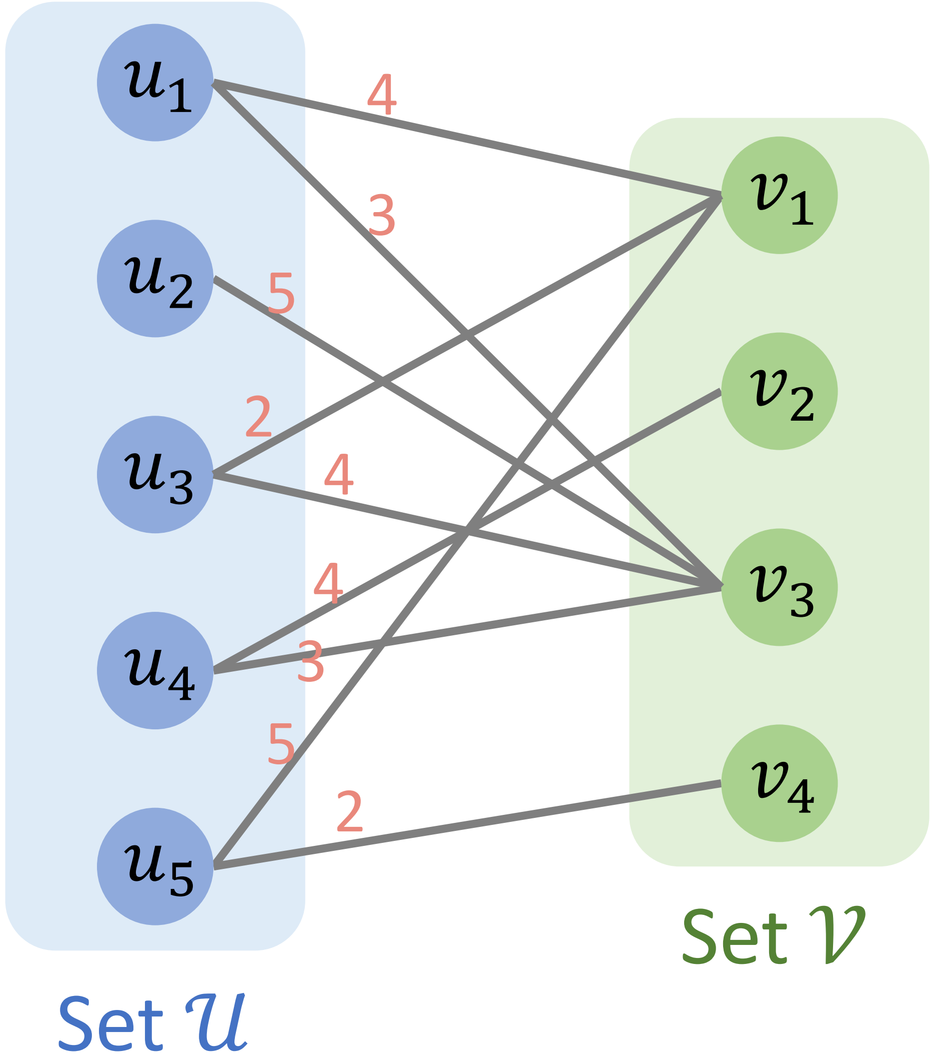
# Maximum-Weight Bipartite Matching

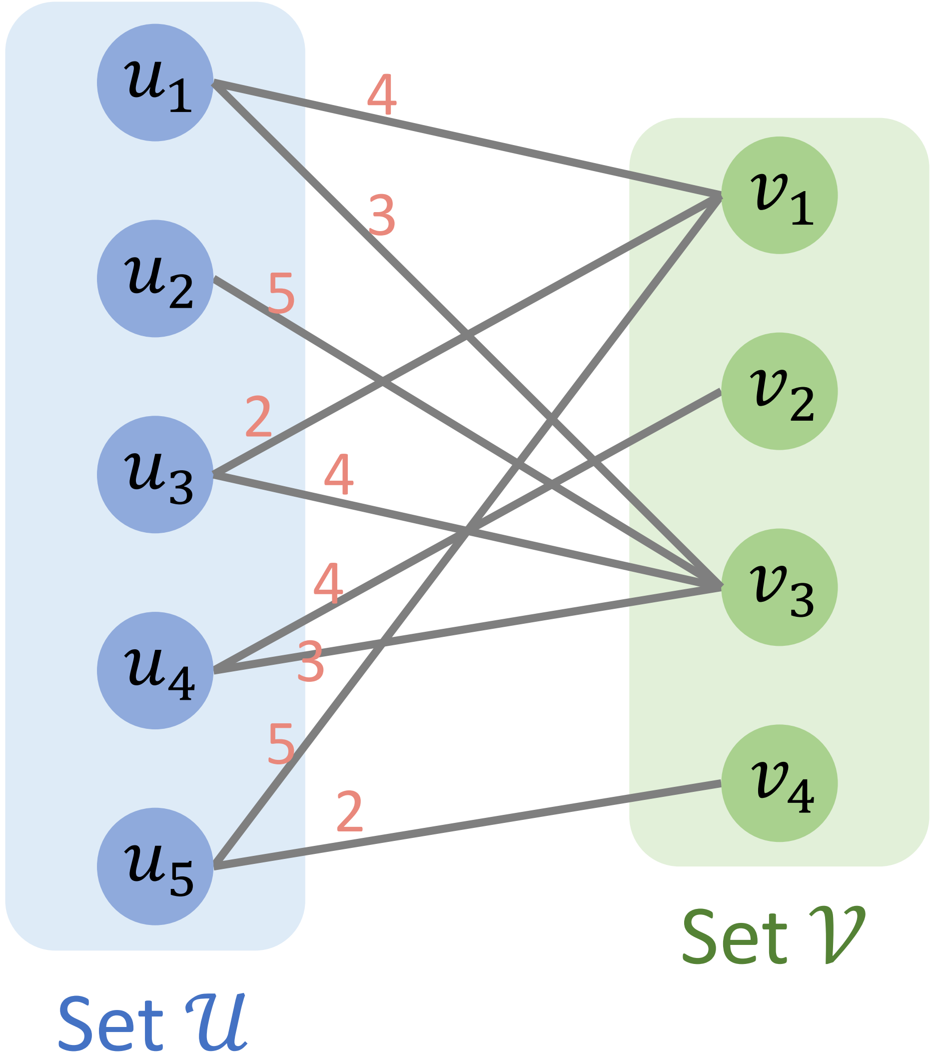
**Shusen Wang**

# Weighted Bipartite Graph




- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Edges have weights: $w_{ij}$.

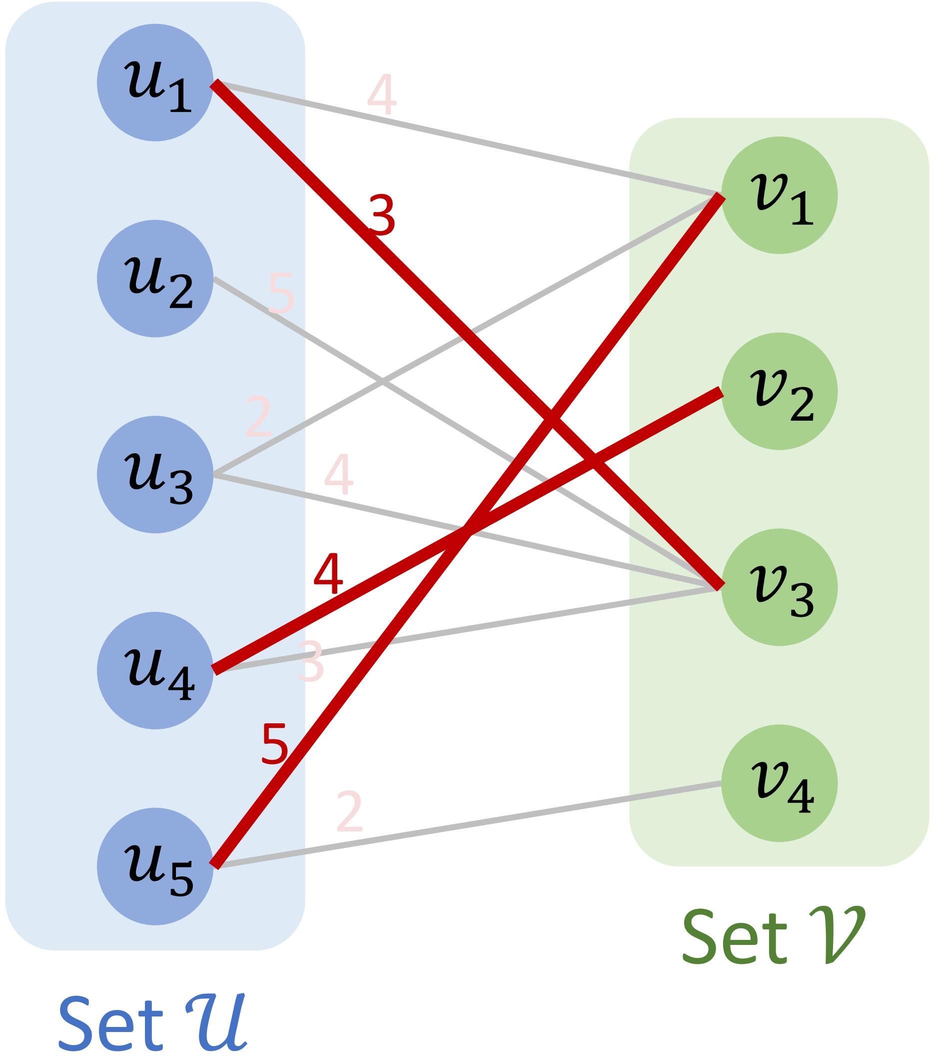
# Weighted Bipartite Graph




- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Edges have weights: $w_{ij}$.
- Adjacency matrix:

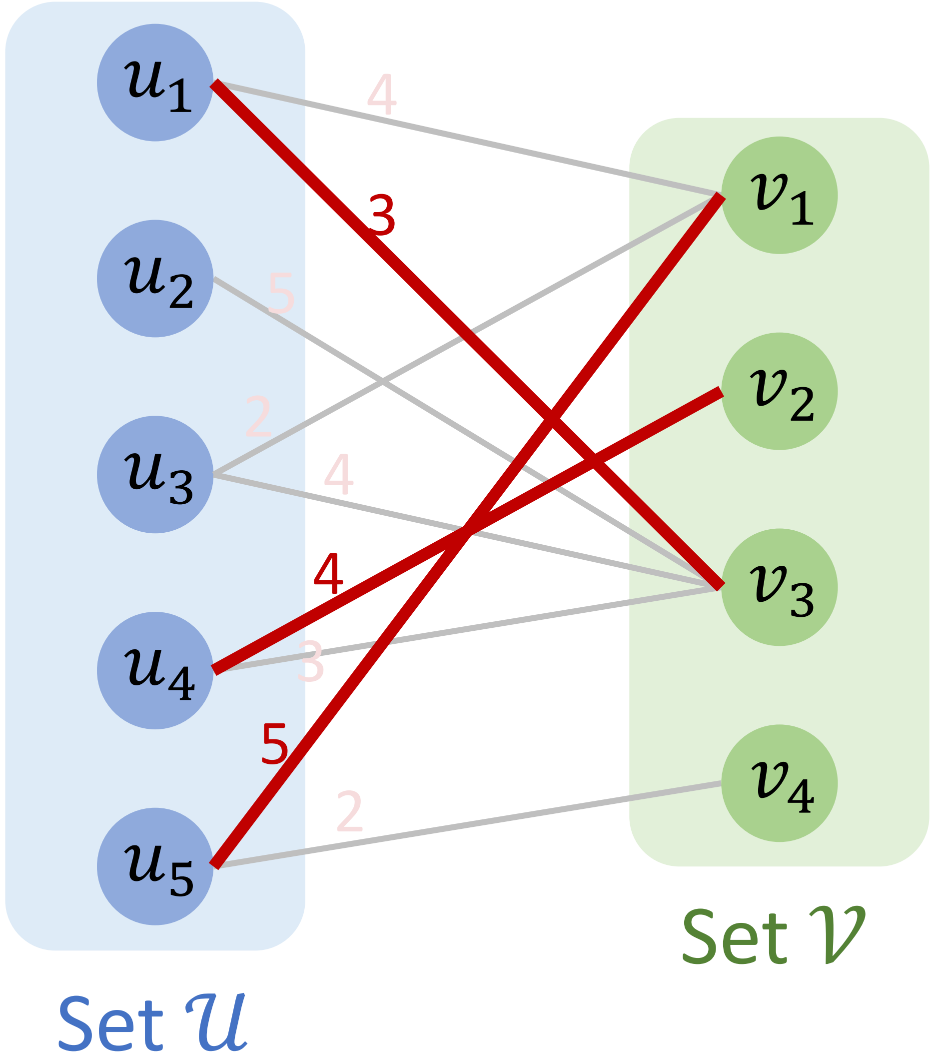
| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | **4** | 0 | **3** | 0 |
| $u_2$ | 0 | 0 | **5** | 0 |
| $u_3$ | **2** | 0 | **4** | 0 |
| $u_4$ | 0 | **4** | **3** | 0 |
| $u_5$ | **5** | 0 | 0 | **2** |

# Bipartite Matching in Weighted Graph




- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Matching is a subset of edges without common vertices.
- Denote the matching by set $\mathcal{S} \subseteq \mathcal{E}$.

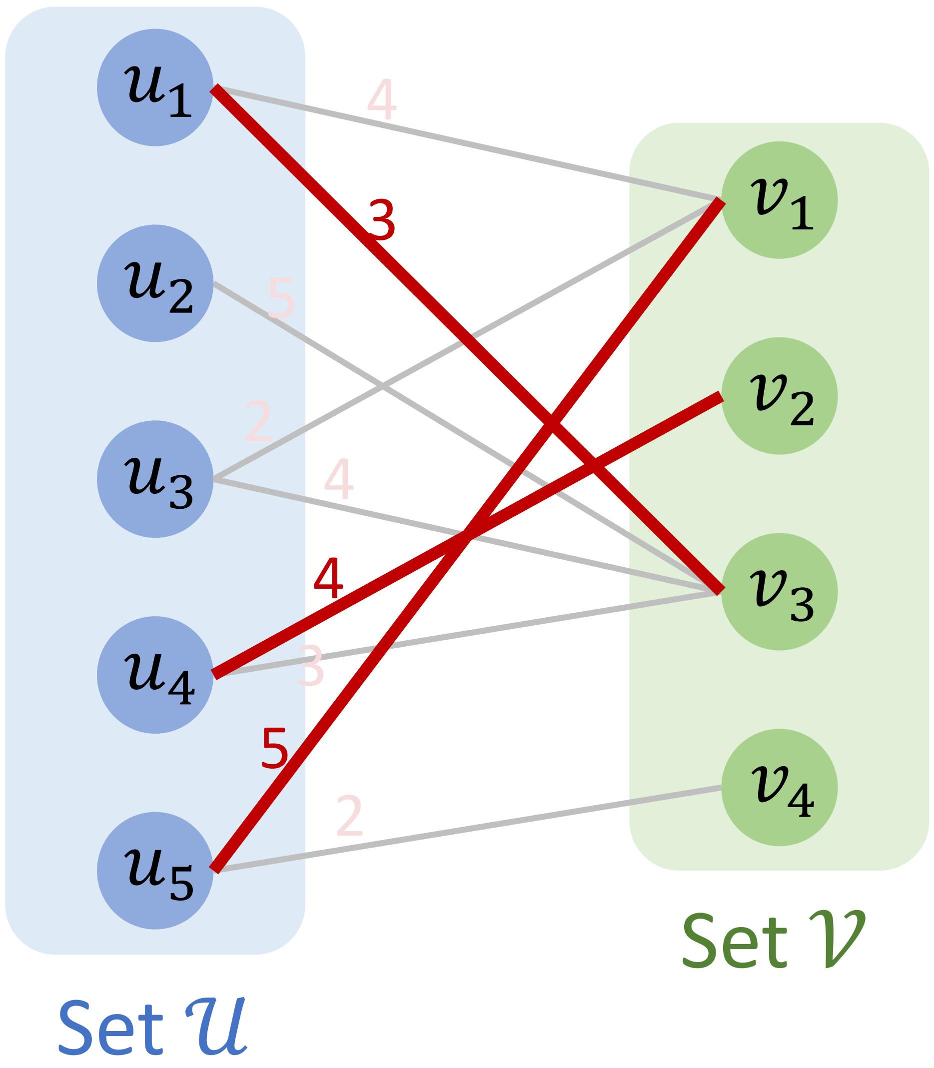
# Bipartite Matching in Weighted Graph




- Sum of weights in matching $\mathcal{S}$:

$$f(\mathcal{S}) = \sum_{(u,v)\in\mathcal{S}} w_{uv}.$$

# Bipartite Matching in Weighted Graph



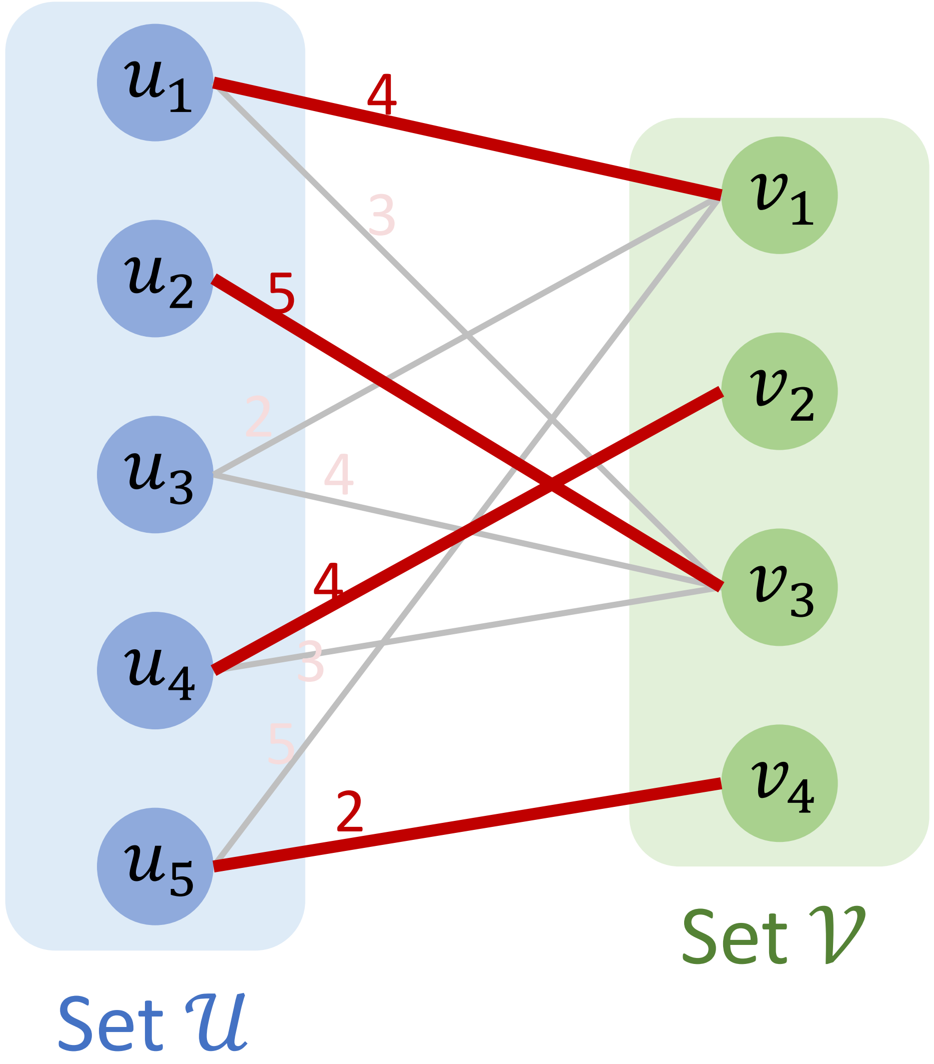

- Sum of weights in matching $\mathcal{S}$:

$$f(\mathcal{S}) = \sum_{(u,v)\in\mathcal{S}} w_{uv}.$$

- In this example,

$$f(\mathcal{S}) = 3 + 4 + 5 = 12.$$

# Bipartite Matching in Weighted Graph



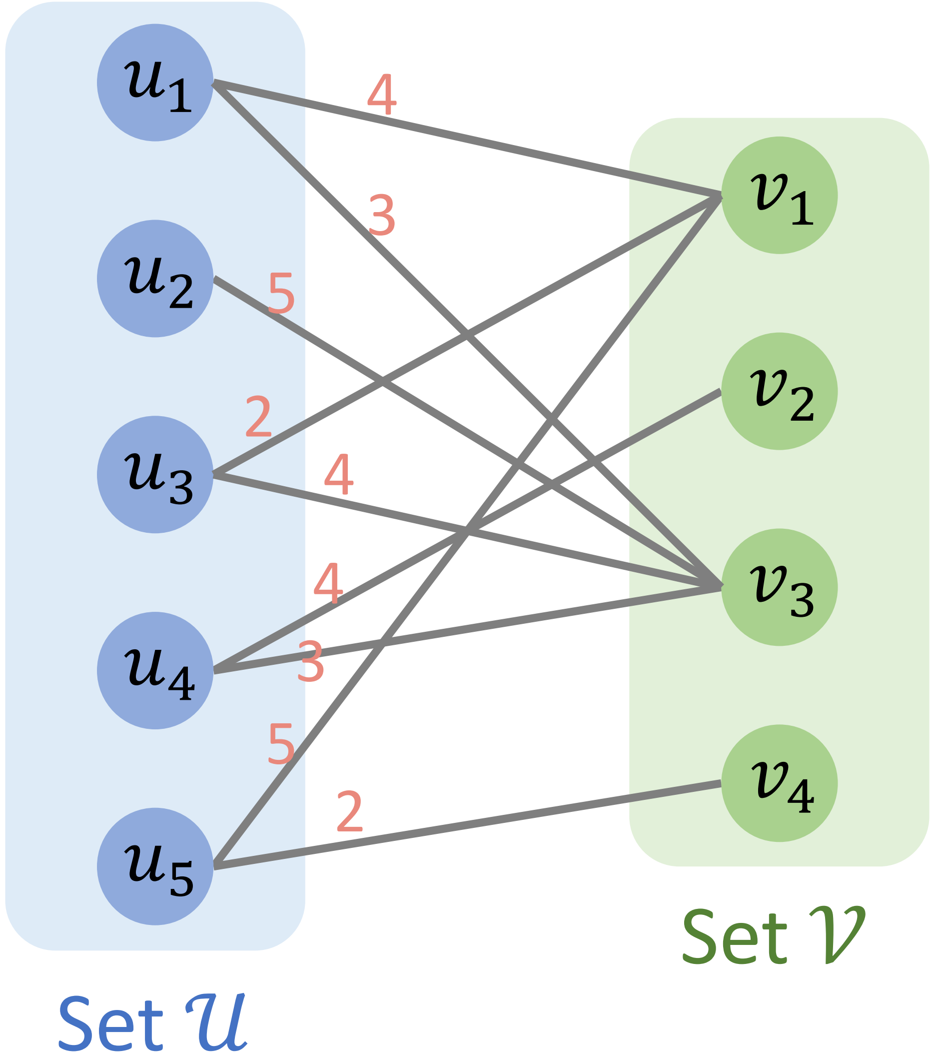

- Sum of weights in matching $\mathcal{S}$:

$$f(\mathcal{S}) = \sum_{(u,v)\in\mathcal{S}} w_{uv}.$$

- In this example,

$$f(\mathcal{S}) = 4 + 5 + 4 + 2 = 15.$$

# Maximum-Weight Bipartite Matching



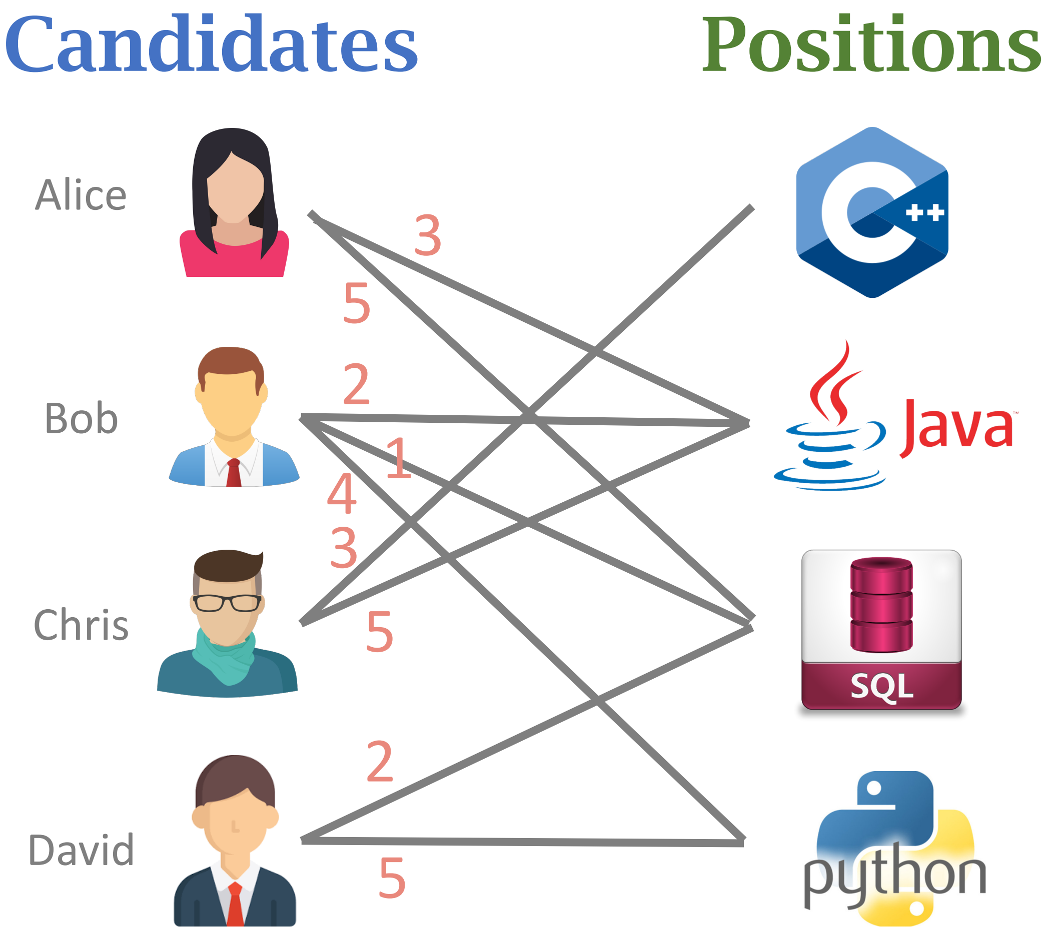

- Sum of weights in matching $\mathcal{S}$:

$$f(\mathcal{S}) = \sum_{(u,v)\in\mathcal{S}} w_{uv}.$$

- **Objective:** Finding matching $\mathcal{S}$ that has the maximum weight:

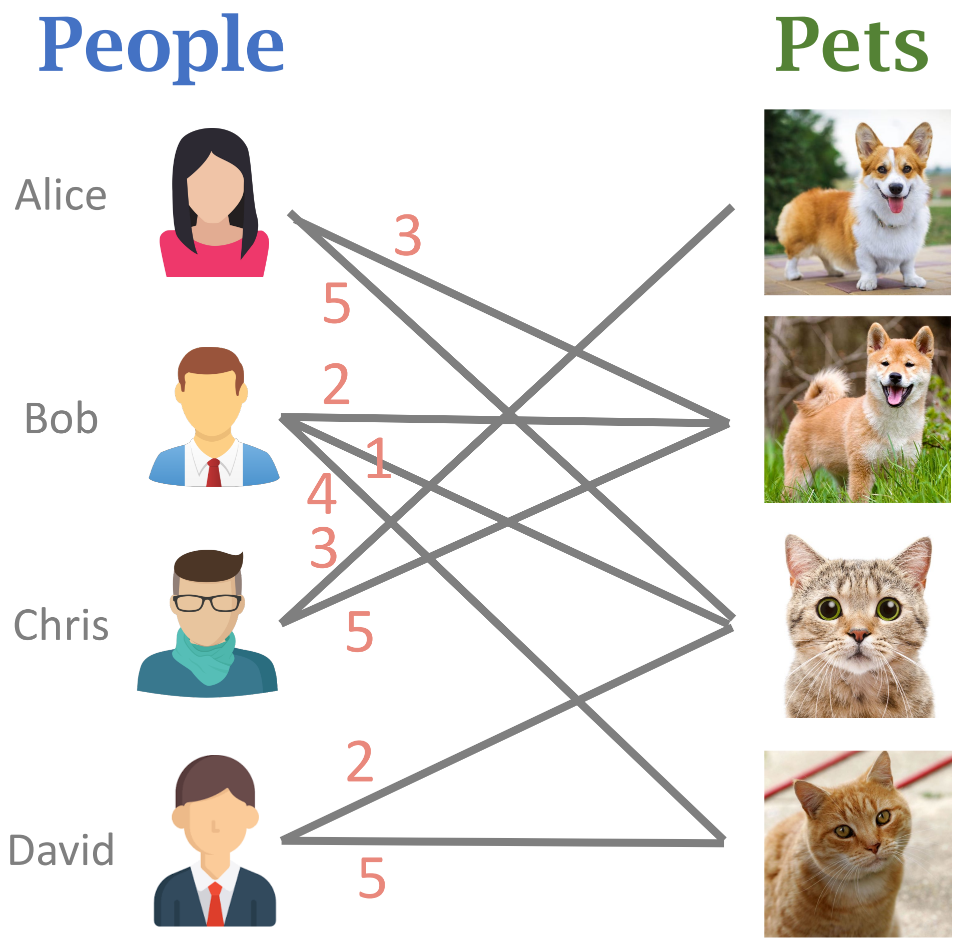
$$\boxed{\max_{\mathcal{S}} f(\mathcal{S}).}$$

# Application 1: Match candidates and positions



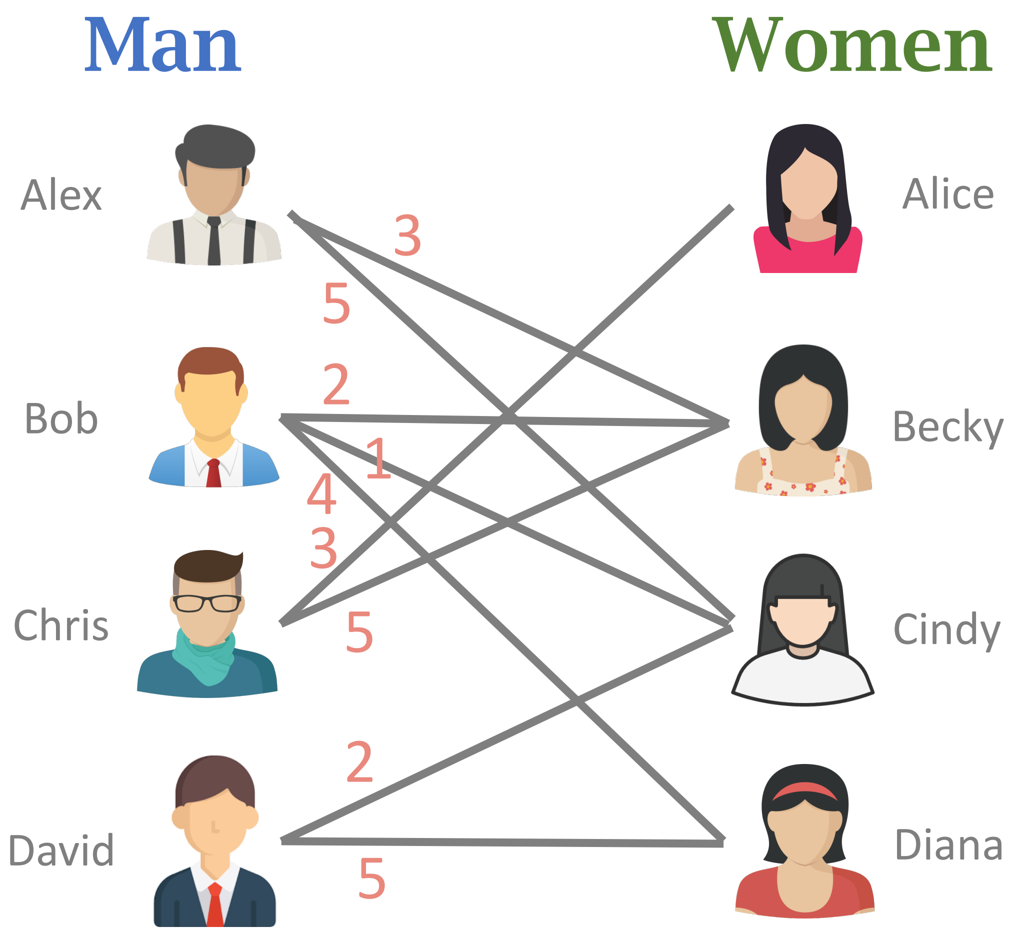

- Edge weights quantify candidates' skills.
- Maximize the weights of matching. (Match the right person with the right job position to maximize the company's interest.)

# Application 2: Pet adoptions




- Edge weight quantifies how much a person loves a pet.
- Maximize the weights of matching.
  (Maximize people's happiness.)

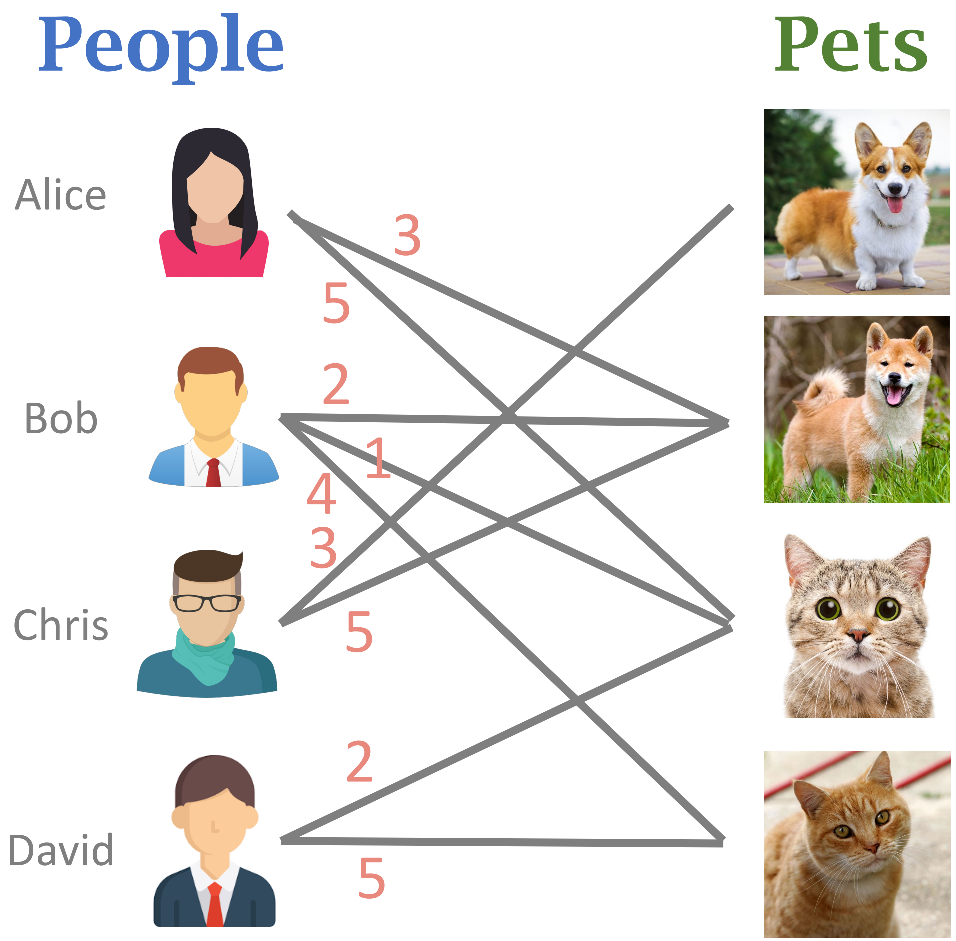
# Application 3: Dating




- Edge weights quantify how well two persons match (e.g., similar hoppy).
- Maximize the weights of matching. (Maximize the change of success.)

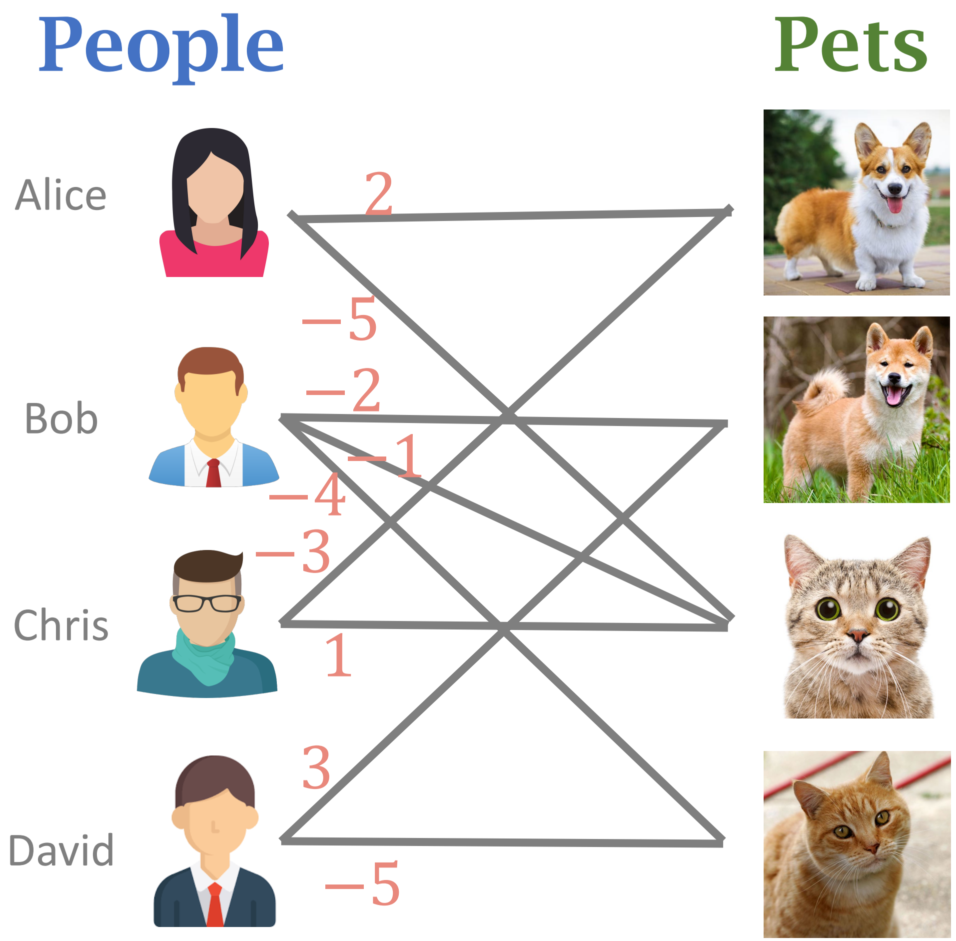
**Maximum Matching $\Longleftrightarrow$ Minimum Matching**

# Maximum Matching



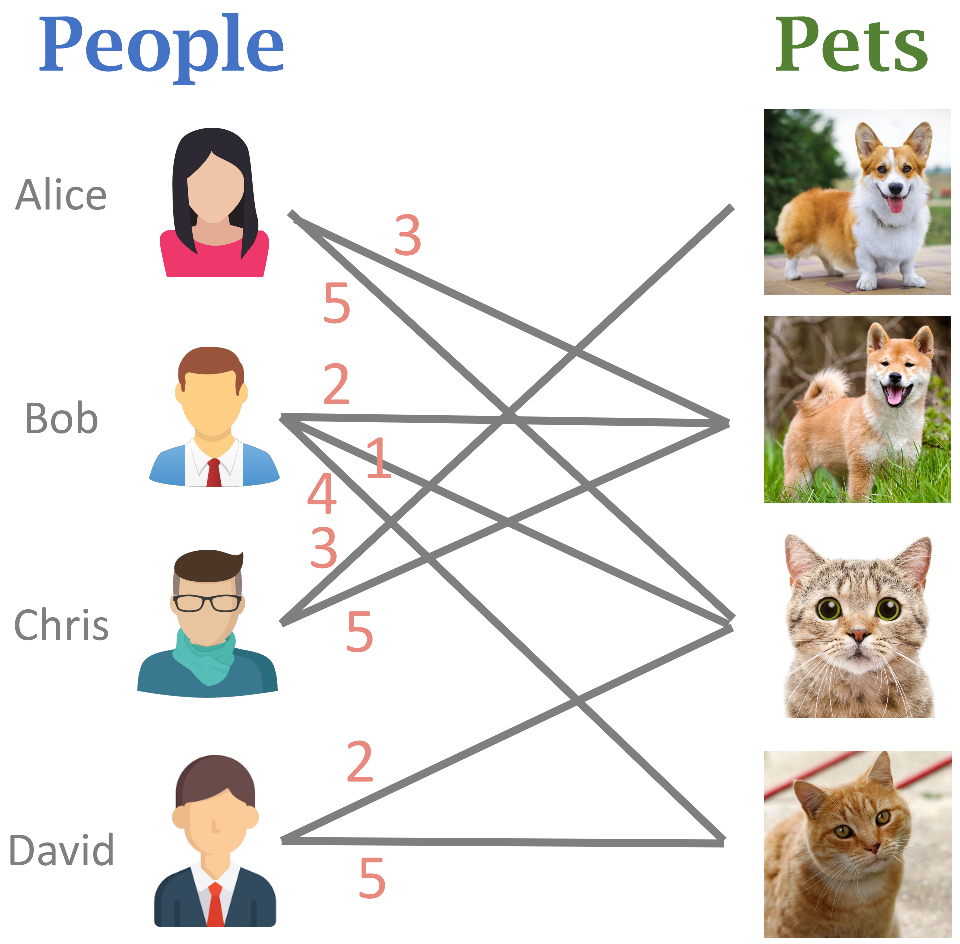

- Adopting a pet can bring happiness to people.
- A weight quantifies how much a person loves a pet.
- Maximize the weights of matching. (Maximize people's happiness.)

# Minimum Matching



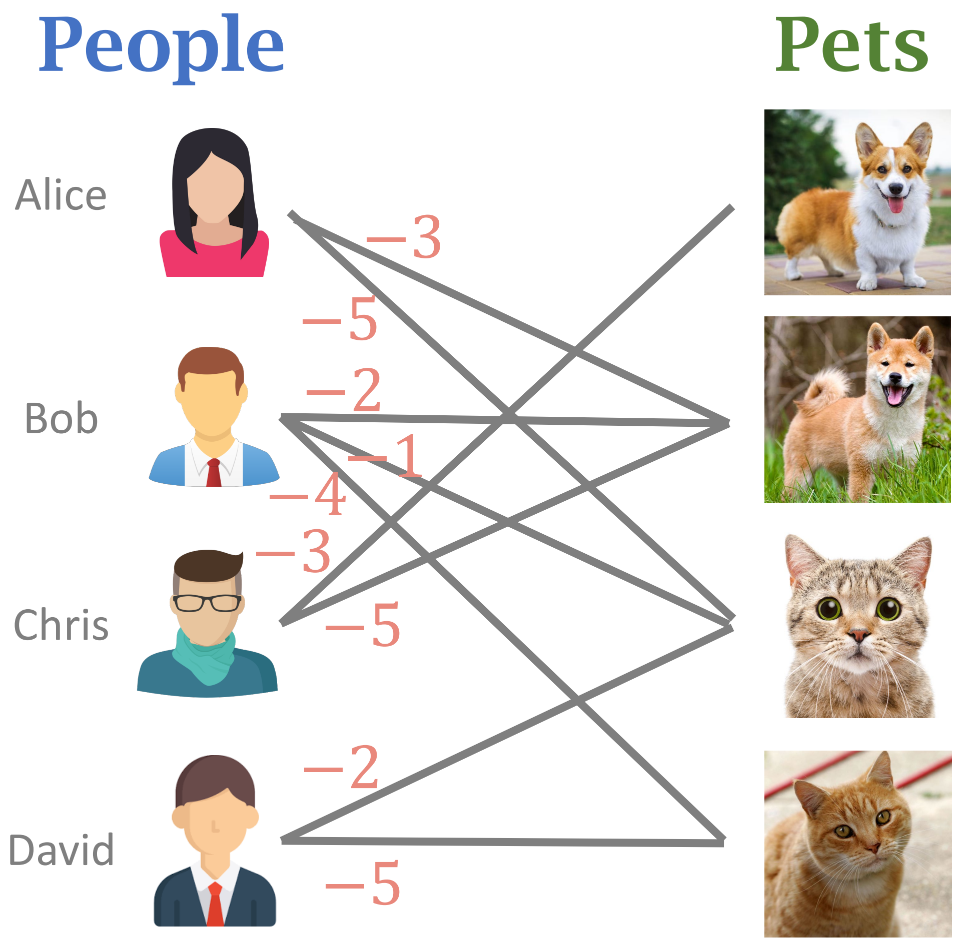

- Adopting a pet can cost time and money.
- A weight quantifies how a person dislike a pet.
- Minimize the weights of matching. (Maximize people's happiness.)

# Maximum Matching ⟺ Minimum Matching




- If we have an algorithm for finding minimum matching.
- Then we can use it for finding maximum matching.

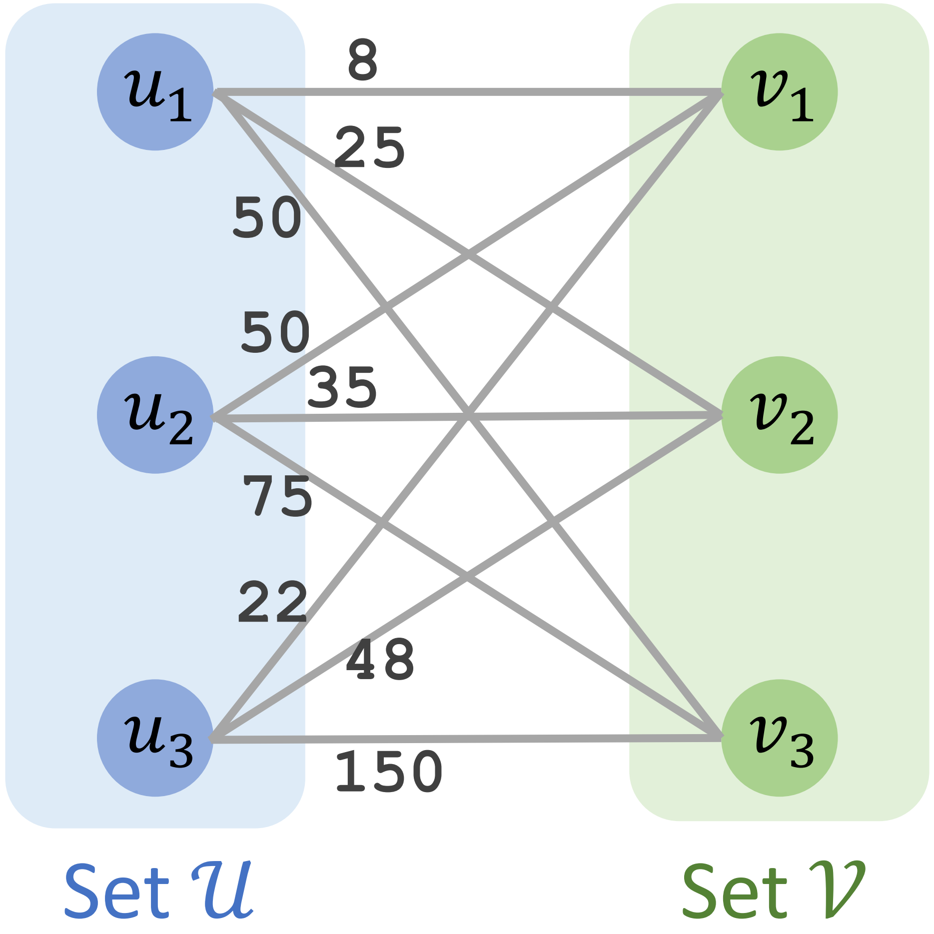
# Maximum Matching ⟺ Minimum Matching




- If we have an algorithm for finding minimum matching.
- Then we can use it for finding maximum matching.
    1. Flip the signs of all the weights.
    2. Run the minimum matching algorithm.

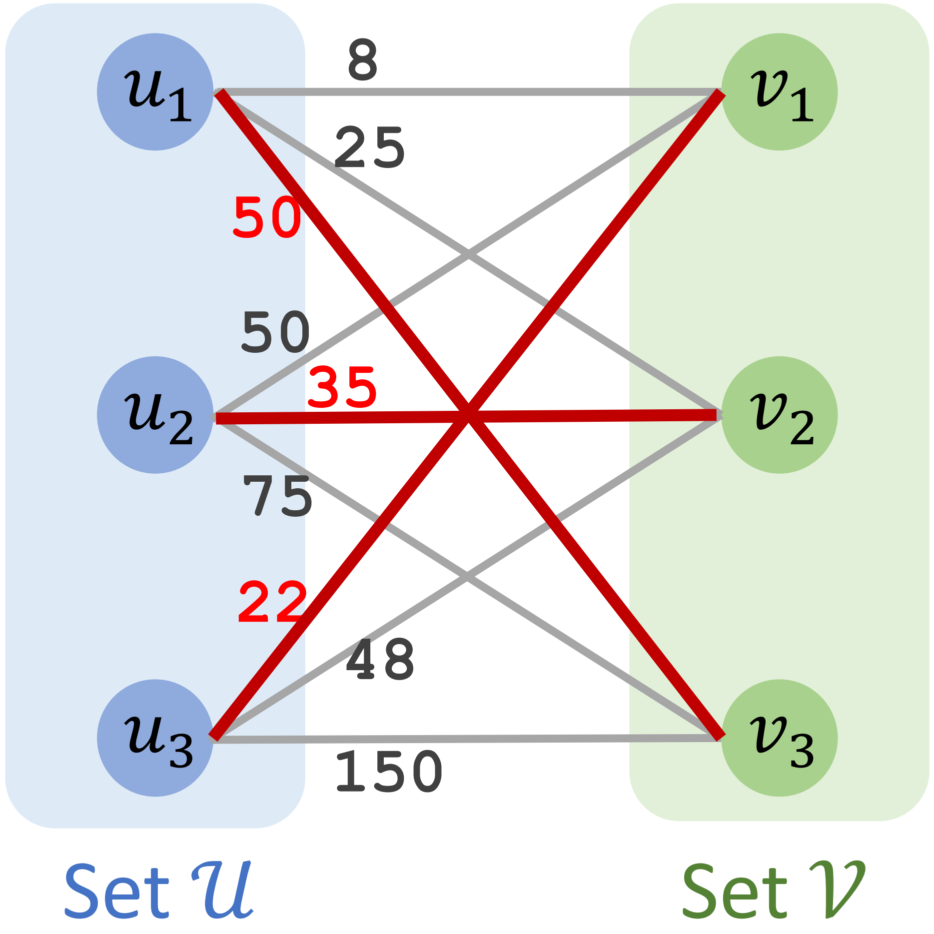
# Hungarian Algorithm for Minimum-Weight Bipartite Matching

# Minimum-Weight Bipartite Matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 | 25 | 50 |
| $u_2$ | 50 | 35 | 75 |
| $u_3$ | 22 | 48 | 150 |

# Minimum-Weight Bipartite Matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 | 25 | 50 |
| $u_2$ | 50 | 35 | 75 |
| $u_3$ | 22 | 48 | 150 |

The minimum sum of weight is $50 + 35 + 22 = 107$.

# Hungarian Algorithm

- Hungarian algorithm is for finding the minimum-weight bipartite matching.
- In the graph, the cardinality of $\mathcal{U}$ and $\mathcal{V}$ must be the same:

$$\boxed{|\mathcal{U}| = |\mathcal{V}| = n.}$$

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 | 25 | 50 |
| $u_2$ | 50 | 35 | 75 |
| $u_3$ | 22 | 48 | 150 |

**Reference:**


- Harold W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*, 2: 83–97, 1955.

# Hungarian Algorithm

- Hungarian algorithm is for finding the minimum-weight bipartite matching.
- In the graph, the cardinality of $\mathcal{U}$ and $\mathcal{V}$ must be the same:

$$|\mathcal{U}| = |\mathcal{V}| = n.$$

- Time complexity: $O(n^3)$.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 | 25 | 50 |
| $u_2$ | 50 | 35 | 75 |
| $u_3$ | 22 | 48 | 150 |

**Reference:**


- Harold W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*, 2: 83–97, 1955.

# Hungarian Algorithm

- Step 1: Subtract row minima.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 | 25 | 50 |
| $u_2$ | 50 | 35 | 75 |
| $u_3$ | 22 | 48 | 150 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.

$$\begin{array}{c|ccc|} & v_1 & v_2 & v_3 \\ \hline u_1 & 8 & 25 & 50 \\ u_2 & 50 & 35 & 75 \\ u_3 & 22 & 48 & 150 \\ \hline \end{array}$$

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 8 −8 | 25 −8 | 50 −8 |
| $u_2$ | 50 −35 | 35 −35 | 75 −35 |
| $u_3$ | 22 −22 | 48 −22 | 150 −22 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 42 |
| $u_2$ | 15 | 0 | 40 |
| $u_3$ | 0 | 26 | 128 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 42 |
| $u_2$ | 15 | 0 | 40 |
| $u_3$ | 0 | 26 | 128 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 42 |
| $u_2$ | 15 | 0 | 40 |
| $u_3$ | 0 | 26 | 128 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 −0 | 17 −0 | 42 −40 |
| $u_2$ | 15 −0 | 0 −0 | 40 −40 |
| $u_3$ | 0 −0 | 26 −0 | 128 −40 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.
  - The minimum of the column is equal to 0.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 2 |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 26 | 88 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
  - Use either horizontal or vertical lines.
  - Minimize the total number of lines.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | ~~0~~ | ~~17~~ | ~~2~~ |
| $u_2$ | ~~15~~ | ~~0~~ | ~~0~~ |
| $u_3$ | ~~0~~ | ~~26~~ | ~~88~~ |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
  - Use either horizontal or vertical lines.
  - Minimize the total number of lines.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | ~~0~~ | ~~17~~ | ~~2~~ |
| $u_2$ | ~~15~~ | ~~0~~ | ~~0~~ |
| $u_3$ | ~~0~~ | ~~26~~ | ~~88~~ |

- The number of line is 3.
- It is NOT the minimum.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
  - Use either horizontal or vertical lines.
  - Minimize the total number of lines.

$$
\begin{array}{c|c|c|c|}
 & v_1 & v_2 & v_3 \\
\hline
u_1 & 0 & 17 & 2 \\
\hline
u_2 & 15 & 0 & 0 \\
\hline
u_3 & 0 & 26 & 88 \\
\hline
\end{array}
$$

- The number of line is 2.
- It is the minimum.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
  - If $n$ lines are required, the algorithm stops.
  - If less than $n$ lines are required, continue with Step 3C.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 2 |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 26 | 88 |

- The number of line is 2.
- Number of vertices is $n = 3$.
- Thus continue to Step 3C.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 2 |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 26 | 88 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 | 2 =k |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 26 | 88 |

The smallest element that is not covered is $k = 2$.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 17 $-2$ | 2 $-2$ |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 26 $-2$ | 88 $-2$ |

The smallest element that is not covered is $k = 2$.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 15 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.
  - Add $k$ to all elements that are covered twice.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 15 +2 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.
  - Add $k$ to all elements that are covered twice.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

At least 3 lines are needed.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
  - If $n$ lines are required, the algorithm stops.

The algorithm stops.

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Output the matching

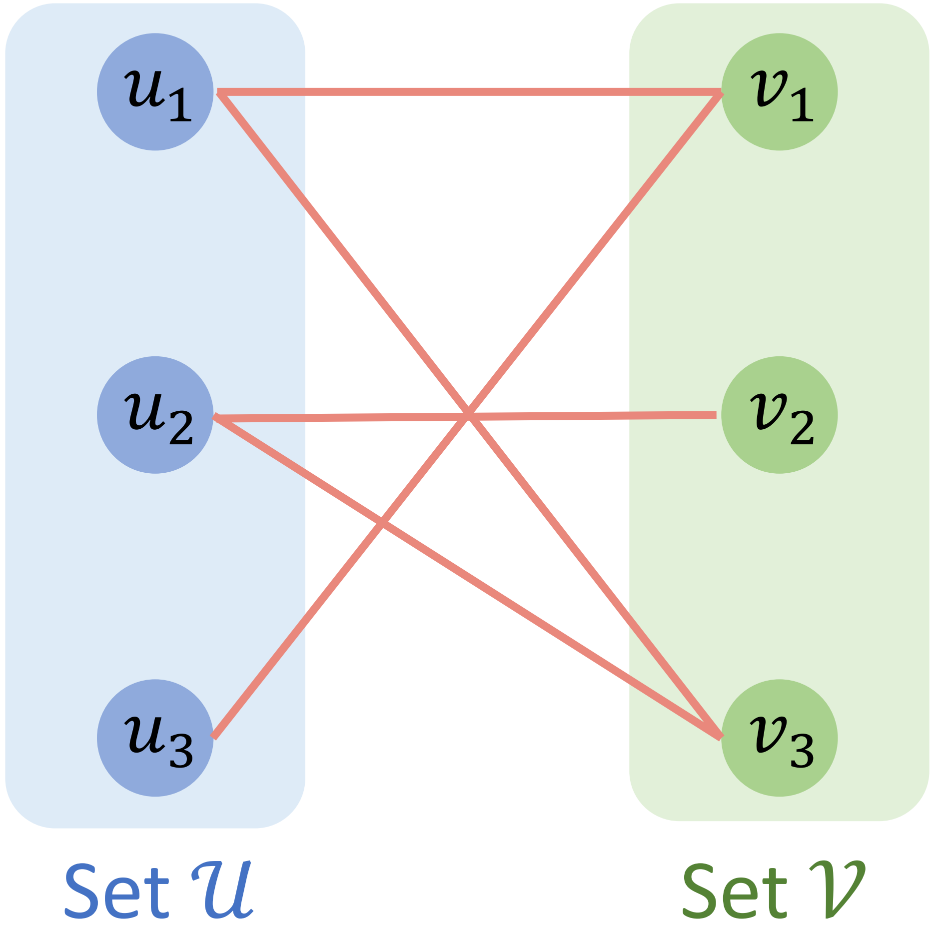
| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

# Output the matching

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | **0** | 15 | **0** |
| $u_2$ | 17 | **0** | **0** |
| $u_3$ | **0** | 24 | 86 |

- Choose a matching among the zeros.
- Think of the zeros as edges.

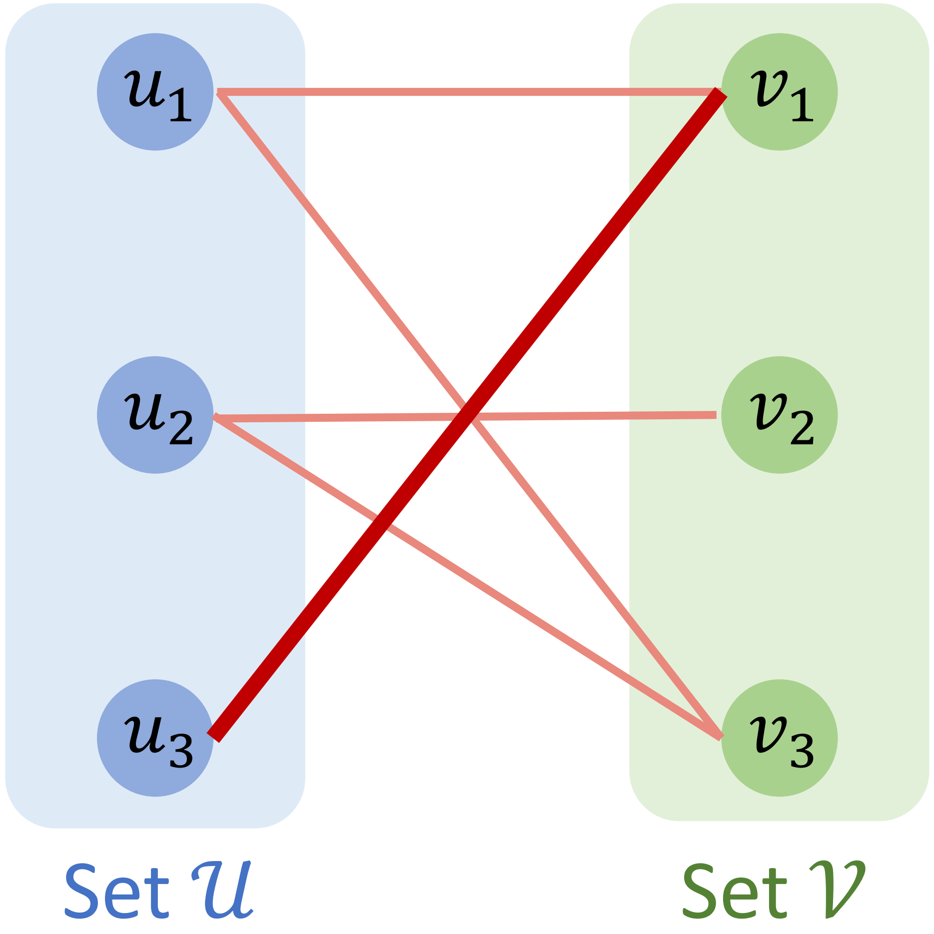
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | **0** | 15 | **0** |
| $u_2$ | 17 | **0** | **0** |
| $u_3$ | **0** | 24 | 86 |

- Choose a matching among the zeros.
- Think of the zeros as edges.

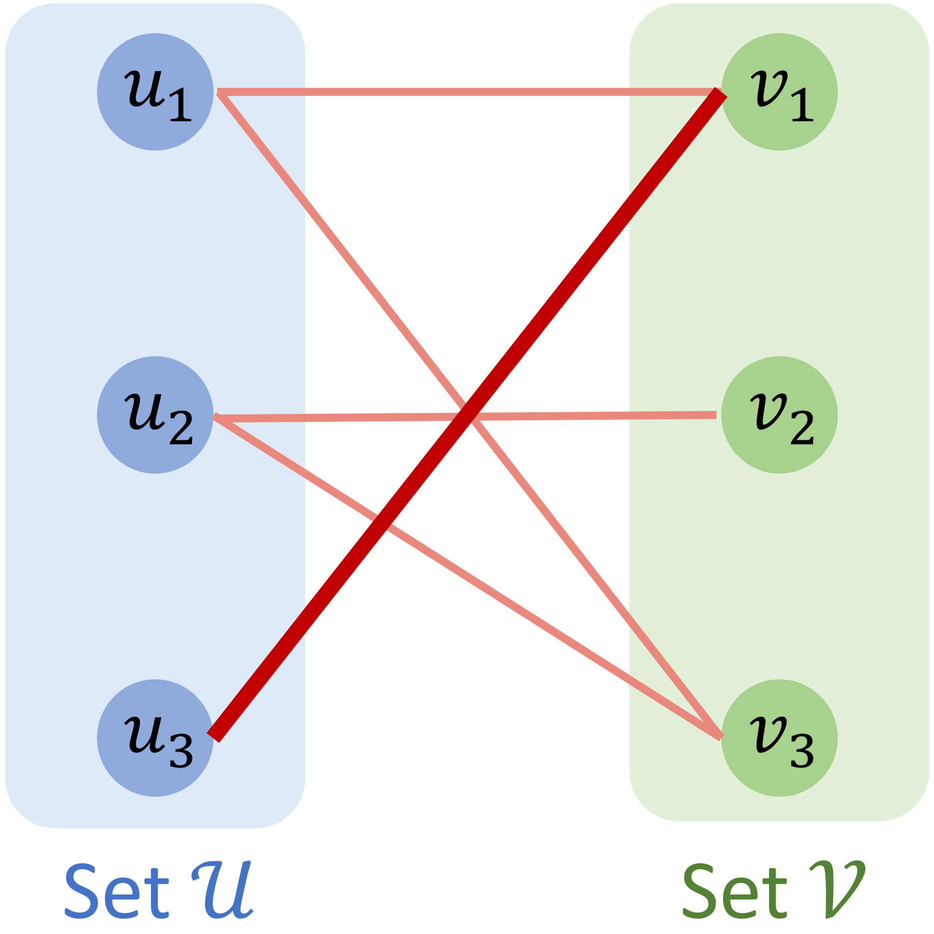
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | **0** | 15 | **0** |
| $u_2$ | 17 | **0** | **0** |
| $u_3$ | **0** | 24 | 86 |

- The edge $(u_3, v_1)$ must be chosen.
- Because it is the only zero in the row.

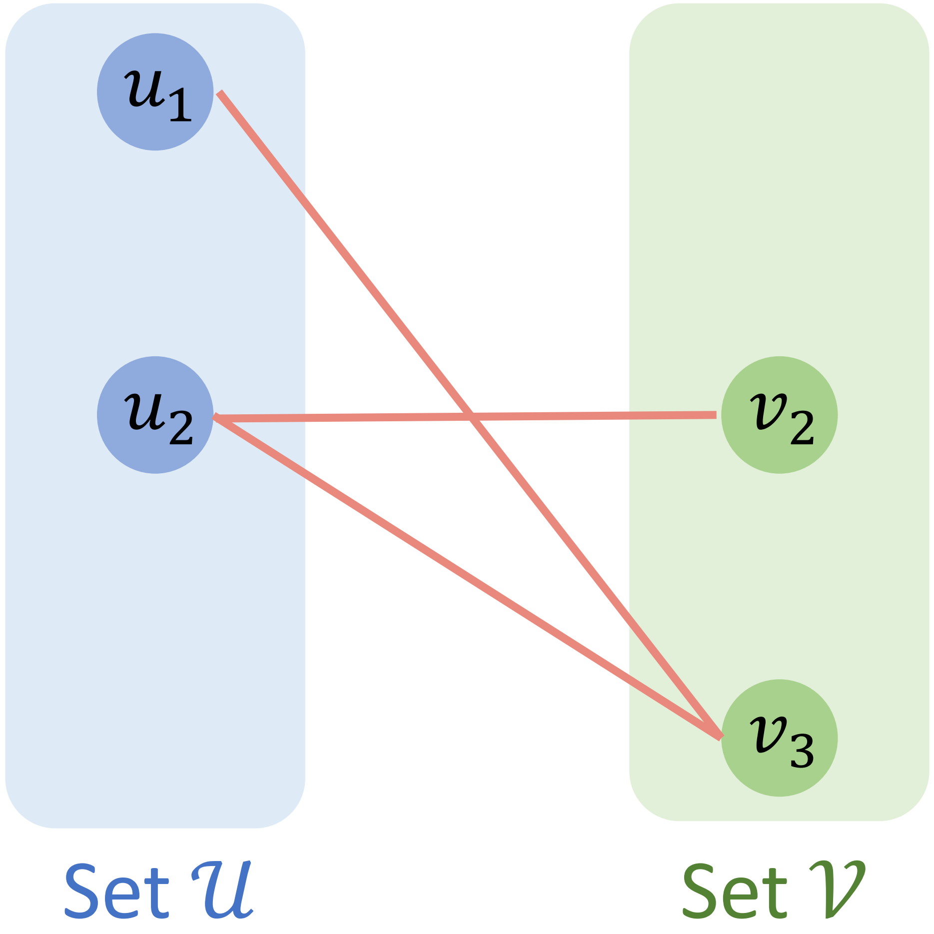
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | **0** | 15 | **0** |
| $u_2$ | 17 | **0** | **0** |
| $u_3$ | **0** | 24 | 86 |

- The edge $(u_3, v_1)$ must be chosen.
- Because it is the only zero in the row.

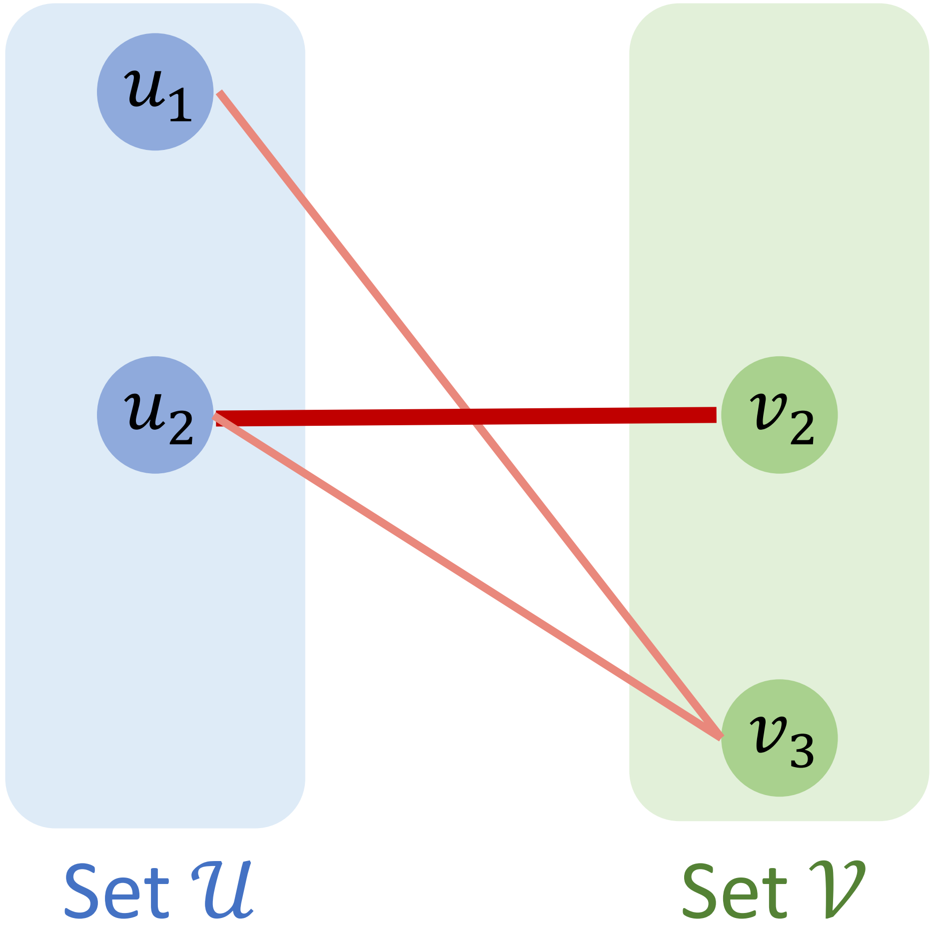
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

- Cover the row of $u_3$.
- Cover the column of $v_1$.

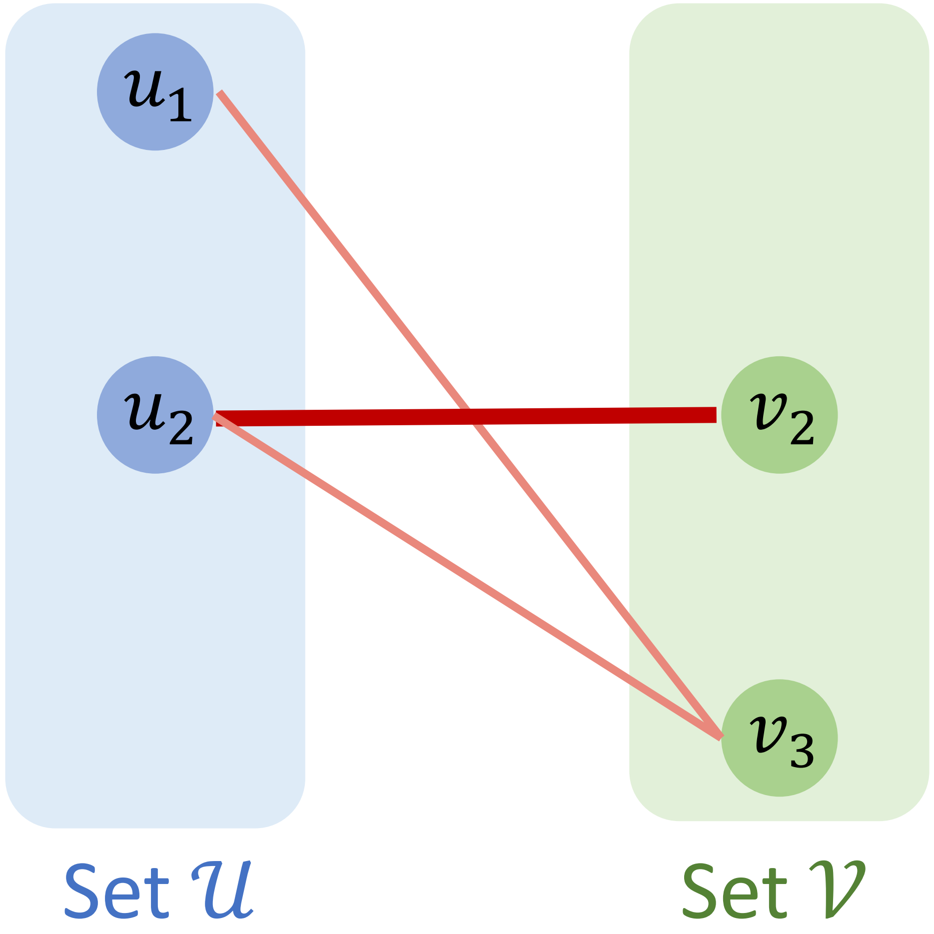
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

- The edge $(u_2, v_2)$ must be chosen.
- Because it is the only zero in the column.

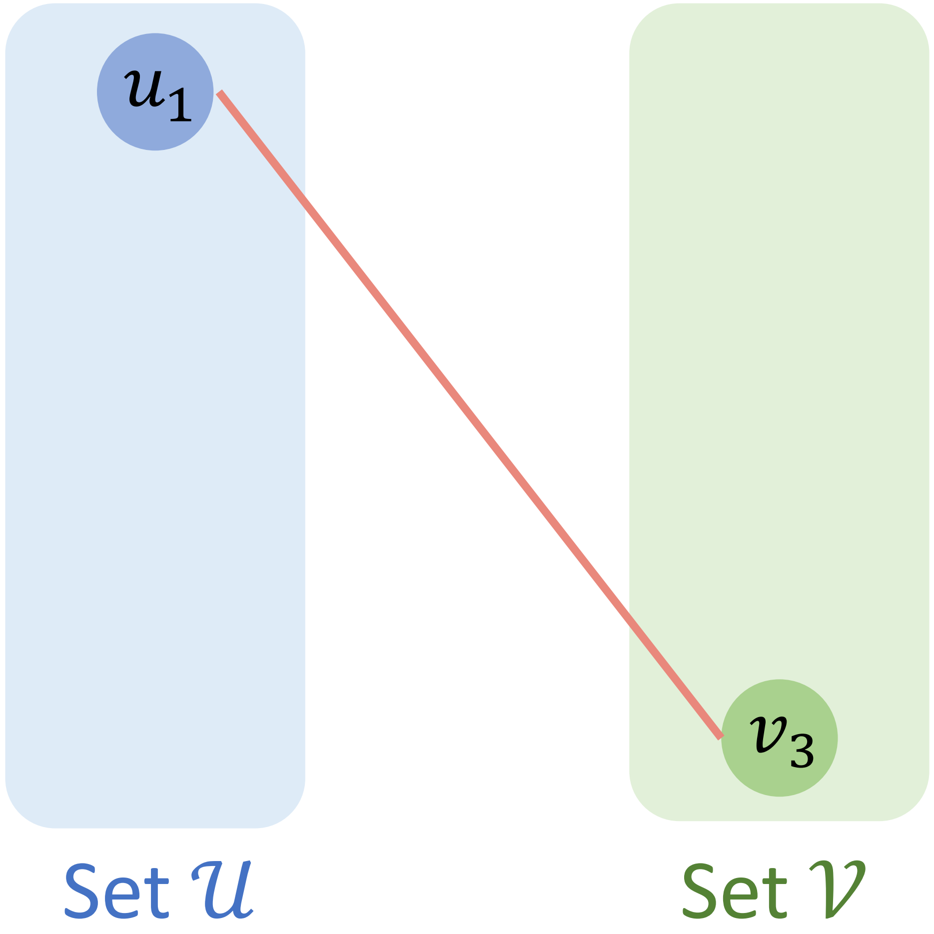
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

- The edge $(u_2, v_2)$ must be chosen.
- Because it is the only zero in the column.

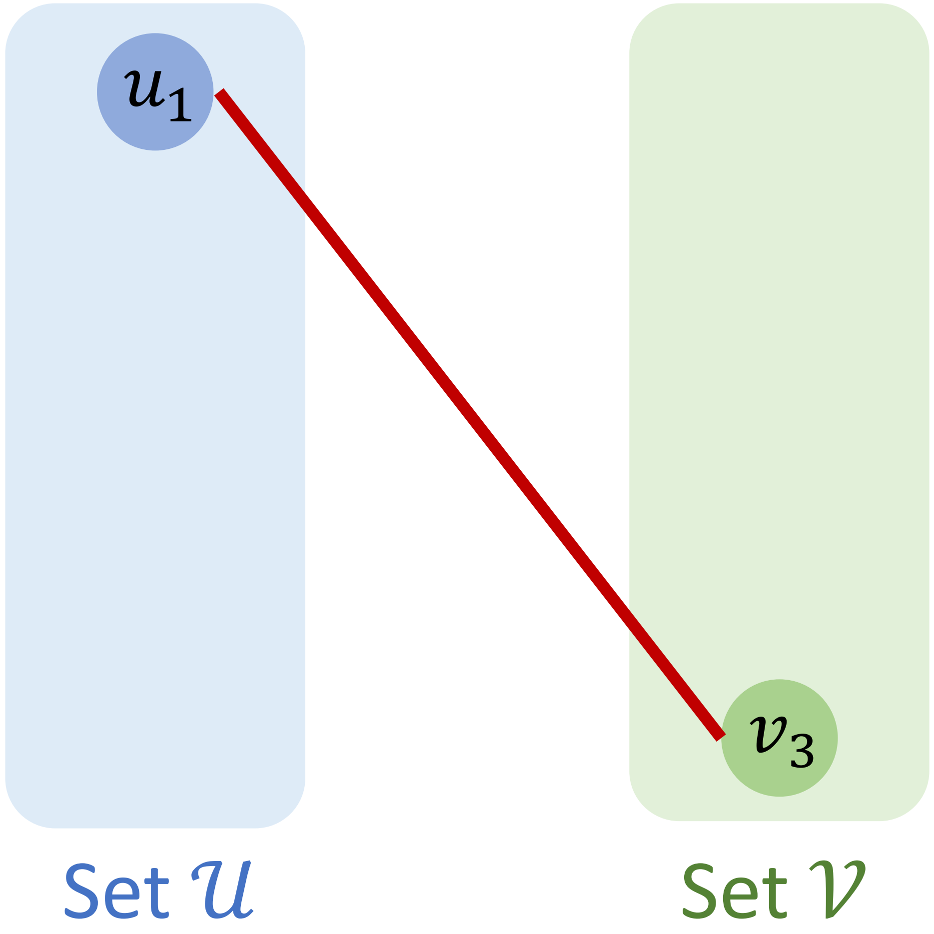
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | **0** |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

- Cover the row of $u_2$.
- Cover the column of $v_2$.

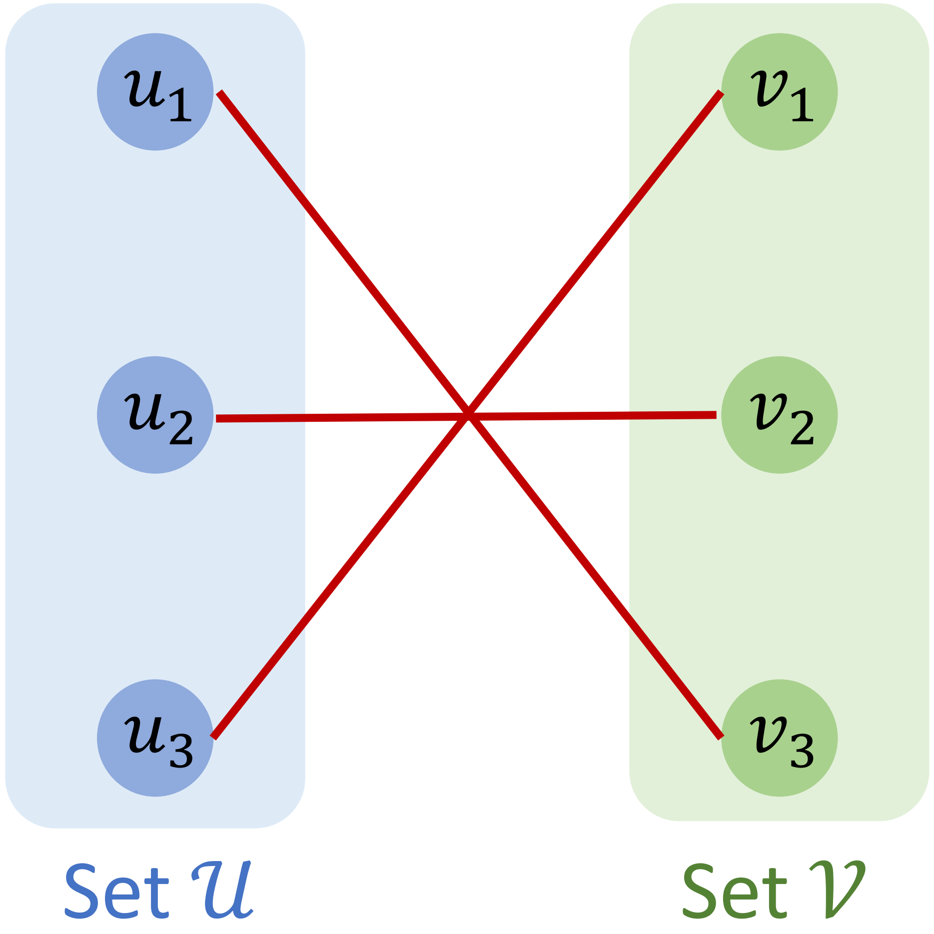
# Output the matching




| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | 0 | 15 | 0 |
| $u_2$ | 17 | 0 | 0 |
| $u_3$ | 0 | 24 | 86 |

- The edge $(u_1, v_3)$ must be chosen.
- Because it is the only zero.

# Output the matching



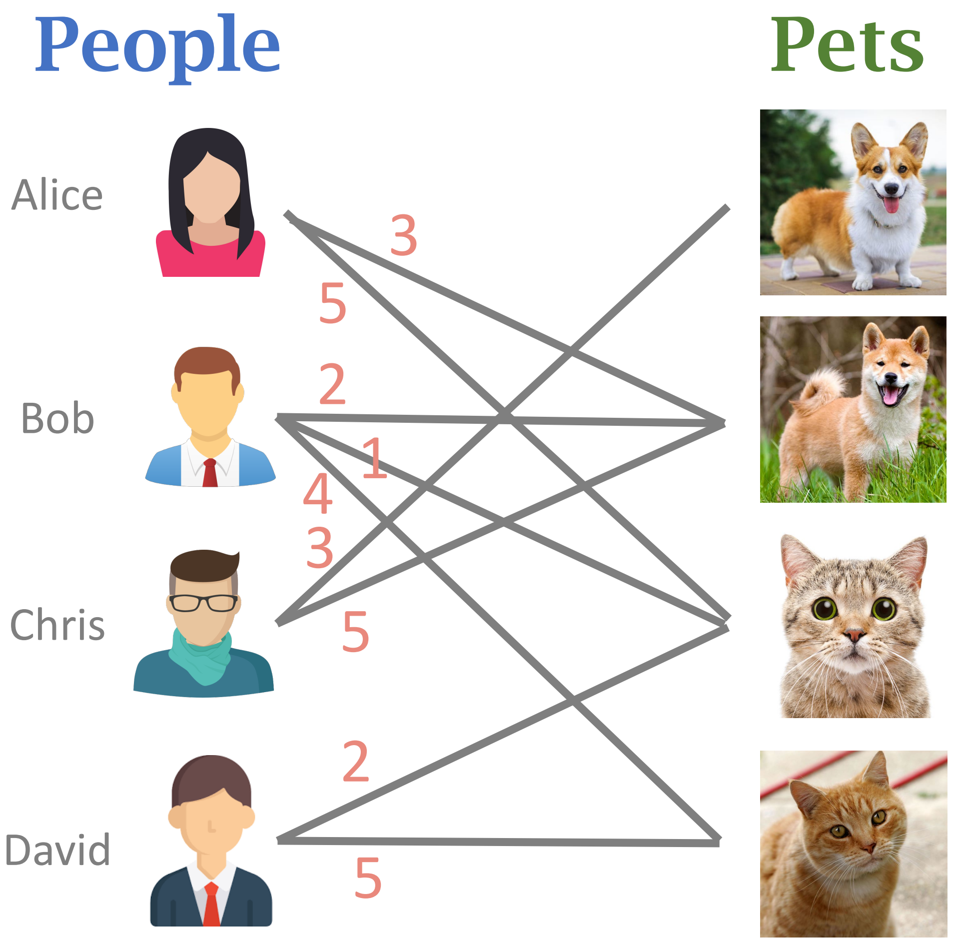

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $u_1$ | **0** | 15 | **0** |
| $u_2$ | 17 | **0** | **0** |
| $u_3$ | **0** | 24 | 86 |

The matching is

$$\mathcal{S} = \{(u_3, v_1),\ (u_1, v_3),\ (u_2, v_2)\}.$$

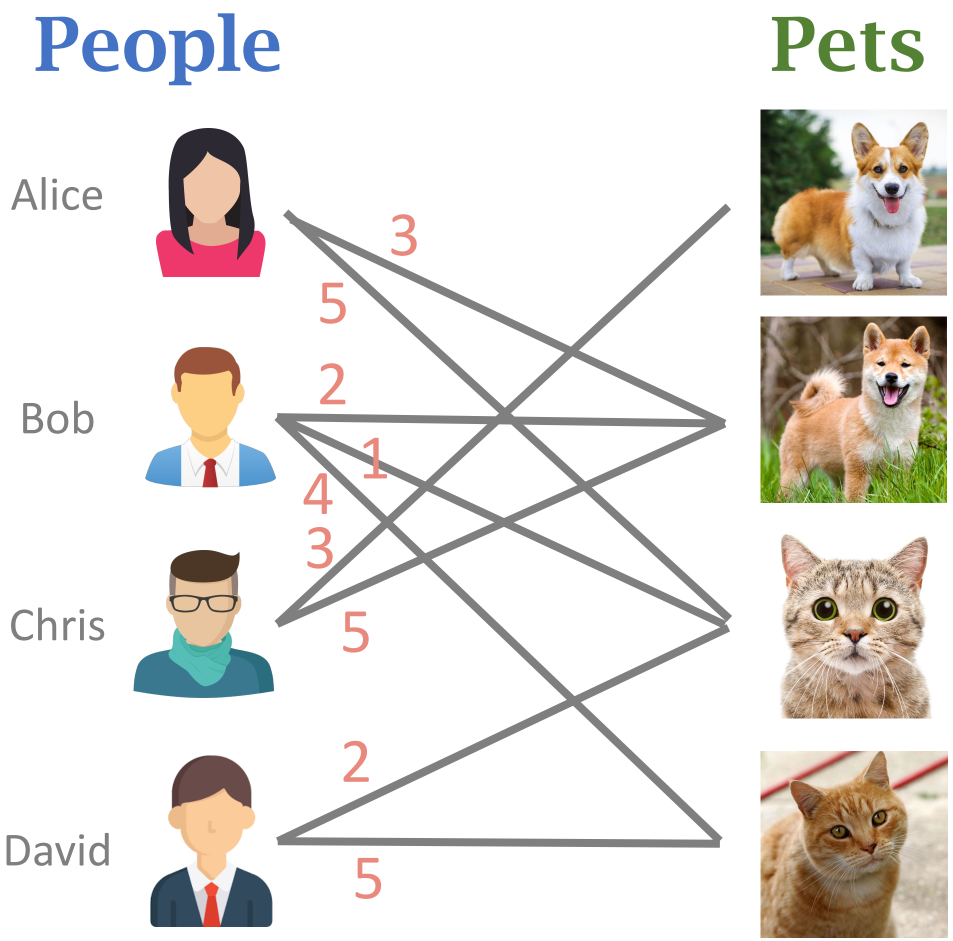
# Hungarian Algorithm for Maximum Matching

# Maximum Matching




- Pet adoption is a max matching problem.
- A weight quantifies how much a person loves a pet.
- Maximize the weights of matching. (Maximize people's happiness.)

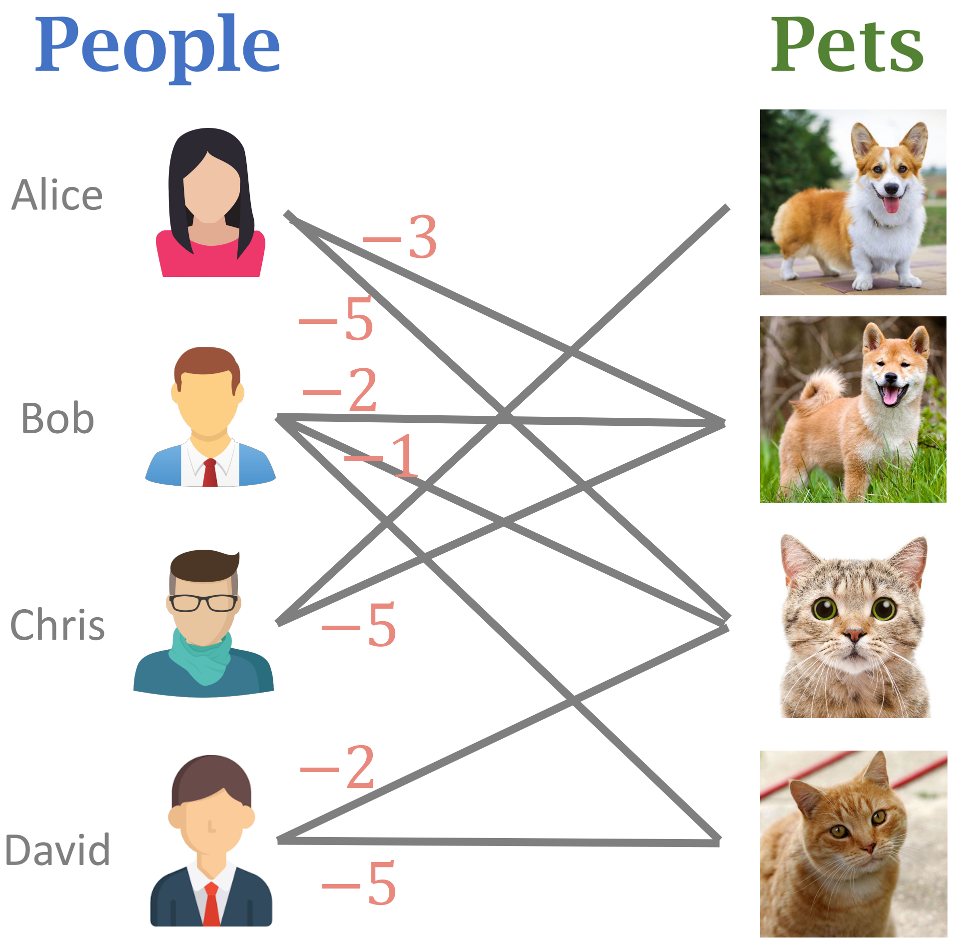
# Hungarian Algorithm for Maximum Matching




**Idea:** Max Matching ➡ Min Matching

- Flip the signs of all the weights.
- It is equivalent to the minimum matching.
- Run the Hungarian algorithm.

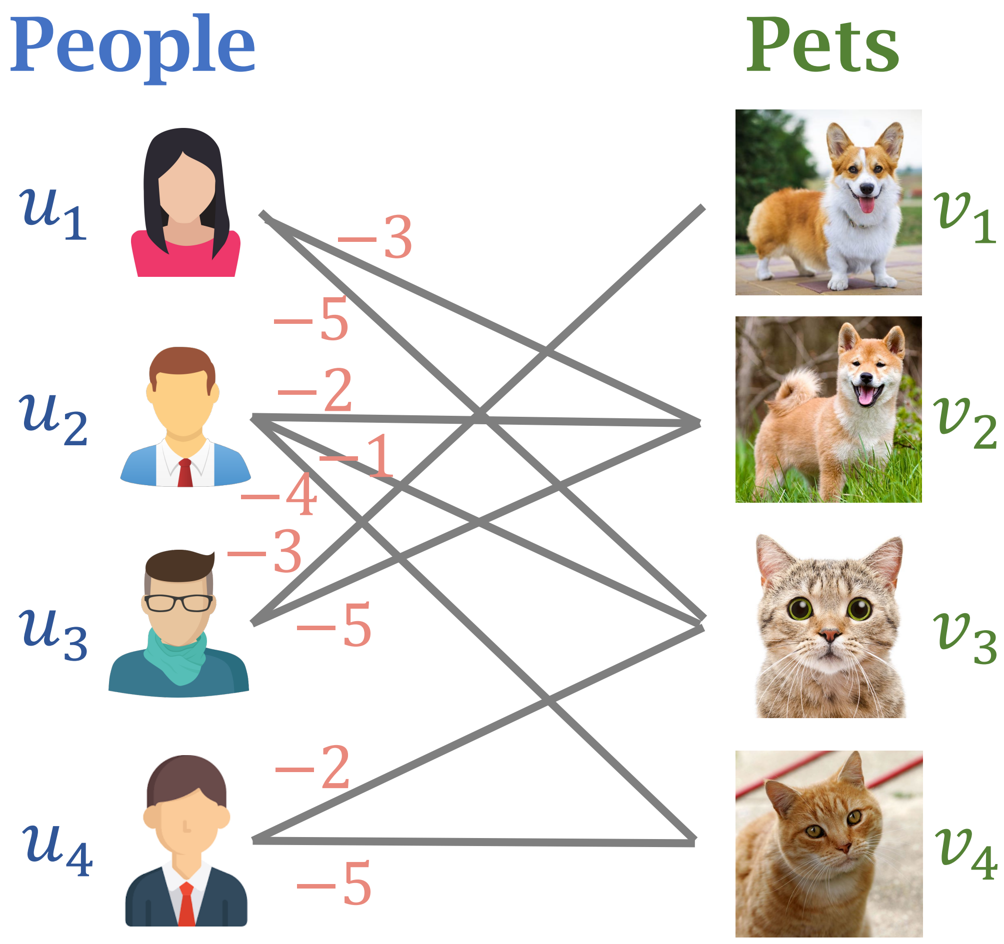
# Hungarian Algorithm for Maximum Matching




**Idea:** Max Matching ➡ Min Matching

- Flip the signs of all the weights.
- It is equivalent to the minimum matching.
- Run the Hungarian algorithm.

# Hungarian Algorithm for Maximum Matching




| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 0 | -3 | -5 | 0 |
| $u_2$ | 0 | -2 | -1 | -4 |
| $u_3$ | -3 | -5 | 0 | 0 |
| $u_4$ | 0 | 0 | -2 | -5 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 0 | -3 | -5 | 0 |
| $u_2$ | 0 | -2 | -1 | -4 |
| $u_3$ | -3 | -5 | 0 | 0 |
| $u_4$ | 0 | 0 | -2 | -5 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 0 −(−5) | −3 −(−5) | −5 −(−5) | 0 −(−5) |
| $u_2$ | 0 −(−4) | −2 −(−4) | −1 −(−4) | −4 −(−4) |
| $u_3$ | −3 −(−5) | −5 −(−5) | 0 −(−5) | 0 −(−5) |
| $u_4$ | 0 −(−5) | 0 −(−5) | −2 −(−5) | −5 −(−5) |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 5 | 2 | 0 | 5 |
| $u_2$ | 4 | 2 | 3 | 0 |
| $u_3$ | 2 | 0 | 5 | 5 |
| $u_4$ | 5 | 5 | 3 | 0 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 5 | 2 | 0 | 5 |
| $u_2$ | 4 | 2 | 3 | 0 |
| $u_3$ | 2 | 0 | 5 | 5 |
| $u_4$ | 5 | 5 | 3 | 0 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 5 | 2 | 0 | 5 |
| $u_2$ | 4 | 2 | 3 | 0 |
| $u_3$ | 2 | 0 | 5 | 5 |
| $u_4$ | 5 | 5 | 3 | 0 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 5 −2 | 2 −0 | 0 −0 | 5 −0 |
| $u_2$ | 4 −2 | 2 −0 | 3 −0 | 0 −0 |
| $u_3$ | 2 −2 | 0 −0 | 5 −0 | 5 −0 |
| $u_4$ | 5 −2 | 5 −0 | 3 −0 | 0 −0 |

# Hungarian Algorithm

- Step 1: Subtract row minima.
  - Subtract the smallest entry of each row from all the entries in the row.
  - The minimum of the row is equal to 0.
- Step 2: Subtract column minima.
  - Subtract the smallest entry of each column from all the entries in the column.
  - The minimum of the column is equal to 0.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 2 | 2 | 3 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 3 | 5 | 3 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
  - Use either horizontal or vertical lines.
  - Minimize the total number of lines.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 2 | 2 | 3 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 3 | 5 | 3 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
  - If $n$ lines are required, the algorithm stops.
  - If less than $n$ lines are required, continue with Step 3C.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 2 | 2 | 3 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 3 | 5 | 3 | 0 |

- The number of line is 3.
- Number of vertices is $n = 4$.
- Thus continue to Step 3C.

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.

$$
\begin{array}{c|cccc|}
 & v_1 & v_2 & v_3 & v_4 \\
\hline
u_1 & 3 & 2 & 0 & 5 \\
u_2 & 2 & 2 & 3 & 0 \\
u_3 & 0 & 0 & 5 & 5 \\
u_4 & 3 & 5 & 3 & 0 \\
\hline
\end{array}
$$

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 2 | 2 =k | 3 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 3 | 5 | 3 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 2 $-2$ | 2 $-2$ | 3 $-2$ | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 3 $-2$ | 5 $-2$ | 3 $-2$ | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.
  - Add $k$ to all elements that are covered twice.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 5 +2 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 5 +2 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
- **Step 3C:** Create additional zeros.
  - Find the smallest element (call it $k$) that is not covered by a line.
  - Subtract $k$ from all uncovered elements.
  - Add $k$ to all elements that are covered twice.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
  - Use either horizontal or vertical lines.
  - Minimize the total number of lines.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Hungarian Algorithm

**Step 3:** Repeat the following:

- **Step 3A:** Cover all zeros with a minimum number of lines.
- **Step 3B:** Decide whether to stop.
  - If $n$ lines are required, the algorithm stops.

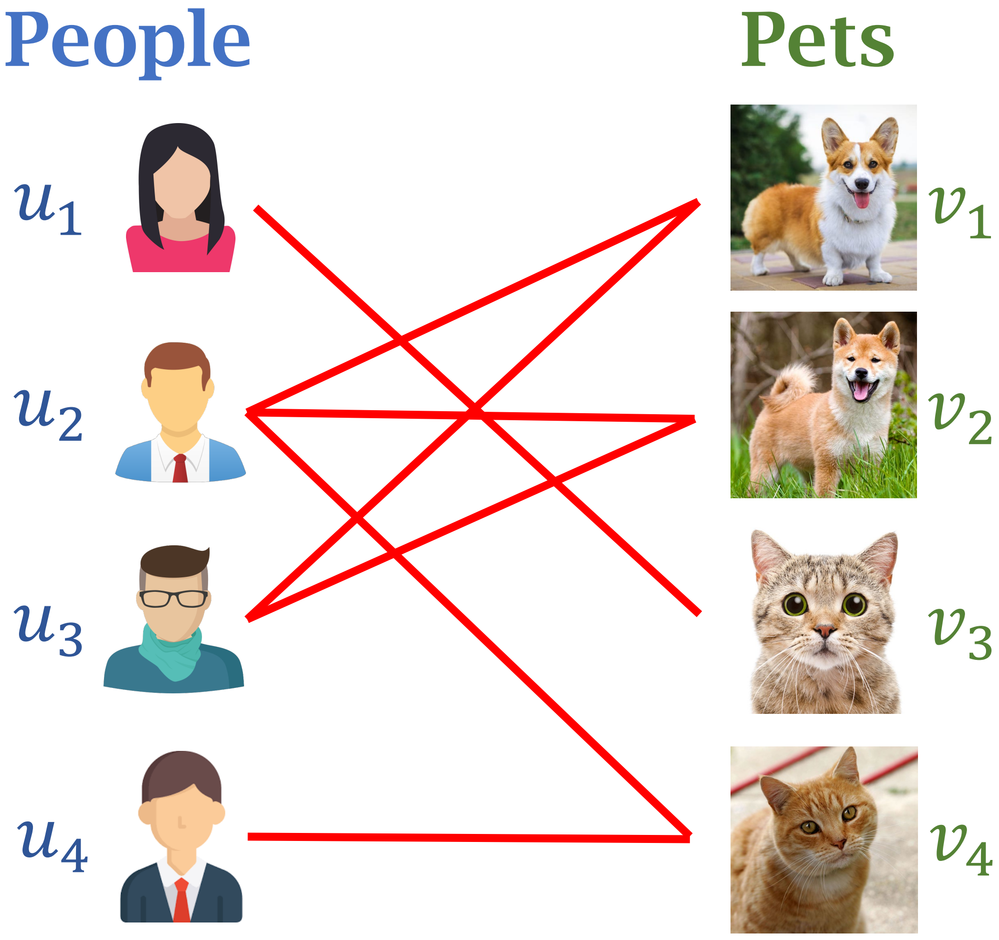
The algorithm stops.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching




| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- The edge $(u_1, v_3)$ must be selected.
  - Otherwise, $u_1$ would have no matching.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- The edge $(u_1, v_3)$ must be selected.
  - Otherwise, $u_1$ would have no matching.
  - Cover the row of $u_1$.
  - Cover the column of $v_3$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- The edge $(u_4, v_4)$ must be selected.
  - Otherwise, $u_4$ would have no matching.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- The edge $(u_4, v_4)$ must be selected.
  - Otherwise, $u_4$ would have no matching.
  - Cover the row of $u_4$.
  - Cover the column of $v_4$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- Select edges $(u_2, v_1)$ and $(u_3, v_2)$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- Select edges $(u_2, v_1)$ and $(u_3, v_2)$.

- Or select edges $(u_3, v_1)$ and $(u_2, v_2)$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- Return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_2, v_1), (u_3, v_2)\}.$

- Or return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_3, v_1), (u_2, v_2)\}.$

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- Return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_2, v_1), (u_3, v_2)\}$.

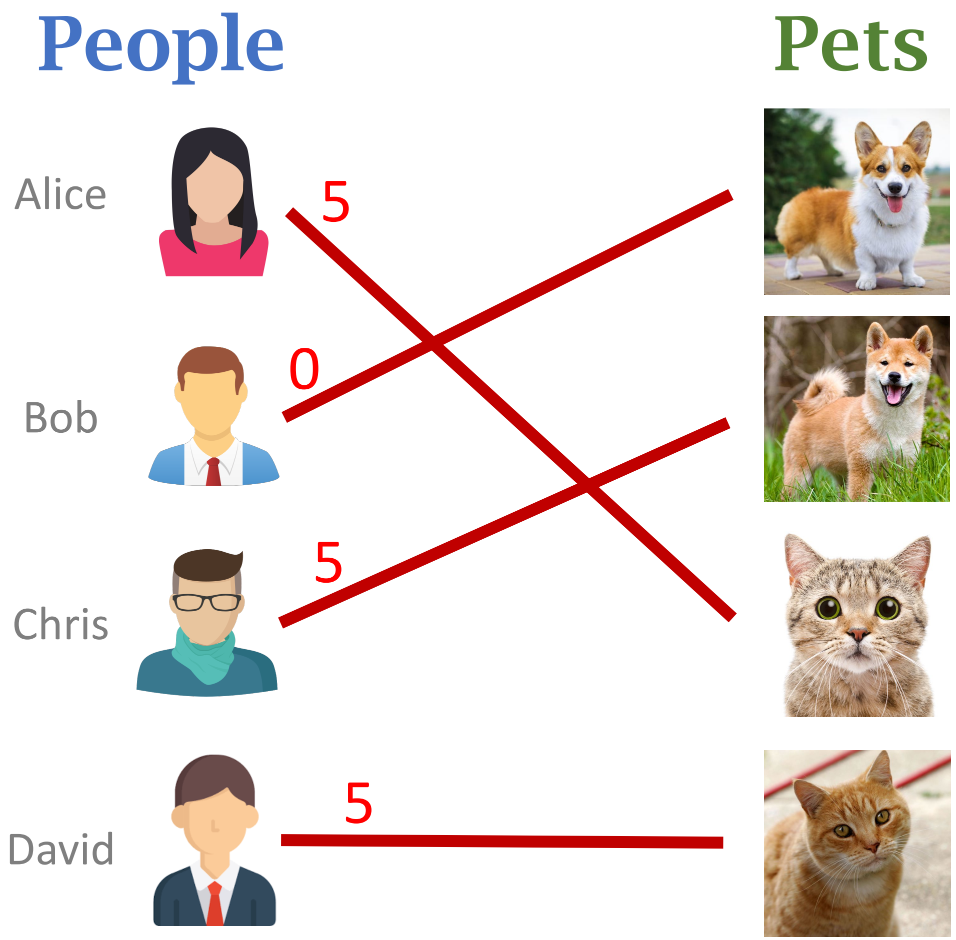
- Or return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_3, v_1), (u_2, v_2)\}$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $u_1$ | 3 | 2 | 0 | 7 |
| $u_2$ | 0 | 0 | 1 | 0 |
| $u_3$ | 0 | 0 | 5 | 7 |
| $u_4$ | 1 | 3 | 1 | 0 |

# Output the matching

- Return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_2, v_1), (u_3, v_2)\}$.
- The matching is equal to 15.

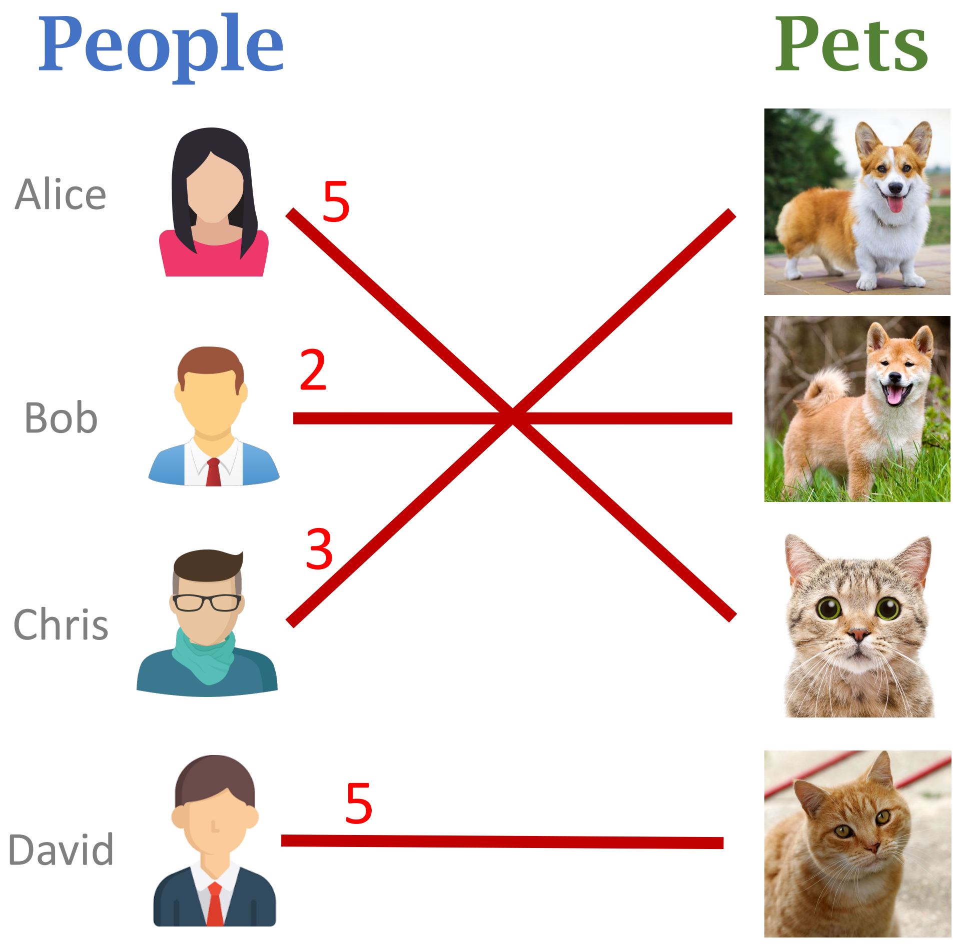
- Or return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_3, v_1), (u_2, v_2)\}$.



# Output the matching

- Return the matching: $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_2, v_1), (u_3, v_2)\}.$
- The matching is equal to 15.

- Or return the matching: $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_3, v_1), (u_2, v_2)\}.$
- The matching is equal to 15.



# Summary

# Maximum-Weight Bipartite Matching

- Weighted bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$. (Edges have weights: $w_{uv}$.)
- Matching is a subset of edges without common vertices.
- Denote the matching by set $\mathcal{S} \subseteq \mathcal{E}$.
- Sum of weights in matching $\mathcal{S}$:

$$f(\mathcal{S}) = \sum_{(u,v) \in \mathcal{S}} w_{uv}.$$

- Find matching $\mathcal{S}$ that has the maximum weight:

$$\max_{\mathcal{S}} f(\mathcal{S}).$$

# Maximum Matching ⟺ Minimum Matching

- Maximum matching: $\max_{\mathcal{S}} f(\mathcal{S})$.
- Minimum matching: $\min_{\mathcal{S}} f(\mathcal{S})$.
- The maximum matching problem can be reduced to minimum matching problem by flipping the signs of weights.
- Algorithms that solve minimum matching can also solve the maximum matching problem.

# Hungarian Algorithm

- Hungarian algorithm finds minimum-weight bipartite matching.
- It requires $|\mathcal{U}| = |\mathcal{V}| = n$.
- Time complexity: $O(n^3)$.

# Questions

# Question 1

- The right is the adjacency matrix of a bipartite graph.
- Find the minimum matching in the graph.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|
| $u_1$ | 20 | 15 | 18 | 24 | 25 |
| $u_2$ | 18 | 20 | 12 | 14 | 15 |
| $u_3$ | 21 | 23 | 25 | 27 | 26 |
| $u_4$ | 17 | 18 | 21 | 23 | 22 |
| $u_5$ | 19 | 22 | 16 | 21 | 20 |

# Question 2

- The right is the adjacency matrix of a bipartite graph.
- Find the maximum matching in the graph.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|
| $u_1$ | 20 | 15 | 18 | 24 | 25 |
| $u_2$ | 18 | 20 | 12 | 14 | 15 |
| $u_3$ | 21 | 23 | 25 | 27 | 26 |
| $u_4$ | 17 | 18 | 21 | 23 | 22 |
| $u_5$ | 19 | 22 | 16 | 21 | 20 |

# Thank You!