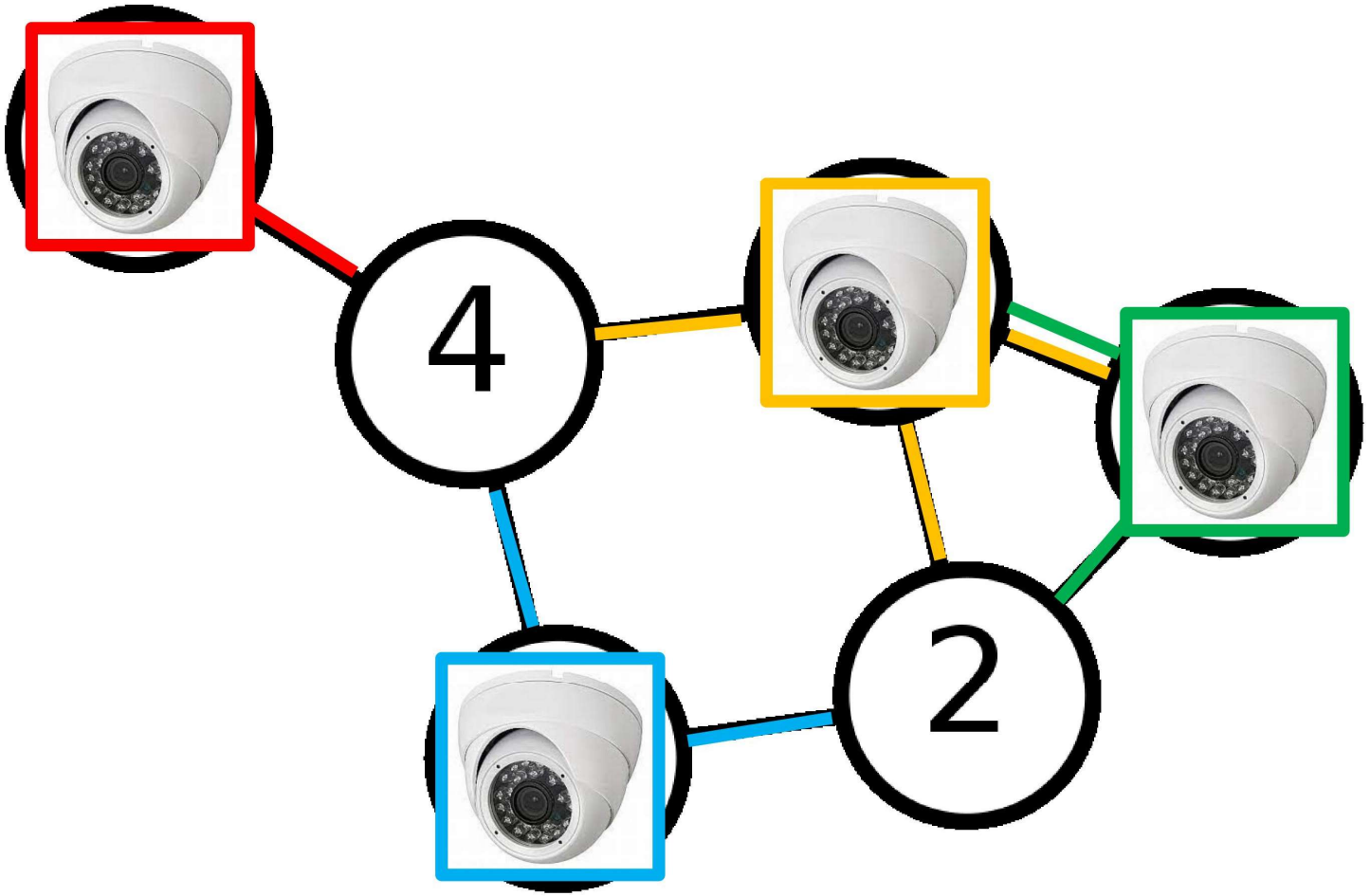
תת קבוצה $V' \subseteq V$ כך שאם $(u,v) \in E$, אזי $u \in V'$ או $v \in V'$ (או שניהם).

למה הבעיה מעניינת?
מה יכולה לייצג בחיים האמתיים?

תיאור חטלחוח ארכוחה רפחחי חחדרוו



תיאור חצלחות ארכיטקטוני חדרון



בעיית כיסוי קודקודיים

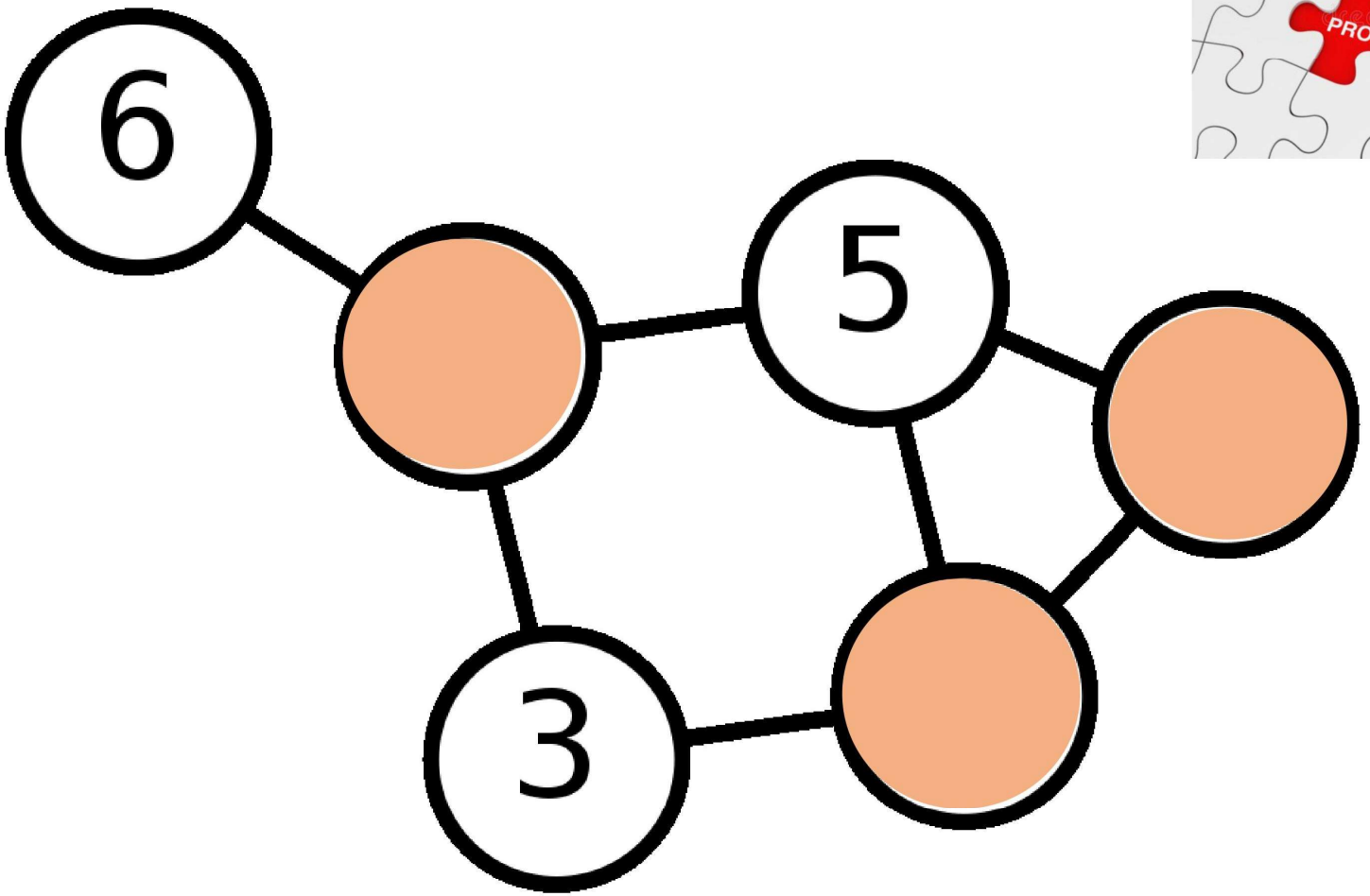
כיסוי על ידי קדקודים (vertex cover) של גרף בלתי מכוון $G=(V,E)$ הוא

תת קבוצה $V' \subseteq V$ כך שאם $(u,v) \in E$, אזי $u \in V'$ או $v \in V'$ (או שניהם).

מציאת הכיסוי המינימלי

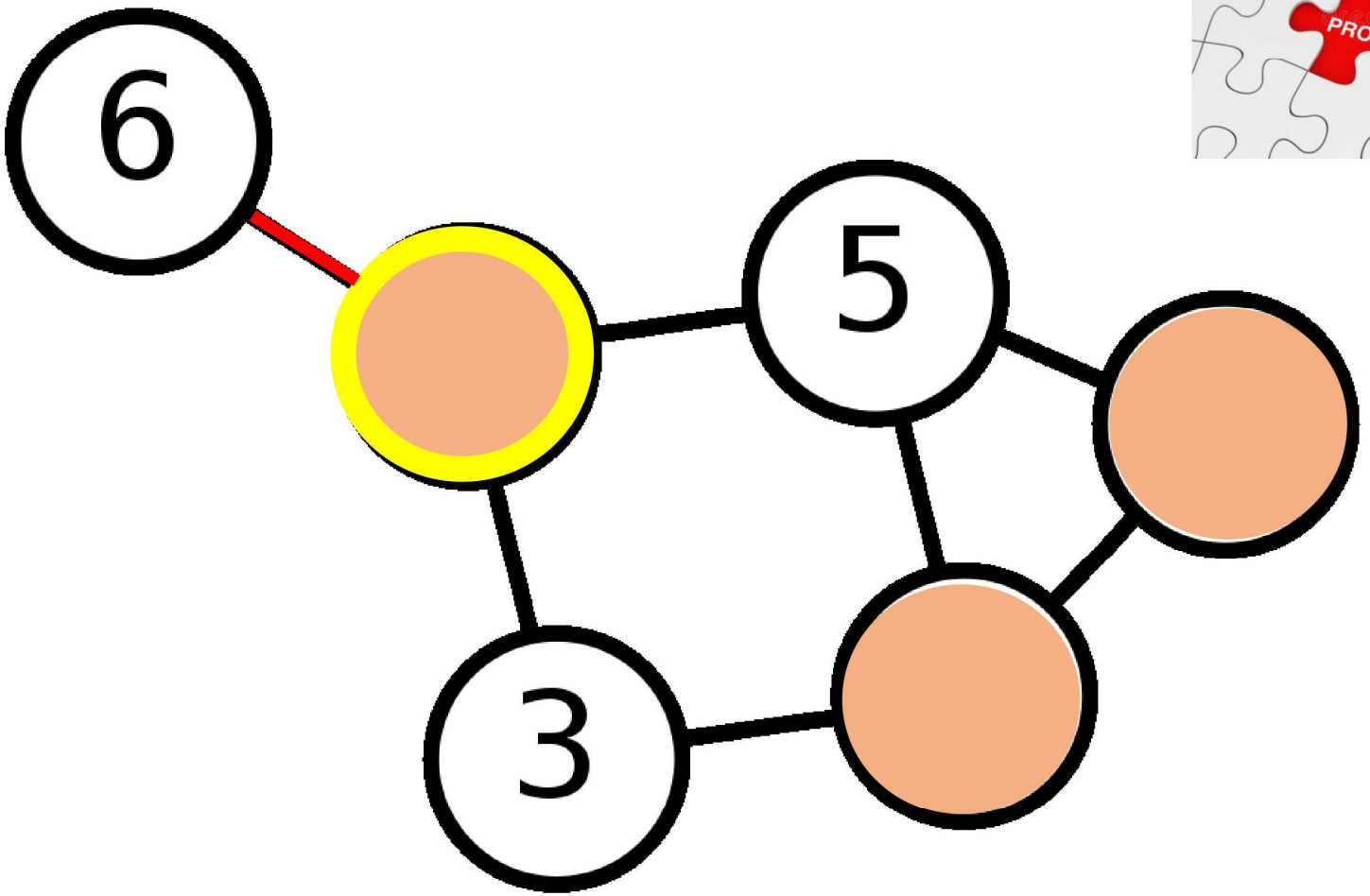
בעיית ריחוי הדדהודיות

2



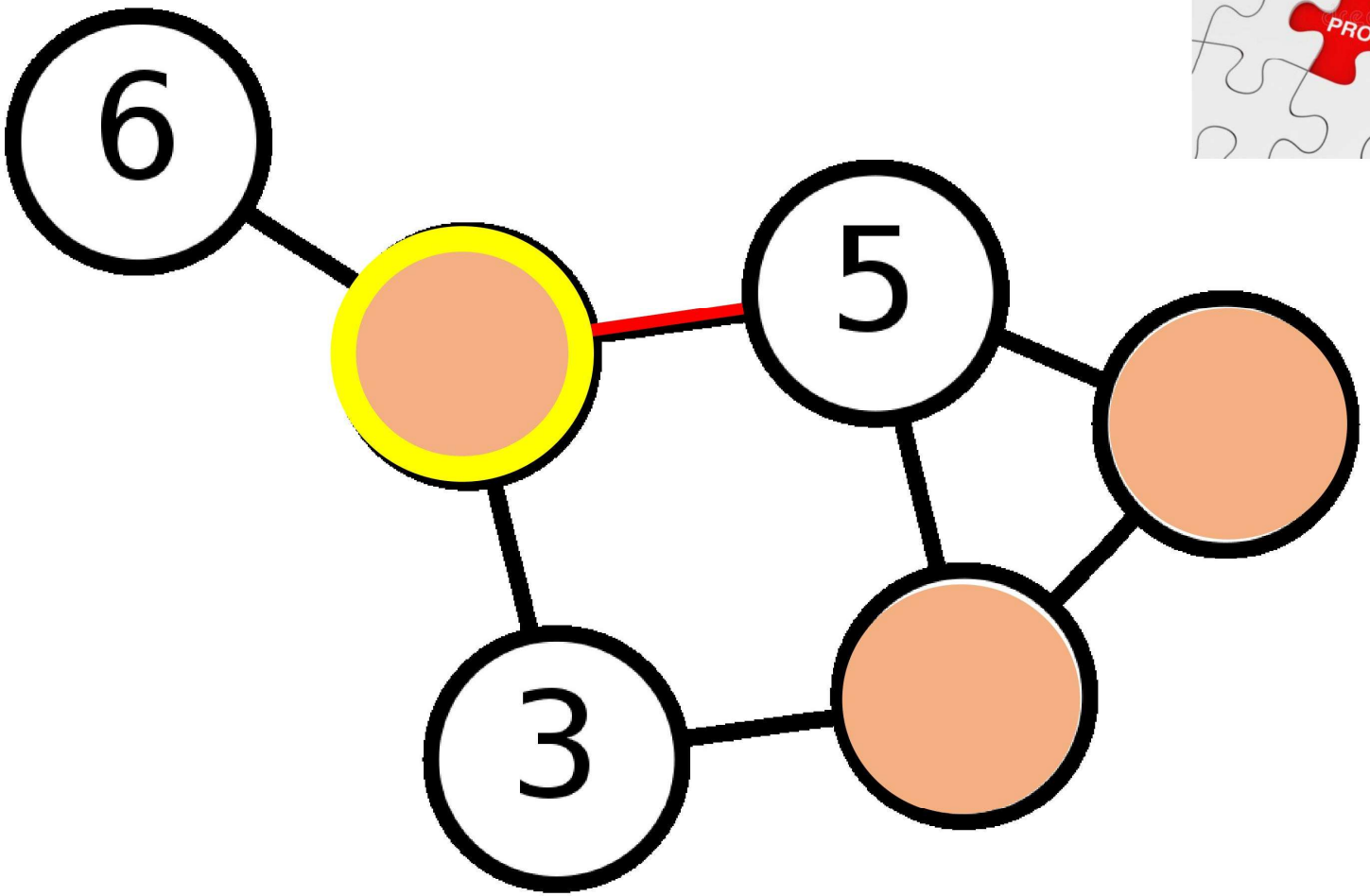
רעיון ריחוי הדדהודיוח

2



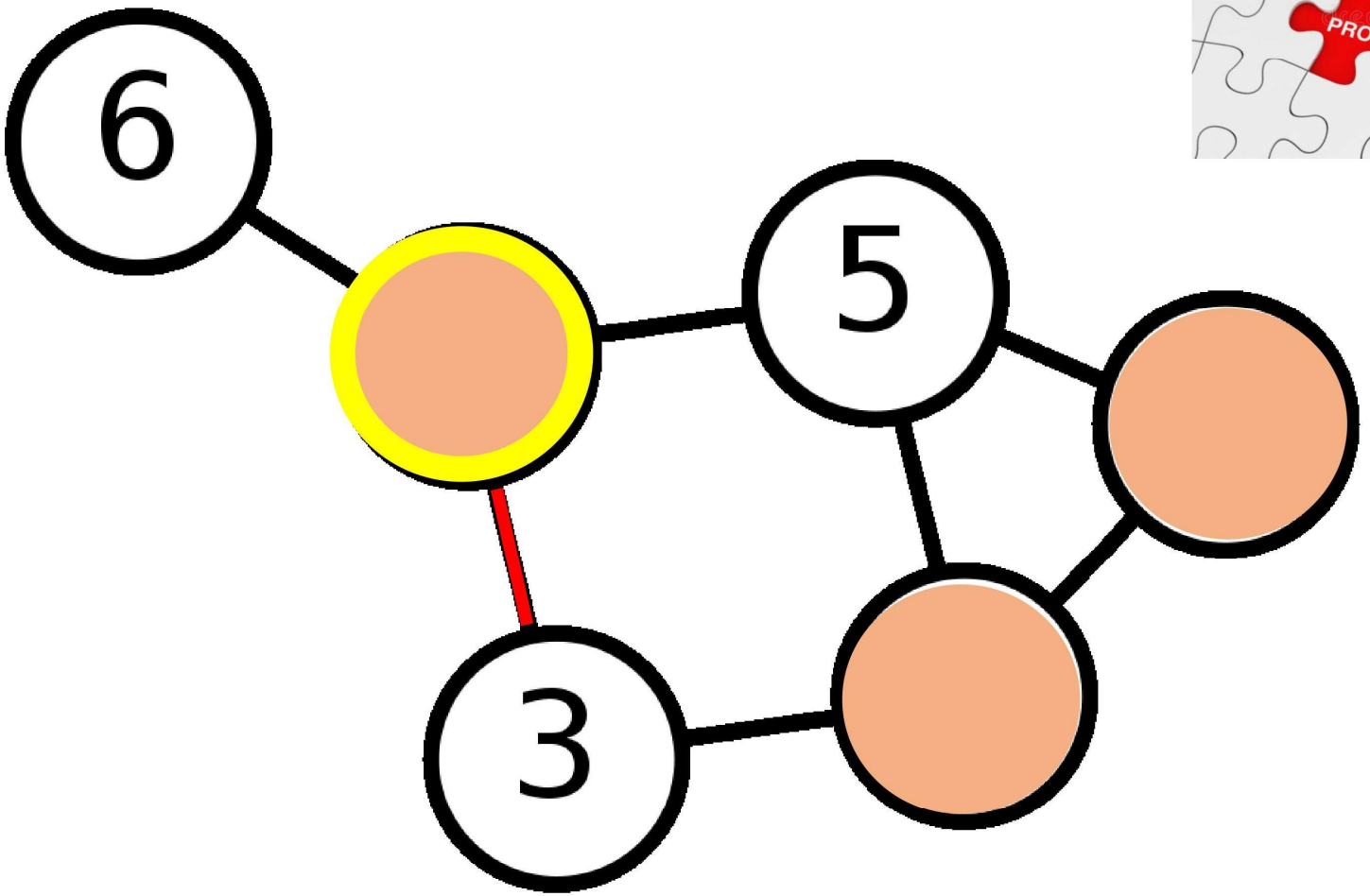
בעיית ריחוי הדדהודיות

2



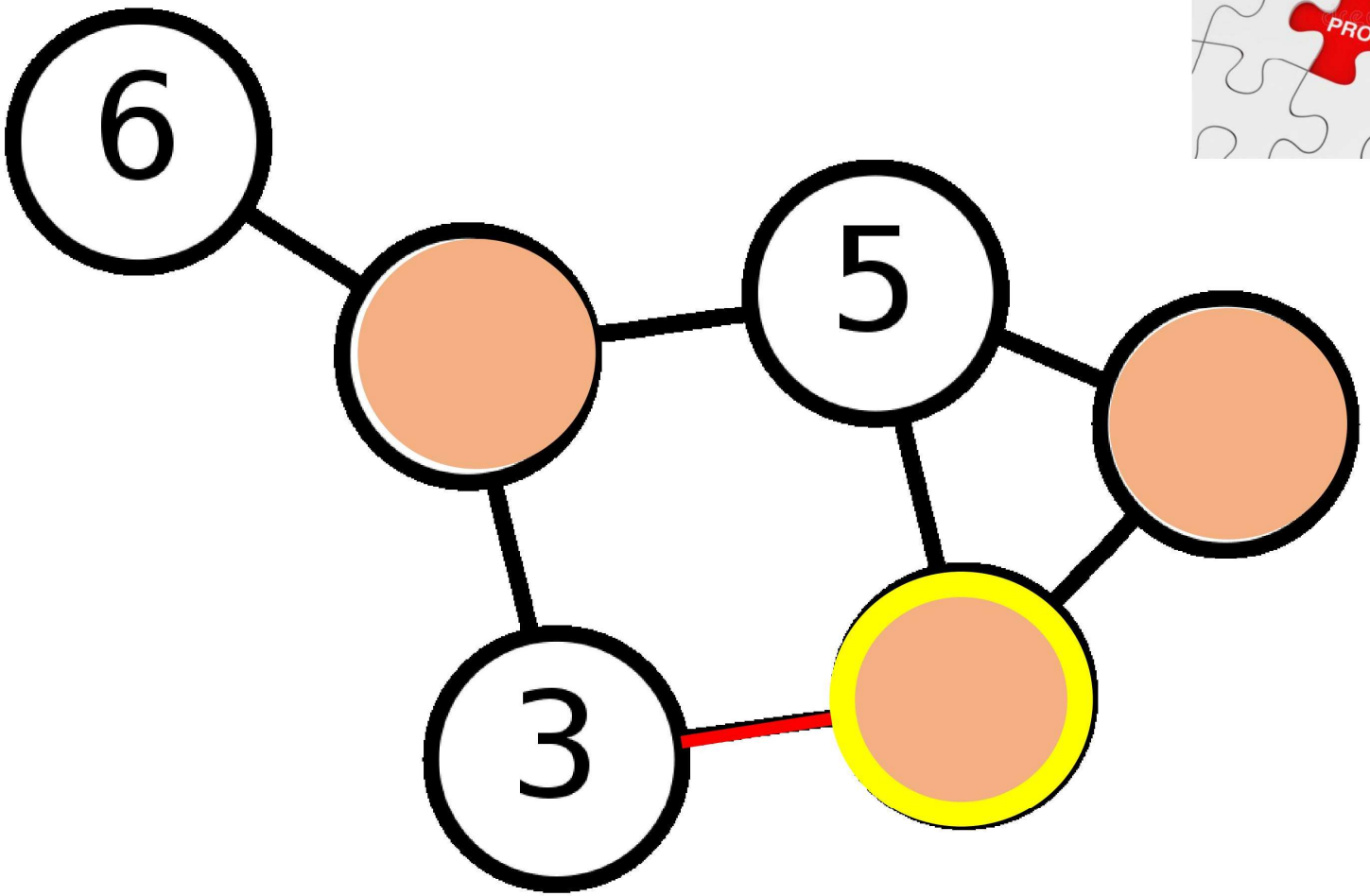
בעיית ריחוי הדדהודיות

2



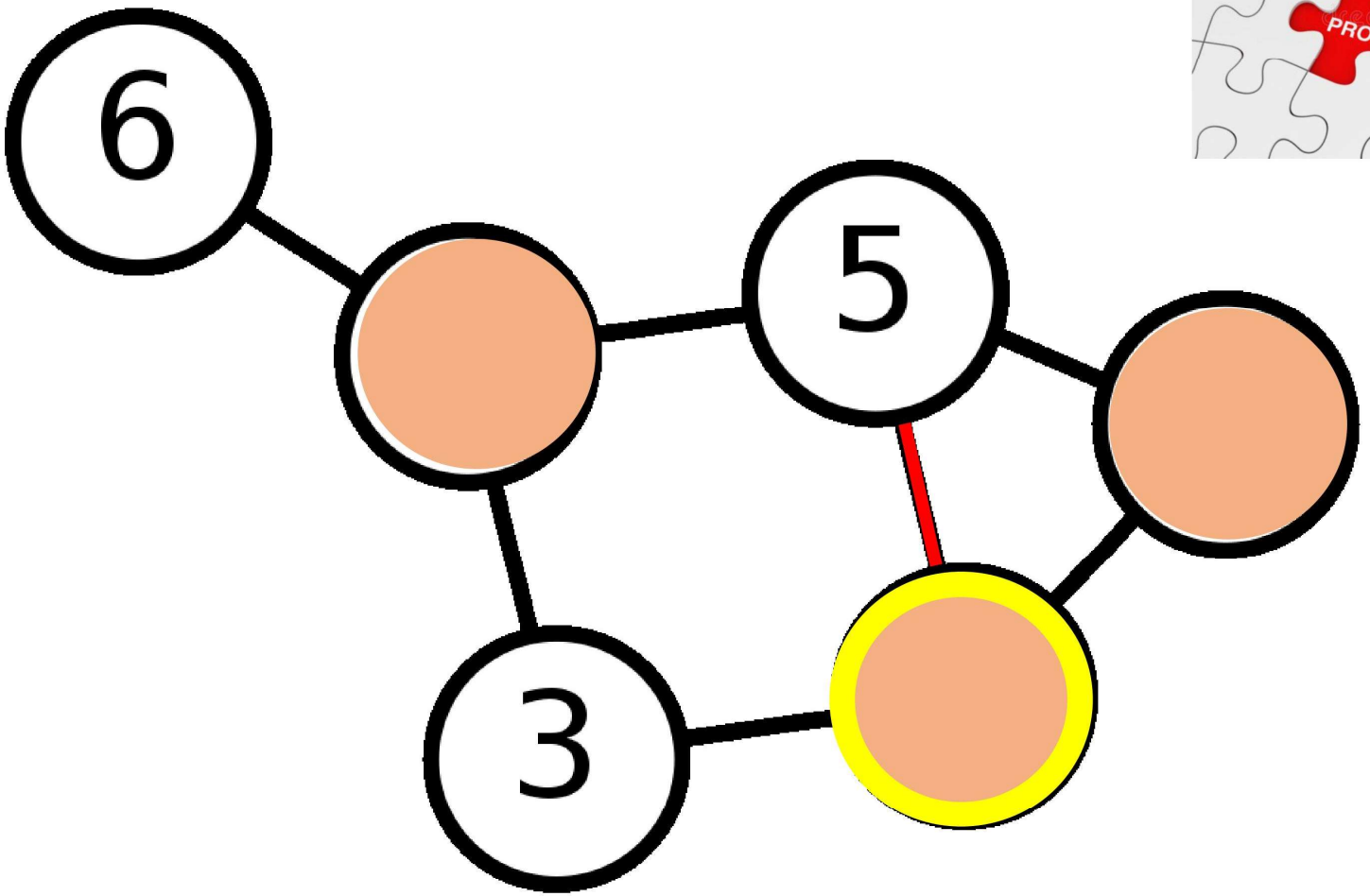
בעיית ריחוי הדדהודיות

2



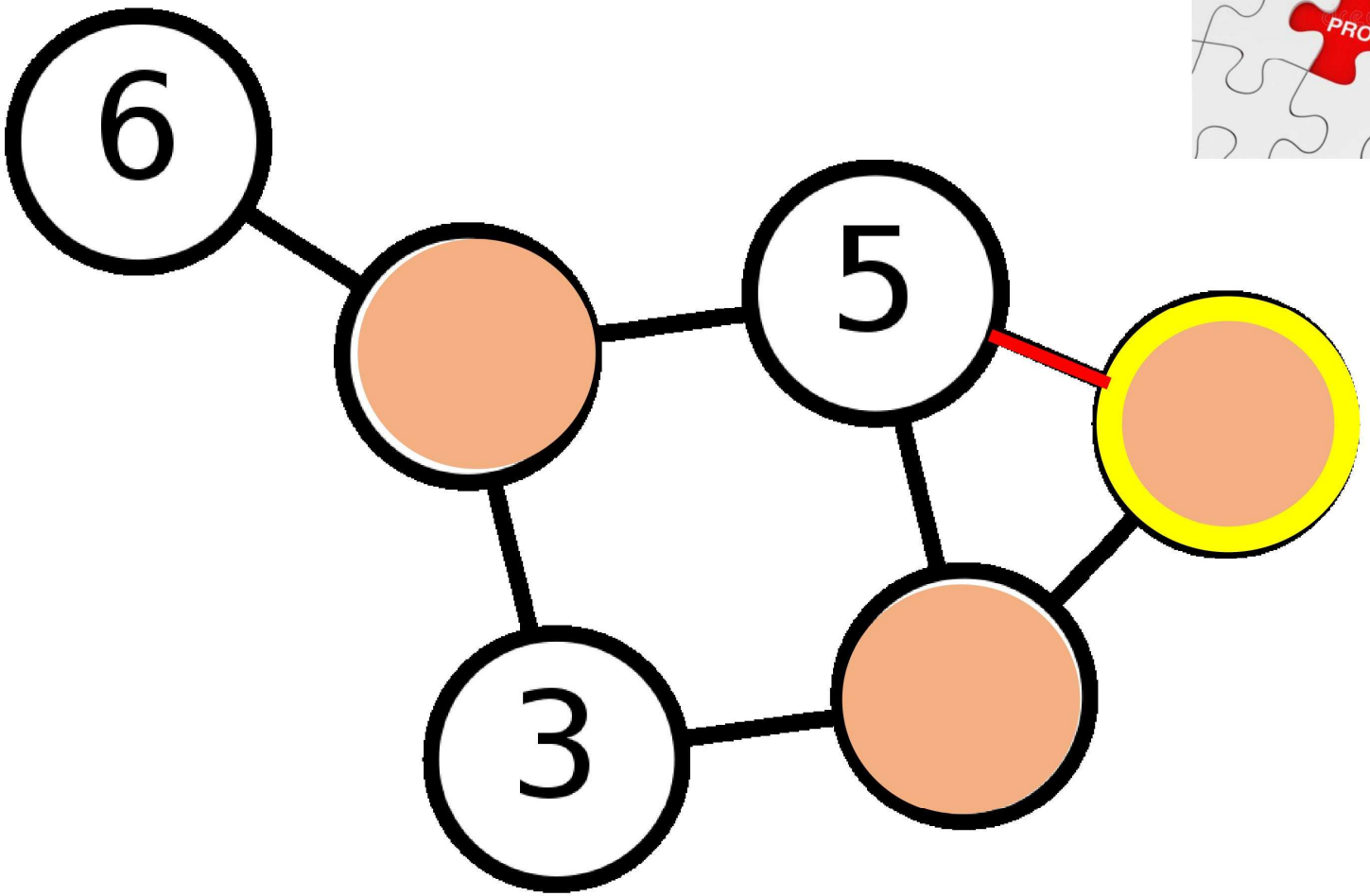
בעיית ריחוי הדדהודיות

2



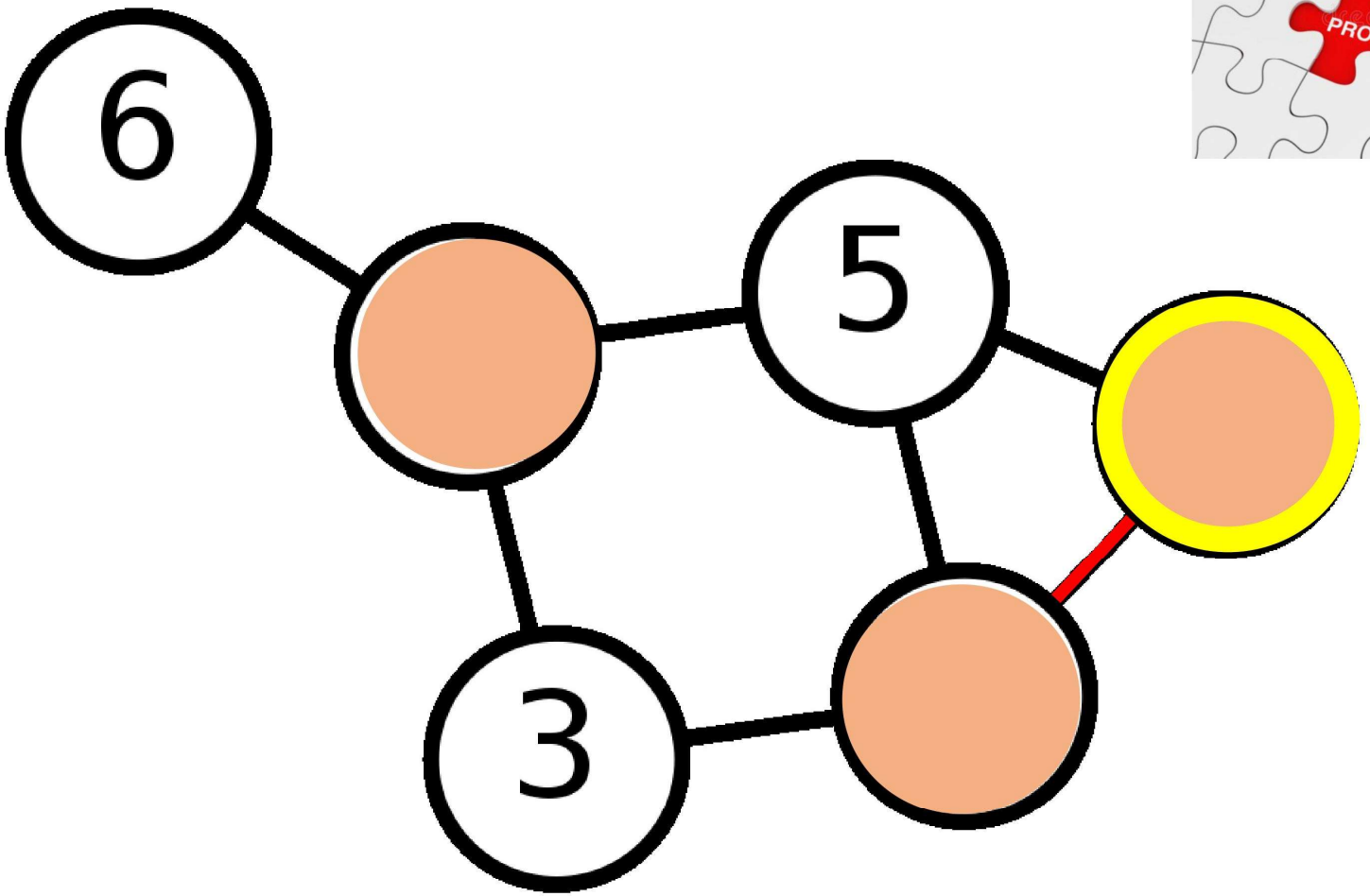
רעיון ריחוי הדדהודיוח

2



רעיון ריחוי הדדהודיוח

2



בעיית כיסוי קודקודיים

כיסוי על ידי קדקודים (vertex cover) של גרף בלתי מכוון $G=(V,E)$ הוא

תת קבוצה $V' \subseteq V$ כך שאם $(u,v) \in E$, אזי $u \in V'$ או $v \in V'$ (או שניהם).

NP שלמה

מציאת הכיסוי המינימלי

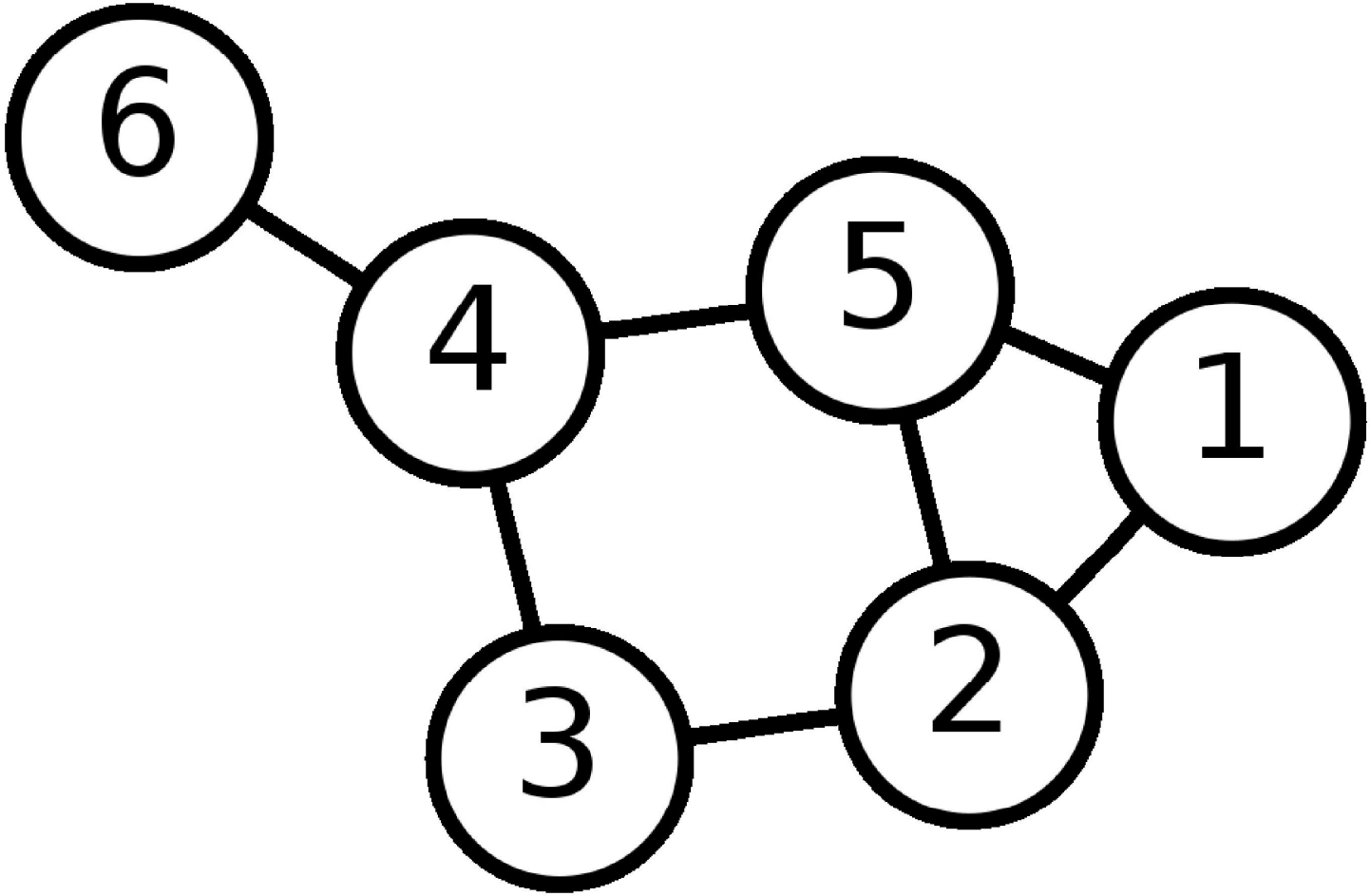
בעיית כיסוי קודקודיים

כיסוי על ידי קדקודים (vertex cover) של גרף בלתי מכוון $G=(V,E)$ הוא

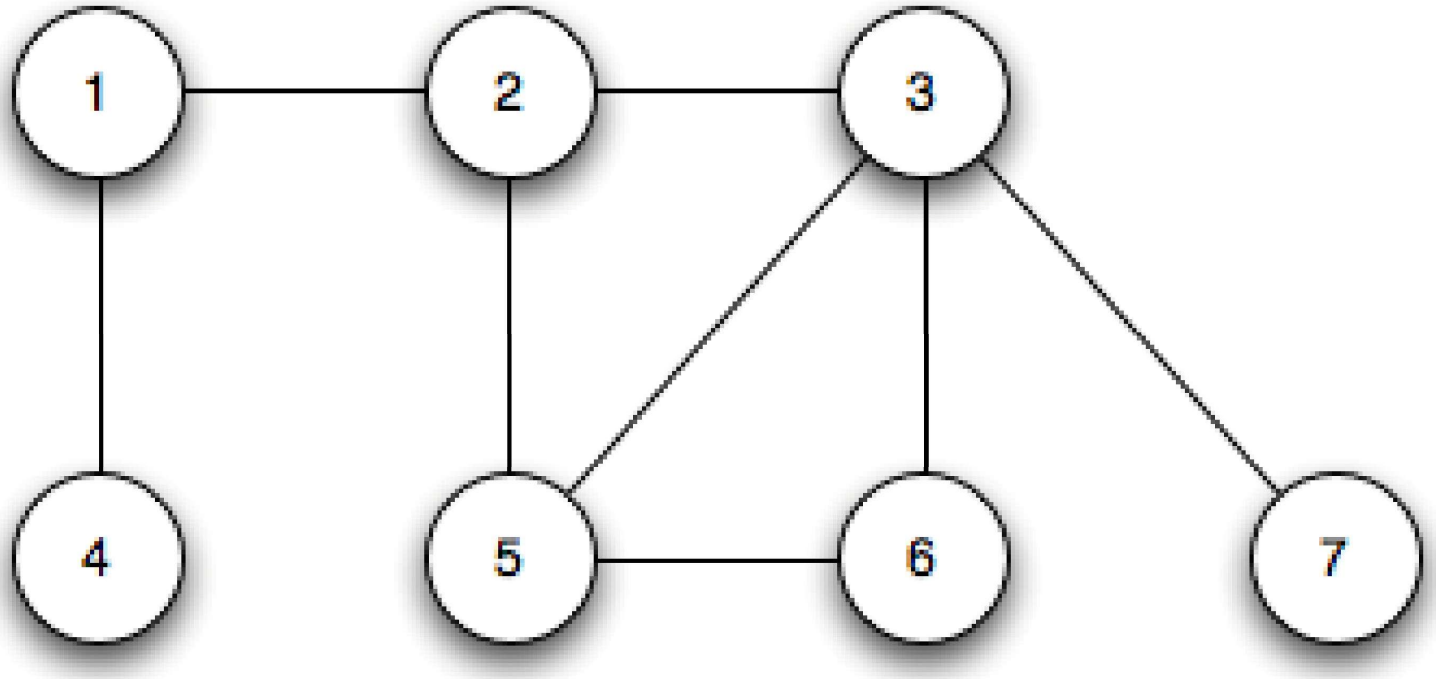
תת קבוצה $V' \subseteq V$ כך שאם $(u,v) \in E$, אזי $u \in V'$ או $v \in V'$ (או שניהם).

מציאת הכיסוי המינימלי

אלגוריתם הירור לרעיית ריחוי הודהודיים



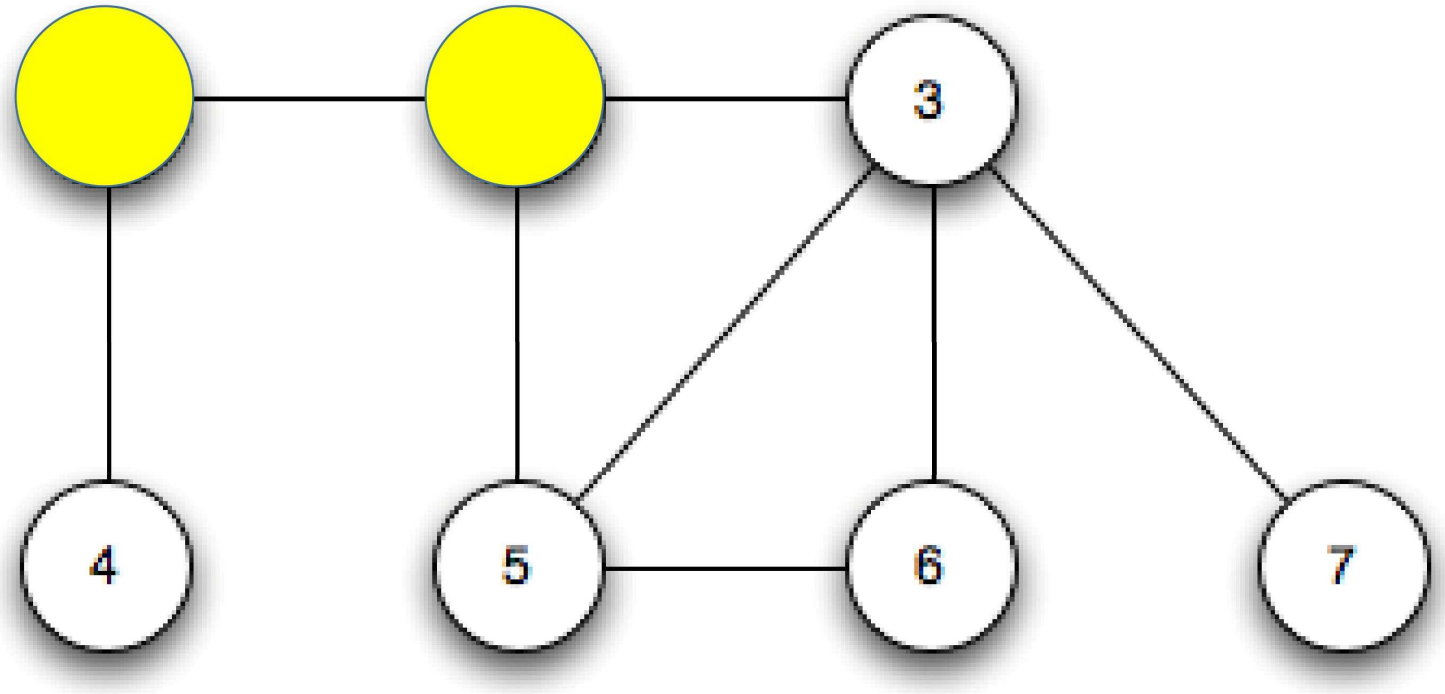
אתחול
אלגוריתם קירוב לבעיית כיסוי קודקודיים



E′	C
(1,4)	
(1,2)	
(2,5)	
(2,3)	
(3,5)	
(3,6)	
(3,7)	
(5,6)	

בחר צלע מהקבוצה E' - הכנס את קדקודיה ל-C

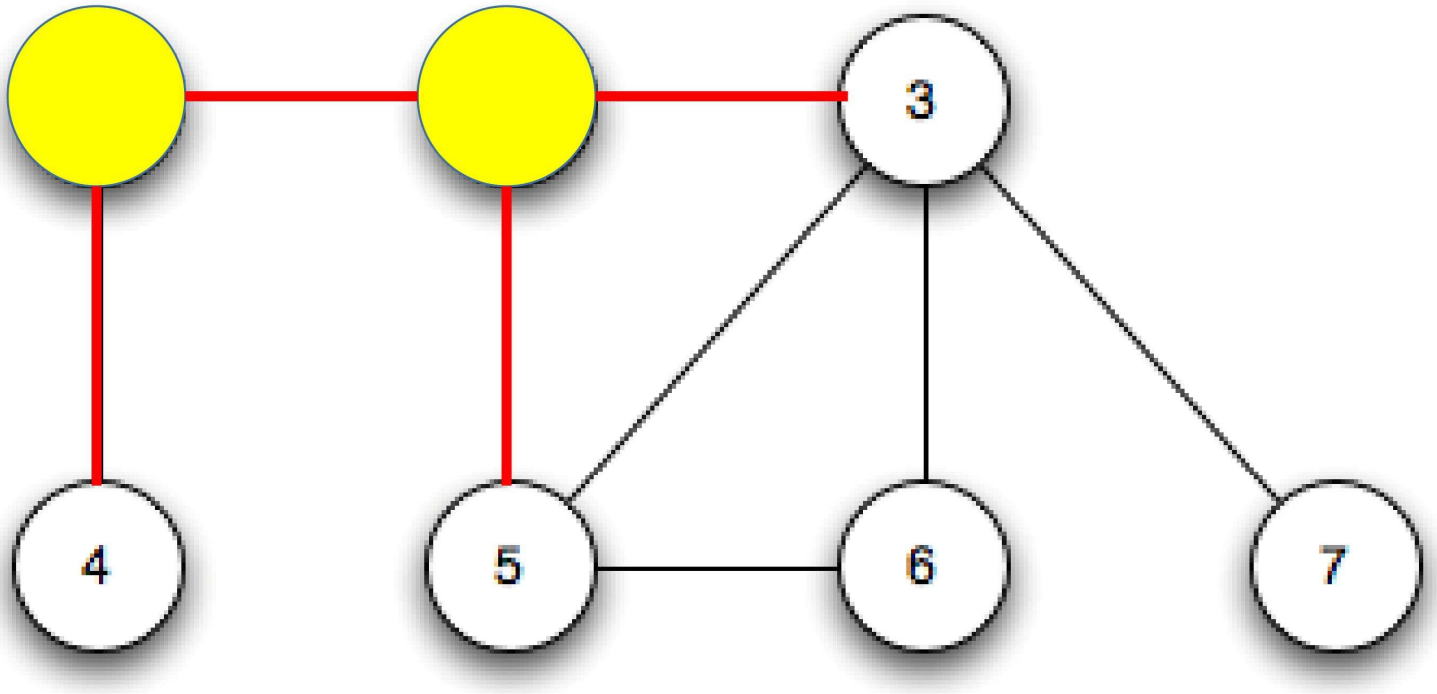
אלגוריתם קירוב לבעיית כיסוי קודקודיים



E'	C
(1,4)	1
(1,2)	2
(2,5)	
(2,3)	
(3,5)	
(3,6)	
(3,7)	
(5,6)	

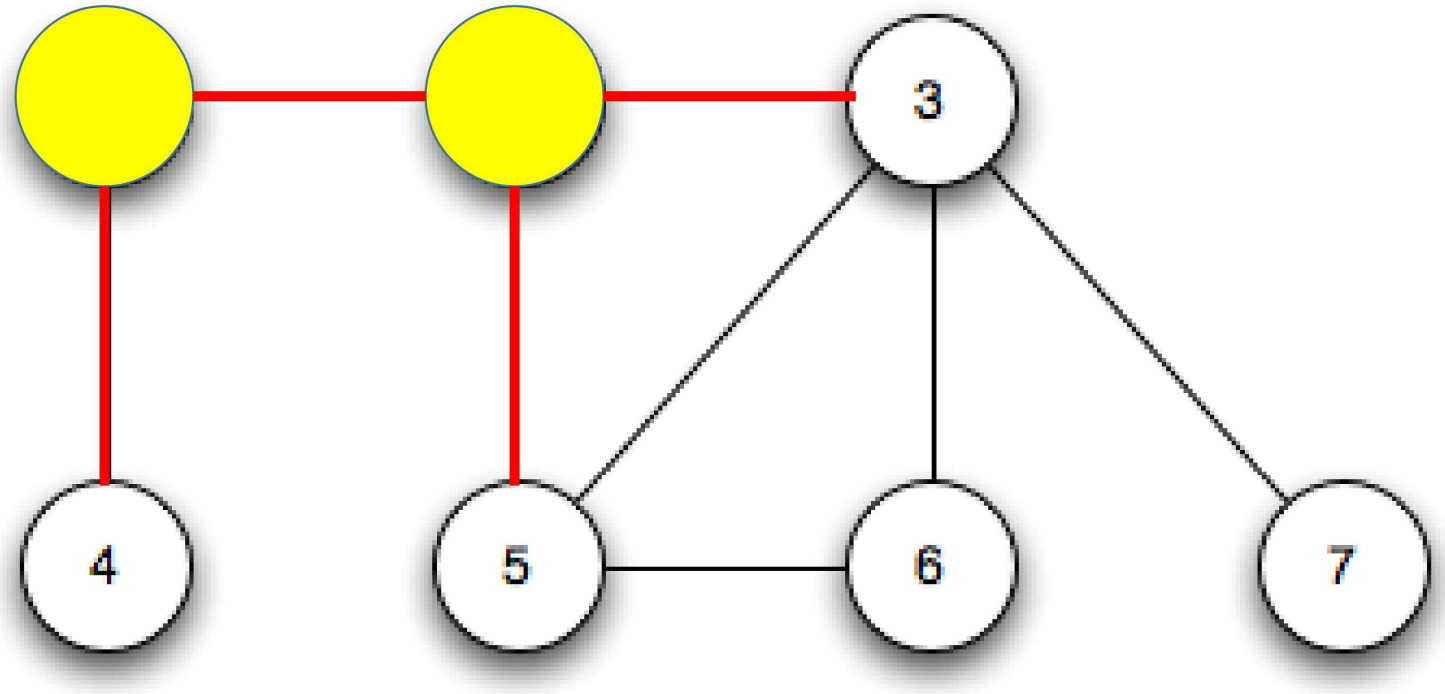
אלגוריתם קירוב לבעיית כיסוי קודקודיים

הסר את הצלעות מ-E′ המכוסות על די הקודקודיים שנוספו ל-C



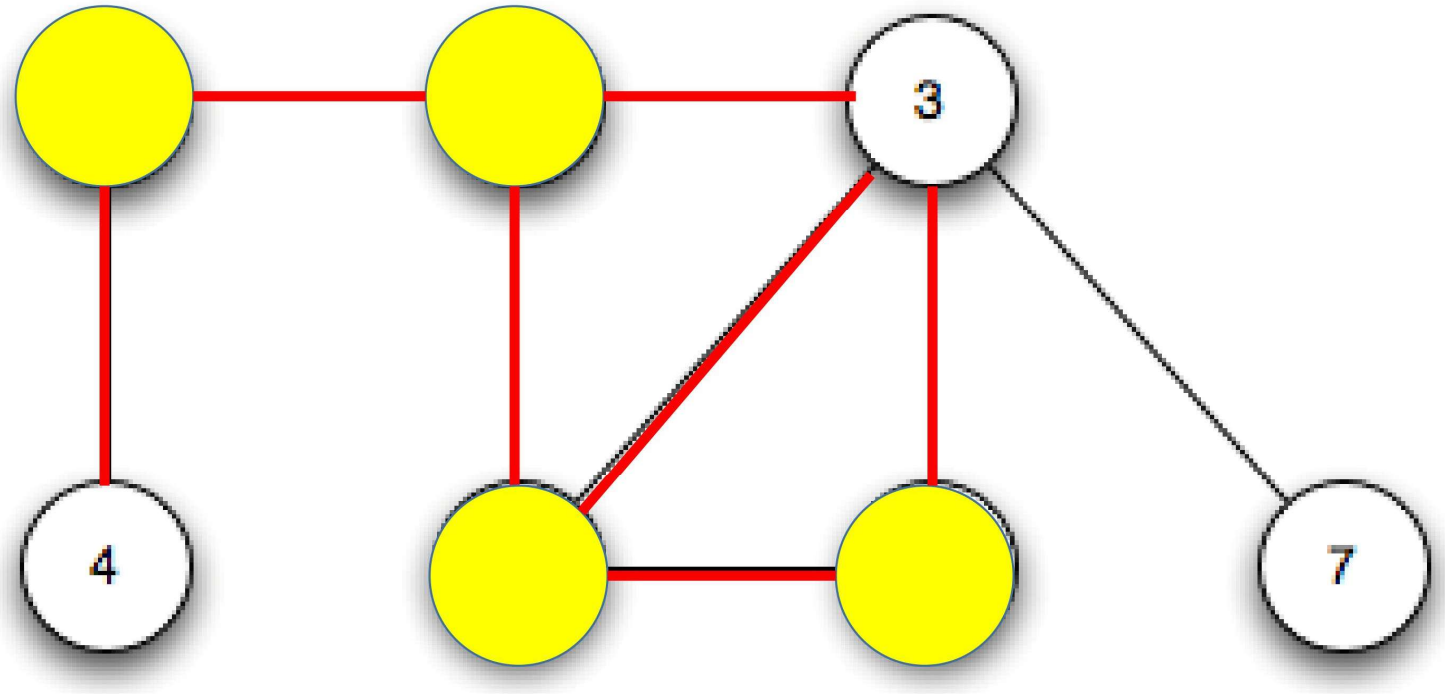
E′	C
(1,4)	1
(1,2)	2
(2,5)	
(2,3)	
(3,5)	
(3,6)	
(3,7)	
(5,6)	

אלגוריתם קירוב לבעיית כיסוי קודקודיים



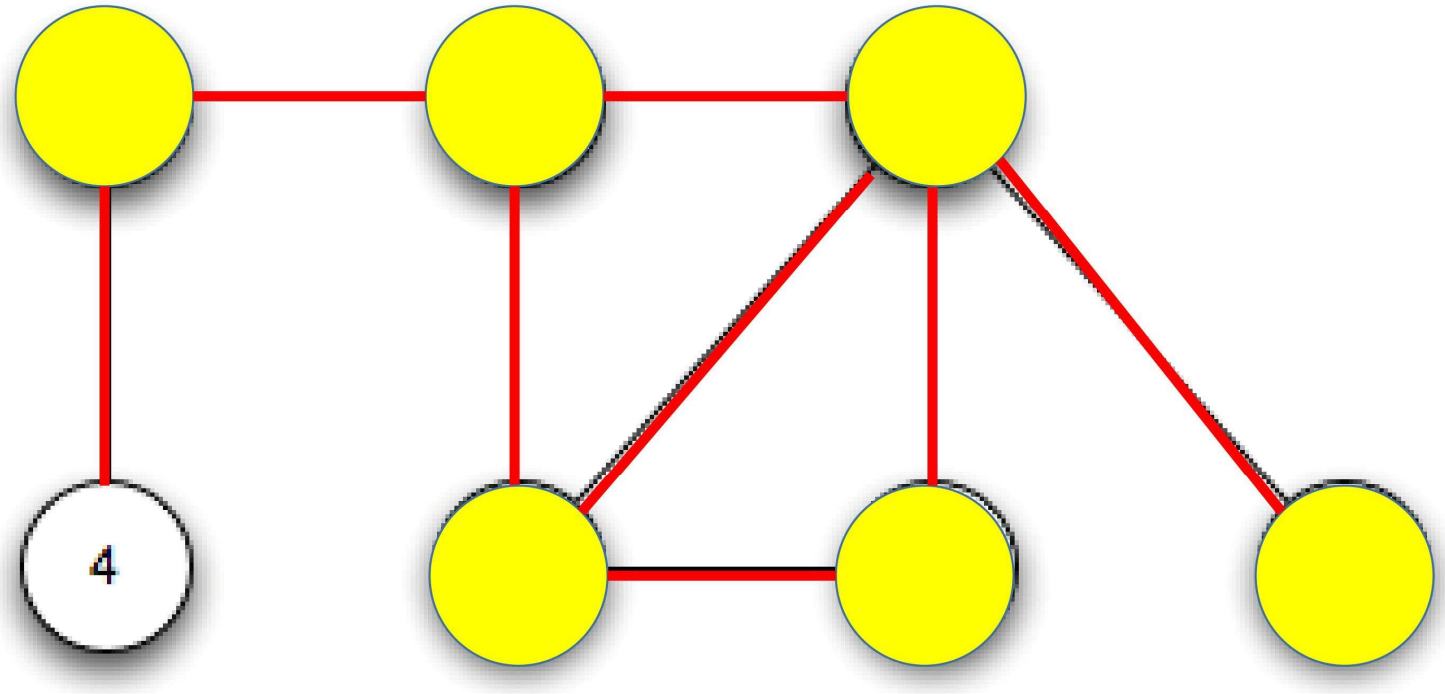
E′	C
	1
(1,2)	2
(3,5)	
(3,6)	
(3,7)	
(5,6)	

אלגוריתם קירוב לבעיית כיסוי קודקודיים



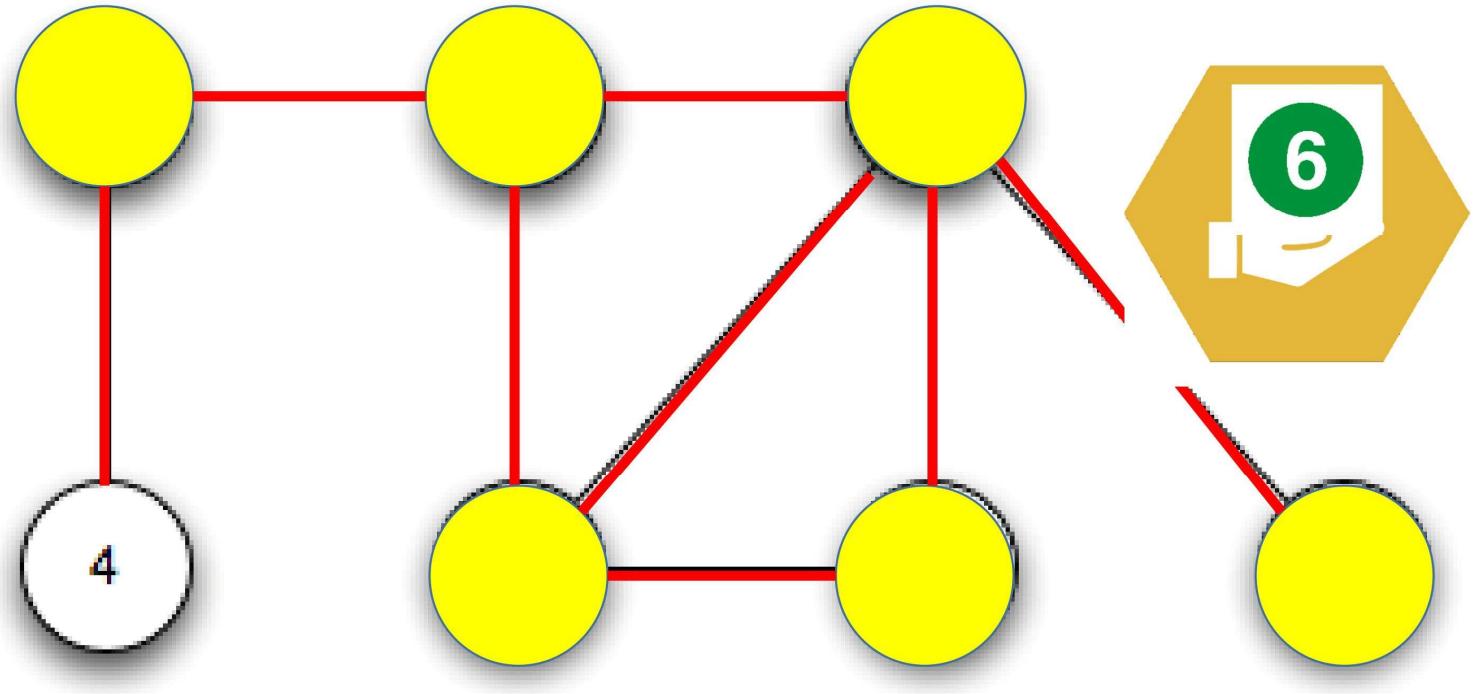
E′	C
	1
(1,2)	2
	5
	6
(3,7)	
(5,6)	

אלגוריתם קירוב לבעיית כיסוי קודקודיים



E'	C
	1
(1,2)	2
	5
	6
	3
	7
(3,7)	
(5,6)	

אלגוריתם קירוב לבעיית כיסוי קודקודיים

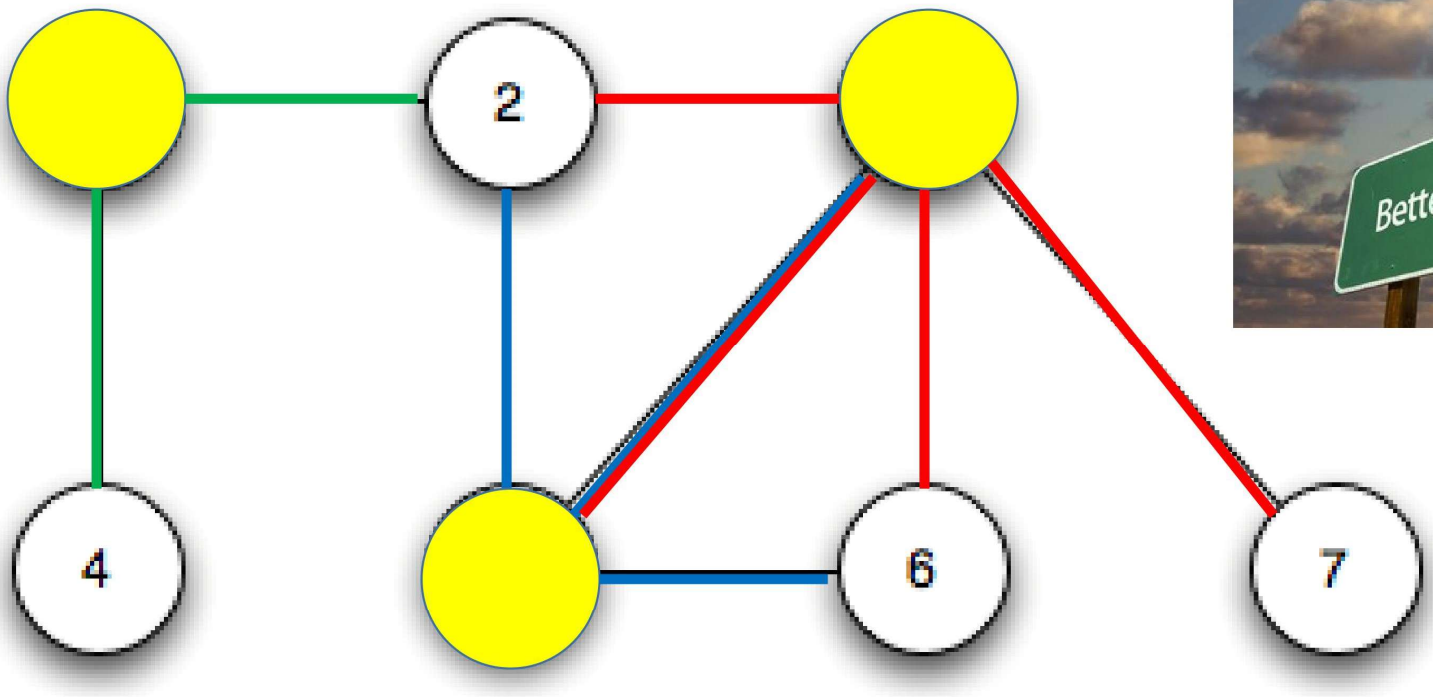


E'	C
	1
(1,2)	2
	5
	6
	3
	7
(3,7)	
(5,6)	

APROX-VERTEX_COVER(G)

1. $C \leftarrow \emptyset$
2. $E' \leftarrow E$
3. **while** $E' \neq \emptyset$
4. **do** let (u,v) be an arbitary edge of E'
5. $C \leftarrow C \cup \{u,v\}$
6. remove from E' every edge incident on either u or v
7. **return** C

אלגוריתם קירוב לבעיית כיסוי קודקודיים



האלגוריתם קירוב לבעיית כיסוי קודקודיים הוא בעל יחס קירוב 2

<u>שלב ראשון:</u> מדוע האלגוריתם מוצא כיסוי ?

האלגוריתם קירוב לבעיית כיסוי קודקודיים הוא בעל יחס קירוב 2

<u>שלב שני:</u> מדוע האלגוריתם מוצא כיסוי ?

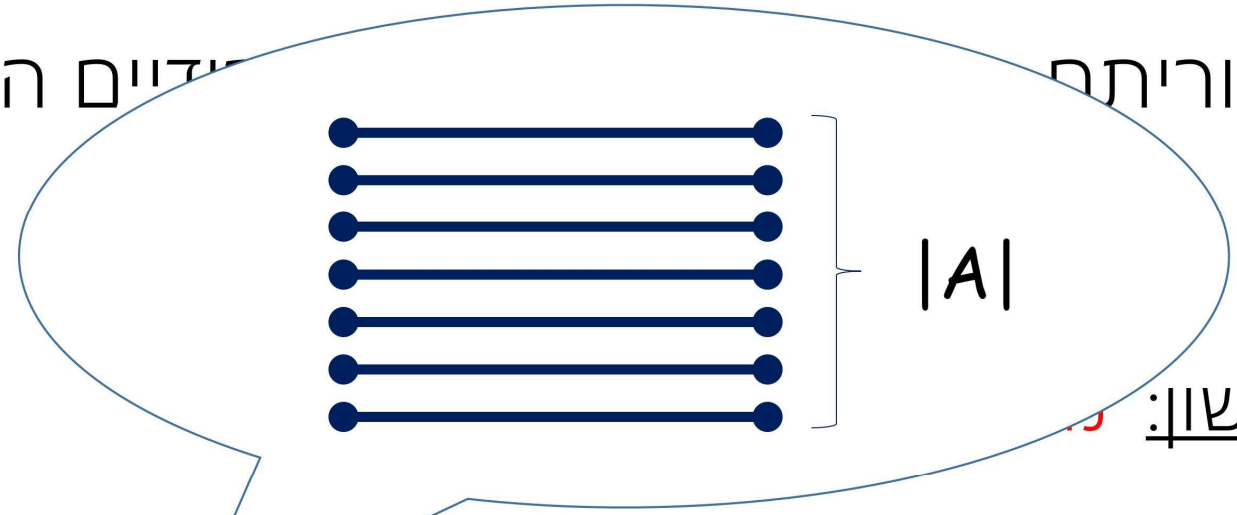
הקבוצה C שמחזיר האלגוריתם היא כיסוי קודקודיים, כיוון שהאלגוריתם חוזר ומבצע את הלולאה עד שכל קשת ב-E מכוסה על ידי איזשהו קדקוד ב-C

האלגוריתם קירוב לבעיית כיסוי קודקודיים הוא בעל יחס קירוב 2

<u>שלב ראשון:</u> מדוע הקירוב הוא 2?

נסמן ב-**A** את קבוצת הקשתות **(u,v)** שנבחרה בשורה 4. קבוצה זו אינה מכילה קשתות שלהן צמתים משותפים, לכן **|C|=2·|A|**.

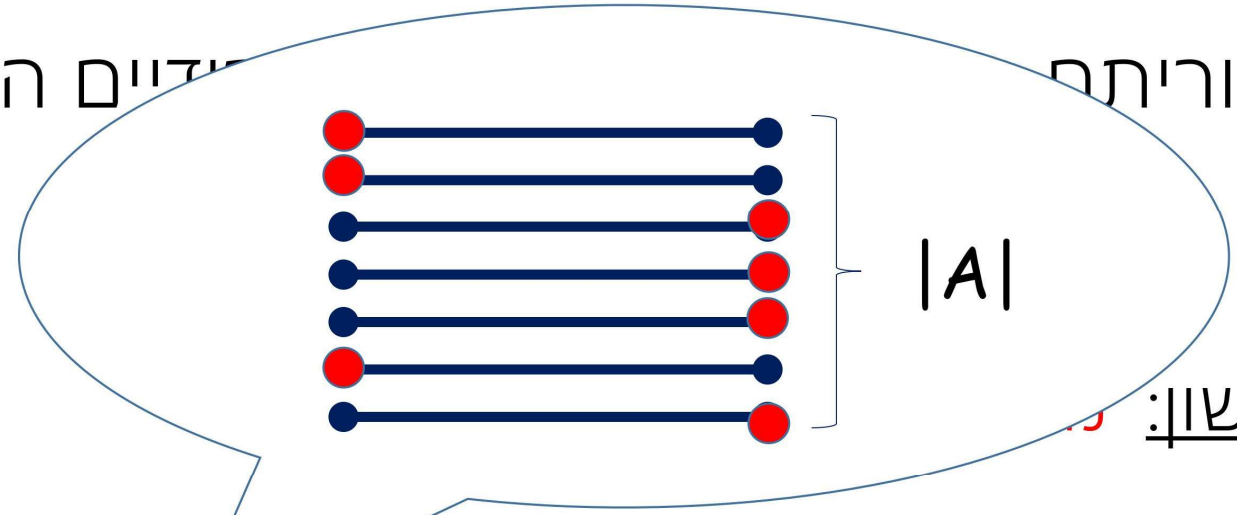
האלגוריתם [illegible] הוא בעל



שלב ראשון: [illegible]

נסמן ב-**A** את קבוצת הקשתות **(u,v)** שנבחרה בשורה 4. קבוצה זו אינה מכילה קשתות שלהן צמתים משותפים, לכן **|C|=2·|A|**.

האלגוריתם [illegible] הוא בעל



שלב ראשון: [illegible]

נסמן ב-A את קבוצת הקשתות (u,v) שנבחרה בשורה 4. קבוצה זו אינה מכילה קשתות שלהן צמתים משותפים, לכן $|C|=2\cdot|A|$.

צמתי הפתרון האופטימלי C^* מכסים את קשתות A ולכן מכילים לפחות צומת אחד לכל קשת ומכאן $|C^*|\geq|A|$.

ולכן נקבל, $|C|\leq 2|C^*|$

Two Glass Balls and a Tower

Amihood Amir

Bar Ilan University

And

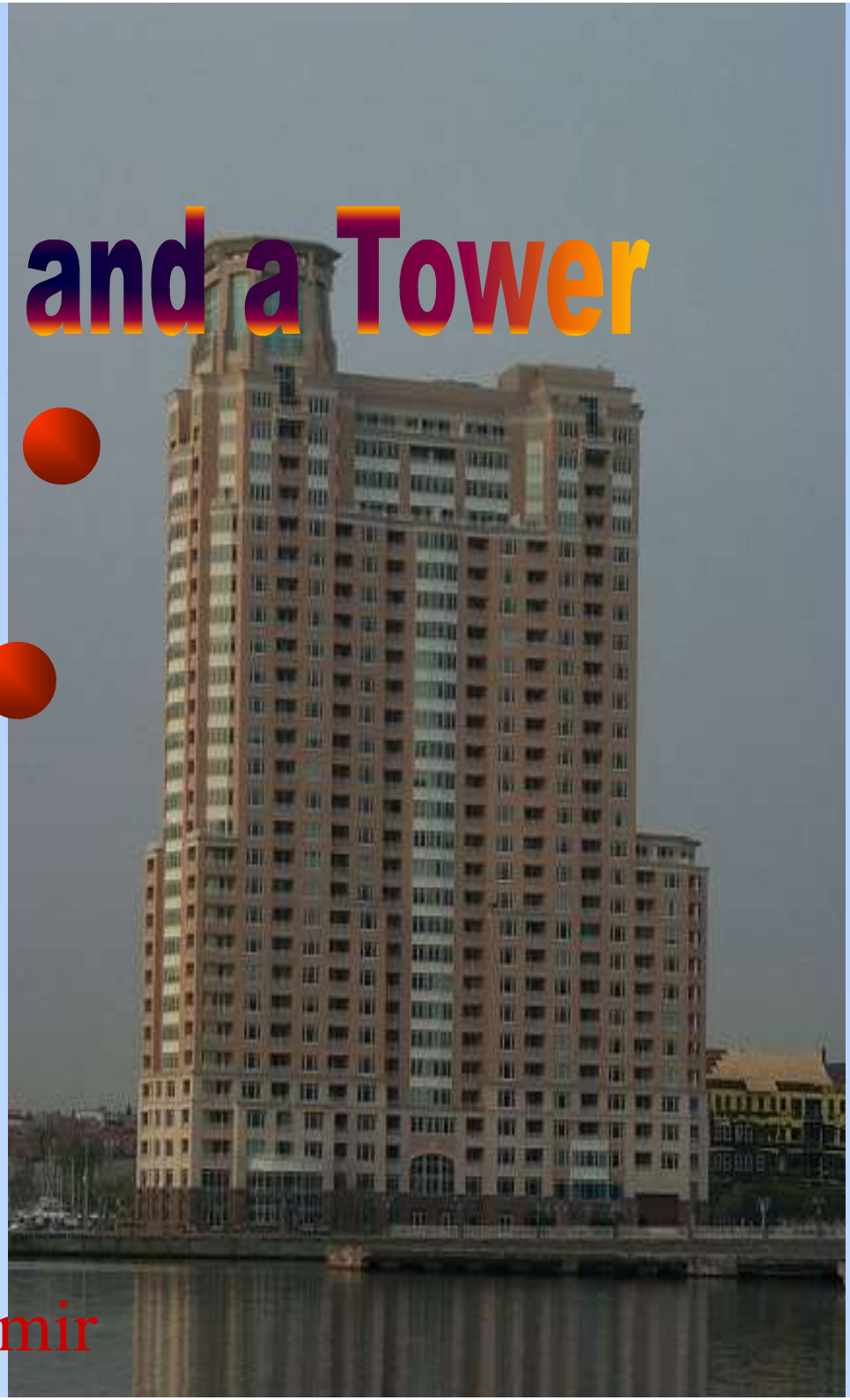
Georgia Tech

Costas Iliopoulos

Oren Kapah

Ely Porat

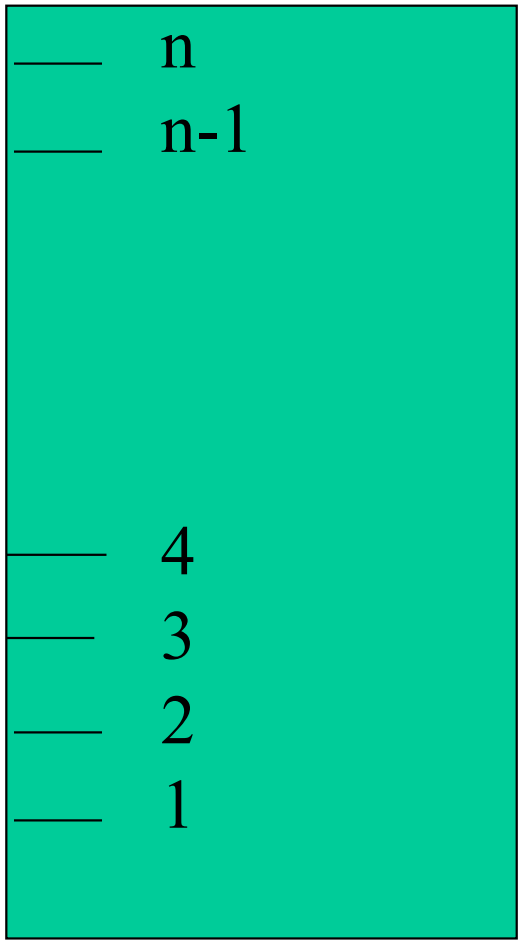
Artistic Consultant: Aviya Amir



Given: A glass ball.
An n storied building.

Find: The floor k such that the ball breaks when dropped from it, but does not break if dropped from floor $k-1$.

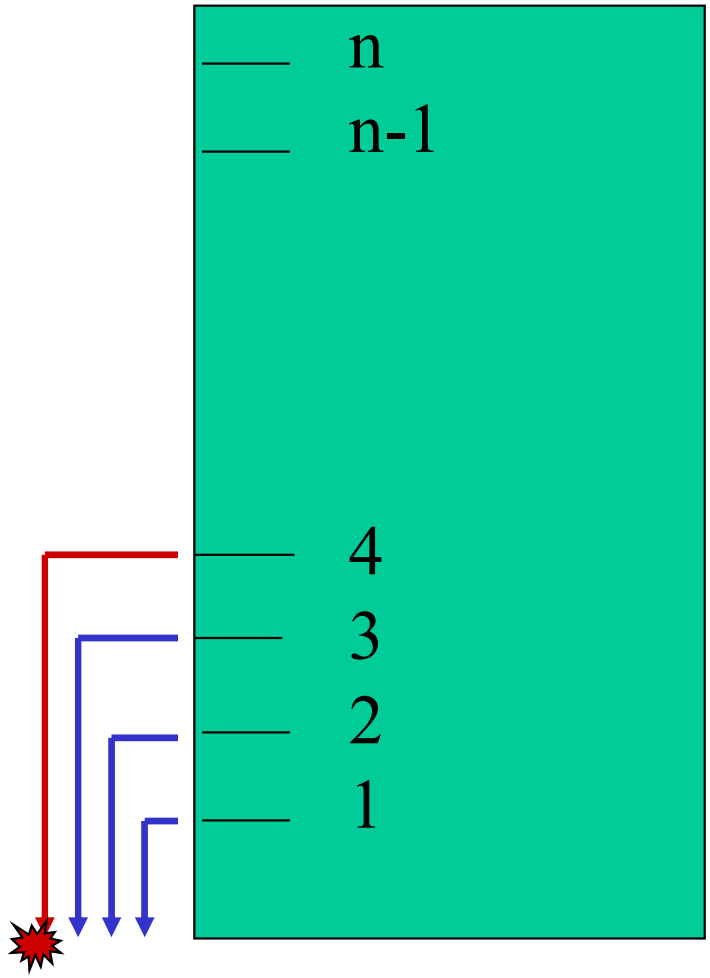
STRATEGY 1: Only *one* ball given to experiment with.



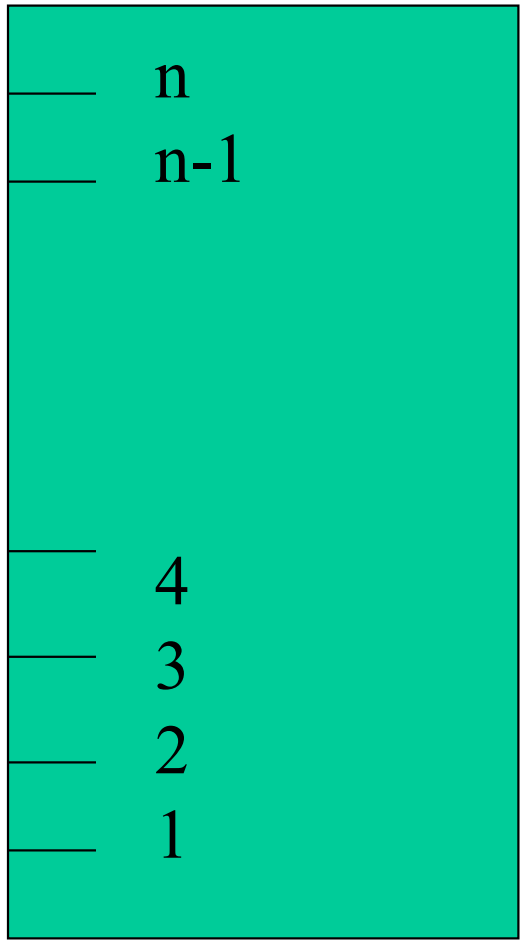
STRATEGY 1: Only *one* ball given to experiment with.

O(n) experiments necessary.

Sequential search.



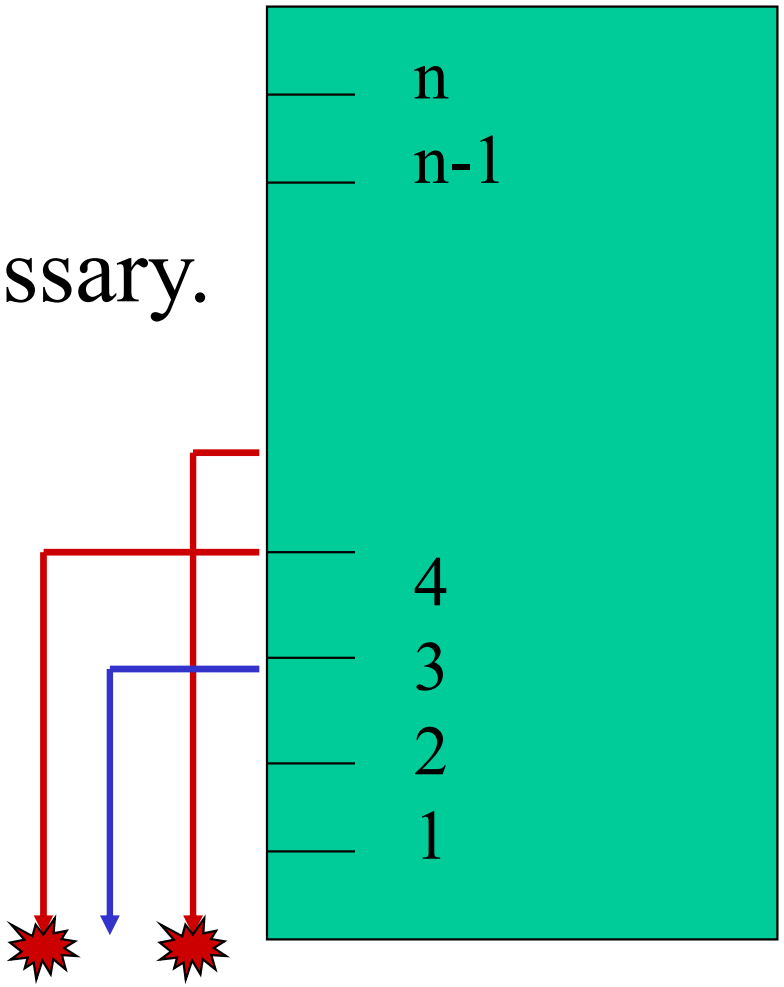
STRATEGY 2: As many balls as necessary are given to experiment with.



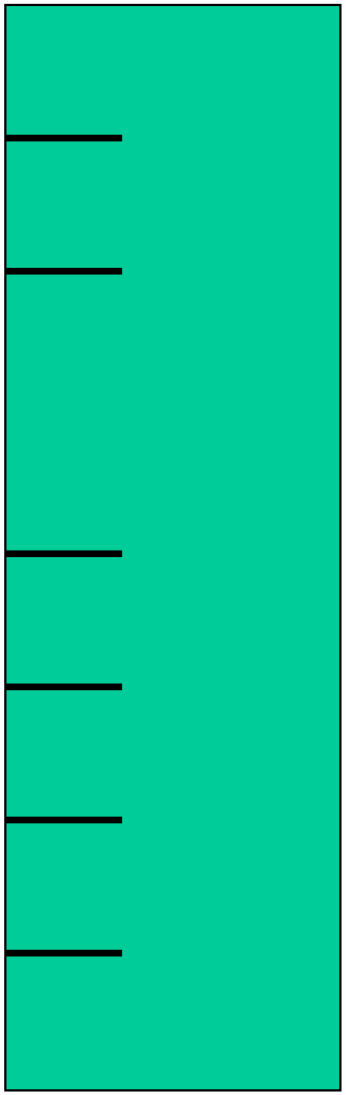
STRATEGY 2: As many balls as necessary are given to experiment with.

$O(\log n)$ experiments necessary.

Binary search.



STRATEGY 3: Only *two* balls given to experiment with.



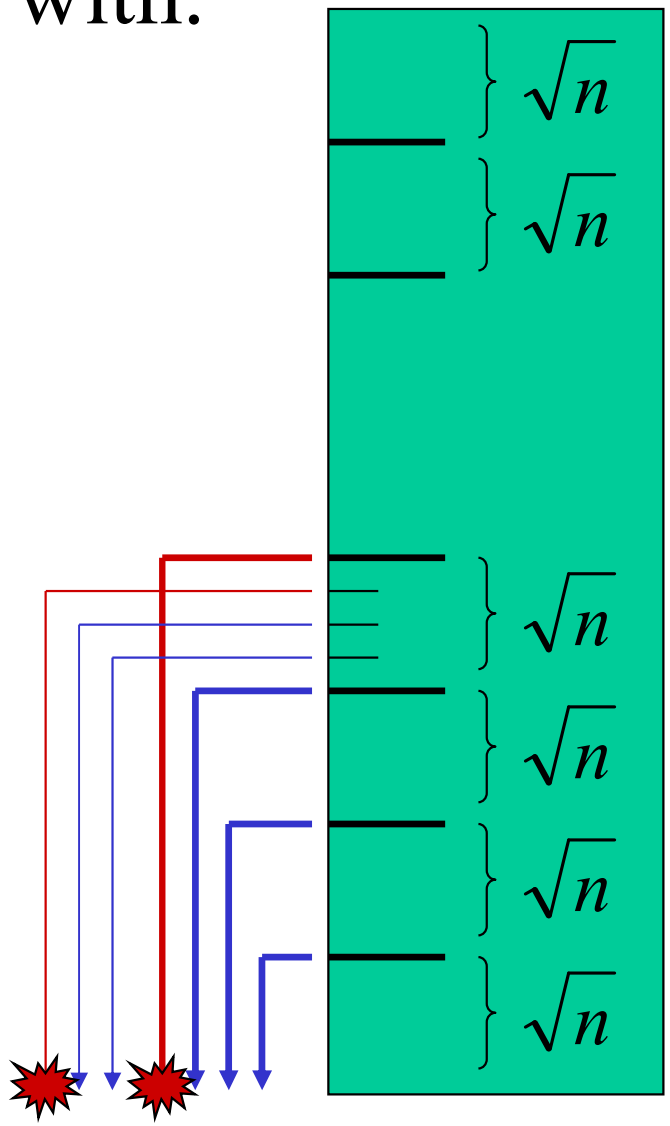
STRATEGY 3: Only *two* balls given to experiment with.

$O(\sqrt{n})$ experiments necessary.

Bounded divide-and-conquer

$O(\sqrt{n})$ Experiments 1st ball

$O(\sqrt{n})$ Experiments 2nd ball



Meaning of two Balls

"Bounded" Divide-and-Conquer.

In reality: Different paradigms.

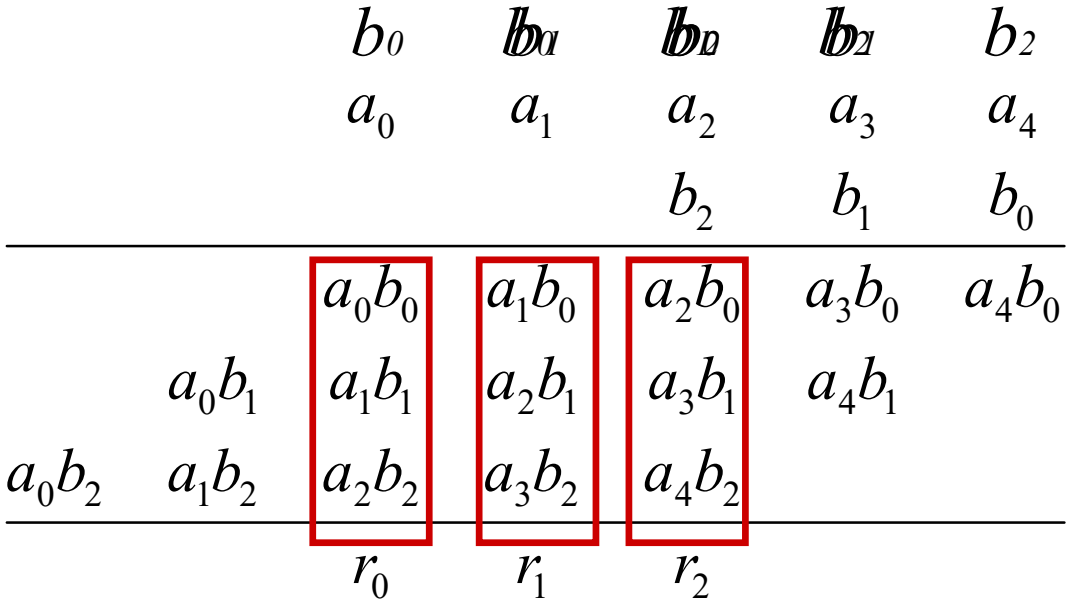
1. Works on large groups.

2. Works within a group.

In Pattern Matching

1. Works on large groups:

Convolutions:



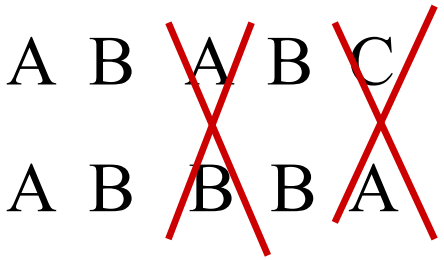
$O(n \log m)$ using FFT

Problem: $O(n \log m)$ only in algebraically closed fields, e.g. $\mathbf{C}$.

Solution: Reduce problem to (Boolean/integer/real) multiplication.

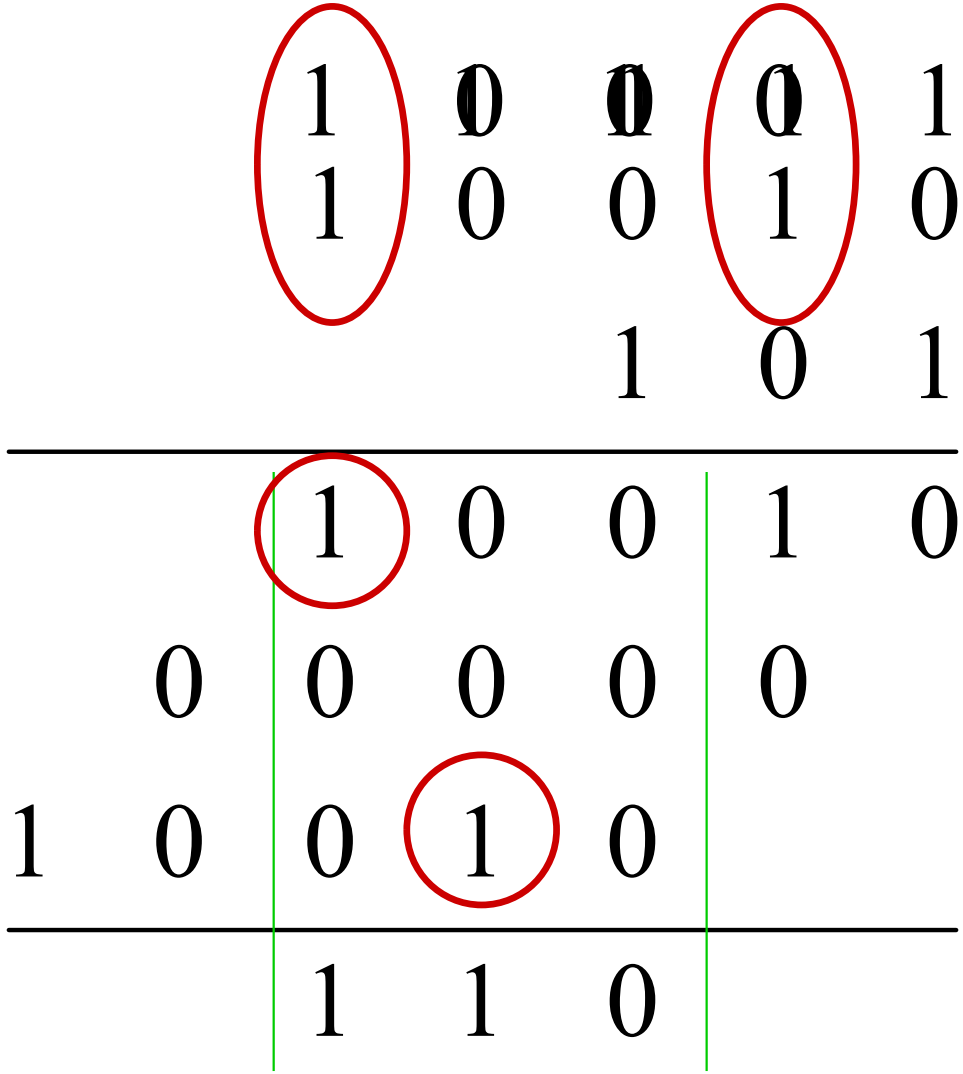
This reduction costs!

Example: Hamming distance.



Counting *mismatches* is equivalent to Counting *matches*

Example:



Count all “hits” of 1 in pattern and 1 in text.

For $a \in \Sigma$

Define:

$$\chi_a(b) = \begin{cases} 1 & \text{if } a=b \\ 0 & \text{o/w} \end{cases}$$

$$\chi_a(S_1 S_2 S_3 \ldots S_n) = \chi_a(S_1)\chi_a(S_2)\chi_a(S_3)\ldots\chi_a(S_n)$$

Example:

$$\chi_a(abbaabb) = 1001100$$

For $\Sigma = \{a, b, c\}$

Do:

$$\begin{aligned} & \chi_a(T) \times \chi_a(P^R) \\ + \; & \chi_b(T) \times \chi_b(P^R) \\ + \; & \chi_c(T) \times \chi_c(P^R) \end{aligned}$$

Result: The number of times a in pattern matches a in text + the number of times b in pattern matches b in text + the number of times c in pattern matches c in text.

So for alphabet with a symbols (a fixed) the time is:

$$O(n\, a \log m) = O(n \log m)$$

Problem: Infinite alphabets.

Without loss of generality:

$$|\Sigma| = m + 1$$

Since every element of T not in P is replaced by some symbol x not in P.

Example:

ABCDEFGH same number of errors as ABXXXXGH

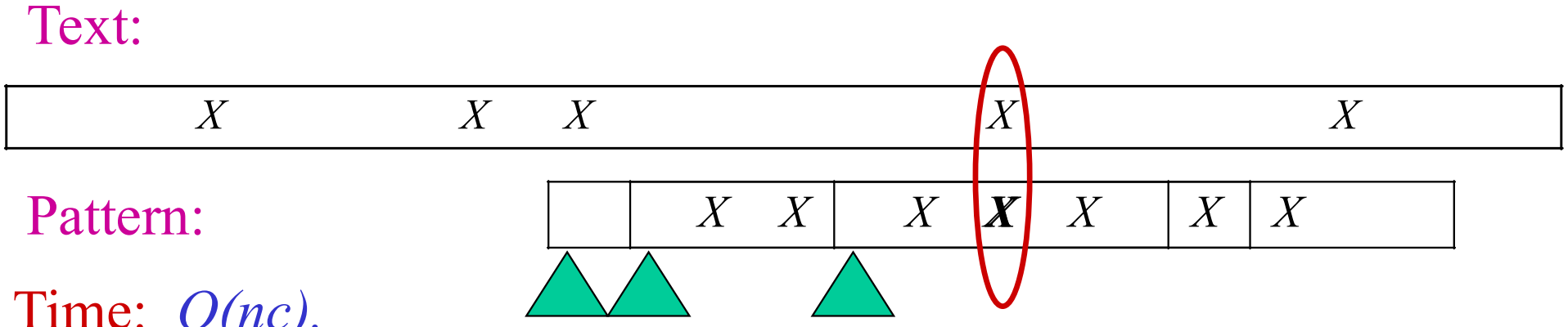
ABBBBBGH ABBBB BGH

Needed: Fast way to compute matches of x to itself.

Such a method exists if x appears in the pattern a very small number of times.

Assume: x appears in pattern c times.

For every occurrence of x in text, update just the appropriate counters of the c occurrences of x in the pattern.



Time: $O(nc)$.

Problem: In general it could be that x occurs in the pattern $O(m)$ times, then total time becomes $O(nm)$. BAD again.

Tradeoff: If x appears in the pattern more than c times, count matches by FFT, in time $O(n \log m)$, per x.

For all x's that appear in the pattern less than c times, count matches (simultaneously) in time $O(nc)$.

How many elements appear at least c times?

$$\frac{m}{c}$$

For these elements , time: *O((m/c) n log m).*

For all other elements, time: *O(nc).*

The optimal case is when they equal, i.e.

$$\frac{mn\log m}{c} = cn$$

$$m\log m = c^2$$

$$\sqrt{m\log m} = c$$

Total Time: $\boxed{O(n\sqrt{m\log m})}$ (A-87,K-87)