# Finding Gale Strings and Orienting Room Partitionings of 1-Oiks

Marta M. Casetti
Julian Merschen
Bernhard von Stengel

Department of Mathematics
London School of Economics

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string**

**s**∈ **G(n,d)** is a bitstring of length **n**

with

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string**
**s**∈ **G(n,d)** is a bitstring of length **n**
with

- exactly **d** bits equal to **1**;

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string**
**s**∈ **G(n,d)** is a bitstring of length **n**
with

- exactly **d** bits equal to **1**;

- (**Gale evenness condition**):

no odd substrings **0**1**0**, **0**111**0**,
**0**11111**0**...

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string** **s**∈ **G(n,d)** is a bitstring of length **n** with

**Example**

**G(6,4) =**

- exactly **d** bits equal to **1**;

- (**Gale evenness condition**):

no odd substrings **0**1**0**, **0**111**0**, **0**11111**0**...

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string** **s**∈ **G(n,d)** is a bitstring of length **n** with

• exactly **d** bits equal to **1**;

• (**Gale evenness condition**):

no odd substrings **0**1**0**, **0**111**0**,

**0**11111**0**...

**Example**

**G(6,4) =**

**111100**

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string** **s**∈ **G(n,d)** is a bitstring of length **n** with

• exactly **d** bits equal to **1**;

• (**Gale evenness condition**):

no odd substrings **0**1**0**, **0**111**0**,

**0**11111**0**...

**Example**

**G(6,4) =**

111100

111001

# What are Gale Strings?

Given **n** > **d**, even **d**, a **Gale string** **s**∈ **G(n,d)** is a bitstring of length **n** with

- exactly **d** bits equal to **1**;

- (**Gale evenness condition**):

no odd substrings 010, 01110, 0111110...

**Example**

**G(6,4) =**

111100
111001
110110
110011
101101
100111
011110
011011
001111

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

`123424`

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
| --- |
| 111100 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
|---|
| 111100 |
| 111001 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
| --- |
| 111100 |
| 111001 |

| 123434 |
| --- |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
| --- |
| 111100 |
| 111001 |

| 123434 |
| --- |
| 111100 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
|---|
| 111100 |
| 111001 |

| 123434 |
|---|
| 111100 |
| 111001 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
| --- |
| 111100 |
| 111001 |

| 123434 |
| --- |
| 111100 |
| 111001 |
| 110110 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
|---|
| 111100 |
| 111001 |

| 123434 |
|---|
| 111100 |
| 111001 |
| 110110 |
| 110011 |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,

the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
|---|
| 111100 |
| 111001 |

| 123434 |
|---|
| 111100 |
| 111001 |
| 110110 |
| 110011 |

| 121314 |
|---|
| (none) |

# When are they completely labeled?

Given a string of labels **{1, . . . , n} =: [n] → [d]**,
the Gale string **s** ∈ **G(n,d)** is **completely labeled**

⇔ the 1 positions in **s** have all labels **1, . . . , d.**

**Examples**

| 123424 |
|---|
| 111100 |
| 111001 |

| 123434 |
|---|
| 111100 |
| 111001 |
| 110110 |
| 110011 |

| 121314 |
|---|
| (none) |

**Do they always come in equal numbers?**

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

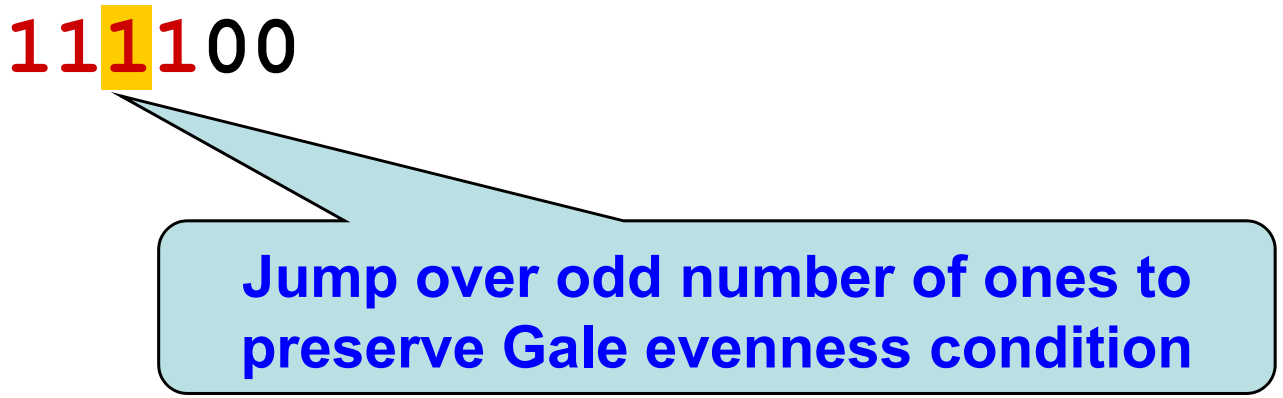
# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**



# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

111100
110110

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

1111100 0111110

110110

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

1111100 0111110

110110 110110

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

```
1111100      0111110      0110111
1101101      1101101
```

# The number of Gale strings is even

**Theorem**

Given a string of labels, the number of completely labeled Gale strings is **even**.

Idea: completely labeled Gale strings = **endpoints of paths** of **almost completely labeled Gale** strings connected by **pivoting steps**.

**What is a pivoting step?**

| | | |
|---|---|---|
| 111100 | 011110 | 011011 |
| 110110 | 110110 | 111001 |

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

123424

111100

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

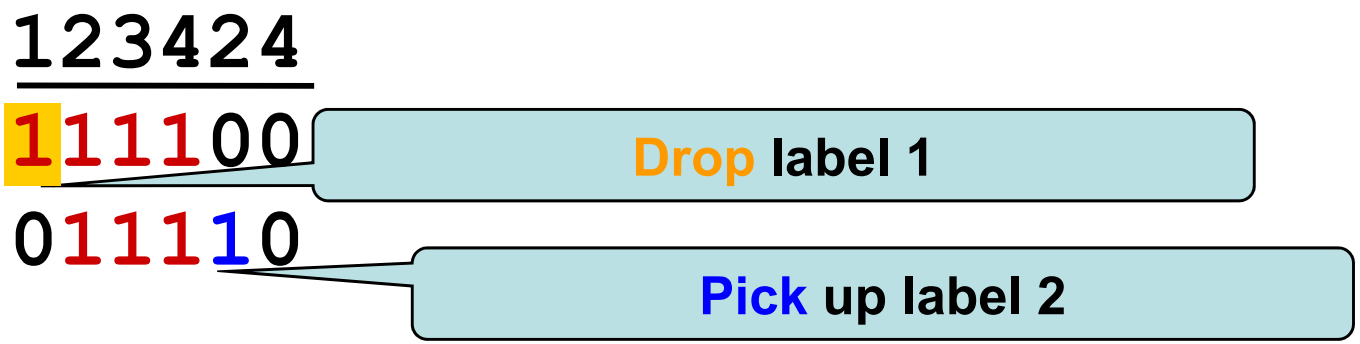
**Example**

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

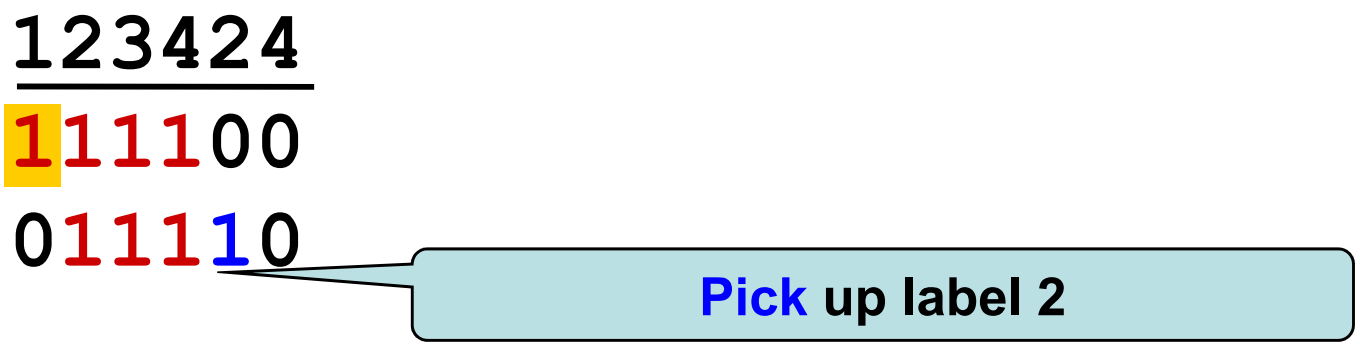


# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**



# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

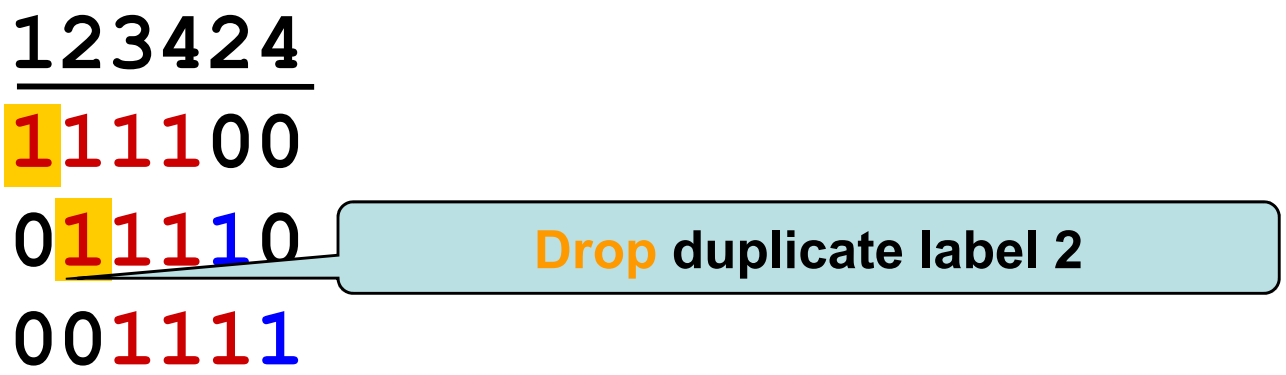
**Example**

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**



# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

```
123424
111100
011110
001111
```

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

123424

111100

011110

001111

011011

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

```
123424
111100
011110
001111
011011
```

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

```
123424
111100
011110
001111
011011
111001
```

# How to obtain second Gale string?

**Almost** completely labeled Gale strings have one missing and one duplicate label.

- drop 1 with a duplicate label, pick up 1 with new label.

**Example**

```
123424
111100
011110
001111
011011
111001
```

**Missing label found again: second Gale string!**

# How good is this pivot algorithm?

**Example [Morris 1994]**

$$\begin{array}{r} 1234564523 \\ \hline 1111110000 \end{array}$$

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
1111110000
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
1111110000
0111111000
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
1111110000
0111111000
0110111100
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
1111110000
0111111000
0110111100
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
1111110000
0111111000
0110111100
0110011110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
0011110011
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
0011110011
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
0011110011
0001111011
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011111010
0011111011
0001111011
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011111010
0011111011
0001111011
0000111111
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011111010
0011111011
0001111011
0000111111
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011111010
0011111011
0001111011
0000111111
1000011111
```

# How good is this pivot algorithm?

**Example [Morris 1994]**

```
1234564523
----------
1111110000
0111111000
0110111100
0110011110
0011011110
0011110110
0011111011
0001111011
0000111111
1000011111
```

In general: exponentially long path in **d** for any missing label.

# Motivation from bimatrix games

**G(n,d)** = facets of cyclic **d**-polytope with **n** vertices

(vertex = position of **1** in Gale string).

# Motivation from bimatrix games

**G(n,d)** = facets of cyclic **d**-polytope with **n** vertices

(vertex = position of **1** in Gale string).

Labels define **d** × **(n − d)** bimatrix game,

completely labeled Gale strings = Nash equilibria.

# Motivation from bimatrix games

**G(n,d)** = facets of cyclic **d**-polytope with **n** vertices

(vertex = position of **1** in Gale string).

Labels define **d** × **(n − d)** bimatrix game,

completely labeled Gale strings = Nash equilibria.

Pivoting = special case of Lemke–Howson algorithm for finding one equilibrium of a bimatrix game.

# Motivation from bimatrix games

**G(n,d)** = facets of cyclic **d**-polytope with **n** vertices

(vertex = position of **1** in Gale string).

Labels define **d** × **(n − d)** bimatrix game,
completely labeled Gale strings = Nash equilibria.

Pivoting = special case of Lemke–Howson algorithm for finding one equilibrium of a bimatrix game.

Labels by [Morris 1994] define **exponentially long** Lemke–Howson paths [McLennan &Tourky 2007; Savani & von Stengel 2006].

# Complexity of finding Gale strings?

# Complexity of finding Gale strings?

GALE STRING

**Input**: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**.

**Question**: Is there a completely labeled Gale string **s**?

# Complexity of finding Gale strings?

GALE STRING

**Input**: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**.

**Question**: Is there a completely labeled Gale string **s**?

ANOTHER GALE STRING

Input: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**, and a completely labeled Gale string **s**.

Output: A completely labeled Gale string **s′** ≠ **s**.

# Complexity of finding Gale strings?

GALE STRING

**Input**: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**.

**Question**: Is there a completely labeled Gale string **s**?

ANOTHER GALE STRING

Input: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**, and a completely labeled Gale string **s**.

Output: A completely labeled Gale string **s′** ≠ **s**.

**Theorem**

GALE STRING and ANOTHER GALE STRING are in **P**.

# Complexity of finding Gale strings?

GALE STRING

**Input**: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**.

**Question**: Is there a completely labeled Gale string **s**?

ANOTHER GALE STRING

Input: Labels **[n]** → **[d]**, where **d** is even and **d** < **n**, and a completely labeled Gale string **s**.

Output: A completely labeled Gale string **s′** ≠ **s**.

**Theorem**

GALE STRING and ANOTHER GALE STRING are in **P**.

# Proof: Reduction to Perfect Matching

# Proof: Reduction to Perfect Matching

PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

# Proof: Reduction to Perfect Matching

PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

# Proof: Reduction to Perfect Matching

PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`

# Proof: Reduction to Perfect Matching

PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`

Idea: neighbours in
label string become
neighbours in graph.

# Proof: Reduction to Perfect Matching

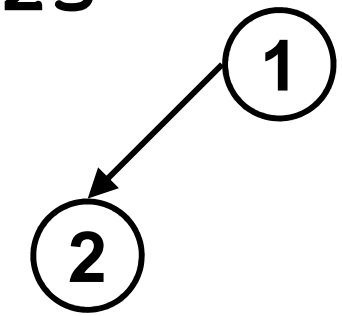
PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`




Idea: neighbours in label string become neighbours in graph.

# Proof: Reduction to Perfect Matching

PERFECT MATCHING

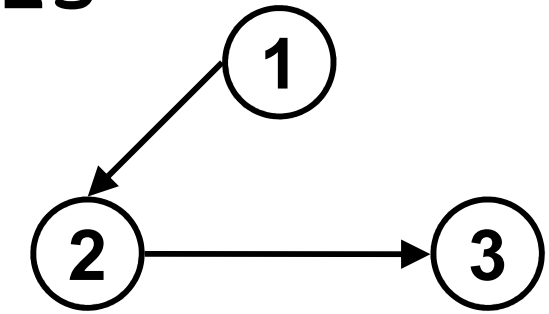
Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`

Idea: neighbours in label string become neighbours in graph.



# Proof: Reduction to Perfect Matching

PERFECT MATCHING

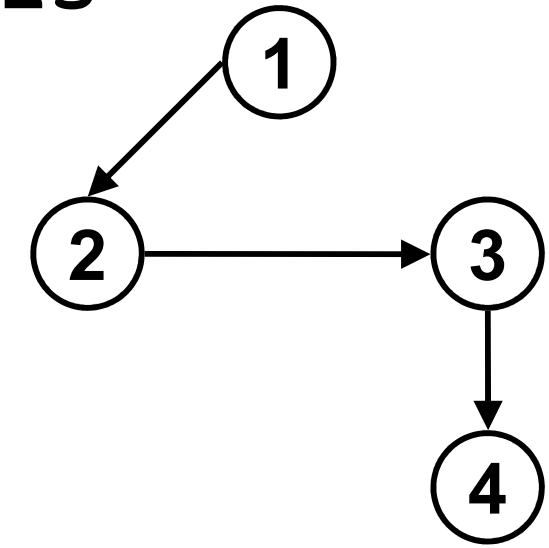
Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`

Idea: neighbours in label string become neighbours in graph.



# Proof: Reduction to Perfect Matching

PERFECT MATCHING

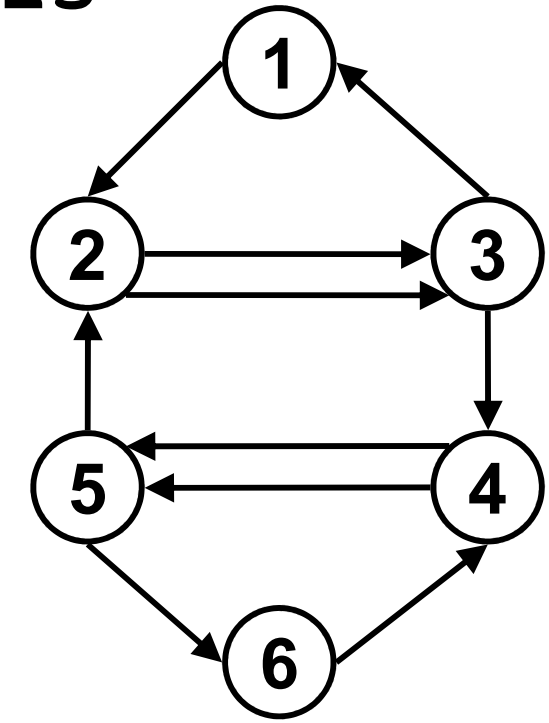
Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`

Idea: neighbours in label string become neighbours in graph.



# Proof: Reduction to Perfect Matching

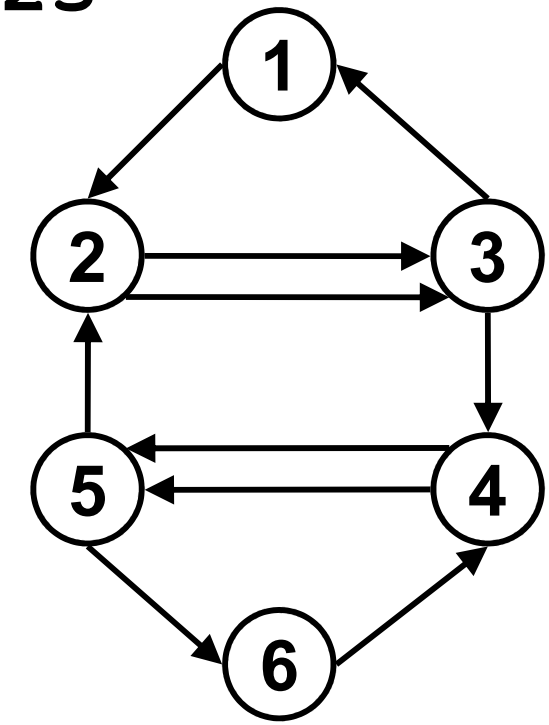
PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else “no”.

Which is in **P** [Edmonds 1965].

**Example** `1234564523`



# Proof: Reduction to Perfect Matching

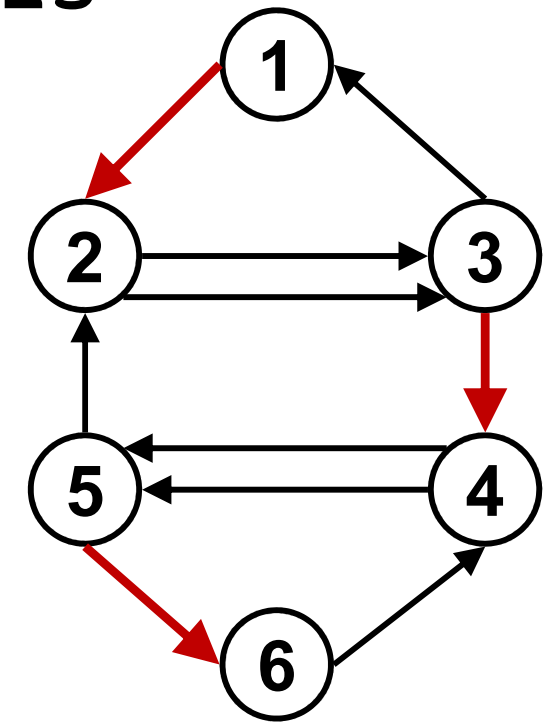
PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else "no".

Which is in **P** [Edmonds 1965].

**Example** 1234564523




**Perfect Matching** =

**Completely labeled Gale String**

1234564523
1111110000

# Proof: Reduction to Perfect Matching

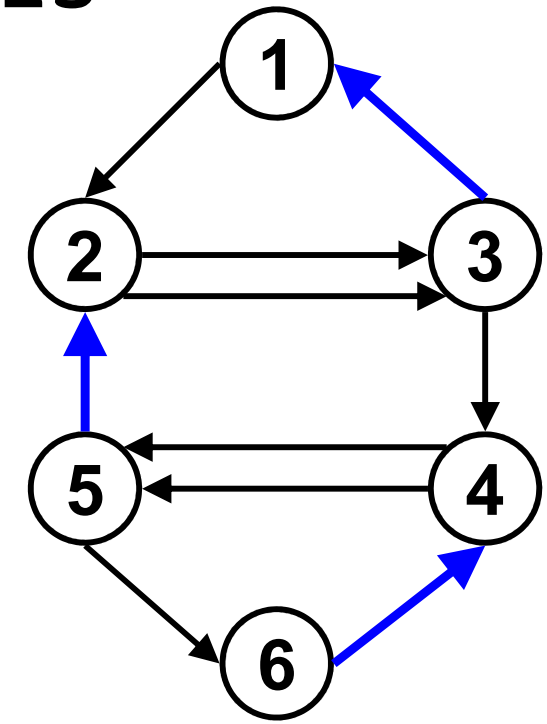
PERFECT MATCHING

Input: Graph **G** on **d** vertices.

Output: Perfect matching of **G**, if one exists; else "no".

Which is in **P** [Edmonds 1965].

**Example** 1234564523




**Perfect Matching** =

**Completely labeled Gale String**

1234564523
1000011111

# Sign of Gale string

# Sign of Gale string

The **sign** $\oplus$ or $\ominus$ of a completely labeled Gale string is the sign of the permutation of labels for the `11` positions in **s**.

# Sign of Gale string

The **sign** $\oplus$ or $\ominus$ of a completely labeled Gale string is the sign of the permutation of labels for the `11` positions in **s**.

**Examples**

| `12 34 24` |
|---|
| `11 11 00` |

# Sign of Gale string

The **sign** ⊕ or ⊖ of a completely labeled Gale string is the sign of the permutation of labels for the **11** positions in **s**.

**Examples**

| **12** | **34** | **24** |
|---|---|---|
| **11** | **11** | **00** |

**12 34** has even number of inversions ⇒ 11 11 00 has sign ⊕.

# Sign of Gale string

The **sign** ⊕ or ⊖ of a completely labeled Gale string is the sign of the permutation of labels for the **11** positions in **s**.

**Examples**

| 12 | 34 | 24 |
|---|---|---|
| 11 | 11 | 00 |

**12 34** has even number of inversions ⇒ 11 11 00 has sign ⊕.

| 1 | 23 | 42 | 4 |
|---|---|---|---|
| 1 | 11 | 00 | 1 |

**41 23** has odd number of inversions ⇒ 1 11 00 1 has sign ⊖.

# Sign of Gale string

The **sign** ⊕ or ⊖ of a completely labeled Gale string is the sign of the permutation of labels for the **11** positions in **s**.

**Examples**

| 12 | 34 | 24 |
|---|---|---|
| 11 | 11 | 00 |

**12 34** has even number of inversions ⇒ 11 11 00 has sign ⊕.

| 1 | 23 | 42 | 4 |
|---|---|---|---|
| 1 | 11 | 00 | 1 |

**41 23** has odd number of inversions ⇒ 1 11 00 1 has sign ⊖.

Given a string of labels, any completely labeled Gale string has sign ⊕ or ⊖.

# For every ⊕ there is a ⊖

# For every ⊕ there is a ⊖

Theorem

Number of completely labeled Gale strings of sign ⊕

= Number of completely labeled Gale strings of sign ⊖

# For every ⊕ there is a ⊖

Theorem

Number of completely labeled Gale strings of sign ⊕

= Number of completely labeled Gale strings of sign ⊖

Proof: Endpoints of paths have opposite sign.

# For every ⊕ there is a ⊖

Theorem

Number of completely labeled Gale strings of sign ⊕
= Number of completely labeled Gale strings of sign ⊖

Proof: Endpoints of paths have opposite sign.

Define **signs** of an **almost completely labeled** Gale string, by replacing **duplicate** with missing label (**two** possibilities one **new** one **old**) ⇒ one ⊕, one ⊖ permutation.

# For every ⊕ there is a ⊖

Theorem

Number of completely labeled Gale strings of sign ⊕
= Number of completely labeled Gale strings of sign ⊖

Proof: Endpoints of paths have opposite sign.

Define **signs** of an **almost completely labeled** Gale string, by replacing **duplicate** with missing label (**two** possibilities one **new** one **old**) ⇒ one ⊕, one ⊖ permutation.

**Example**

$$\frac{123424}{011110}$$

# For every ⊕ there is a ⊖

Theorem

Number of completely labeled Gale strings of sign ⊕
= Number of completely labeled Gale strings of sign ⊖

Proof: Endpoints of paths have opposite sign.

Define **signs** of an **almost completely labeled** Gale string, by replacing **duplicate** with missing label (**two** possibilities one **new** one **old**) ⇒ one ⊕, one ⊖ permutation.

**Example**

| 1342 | 123424 |
|---|---|
| sign ⊕ | 011110 |

# For every ⊕ there is a ⊖

Theorem

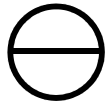
Number of completely labeled Gale strings of sign ⊕
= Number of completely labeled Gale strings of sign ⊖

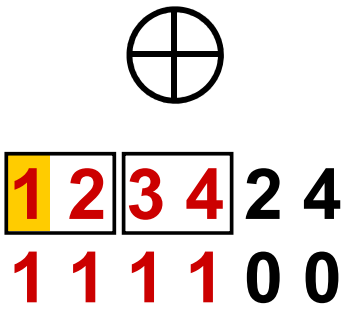
Proof: Endpoints of paths have opposite sign.

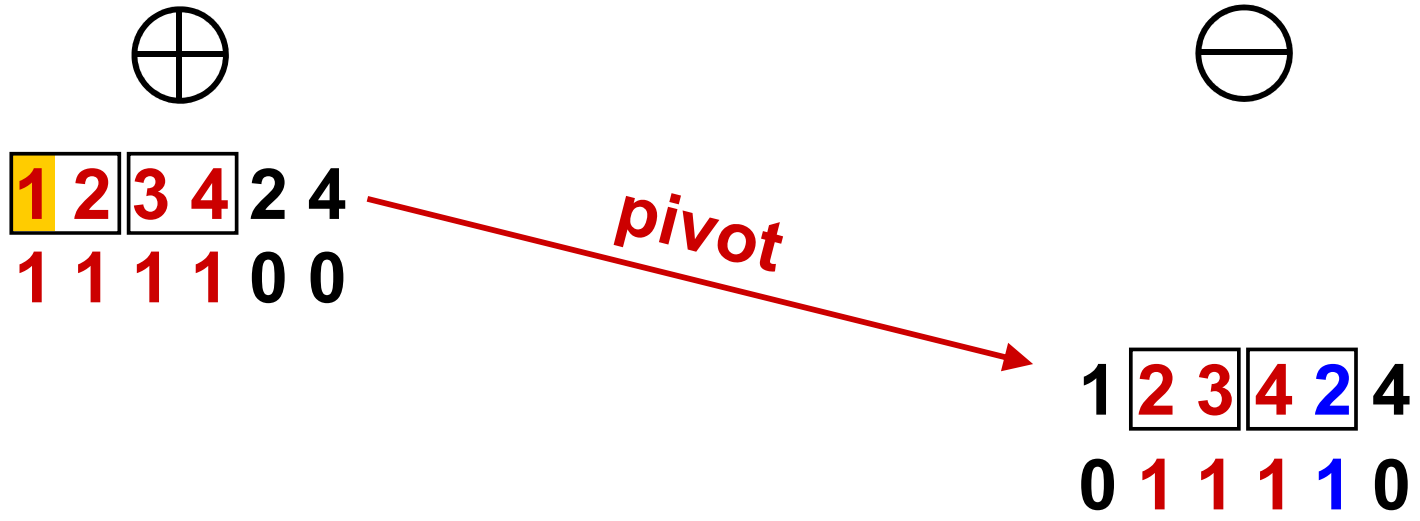
Define **signs** of an **almost completely labeled** Gale string, by replacing **duplicate** with missing label (**two** possibilities one **new** one **old**) ⇒ one ⊕, one ⊖ permutation.

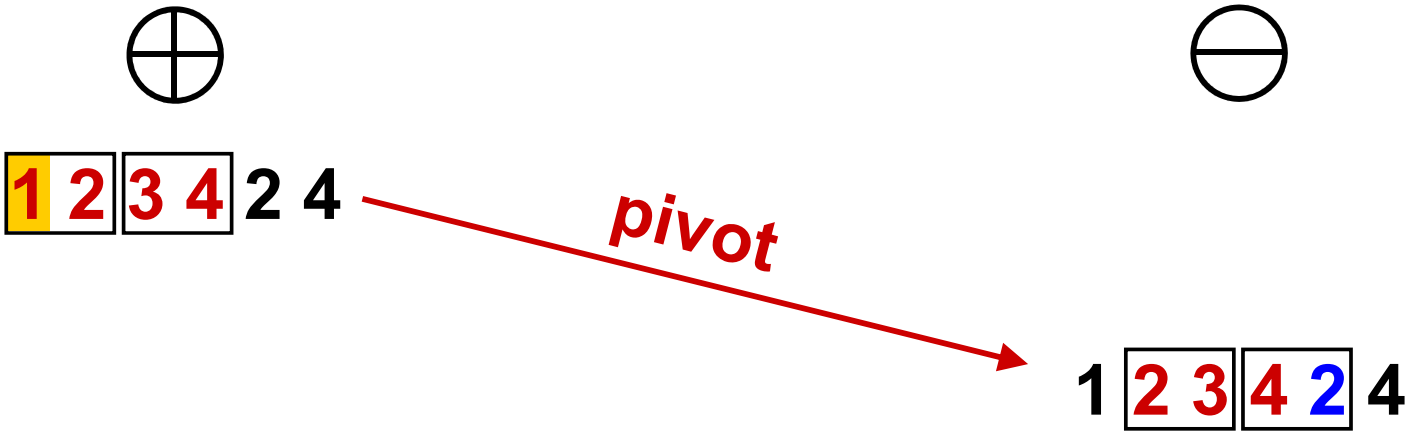
**Example**

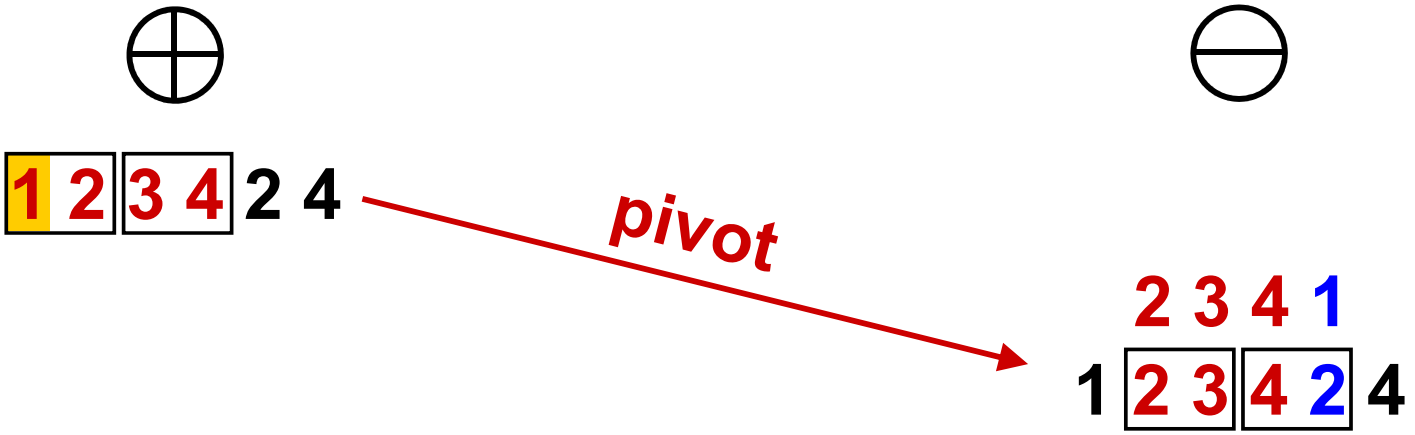
| 1342 | 123424 | 2341 |
|---|---|---|
| sign ⊕ | 011110 | sign ⊖ |

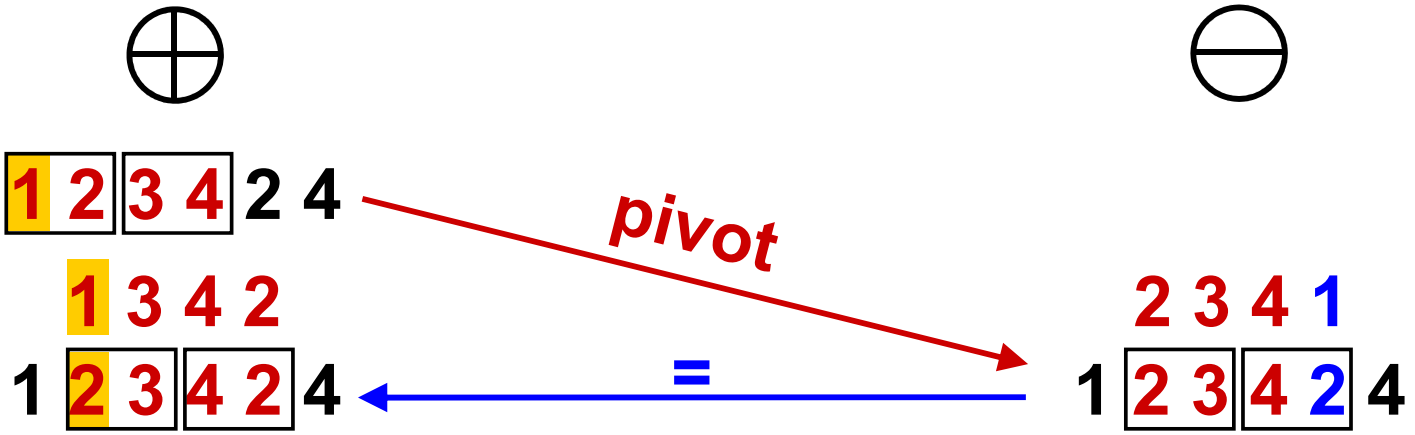


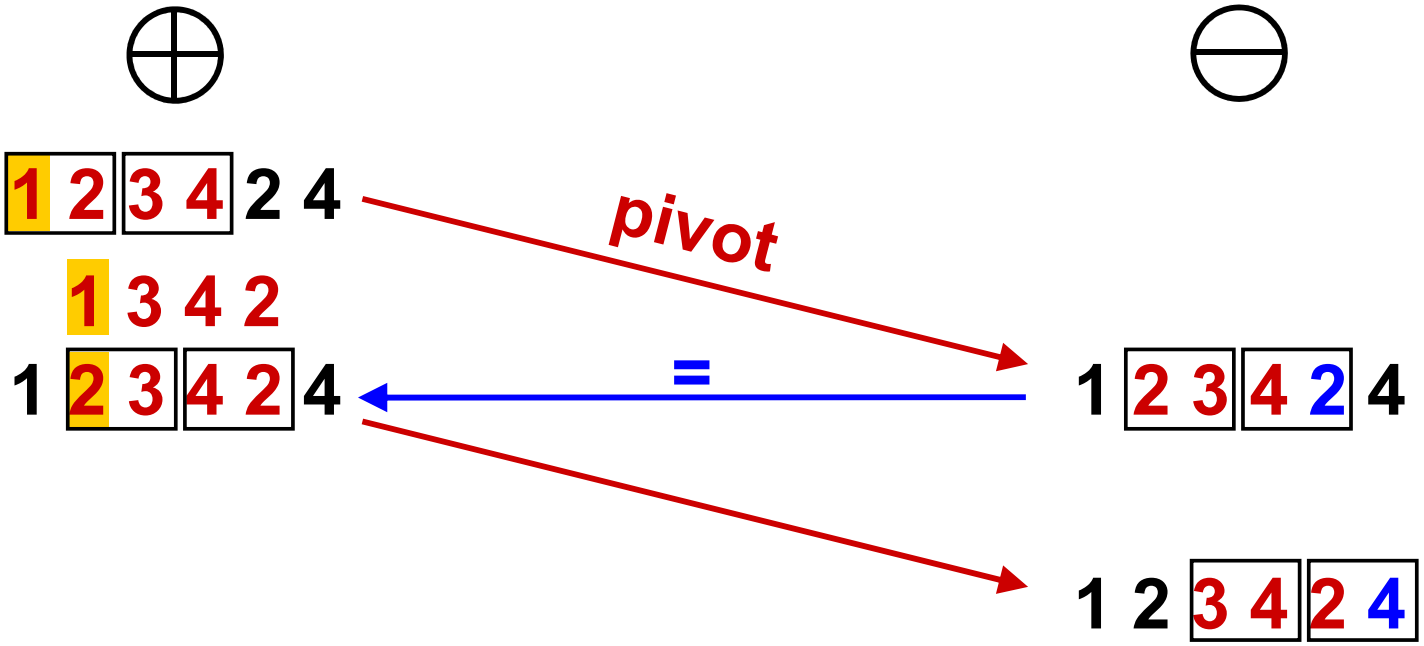

1 2 3 4 2 4
1 1 1 1 0 0

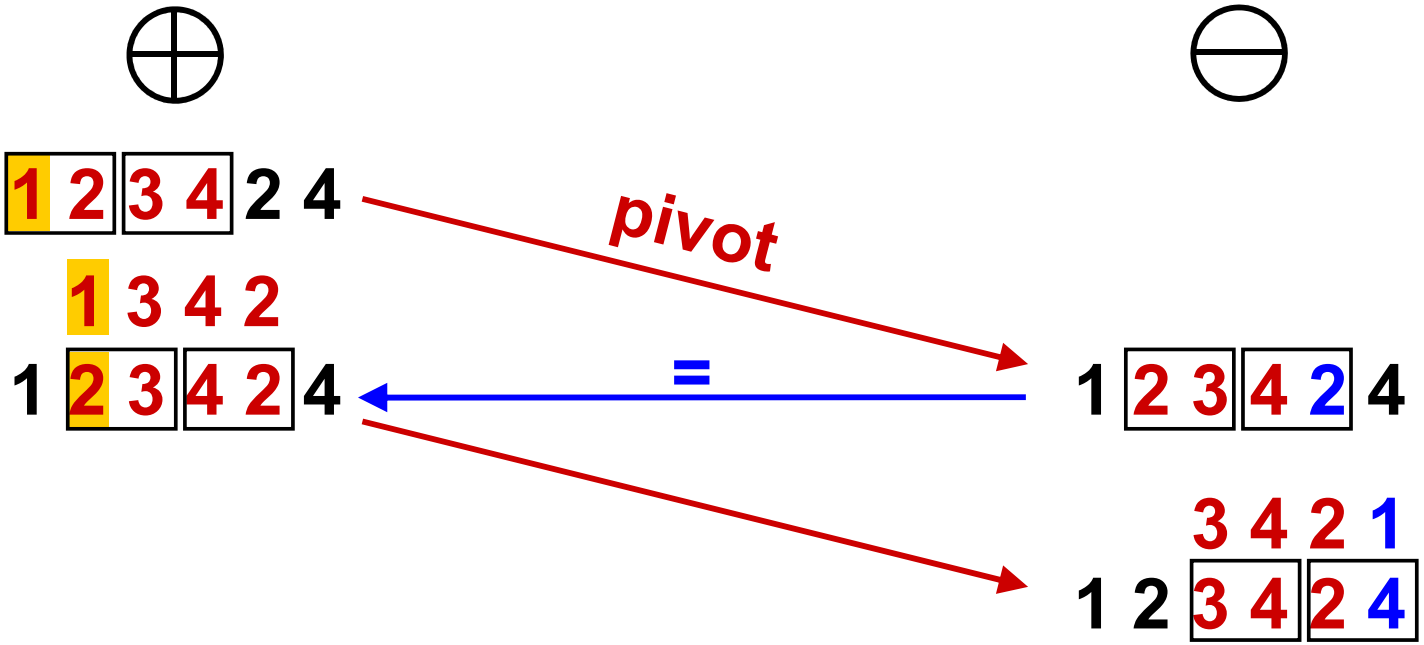

1 2 3 4 2 4
1 1 1 1 0 0
pivot
1 2 3 4 2 4
0 1 1 1 1 0

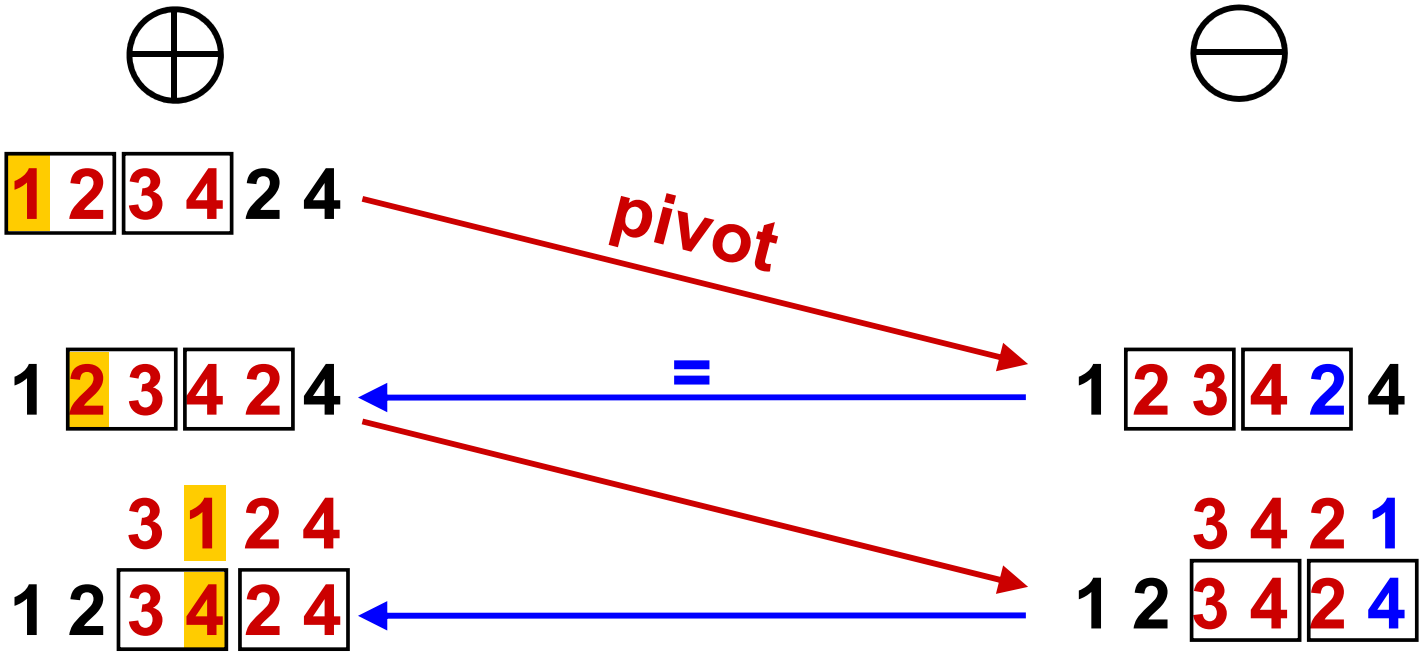

1 2 3 4 2 4
pivot
1 2 3 4 2 4

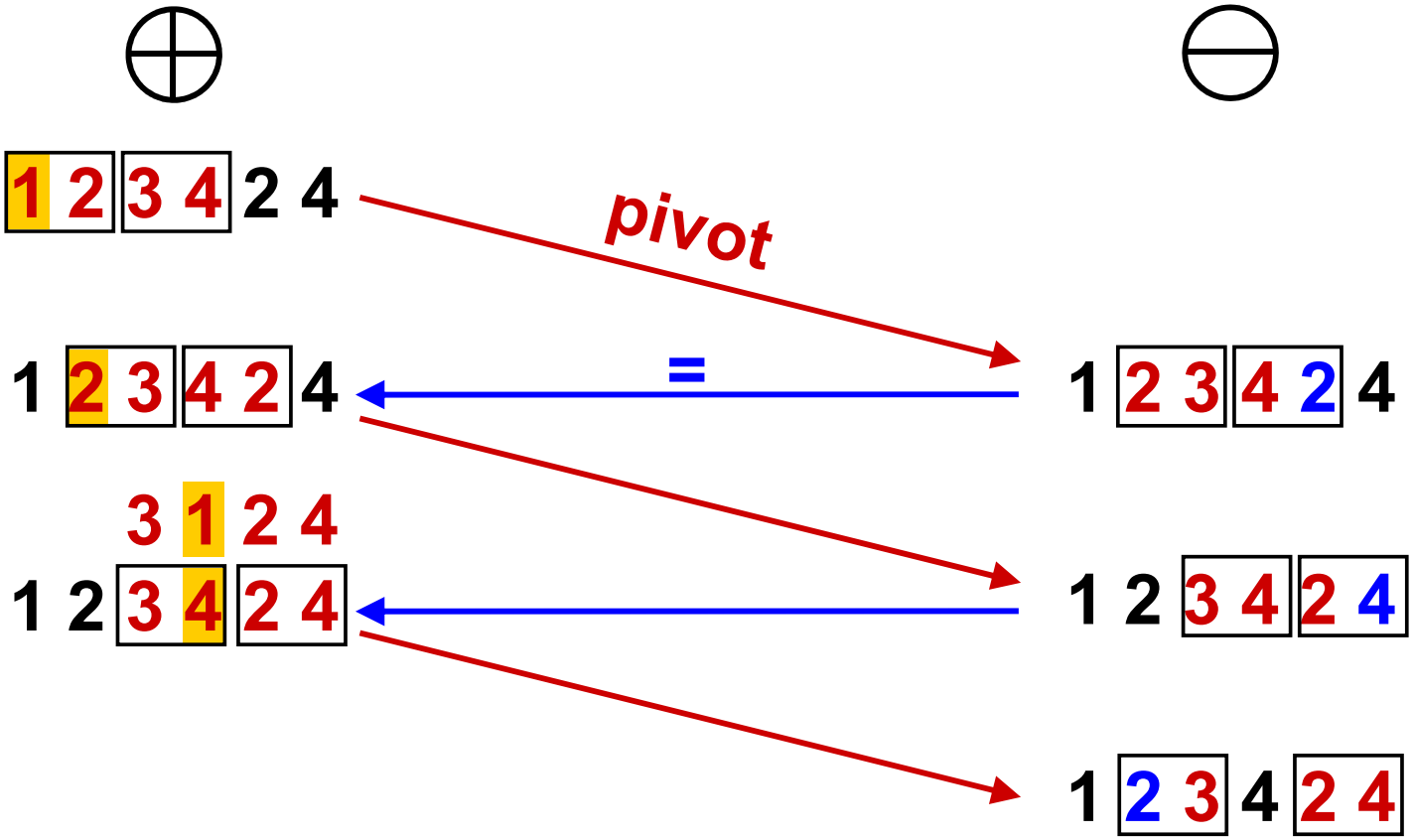

1 2 3 4 2 4
pivot
2 3 4 1
1 2 3 4 2 4

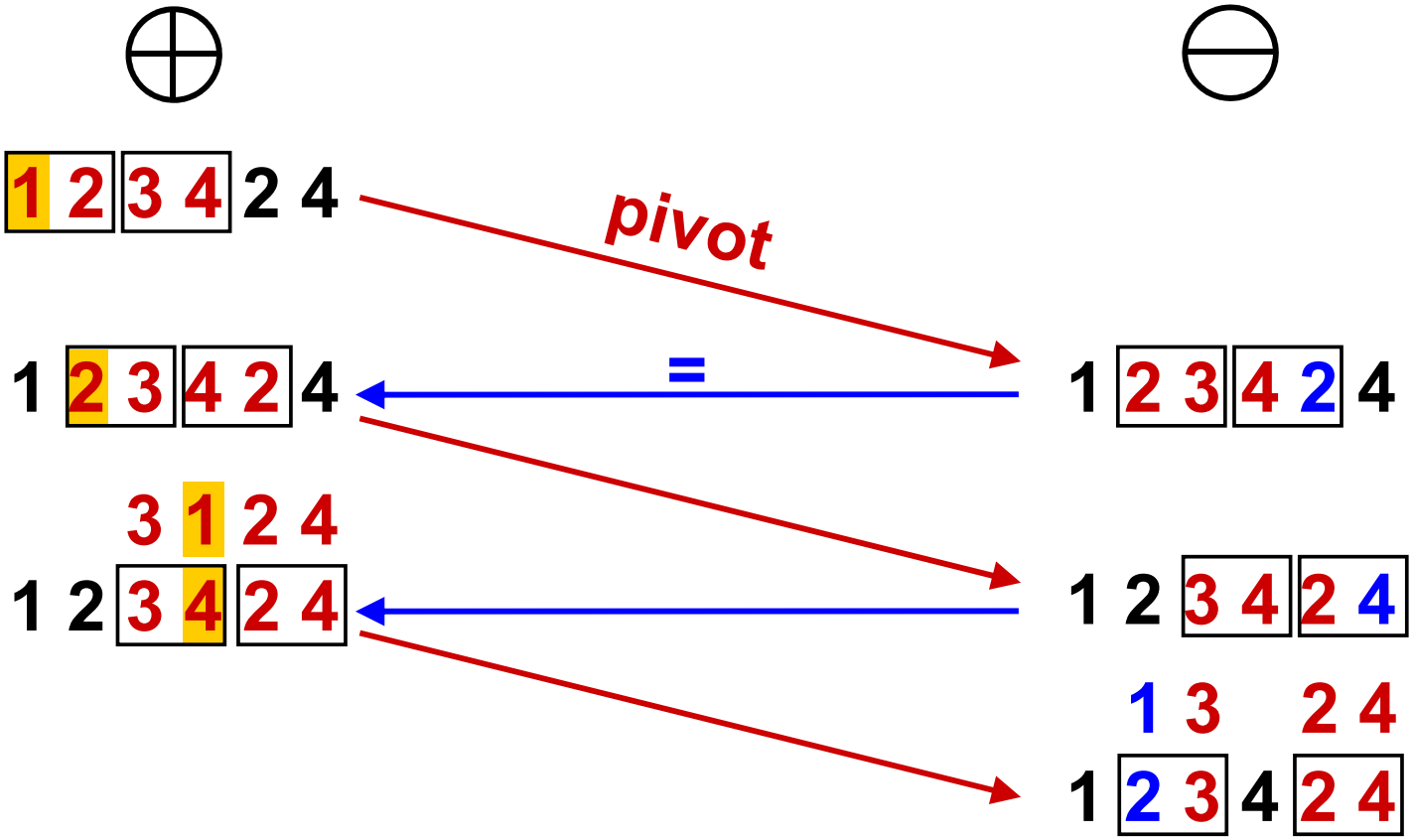

1 2 3 4 2 4
pivot
1 3 4 2
2 3 4 1
1 2 3 4 2 4
=
1 2 3 4 2 4

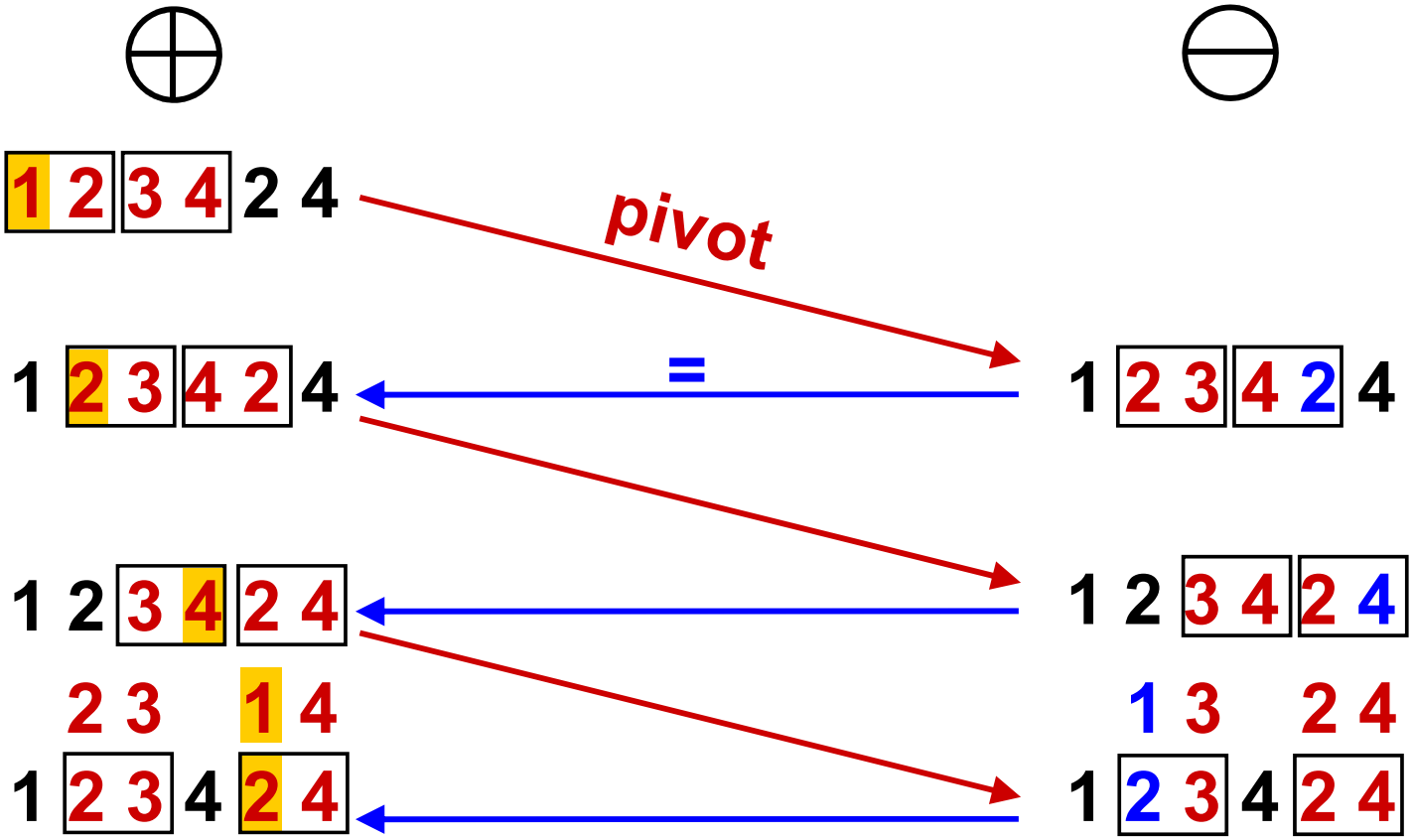

⊕
⊖
1 2 3 4 2 4
pivot
1 3 4 2
1 2 3 4 2 4
=
1 2 3 4 2 4
1 2 3 4 2 4

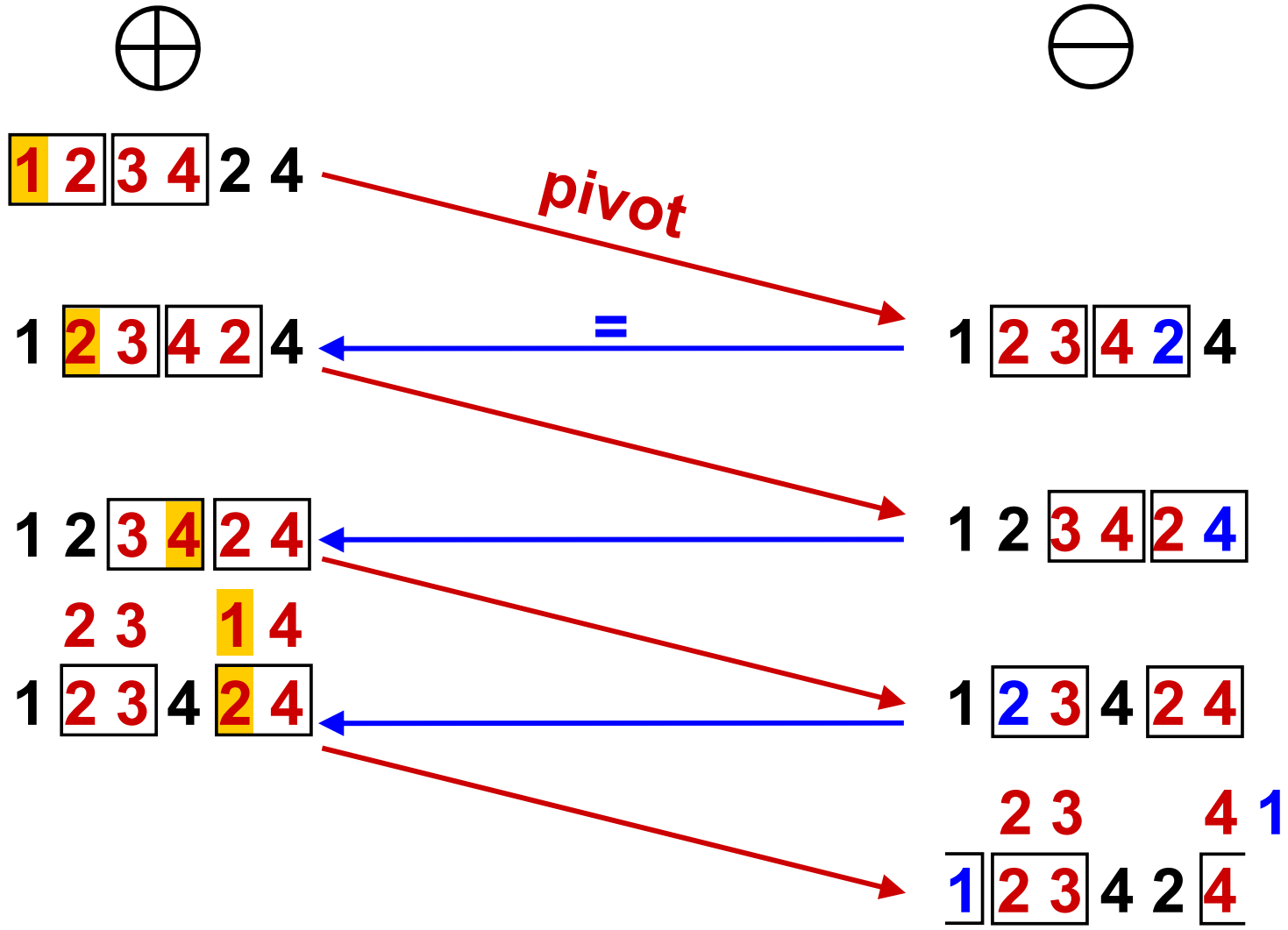

⊕
⊖
1 2 3 4 2 4
pivot
1 3 4 2
1 2 3 4 2 4
=
1 2 3 4 2 4
3 4 2 1
1 2 3 4 2 4

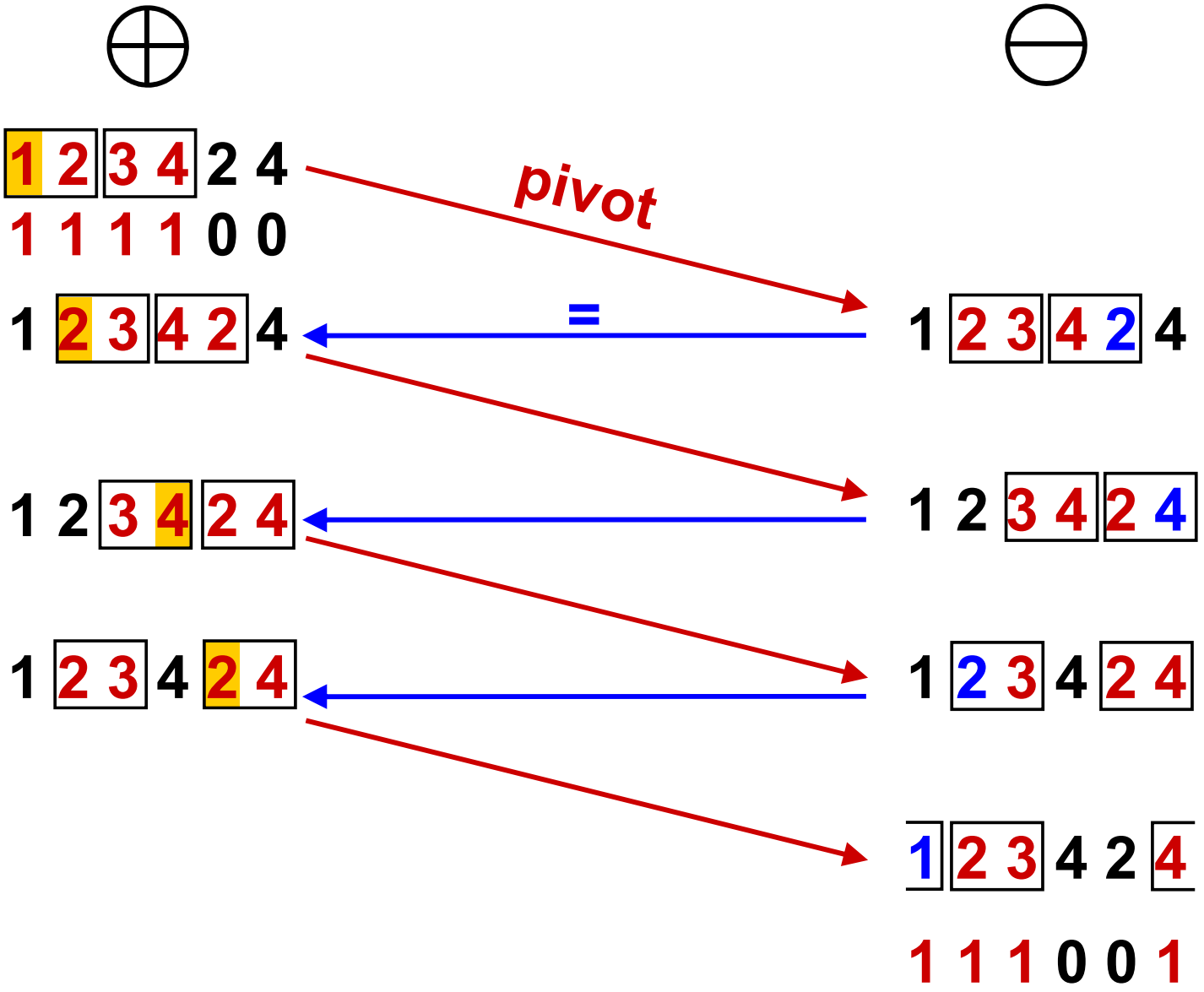

⊕
⊖
1 2 3 4 2 4
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
3 1 2 4
1 2 3 4 2 4
3 4 2 1
1 2 3 4 2 4

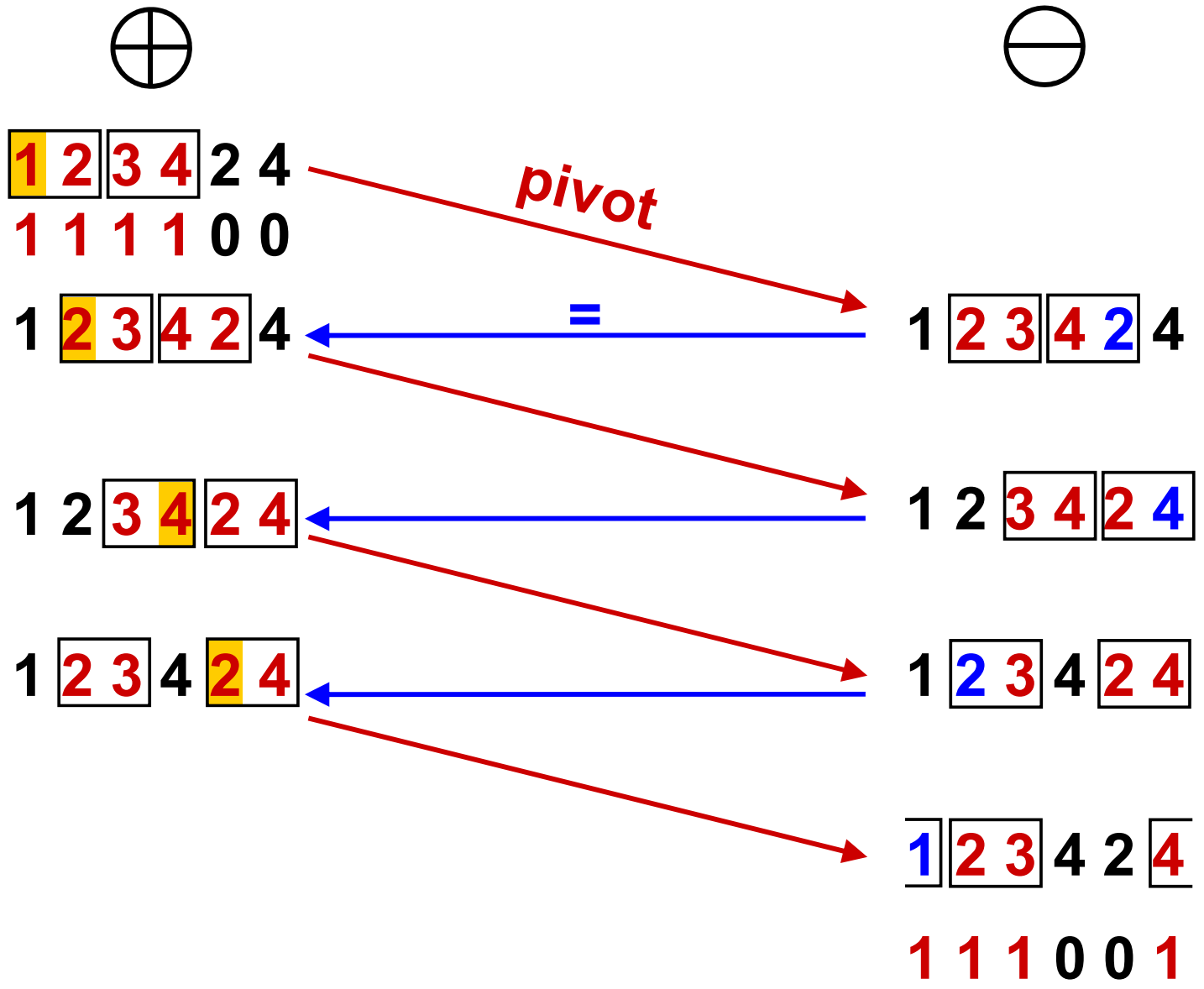

⊕
⊖
1 2 3 4 2 4
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
3 1 2 4
1 2 3 4 2 4
1 2 3 4 2 4
1 2 3 4 2 4


⊕
⊖
1 2 3 4 2 4
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
3 1 2 4
1 2 3 4 2 4
1 2 3 4 2 4
1 3 2 4
1 2 3 4 2 4

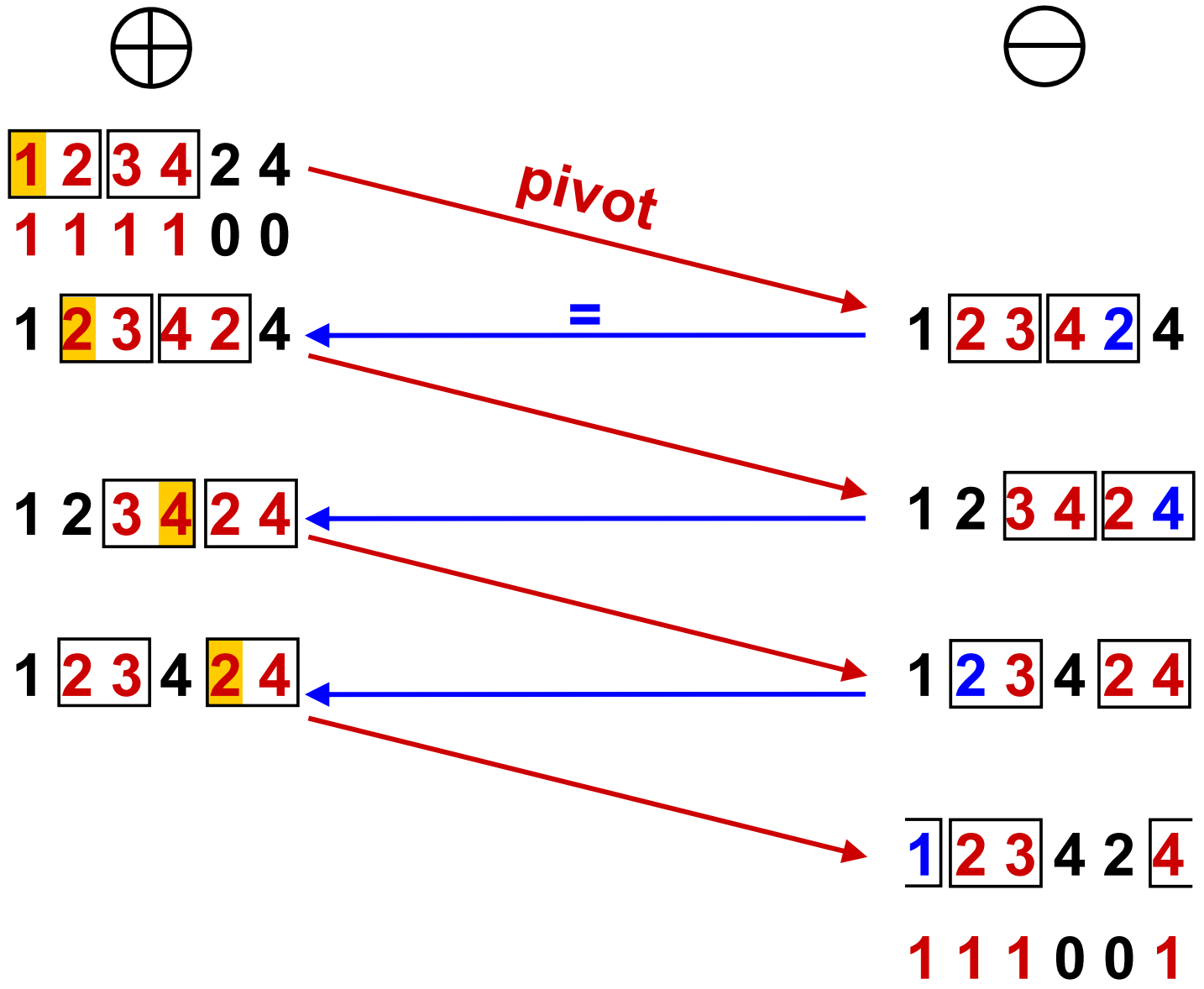

⊕
⊖
1 2 3 4 2 4
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
1 2 3 4 2 4
2 3 1 4
1 3 2 4
1 2 3 4 2 4
1 2 3 4 2 4


⊕
⊖
1 2 3 4 2 4
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
1 2 3 4 2 4
2 3 1 4
1 2 3 4 2 4
1 2 3 4 2 4
2 3 4 1
1 2 3 4 2 4


⊕
⊖
1 2 3 4 2 4
1 1 1 1 0 0
pivot
1 2 3 4 2 4
=
1 2 3 4 2 4
1 2 3 4 2 4
1 2 3 4 2 4
1 2 3 4 2 4
1 2 3 4 2 4
1 1 1 0 0 1




**Directed path: PPAD membership.**




**Directed path: PPAD membership.**

# What are oiks?

# What are oiks?

**So far**: Completely labeled Gale strings

= perfect matchings in an Euler graph (with multiple edges).

# What are oiks?

**So far**: Completely labeled Gale strings
= perfect matchings in an Euler graph (with multiple edges).

**Euler complex** or **Oik** of dimension **d−1** (or **(d−1)-oik**)
= collection of sets of size **d** called **rooms**, so that for each room **R** and vertex **v** ∈ **R** the wall **R** − {**v**} is contained in an **even** number of rooms. [Todd 1974, Edmonds 2009]

# What are oiks?

**So far**: Completely labeled Gale strings
= perfect matchings in an Euler graph (with multiple edges).

**Euler complex** or **Oik** of dimension **d−1** (or **(d−1)-oik**)
= collection of sets of size **d** called **rooms**, so that for each room **R** and vertex **v** ∈ **R** the wall **R** − {**v**} is contained in an **even** number of rooms. [Todd 1974, Edmonds 2009]

if the wall is in exactly **two** rooms:
oik = (combinatorial) manifold.

# What are oiks?

**So far**: Completely labeled Gale strings
= perfect matchings in an Euler graph (with multiple edges).

**Euler complex** or **Oik** of dimension **d−1** (or **(d−1)-oik**)
= collection of sets of size **d** called **rooms**, so that for each room **R** and vertex **v** ∈ **R** the wall **R** − {**v**} is contained in an **even** number of rooms. [Todd 1974, Edmonds 2009]

if the wall is in exactly **two** rooms:
oik = (combinatorial) manifold.

**Example** **1** **2 3** **4 2** **4**

# What are oiks?

**So far**: Completely labeled Gale strings
= perfect matchings in an Euler graph (with multiple edges).

**Euler complex** or **Oik** of dimension **d−1** (or **(d−1)-oik**)
= collection of sets of size **d** called **rooms**, so that for each room **R** and vertex **v** ∈ **R** the wall **R** − {**v**} is contained in an **even** number of rooms. [Todd 1974, Edmonds 2009]

if the wall is in exactly **two** rooms:
oik = (combinatorial) manifold.

**Example** **1 2 3 4 2 4**

**Let R be 2 3 and R – {2} = {3}.**
**The wall 3 is in two rooms, 2 3 and 3 4.**

# Rooms partitions in 1-oiks have a sign

# Rooms partitions in 1-oiks have a sign

**Room partitioning** of oik = partition of all vertices into rooms.

# Rooms partitions in 1-oiks have a sign

**Room partitioning** of oik = partition of all vertices into rooms.

**Theorem** [Edmonds & Sanita 2010]

In any oik, the number of room partitionings is even.

# Rooms partitions in 1-oiks have a sign

**Room partitioning** of oik = partition of all vertices into rooms.

**Theorem** [Edmonds & Sanita 2010]

In any oik, the number of room partitionings is even.

**Theorem**

In a 1-oik, number of $\oplus$ room partitionings

= number of $\ominus$ room partitionings.

# Rooms partitions in 1-oiks have a sign

**Room partitioning** of oik = partition of all vertices into rooms.

**Theorem** [Edmonds & Sanita 2010]

In any oik, the number of room partitionings is even.

**Theorem**

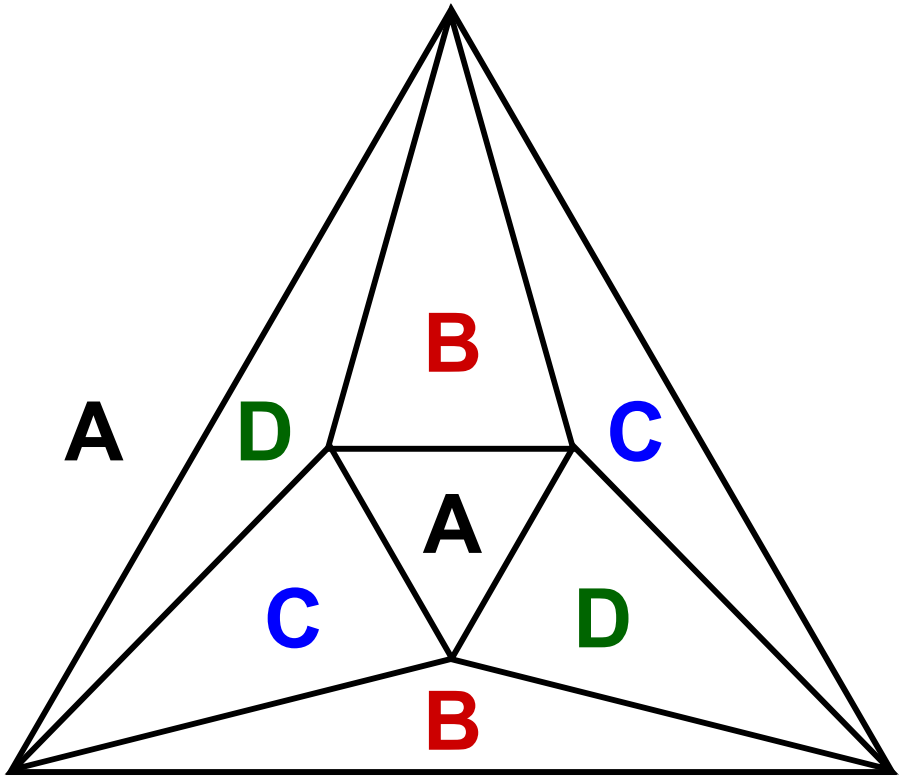
In a 1-oik, number of $\oplus$ room partitionings

= number of $\ominus$ room partitionings.

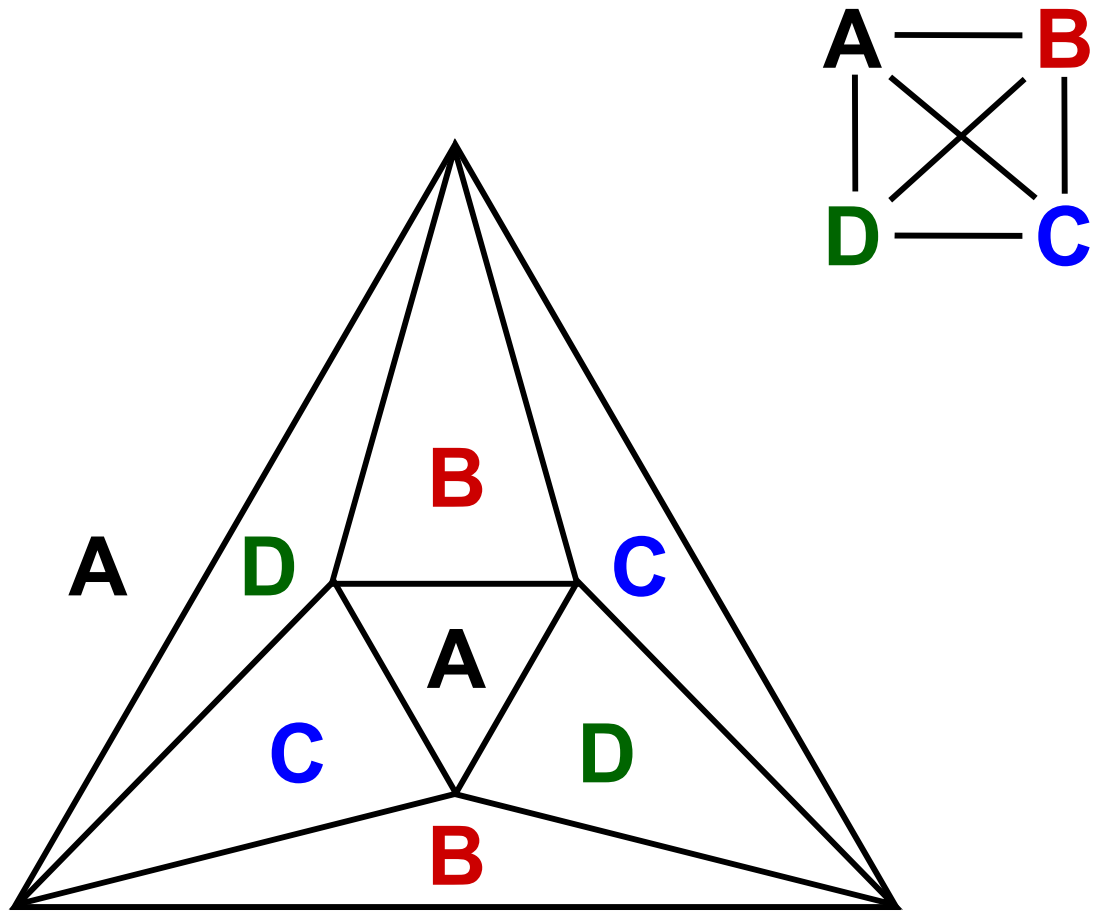
Idea:
1-oiks are Euler graphs, Euler tour = label string,
completely labeled Gale string = perfect matching
= room partitioning.

# Sign for room partitionings if d > 2?

# Sign for room partitionings if d > 2?



# Sign for room partitionings if d > 2?



# Thank you