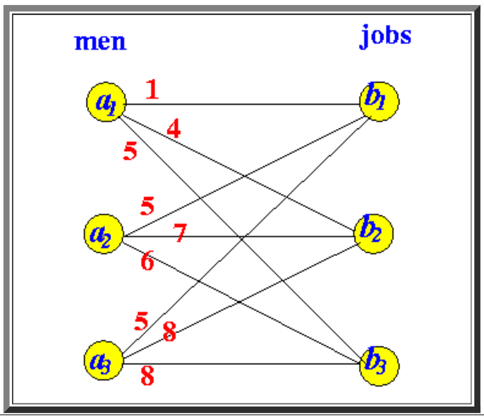
The Hungarian Algorithm for the Assignment Problem

- **Historical background on the Hungarian Algorithm**
  - History:
    - The **Hungarian method** is a **combinatorial optimization algorithm** which solves the **assignment problem** in polynomial time
    - Later it was discovered that it was a **primal-dual Simplex method**.
    - It was developed and published by **Harold Kuhn in 1955**, who gave the name "Hungarian method" because the algorithm was largely **based on the earlier works** of two Hungarian mathematicians: **Denes Konig and Jeno Egervary**.

- **Overview of the Hungarian Algorithm**
  - **Recall the Goal:**
    - Find a **minimum cost complete matching** in a **weighted bi-partite graph**

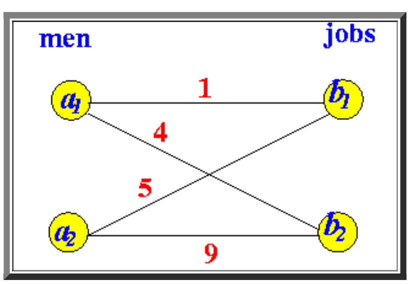
      Example weighted bi-partite graph:



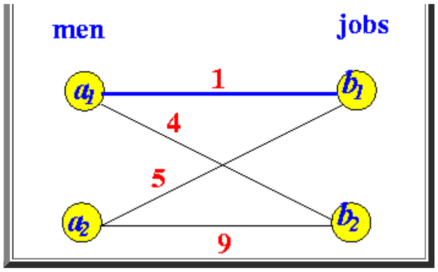
  - **Pseudo code:**

```
Construct a subgraph graph G consisting of the "best cost edges";
            // We will discuss what is a "best cost edge" later

Find a maximal matching M in subgraph G

repeat until M is a complete matching
{
   Add the "next best cost edges" to G;
              // Notice the quotes: it's more complex that just
              // looking at the cost of an edge

   Find a maximal matching M in (modified) subgraph G;
}
```

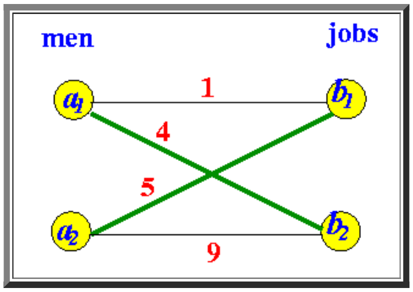
- **Caveat on finding the "best cost edges"**
  - **Caveat:**
    - The **best cost edges** may ***not*** be the **least cost edges**
  - Consider the following **simple example:**



The **least cost edge** in the graph is **(a_1, b_1)**:



However, the **minimum cost solution does *not*** contain the edge **(a_1, b_1)**:

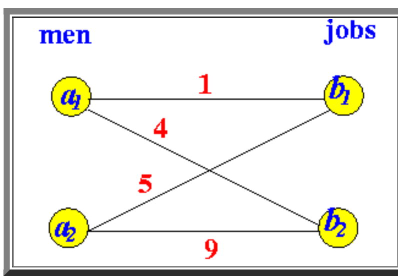


Reason:

- When we **match** vertex a_1 with vertex b_1, we are **also** ***forcing***:
  - vertex a_2 to be **matched up** with vertex b_2 !!!!
  - The **cost** of this **matchup more than *nullify*** the **minimum cost** achieved by **matching** vertex a_1 with vertex b_1

- **Fact:**
  - When a **vertex** b_j is **matched** with some **vertex** a_i, it will:
    - **Remove** the **opportunity** to **match vertex** b_j with some ***other* vertex** a_k
    - This **lost opportunity carries** a **cost** !!!

- **Simple example to illustrate how to find "best cost edges"**
  - **Example:**



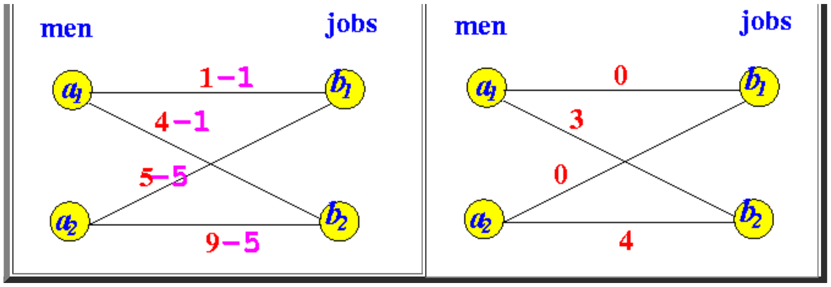

```
Cost matrix:

     | b1   b2
 ----+----------
  a1 |  1    4
  a2 |  5    9
```

- The **minimum cost** incurred to match **a_1 = 1** (by matching a_1 to b_1)

  The **minimum cost** incurred to match **a_2 = 5** (by matching a_2 to b_1)

  Compute the **additional incurred cost** when using **non-optimal edged** by **subtracting** the **minimum cost** from the other edges that is **incident** to that **same node**:

Subtract min cost:	Result:

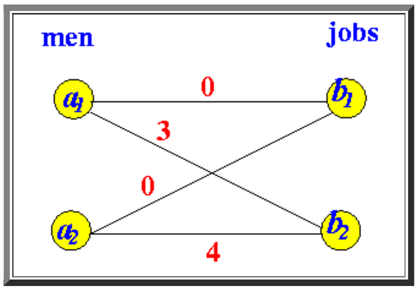


The **0-weight edges** are the **best edges** to **match** a_1 and a_2 when **nothing else matters**

However, we know that **something else does matter**:

- a_1 and a_2 **cannot** be **matched** to the **same vertex** b_1

- Consider the **normalized cost graph**:



How to read the cost **3** and **4**:

- It will cost **3 extra $** (or $3 more than best edge) to **match** a_1 with b_2
- It will cost **4 extra $** (or $4 more than best edge) to **match** a_2 with b_2

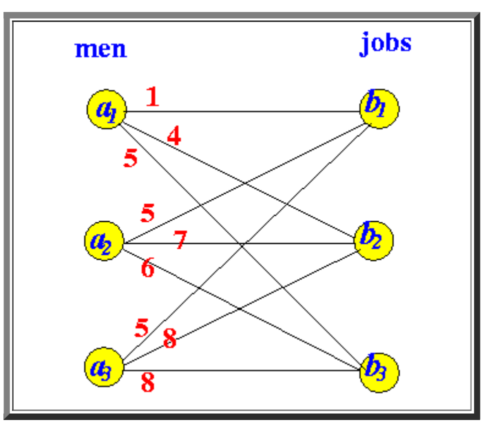
Now you can tell exactly what you need to do:

- It is **cheaper** to force a_1 to **match up** with b_2 !!!

Therefore:

- The edge **(a_1,b_2)** is ***also*** one of the **best cost edges** !!!

- **The Hungarian Algorithm (with examples)**
  - **Example assignment problem:**

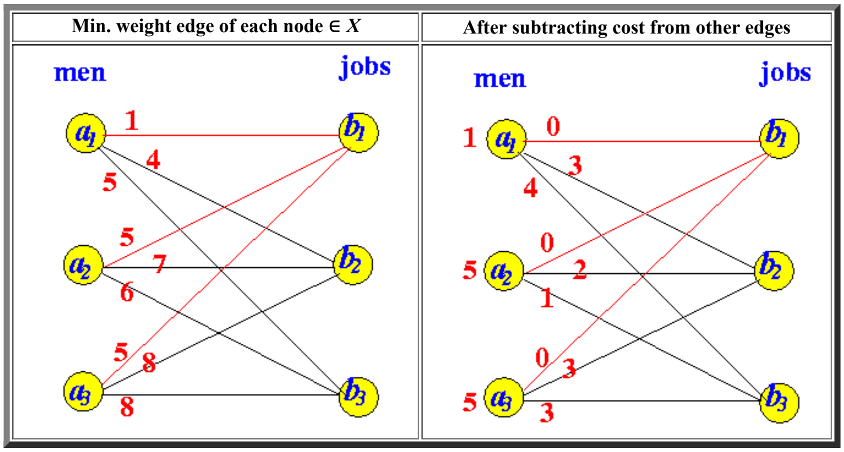


- **The Hungarian algorithm: initial step**
  - **Initial step:**
    - For **each vertex** ∈ *X* **(men)**:

1. find the **minimal cost edge** and
2. **subtract its weight** from all weights **connected with that vertex**.

You will get **at least one 0-weight edges** for each node.

Example:

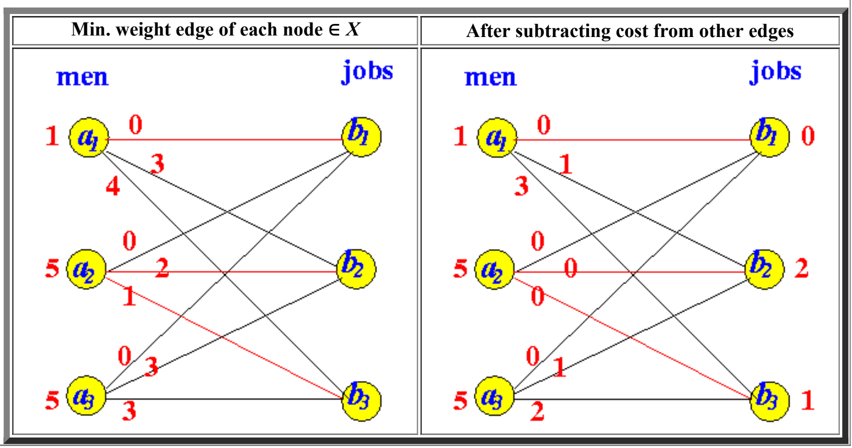


- For **each vertex** ∈ ***Y*** **(jobs)**:

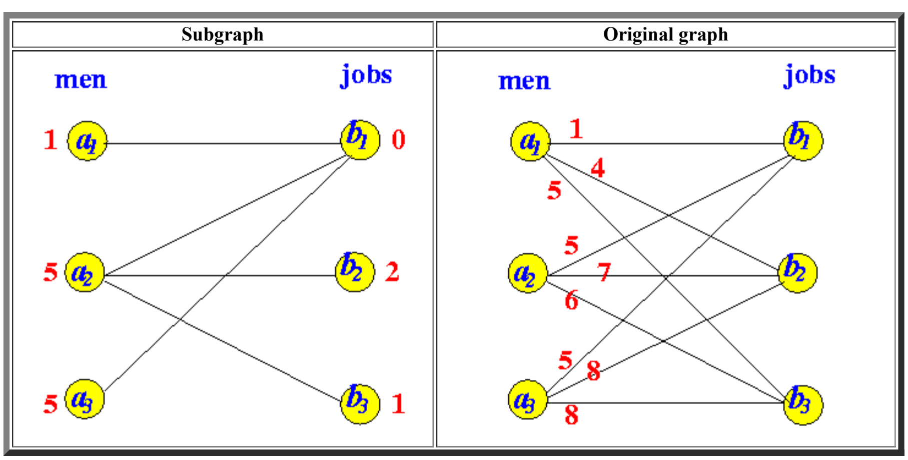
1. find the **minimal cost edge** and
2. **subtract its weight** from all weights **connected with that vertex**.

You ***may*** get more **0-weight edges** (and you not get any more)

Example:



- Consider the **subgraph** consisting ***only*** of the **0-weight edges** after **step 0**:

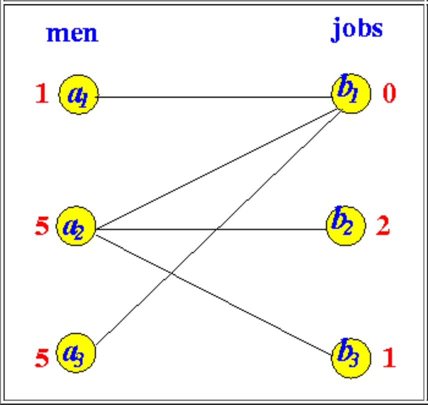
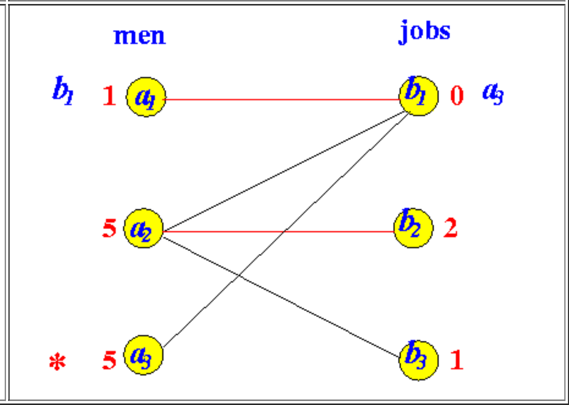


Note:

- These **edges** will provide the **lowest possible cost** ***if*** we can **find a maximum matching** (involving all element in **set *X* (all men)**)

- Find a **maximal matching** in the **subgraph** consisting ***only*** of the **0-weight edges**

  Example:

Subgraph with 0-weight edges	A maximal matching
	

  Note:

  - The **graph** includes the **labels** made by the ***last*** **step** of the **max. flow algorithm**

- If **matching = *maximum* (all vertices ∈ *X***, then we are **done**

  But in this case, we are **not done (yet)** and enter the **iterative step**

  Iterative step:

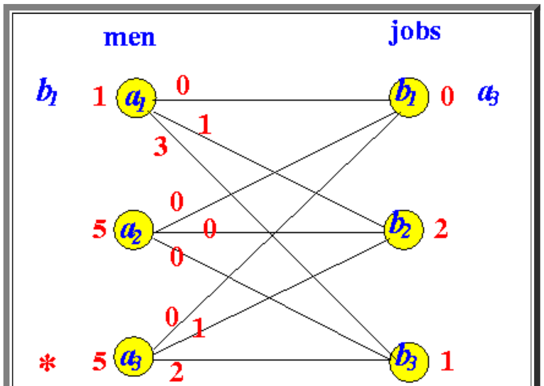
  - **Add** the **"next least cost edge(s)"** to the **subgraph** (it's **more complex** that just finding the smallest cost, because you have to consider the effect of **other nodes**) and
  - Find a new **maximal mathcing**

- **Iterative step:**

  - **Iterative step (only doen when the matching is *not* maximum (complete)):**

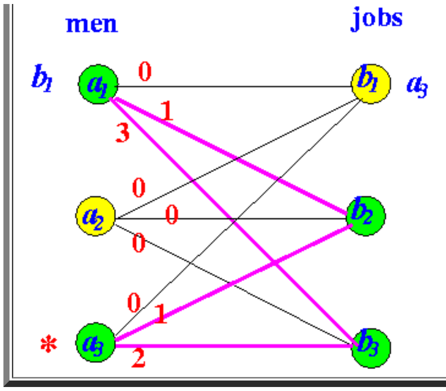
    - ***Add*** **the next least cost edge(s):**

      1. **Look** in the **original bi-partite graph** (with the **label** of the **maximum flow** algorithm added):

      ---

      2. **Find** all **edges** (with **cost > 0**) going from a ***labeled* vertex ∈ *X* (men)** to an ***unlabeled* vertex ∈ *Y* (jobs)**



Edges: + = a labeled node

	b_1 (+)	b_2	b_3
+ a_1	0	1	3
a_2	0	0	0
+ a_3	0	1	2

Find the **minimum cost δ**:

- δ = 1

---

3. For **each edge** with **cost > 0** such that: ***labeled* vertex** ∈ ***X* (men)** → ***unlabeled* vertex** ∈ ***Y* (jobs)**:

   - ***subtract* δ** from the **cost** of the edge

   For **each edge** with **cost > 0** joining an ***unlabeled* vertex** ∈ ***X* (men)** → ***labeled* vertex** ∈ ***Y* (jobs)**:

   - ***add* δ** from the **cost** of the edge

   (This **addition** and **subtraction** is actually a **pivoting operation** in the **Simplex Algorithm** !!)

   Example:

Before any operations

	b_1 (+)	b_2	b_3
+ a_1	0	1	3
a_2	0	0	0
+ a_3	0	1	2

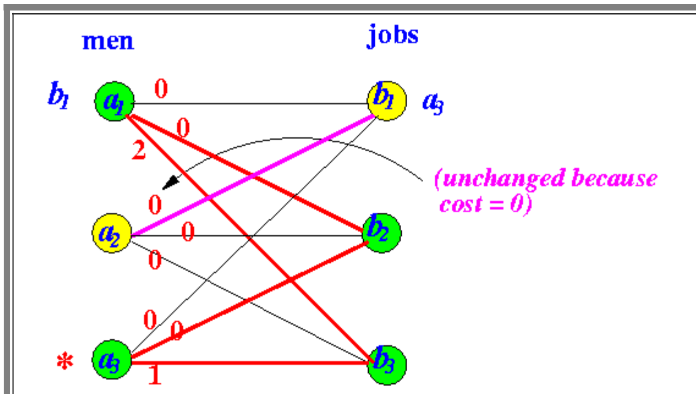
After the *subtract* step

	b_1 (+)	b_2	b_3
+ a_1	0	0	2
a_2	0	0	0
+ a_3	0	0	1

After the *addition* step

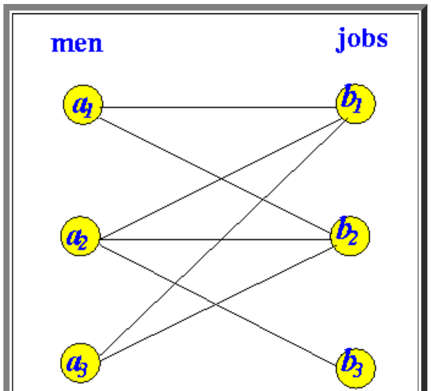
	b_1 (+)	b_2	b_3
+ a_1	0	0	2
a_2	0	0	0
+ a_3	0	0	1

Result:

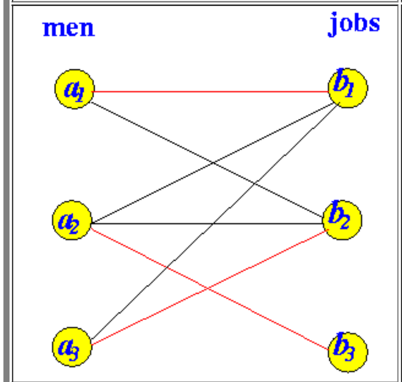
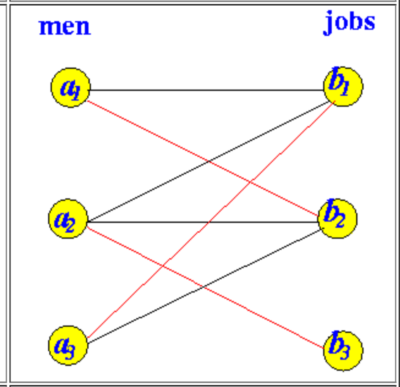
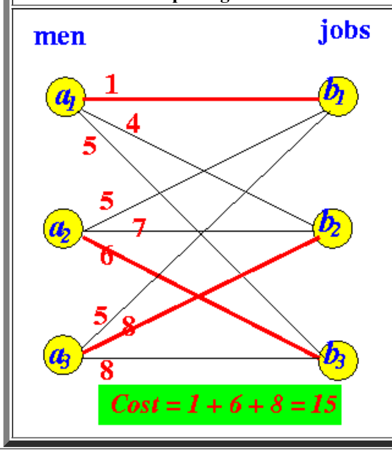
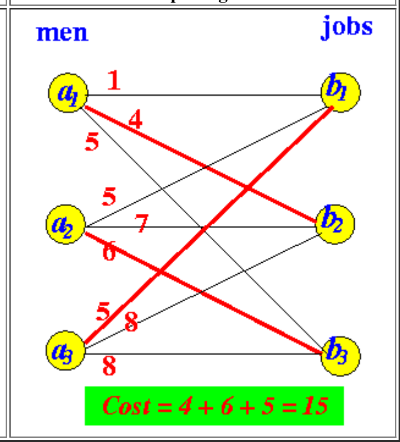


---

4. **Add** the **new 0-cost edges** and **repeat** the **matching step**:



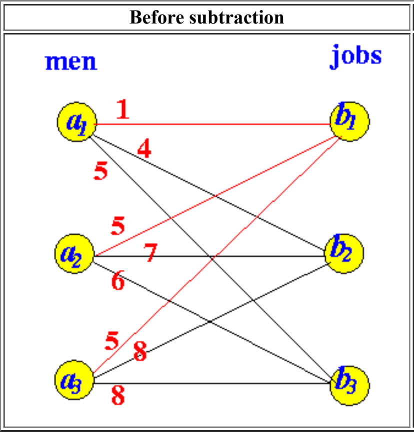
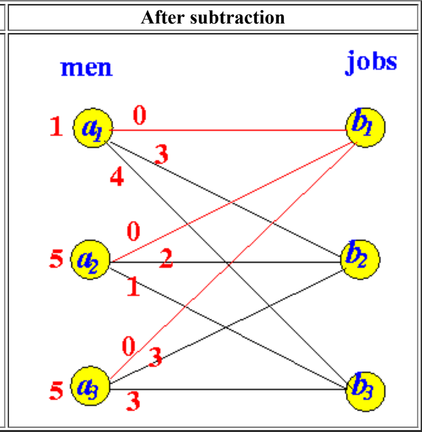
Result:

Max. matching 1	Max. matching 2
	
Corresponing answer	**Corresponing answer**
	

- **Final note:**
  - There were **2 optimum solution** possible because in the **last step** of the algorithm, we have **added** **2 *new* edges** of the **same cost** to the subgraph.
  - **Each edge** gave rise to an **maximum matching**

    But, the **cost** of the optimum solution is **equal**

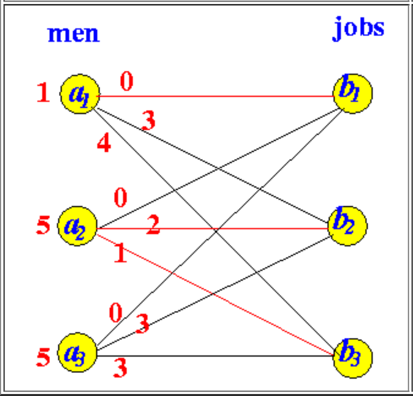
- **Rationale in some of the steps**
  - **Reason to do the *first* subtract step:**
    - The **first subtract step:**

Before subtraction	After subtraction
	

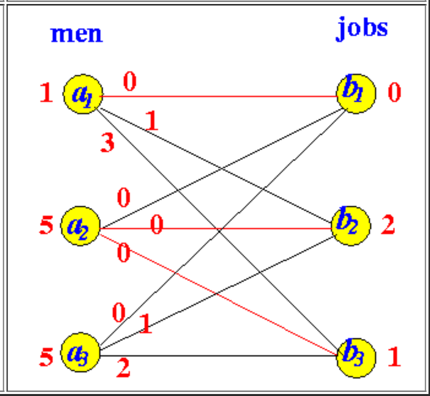
- Observe that:
  - The **0-weight edges** are the **minimum cost edges** to connect a **source node** to a **destination node**

- **Reason to do the *second* subtract step:**
  - The **second subtract step:**

    Before subtraction

    After subtraction

  - Observe that:
    - ***Before* the subtraction** there is **no "minimum" cost path** from ***any* source** to **nodes b_2 and b_3**

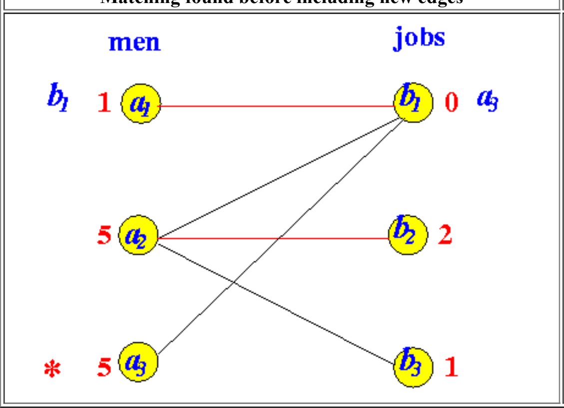
      Therefore:

      - There is **no way** that **nodes b_2 and b_3** will be **assigned** (using only **0-weight edges**) !!!

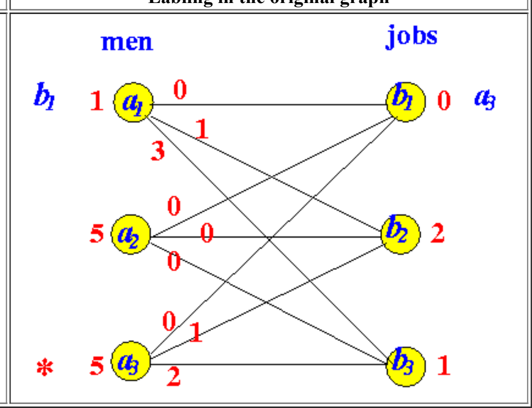
    - ***After* the subtraction** there are ***two* new 0-weight edges** that are the **minimum cost edges** to connect to **nodes b_2 and b_3** !!!!

- Why consider **adding *only* edges** (with **cost > 0**) going from a ***labeled* vertex ∈ *X* (men)** to an ***unlabeled* vertex ∈ *Y* (jobs)**
  - Consider the **labeling** of the **original graph** when we decide to **add new (minimum cost) edges** (to get a **complete matching**):

    Matching found before including new edges

    Labling in the original graph

  - There are **4 types** of **edges**:
    - **unlabeled x → unlabeled y**
    - **unlabeled x → labeled y**
    - **labeled x → unlabeled y**
    - **labeled x → labeled y**

    Remember: we **add** edges because we **cannot find** a **maximum matching** with the **current set** of minimum cost edges

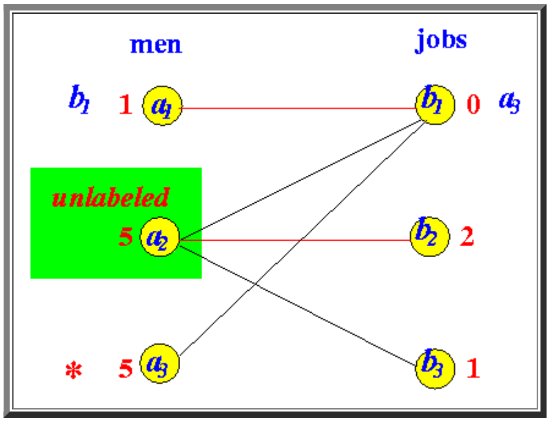
    I.e.:

- **add** edges to allow the **unmatched nodes** ∈ *X* to be **matched** with some node ∈ *Y*

- **Fact:**

  - An **unlabeled node** ***x*** is a node that **has been *matched*** with **some node** ∈ *Y*

    Example:

    (This is how the **labeling algorithm** works.)

  Therefore:

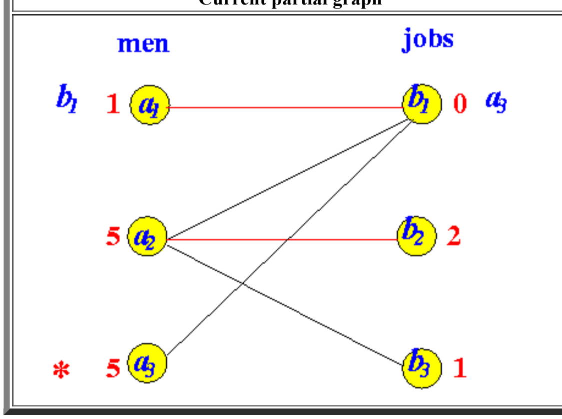
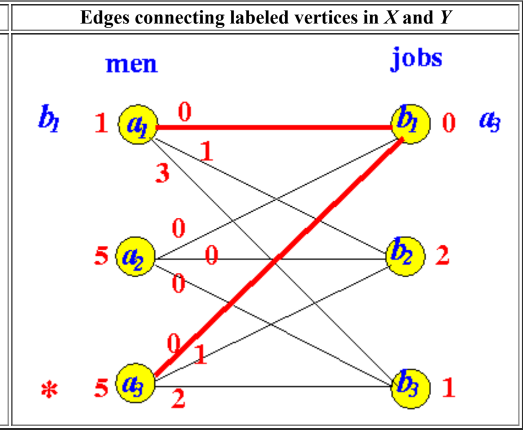
  - There is **no need** to **include** edges of the type:
    - **unlabeled *x* → unlabeled *y***
    - **unlabeled *x* → labeled *y***

    because the **vertex *x*** has **already been matched** !!!

- **Fact:**

  - An ***edge*** connecting a **labeled node *x*** to a **labeled node *y*** is an **edge in the *current* partial graph**

    Example:

Current partial graph	Edges connecting labeled vertices in *X* and *Y*
	

    (**Again**, this is how the **labeling algorithm** works.)

  Therefore:

  - There is **no need** to **include** edges of the type:
    - **labeled *x* → labeled *y***
    - **unlabeled *x* → labeled *y***

    because these ***edges*** are **already included** !!!

- The only remaining type of edges to consider for inclusion is: **labeled *x* → unlabeled *y***

- Why do we **subtract** and **add** in a particular way ?

  - That is the **elementary row operation** performed in the **Simplex method**
  - Sometimes you have to **add** the **pivot row** to another **row**

    Sometimes you have to **subtract** the **pivot row** from another **row**

  - Without knowing **details** of the **Dual Simplex Algorithm**, it is not possible to explain the reason completely

---

- **Another worked out example**

  - **Problem description:**

    - 4 applicants $a_1, a_2, a_3,$ *and* a_4 apply for 4 jobs $b_1, b_2, b_3,$ *and* b_4
    - The **cost matrix** is:

```
        b1  b2  b3  b4
    ----+-----------------
     a1 |   6  12  15  15
     a2 |   4   8   9  11
     a3 |  10   5   7   8
     a4    12  10   6   9
```

    - Find the **optimum** (**least cost**) assignment of applicants to jobs.

  - **Solution:**

    - **Initialization:**

      1. **Subtract** the **least cost** of **each vertex** ∈ ***X*** from all weights connected to that vertex

         This is the same as:

         - Subtract the **smallest value** in **each row** from **all other values in that row**

         Result:

Before subtraction:	After subtraction:

<pre>   b1 b2 b3 b4
    ----+-----------------
     a1
	6 12 15 15
	     a2
		4 8 9 11
		     a3
			10 5 7 8
			     a4
				12 10 6 9</pre>	<pre> b1 b2 b3 b4
					    ----+-----------------
					     a1
						0 6 9 9
						     a2
							0 4 5 7
							     a3
								5 0 2 3
								     a4
									6 4 0 3</pre>

         ---

      2. **Subtract** the **least cost** of **each vertex** ∈ ***Y*** from all weights connected to that vertex

         This is the same as:

         - Subtract the **smallest value** in **each** ***column*** from **all other values in that row**

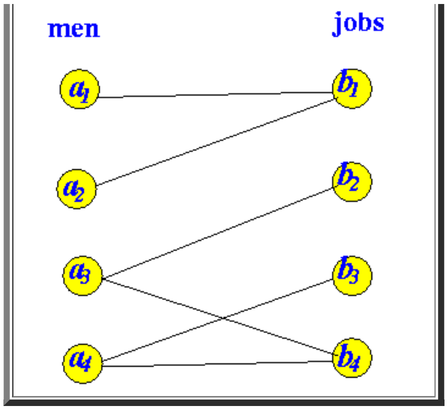
         Result:

Before subtraction:	After subtraction:

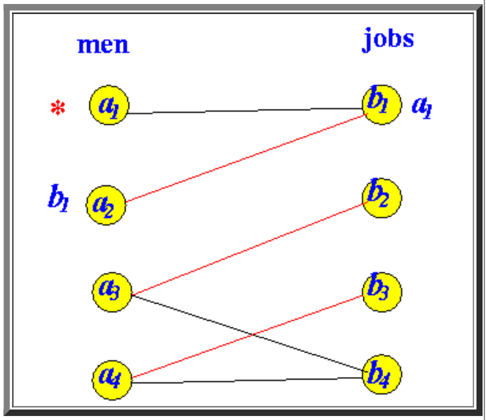
<pre>   b1 b2 b3 b4
    ----+-----------------
     a1
	0 6 9 9
	     a2
		0 4 5 7
		     a3
			5 0 2 3
			     a4
				6 4 0 3</pre>	<pre> b1 b2 b3 b4
					    ----+-----------------
					     a1
						0 6 9 6
						     a2
							0 4 5 4
							     a3
								5 0 2 0
								     a4
									6 4 0 0</pre>

    - **Initial matching:**

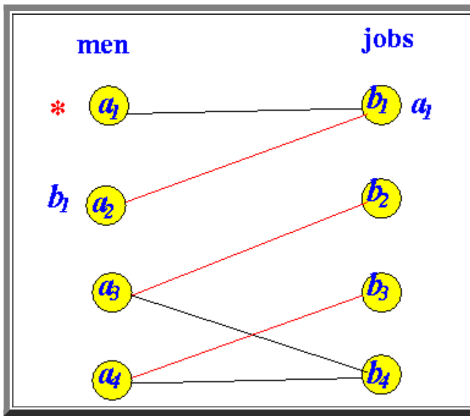
      - **Subgraph** with **0-weight edges**:



- **Maximal matching:**



- **Iteration 1:**
  - **The incomplete maximal matching:**



Edges labeled x_i → unlabeled y_j:
(marked red)

	+ b1	b2	b3	b4
+ a1	0	6	9	6
+ a2	0	4	5	4
a3	5	0	2	0
a4	6	4	0	0

δ = 4

  - **Recalculate edge costs to find new 0-weight edges:**

Subtract δ from the cost of edges

labeled x_i → unlabeled y_j:

Result:

	b1	- b2	- b3	- b4
+ a1	0	2	5	2
+ a2	0	0	1	0
a3	5	0	2	0
a4	6	4	0	0

Add δ to the cost of edges

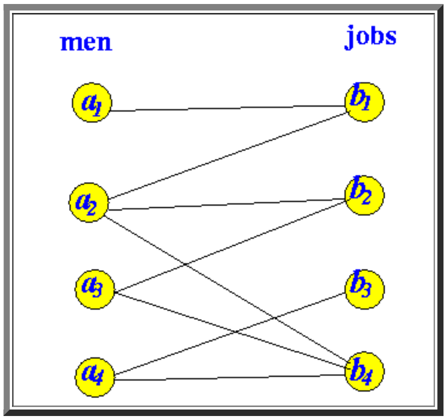
unlabeled x_i → labeled vertex y_j

(only when cost of edge > 0)

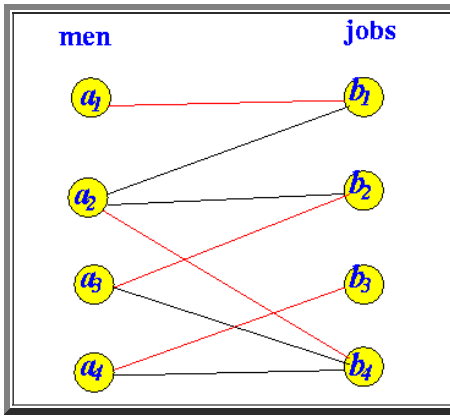
Result:

	b1	- b2	- b3	- b4
+ a1	0	2	5	2
+ a2	0	0	1	0
a3	9	0	2	0
a4	10	4	0	0

- **New subgraph** with **additional 0-weight edges**:



- **Maximal matching:**




```
Minimum cost assignment:

        b1  b2  b3  b4
    ----+-----------------
    a1 |   6  12  15  15
    a2 |   4   8   9  11
    a3 |  10   5   7   8
    a4 |  12  10   6   9

Cost = 6 + 5 + 6 + 11 = 28
```