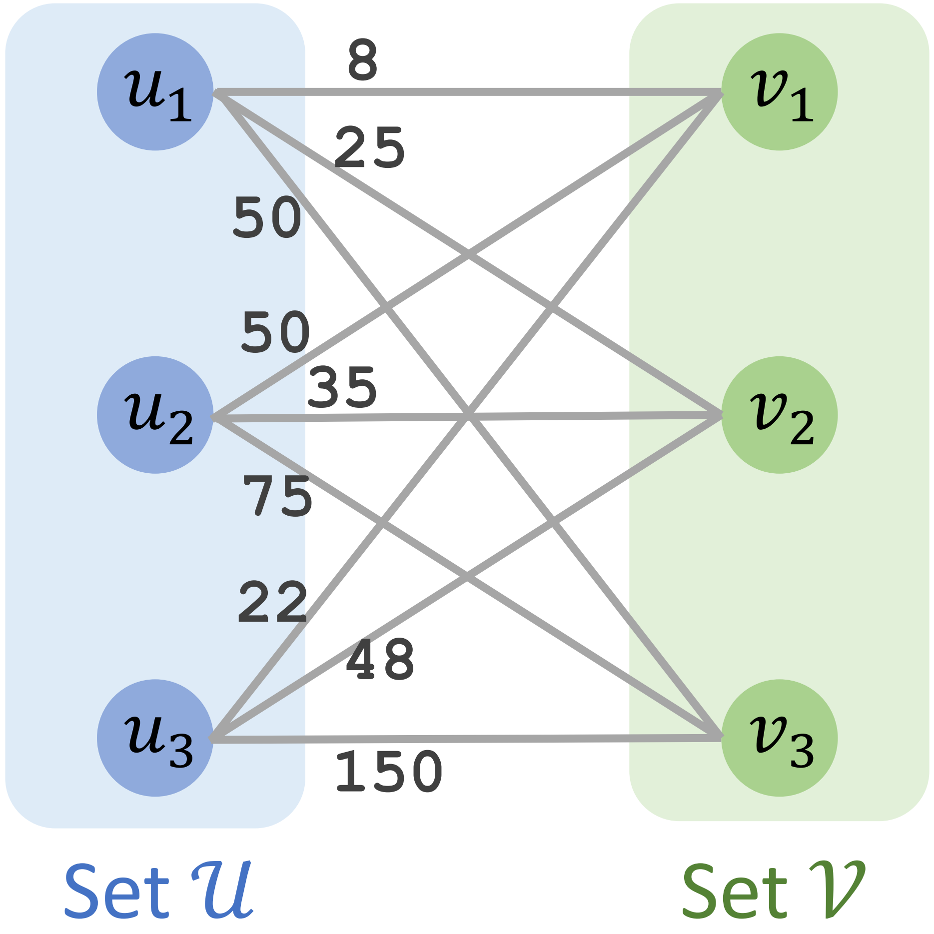
Hungarian Algorithm

Shusen Wang

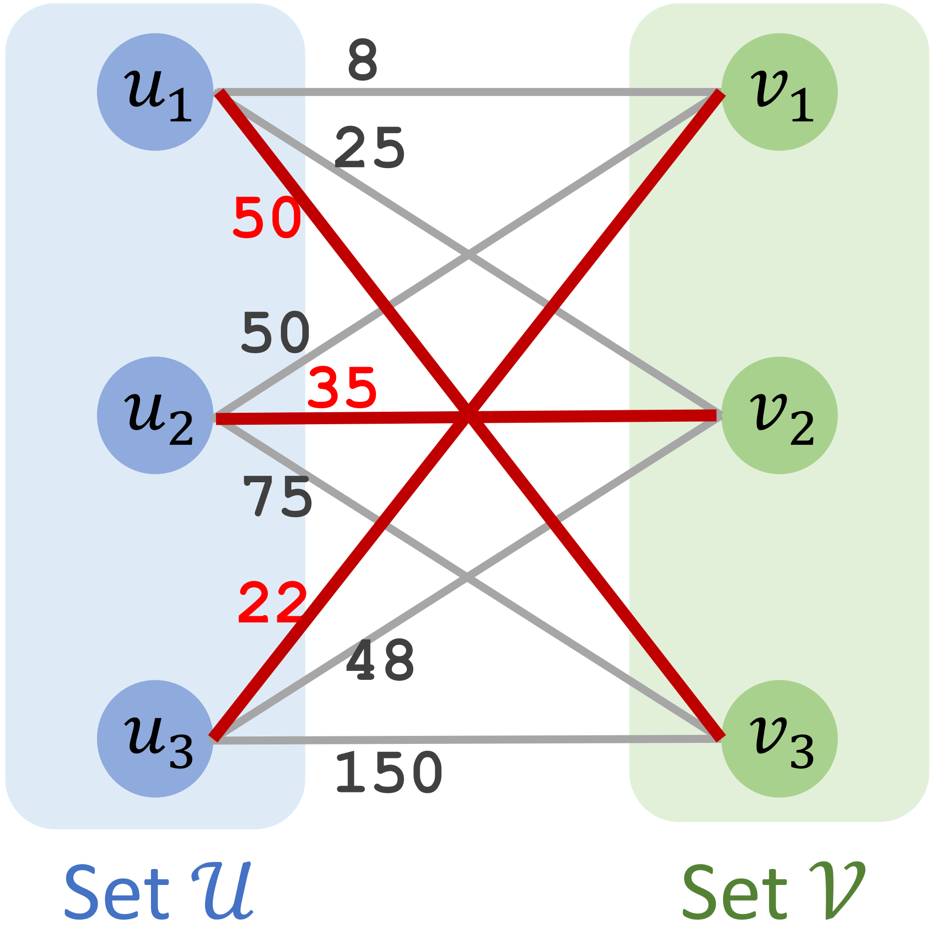
Minimum-Weight Bipartite Matching

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 $50 + 35 + 22 = 107$.

Subtract Row Minima

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

Subtract Row Minima

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

Subtract Row Minima

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

Subtract Row Minima

Now, the row minima are zeros.

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

Subtract Column Minima

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

Subtract Column Minima

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

Subtract Column Minima

	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

Subtract Column Minima

Now, the column minima are zeros.

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

Iteration 1

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

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

Iteration 1A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

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

Iteration 1A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

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

Not optimal!

Iteration 1A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

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

Iteration 1B

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

- If n lines are required, the algorithm stops.
- If less than n lines are required, then continue with Step C.

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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

First, find the smallest element (denote k) that is not covered by a line.

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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

First, find the smallest element (denote 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Second, 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Second, 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Third, add k to all the 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Third, add k to all the 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

Iteration 2

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

$$
\begin{array}{c|ccc|}
 & v_1 & v_2 & v_3 \\
\hline
u_1 & 0 & 15 & 0 \\
u_2 & 17 & 0 & 0 \\
u_3 & 0 & 24 & 86 \\
\hline
\end{array}
$$

Iteration 2A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

	v_1	v_2	v_3
u_1	0	15	0
u_2	17	0	0
u_3	0	24	86

Iteration 2A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

	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.

Iteration 2B

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

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

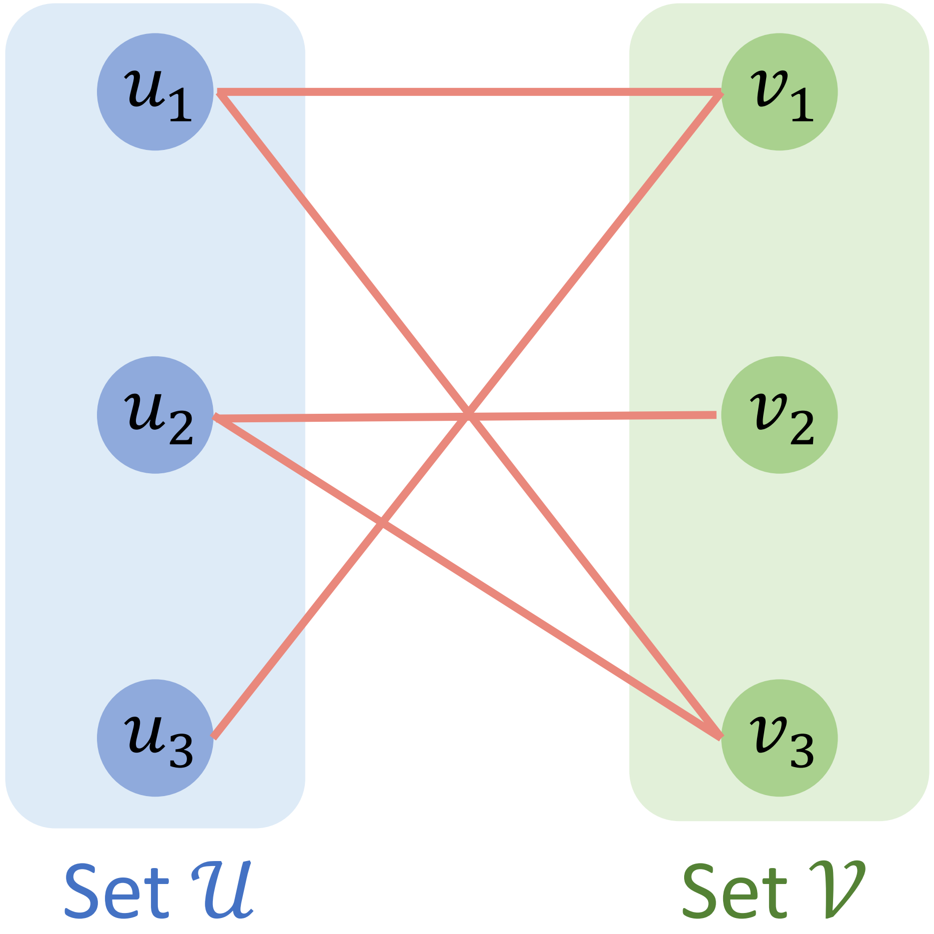
	v_1	v_2	v_3
u_1	0	15	0
u_2	17	0	0
u_3	0	24	86

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.

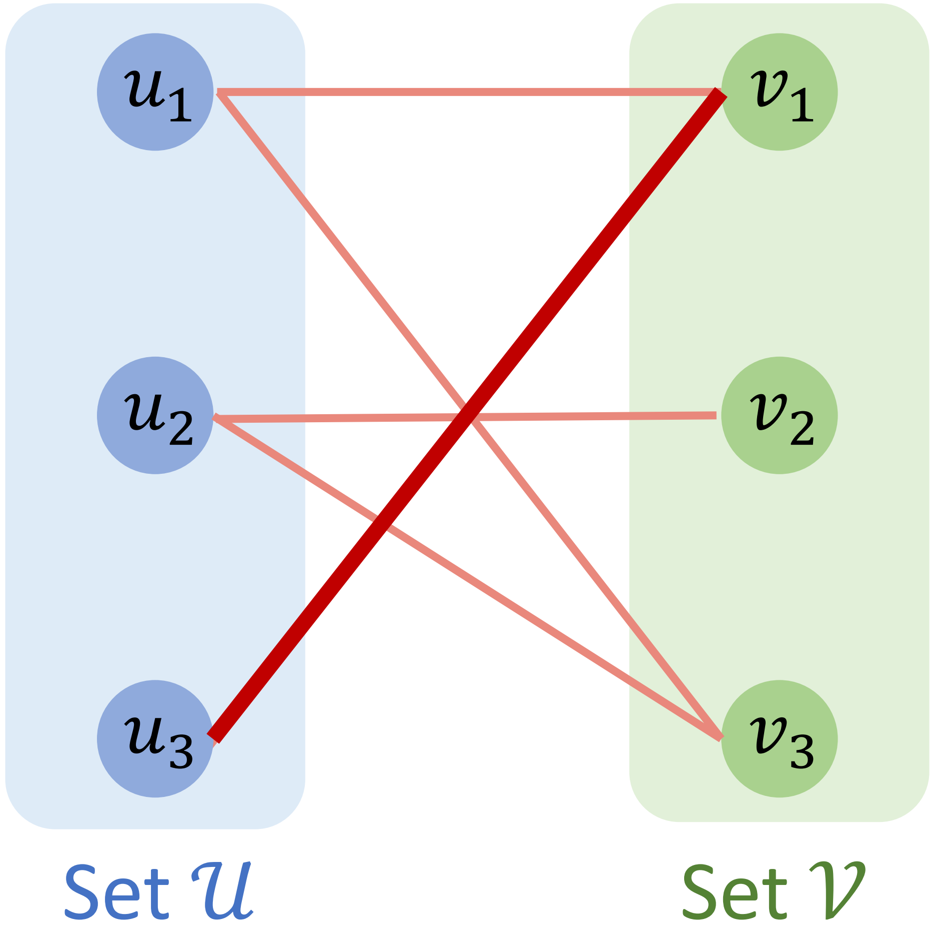
Output the matching



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

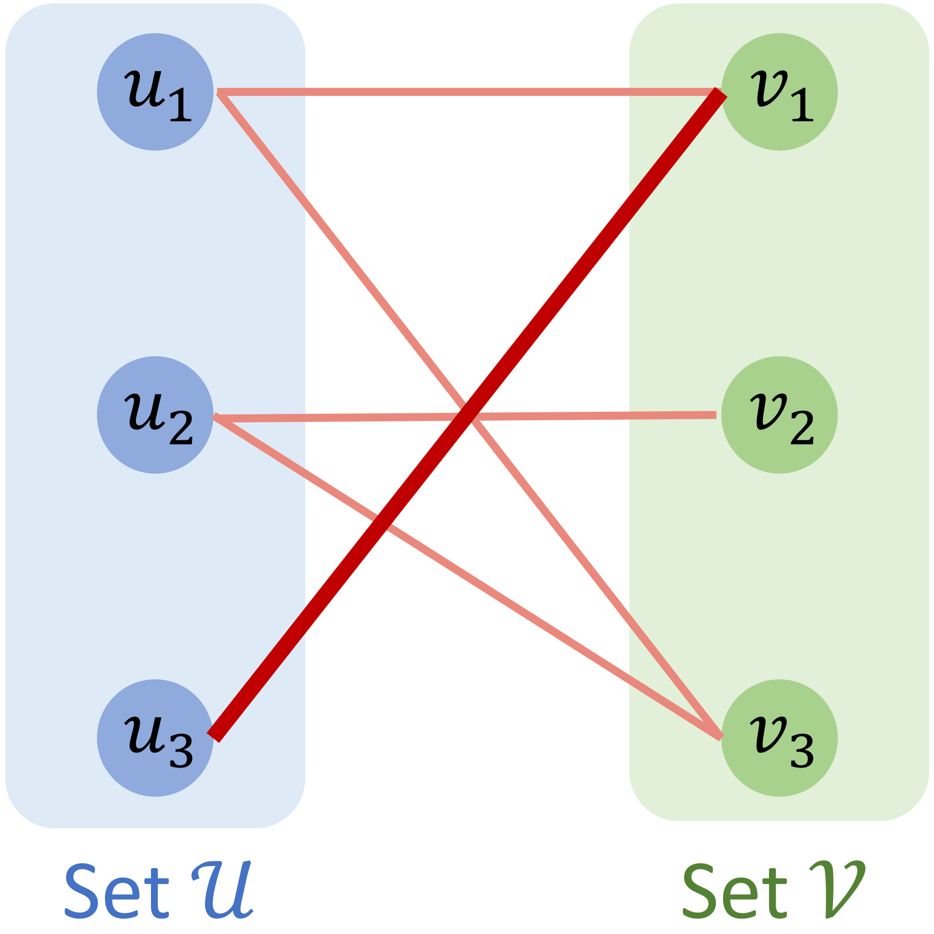
- Choose a matching among the zeros.
- Think of the zeros as edges.

Output the matching



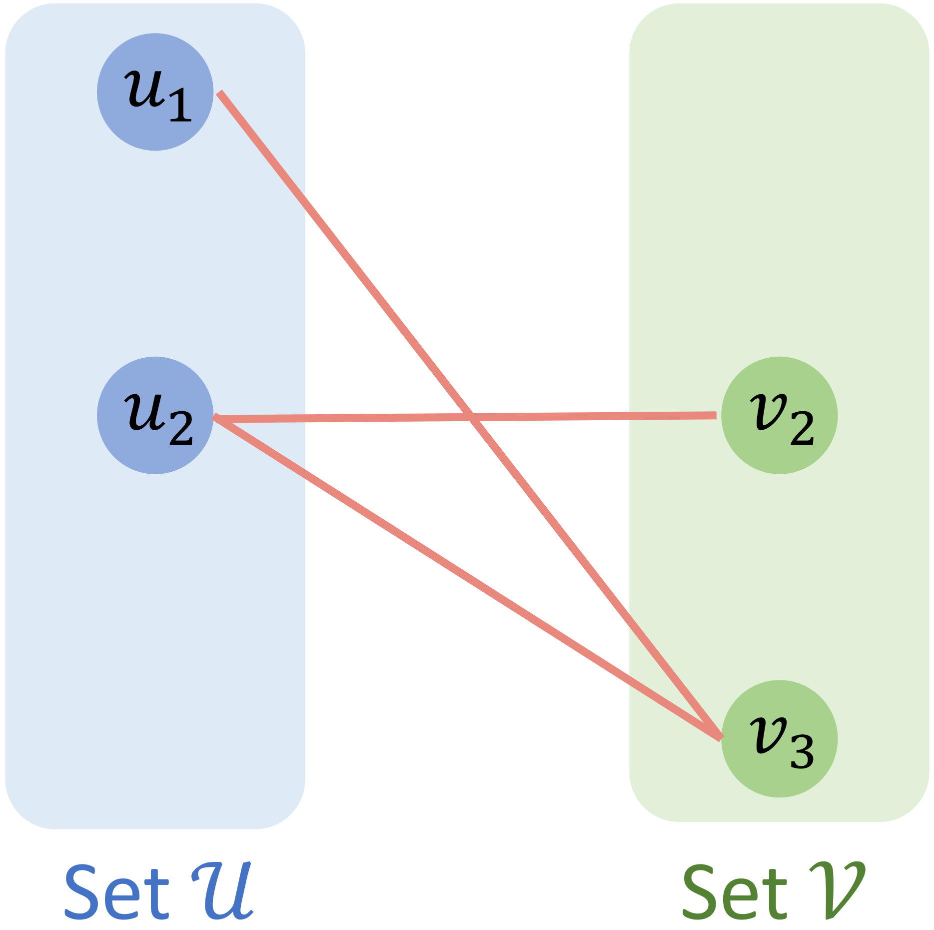
	v_1	v_2	v_3
u_1	**0**	15	**0**
u_2	17	**0**	**0**
u_3	**0**	24	86

Output the matching



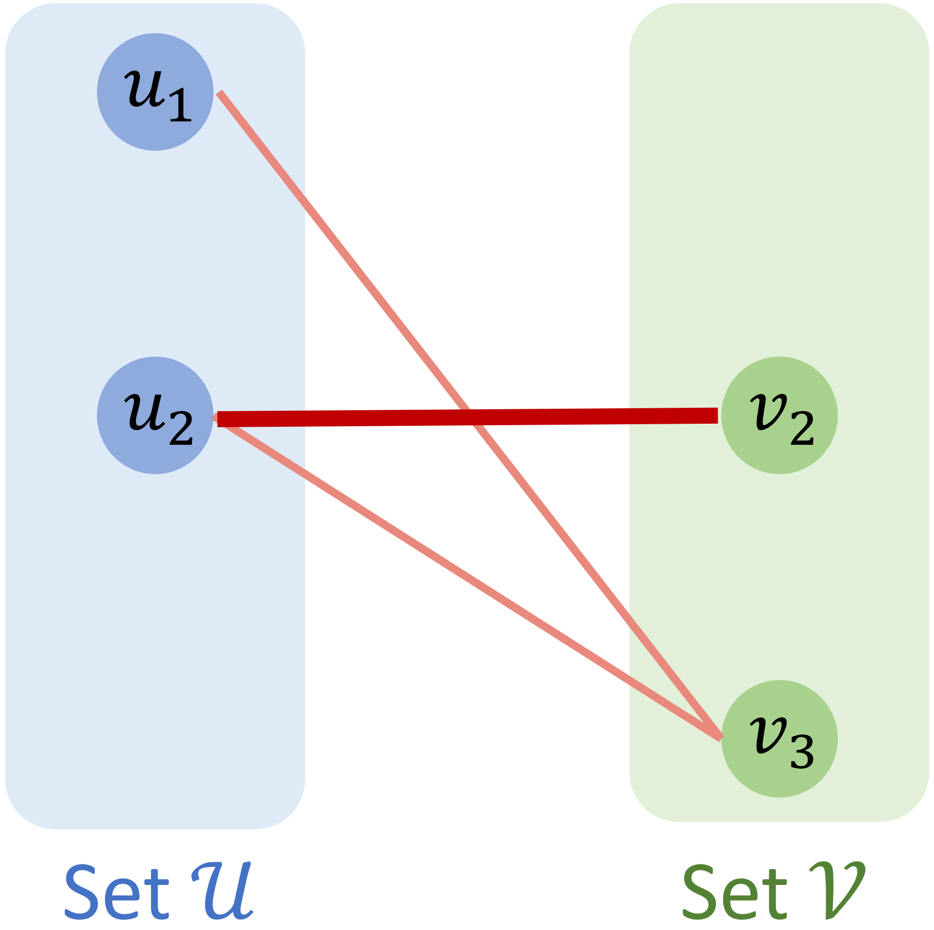
	v_1	v_2	v_3
u_1	**0**	15	**0**
u_2	17	**0**	**0**
u_3	**0**	24	86

Output the matching



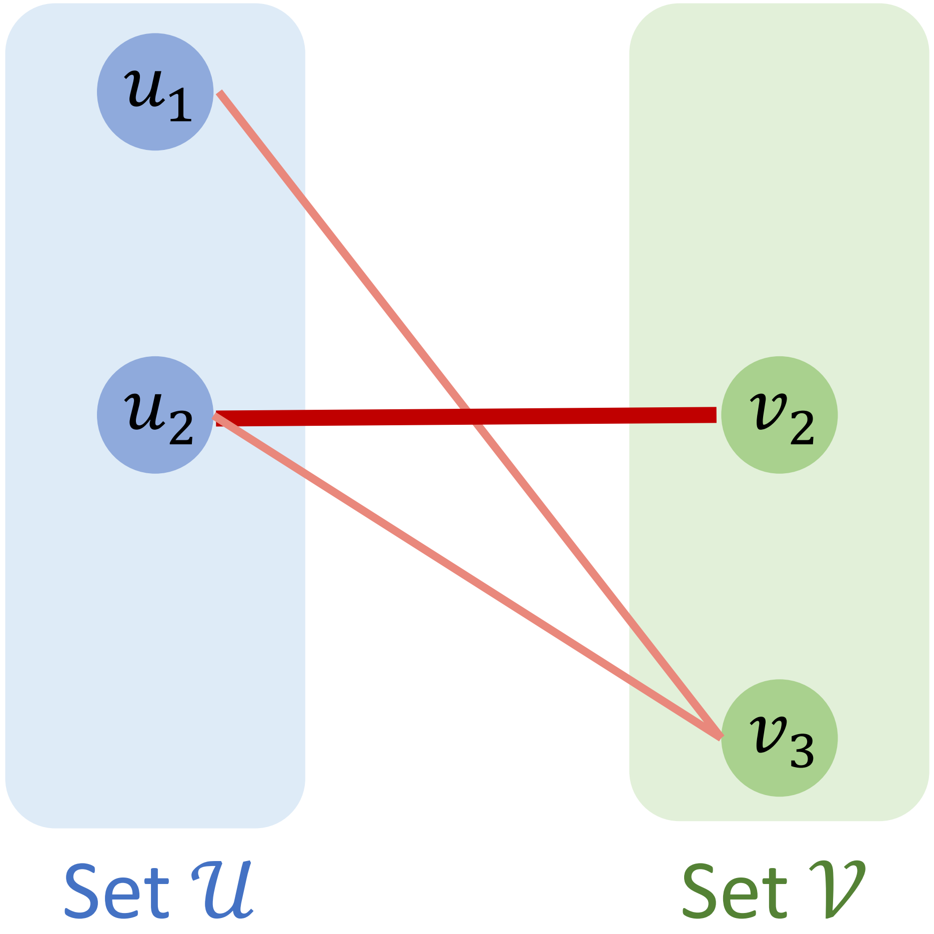
	v_1	v_2	v_3
u_1	0	15	**0**
u_2	17	**0**	**0**
u_3	0	24	86

Output the matching



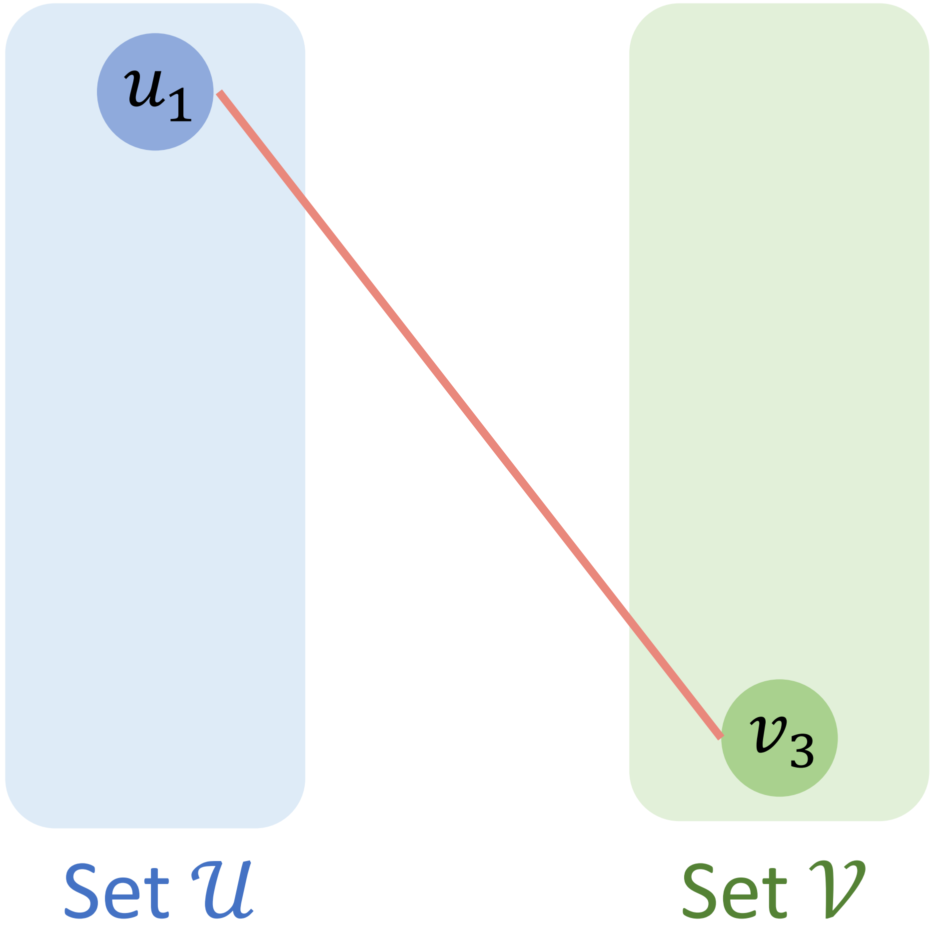
	v_1	v_2	v_3
u_1	0	15	0
u_2	17	0	0
u_3	0	24	86

Output the matching



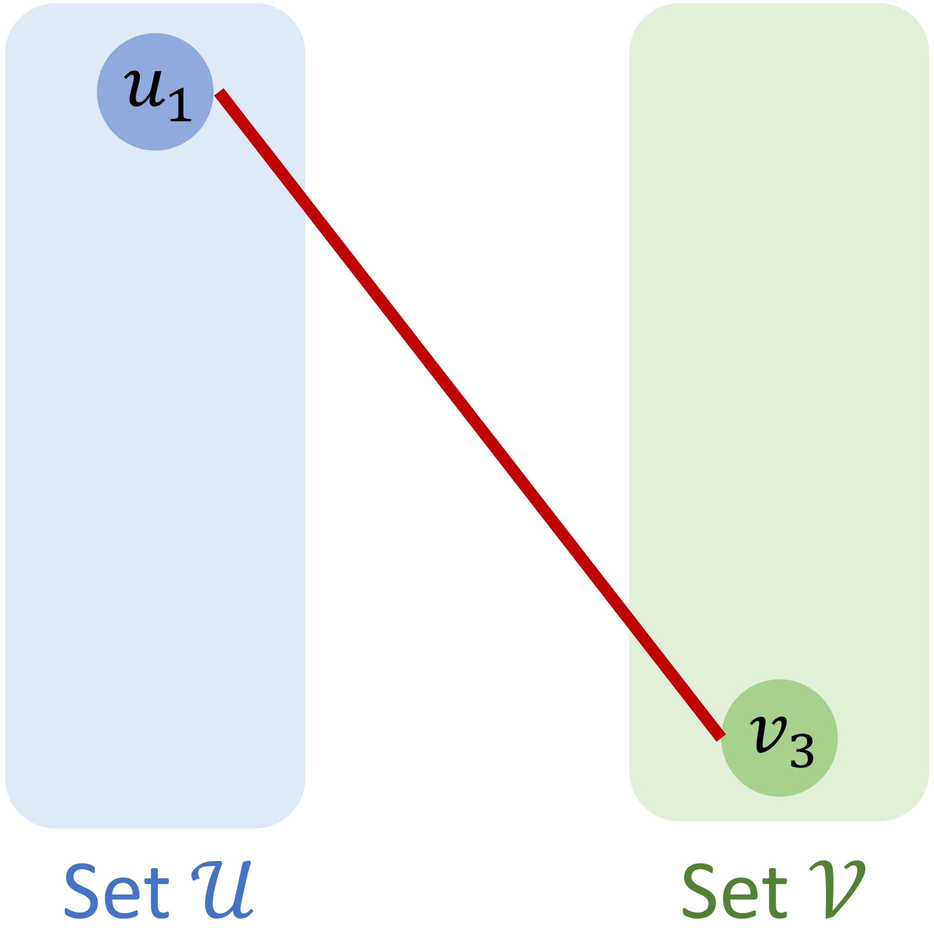
	v_1	v_2	v_3
u_1	0	15	0
u_2	17	0	0
u_3	0	24	86

Output the matching



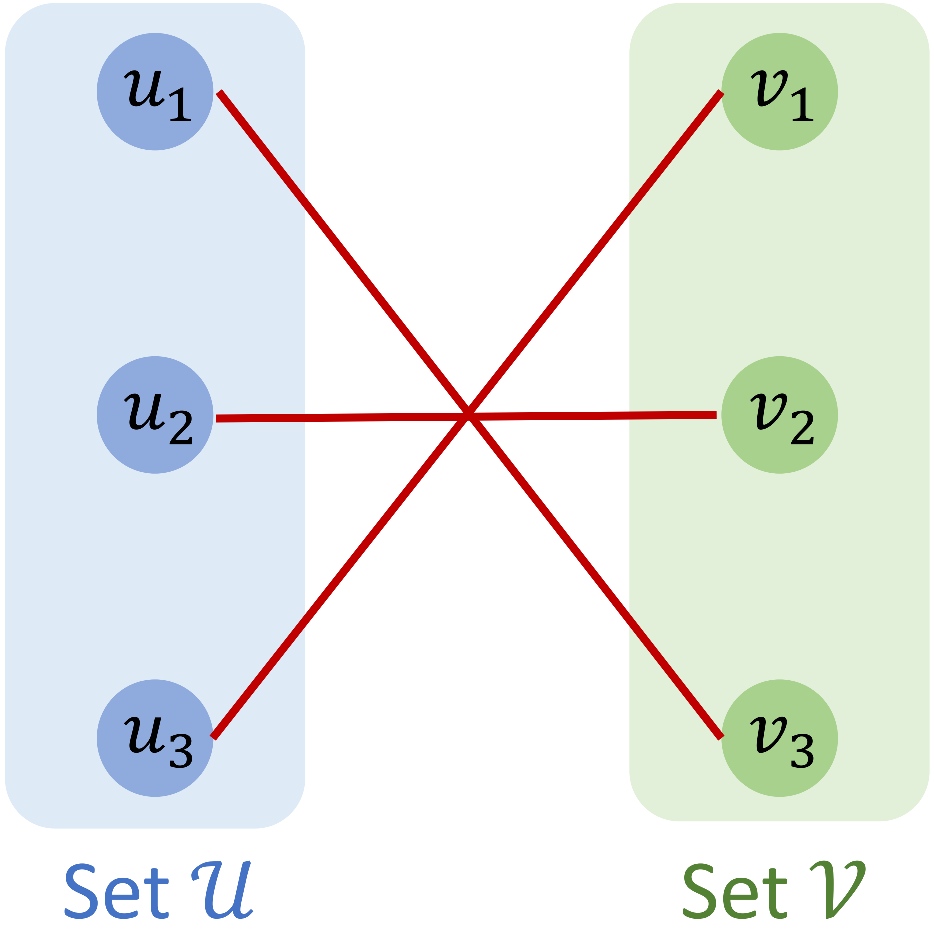
	v_1	v_2	v_3
u_1	0	15	**0**
u_2	17	0	0
u_3	0	24	86

Output the matching



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

Output the matching



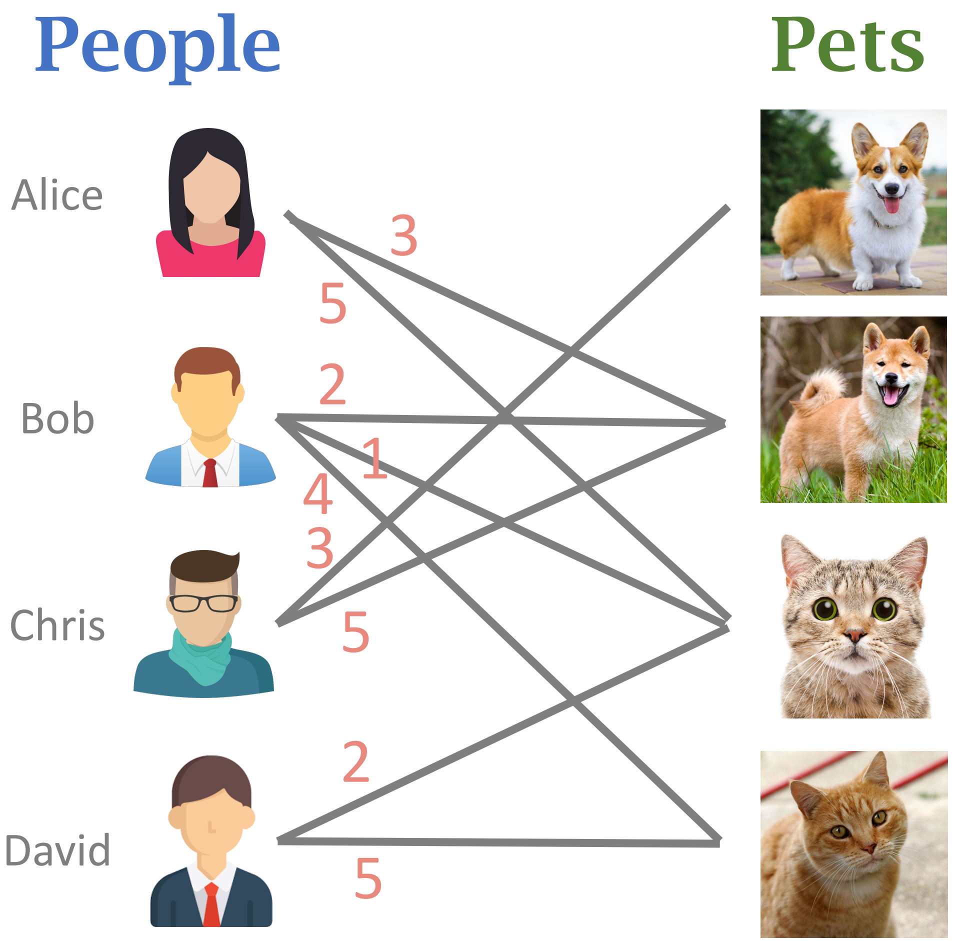
	v_1	v_2	v_3
u_1	**0**	15	**0**
u_2	17	**0**	**0**
u_3	**0**	24	86

The matching is

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

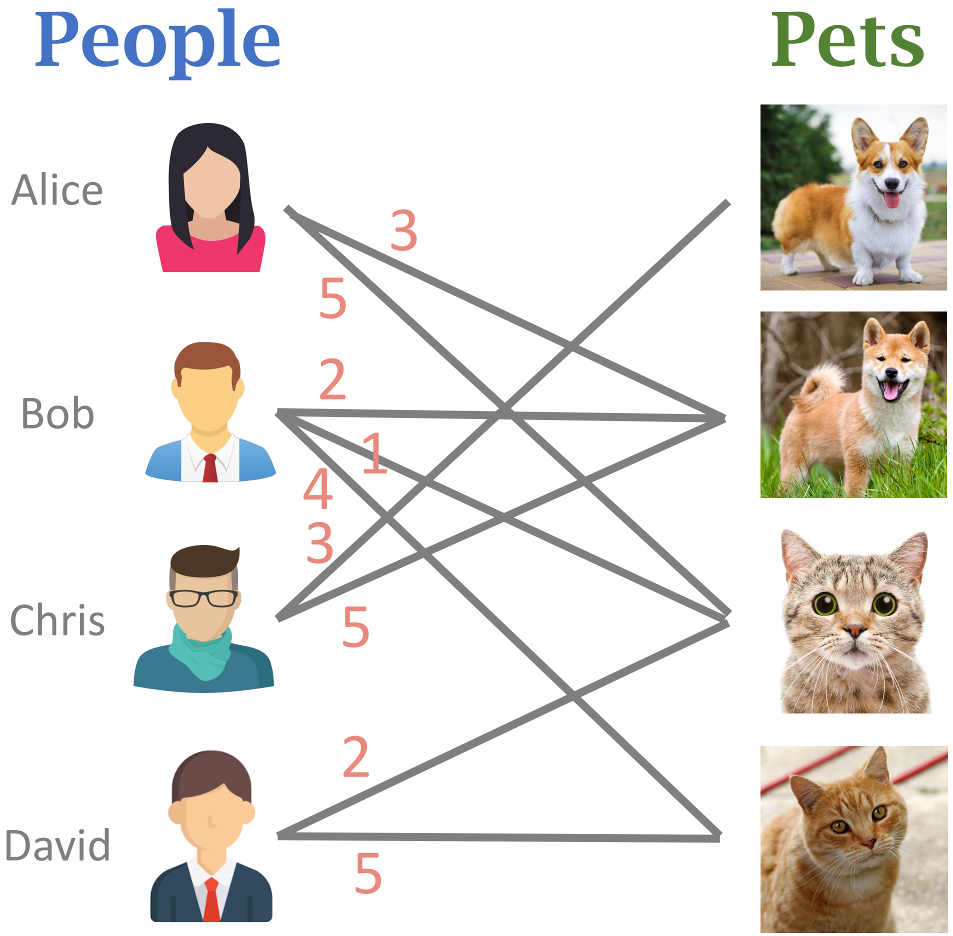
Maximum-Weight Bipartite Matching

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.)

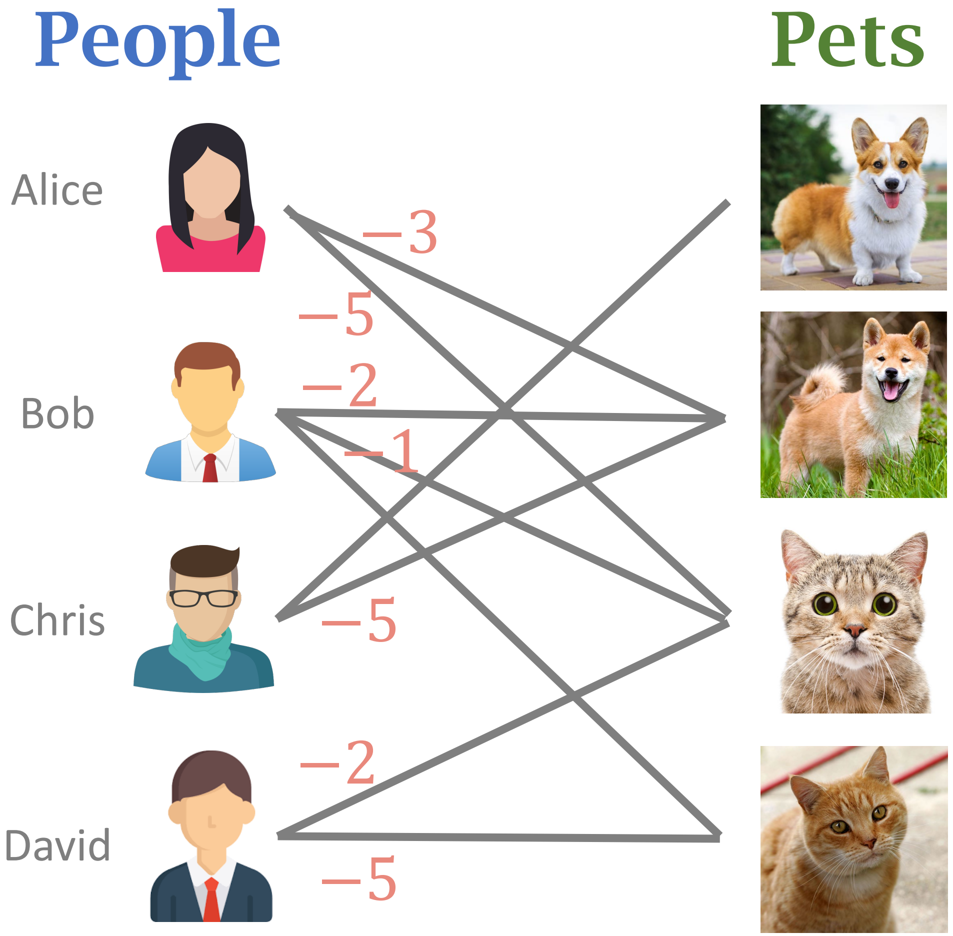
Hungarian Algorithm for Maximum Matching



Idea: Max Matching ➡ Min Matching

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

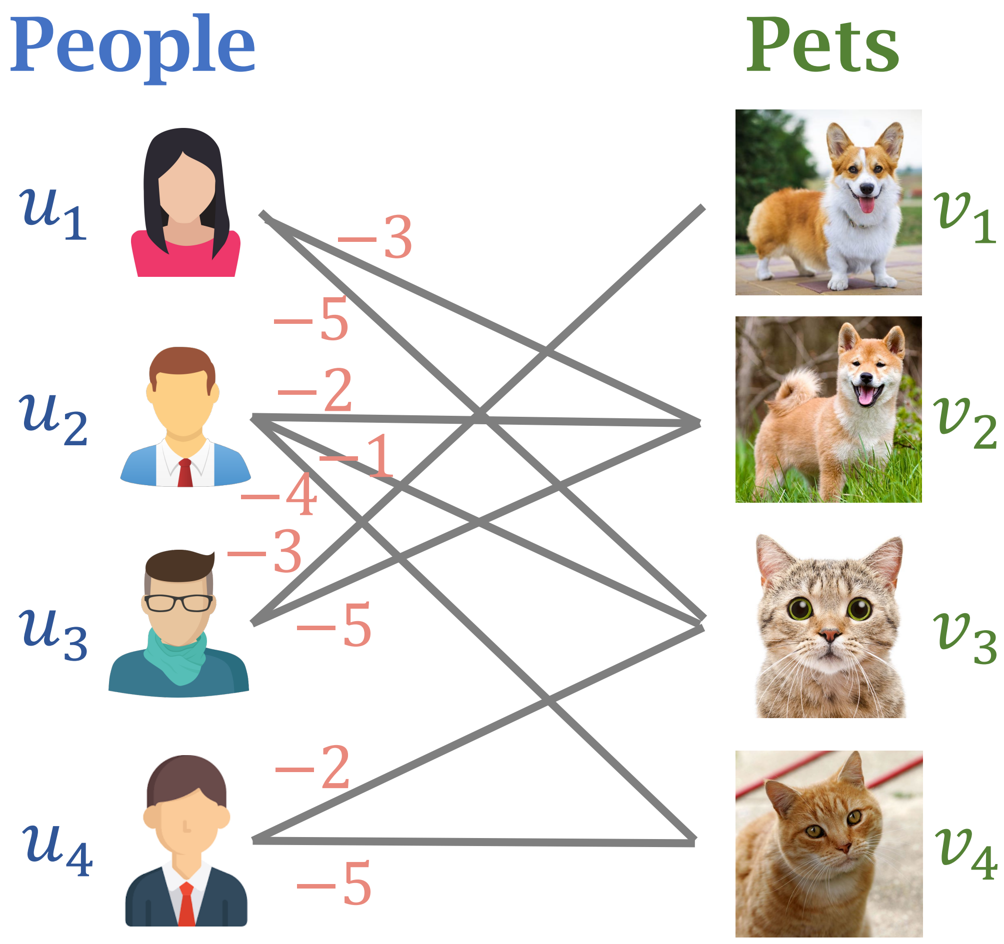
Hungarian Algorithm for Maximum Matching



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

Subtract Row Minima

	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

Subtract Row Minima

	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)

Subtract Row Minima

Now, the row minima are zeros.

	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

Subtract Column Minima

	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

Subtract Column Minima

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline u_1 & 5 & 2 & 0 & 5 \\ u_2 & 4 & 2 & 3 & 0 \\ u_3 & 2 & 0 & 5 & 5 \\ u_4 & 5 & 5 & 3 & 0 \end{array}$$

Subtract Column Minima

	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

Subtract Column Minima

Now, the column minima are zeros.

	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

Iteration 1

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

	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

Iteration 1A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

	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

Iteration 1B

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

- If n lines are required, the algorithm stops.
- If less than n lines are required, then continue with Step C.

	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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

First, find the smallest element (denote 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	3	0
u_3	0	0	5	5
u_4	3	5	3	0

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

First, find the smallest element (denote 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Second, 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Second, 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Third, add k to all the 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

Iteration 1C

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

Third, add k to all the 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

Iteration 2

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline u_1 & 3 & 2 & 0 & 7 \\ u_2 & 0 & 0 & 1 & 0 \\ u_3 & 0 & 0 & 5 & 7 \\ u_4 & 1 & 3 & 1 & 0 \end{array}$$

Iteration 2A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

	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

Iteration 2A

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

At least 3 lines are needed.

$$
\begin{array}{c|cccc|}
 & v_1 & v_2 & v_3 & v_4 \\
\hline
u_1 & 3 & 2 & 0 & 7 \\
u_2 & 0 & 0 & 1 & 0 \\
u_3 & 0 & 0 & 5 & 7 \\
u_4 & 1 & 3 & 1 & 0 \\
\hline
\end{array}
$$

Iteration 2B

Repeat the followings:

A. Cover all the zeros with a minimum number of lines.

B. Decide whether to stop.

C. Create additional zeros.

If n lines are required, the algorithm stops.

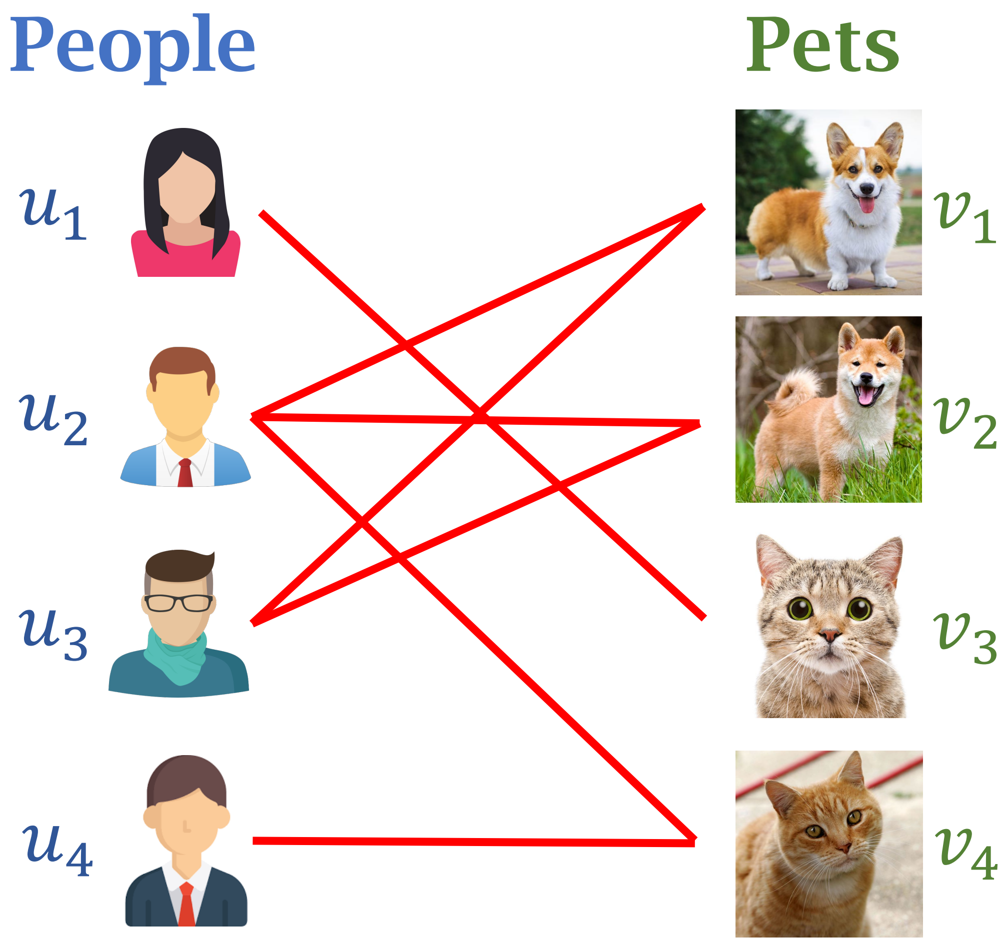
The algorithm stops.

	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

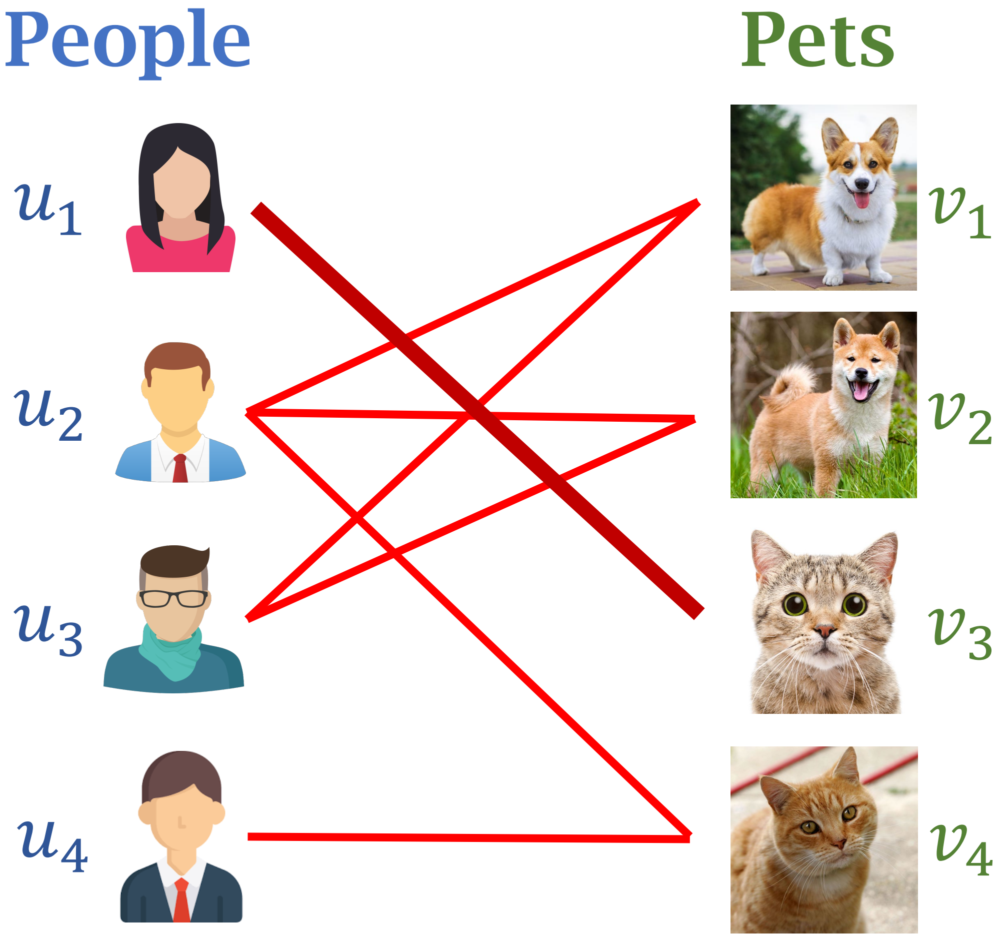
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

Output the matching



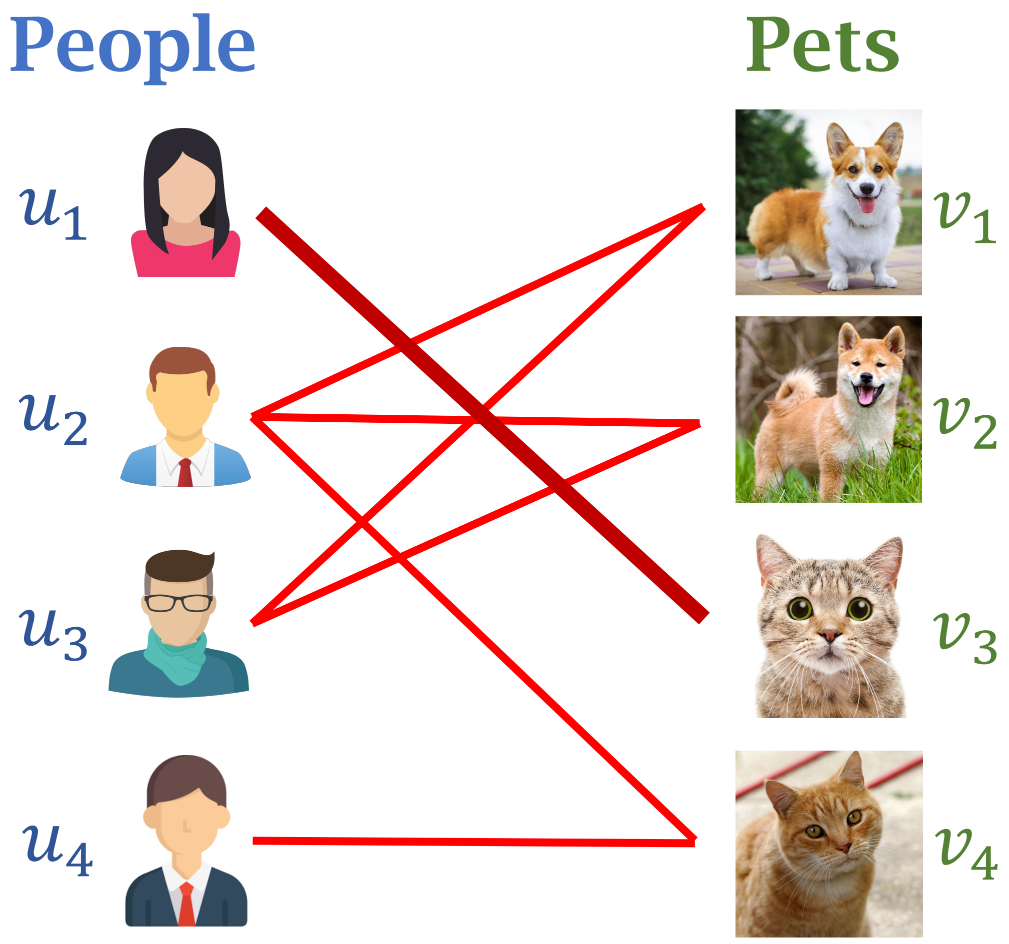
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

Output the matching



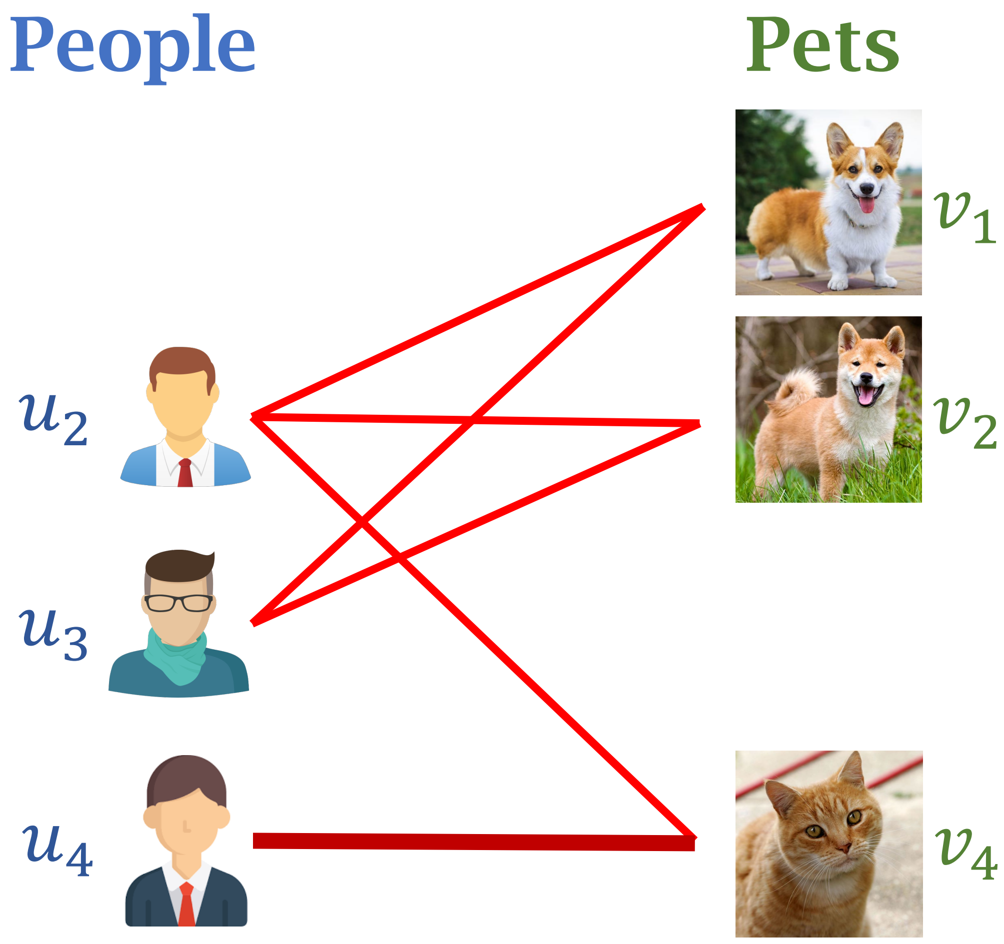
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

Output the matching



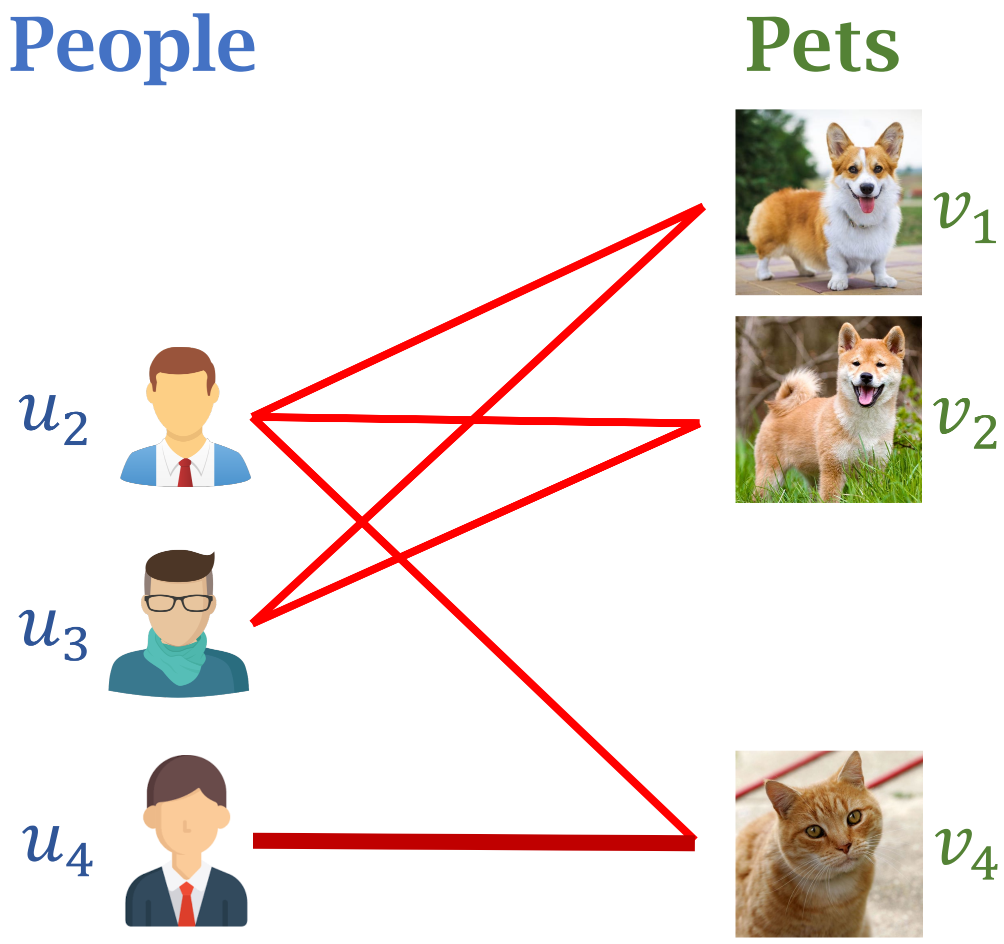
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

Output the matching



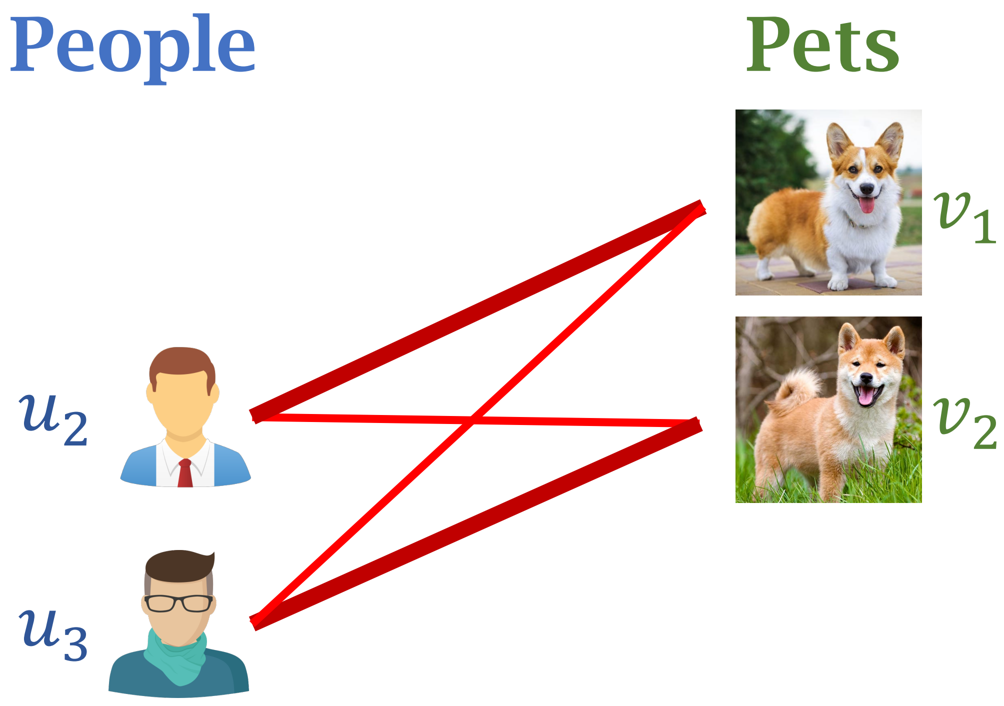
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

Output the matching



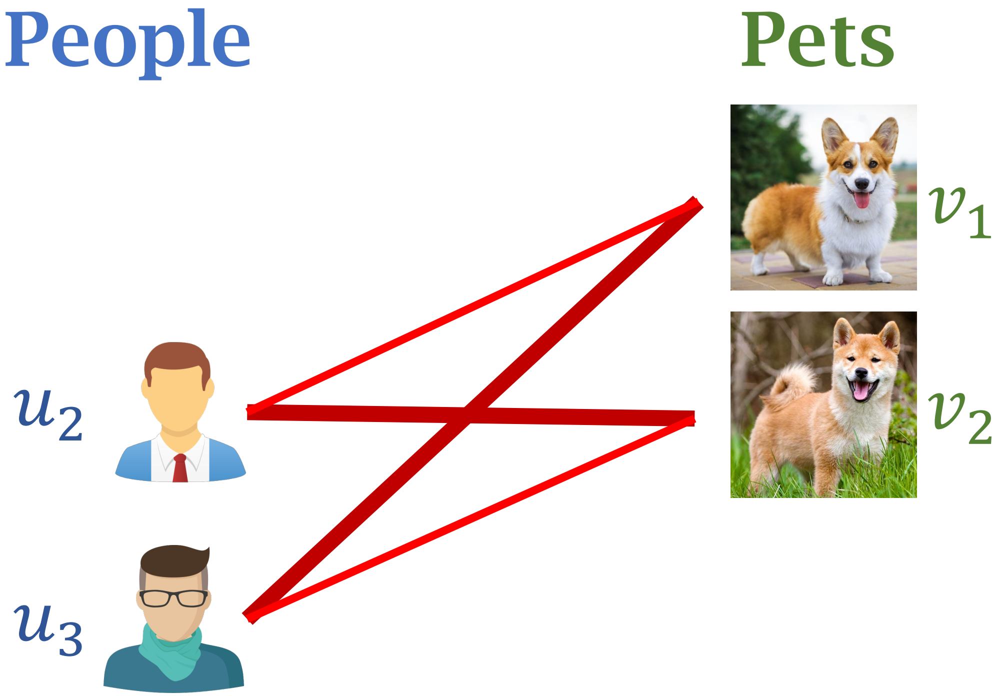
	v_1	v_2	v_3	v_4
u_1	3	2	0	7
u_2	0	0	1	0
u_3	0	0	5	7
u_4	1	3	1	0

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

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

$$
\begin{array}{c|cccc|}
 & v_1 & v_2 & v_3 & v_4 \\
\hline
u_1 & 3 & 2 & 0 & 7 \\
u_2 & 0 & 0 & 1 & 0 \\
u_3 & 0 & 0 & 5 & 7 \\
u_4 & 1 & 3 & 1 & 0 \\
\hline
\end{array}
$$

Output the matching

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

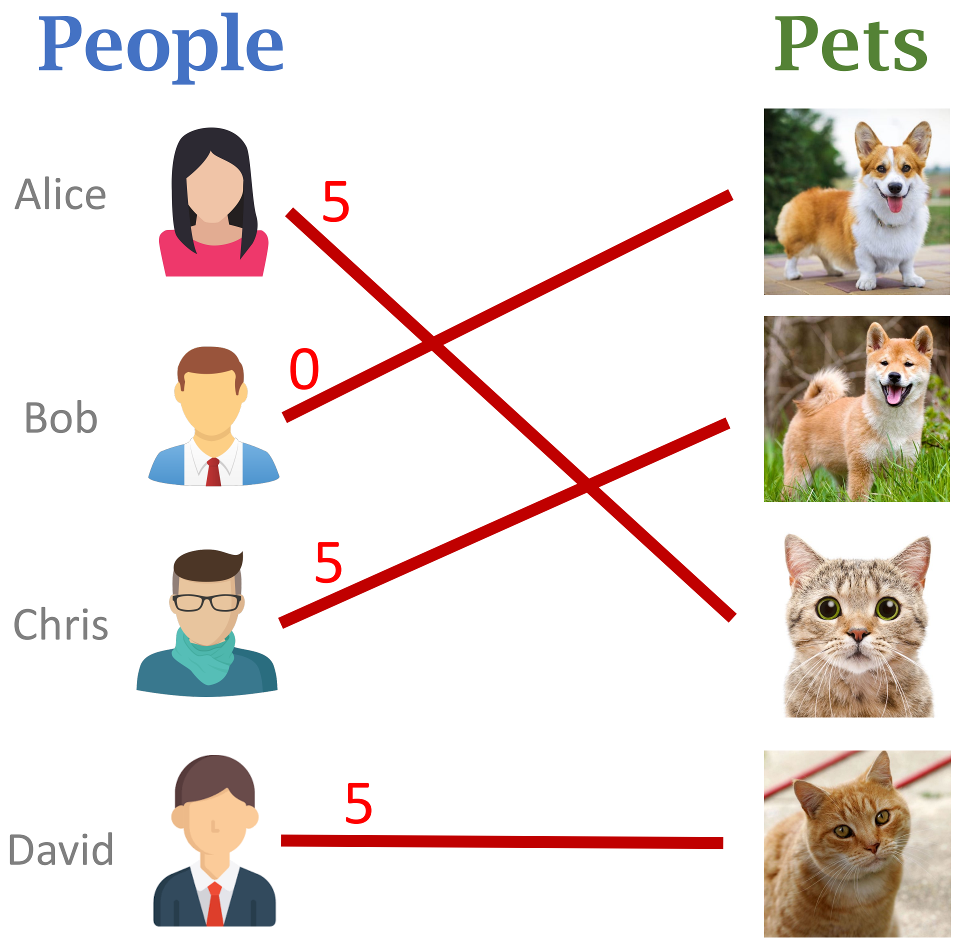
- Or return the matching:
  $\mathcal{S} = \{(u_1, v_3), (u_4, v_4), (u_3, v_1), (u_2, v_2)\}.$

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



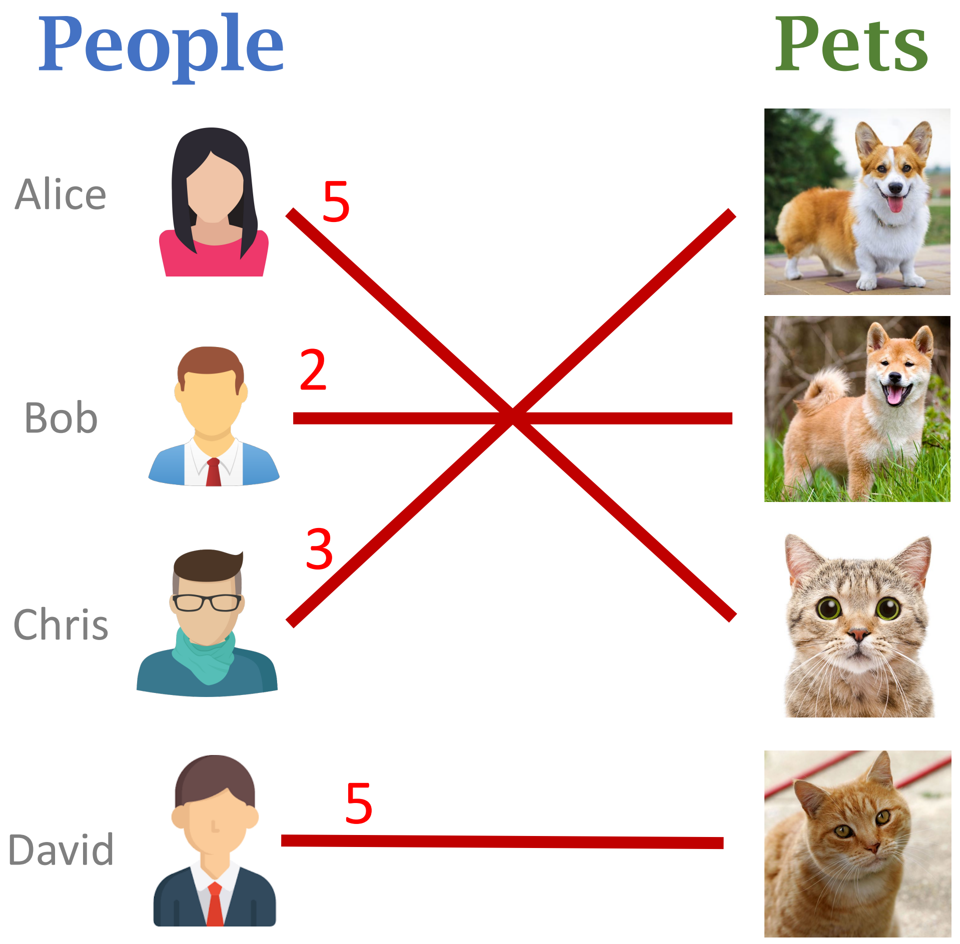
Output the matching

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

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 on 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 on 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!