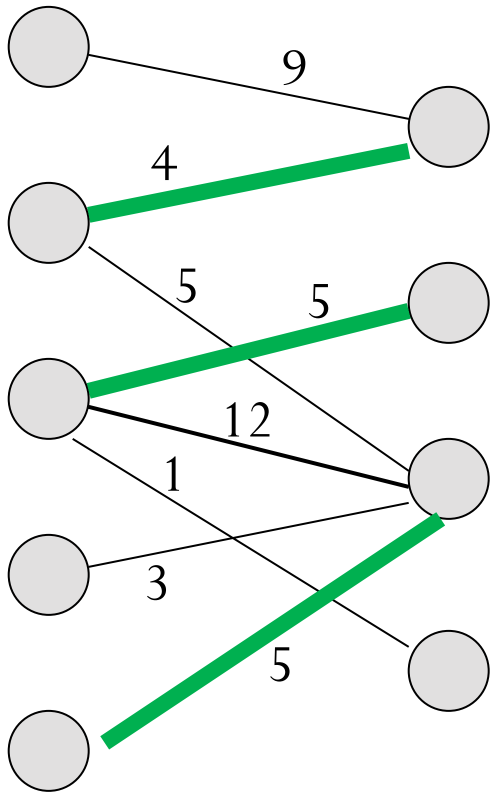
Weighted Bipartite Matching

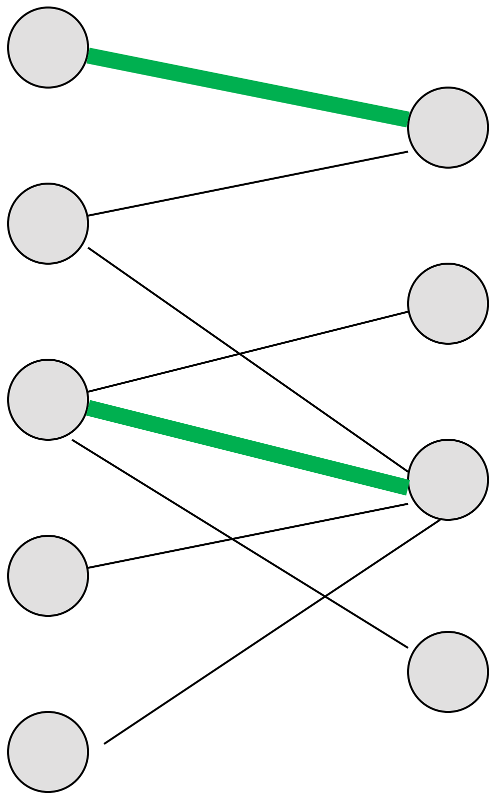
CS31005: Algorithms-II
Autumn 2020
IIT Kharagpur

Matching

- A matching in an undirected graph $G = (V, E)$ is a subset of edges $M \subseteq E$, such that for all vertices $v \in V$, at most one edge of M is incident on v
- Size of the matching $M = |M|$
- Weight of the matching M (for weighted graphs) = sum of the weights of the edges in M
- A maximum cardinality matching is a matching with maximum number of edges among all possible matchings

- A maximum weighted matching is a matching with highest weight among all other matchings in the graph
- Our problem: Given a weighted bipartite graph $G = (V, E)$ with partitions X and Y, and positive weights on each edge, find a maximum weighted matching in G
- Models assignment problems with cost in practice
- Simple flow based techniques that we used for unweighted bipartite graphs no longer work for weighted graphs…





A matching with weight 14 (maximum cardinality matching but not maximum weighted)

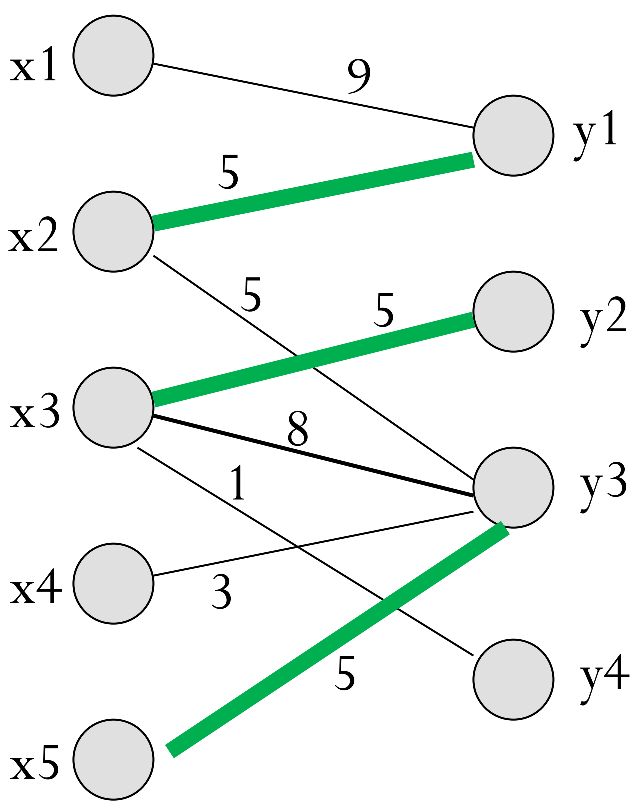
A maximum weighted matching with weight 21 (maximum weighted matching but not maximum cardinality)

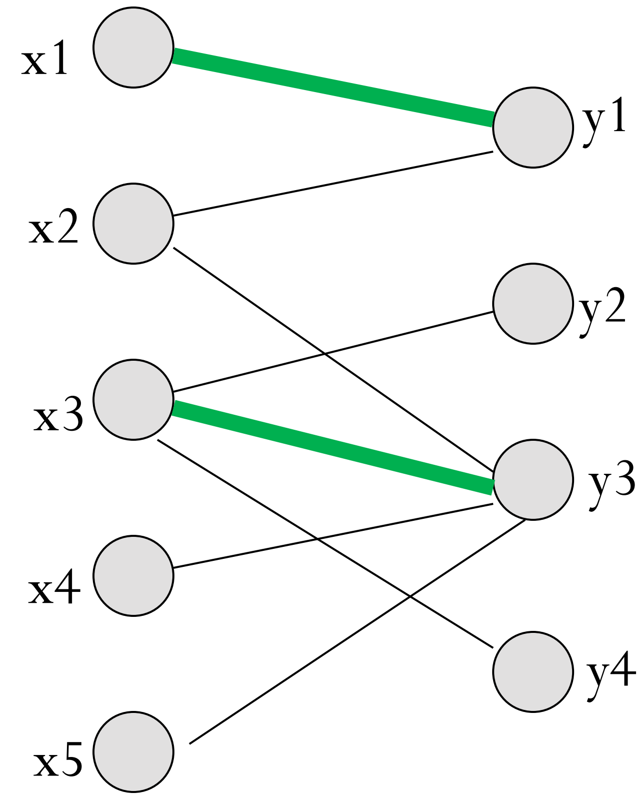
Perfect Matching

- Given a matching M
  - The vertices belonging to the edges of a matching are saturated by the matching; the others are unsaturated (also called free vertices)
- If a matching saturates every vertex of G, then it is a perfect matching
- For a perfect matching to exist, number of vertices must be even
  - For bipartite graphs, the number of vertices in each partition must be the same
  - For any graph with n vertices, size of a perfect matching is n/2

Augmenting Paths

- Given a matching M, a path between two distinct vertices u and v is called an alternating path if the edges in the path alternate between in M and not in M
- An alternating path P that begins and ends at unsaturated vertices is an augmenting path
  - Replacing $M \cap E(P)$ by $(E(P) - M)$ produces a new matching M' with one more edge than M (i.e., augments M)





{x1, x4, y4} are unsaturated
(x2, x3, x5, y1, y2, y3} are saturated
P = <x1,y1,x2,y3,x5> is an alternating path but not augmenting

P = <y2, x3, y3, x4> is an augmenting path
M ∩ E(P) = {(x3,y3)}
E(P) – M = {(x3, y2), (x4, y3)}
M’ = {(x1,y1), (x3,y2), (x4, y3)} is a higher cardinality matching

Key Result

[Berge's Theorem] A matching M in a graph G is a maximum matching in G if and only if G has no augmenting path

- This gives another way of finding maximum cardinality matchings in bipartite graphs without depending on flows
- But does not help directly in finding a maximum weighted matching (can you show a counterexample?)
- Instead, the algorithm we learn will use it in a related graph

Hungarian Algorithm

- Also called Kuhn-Munkres algorithm
  - Finds a maximum weighted perfect matching in a complete bipartite graph
    - $|X| = |Y|$
    - An edge (x, y) exists between each pair $x \in X$ and $y \in Y$
- So what if your input graph is not complete?
  - Just add dummy vertices (if needed) to make the no. of vertices on both sides equal, and add edges that do not exist with weight 0
  - Find the maximum weighted matching in this new graph, then throw away any dummy edge included in the matching
  - Remaining edges will be the maximum weighted matching in your original input graph

Equality Subgraph

- Assign a label $\ell(u)$ to every vertex u
- Feasible labelling

  $$\ell(x) + \ell(y) \geq w(x,y) \text{ for any edge } (x,y)$$

- Given a feasible labelling ℓ, Equality Subgraph $G_\ell = (V, E_\ell)$ where
  - $E_\ell = \{(x,y) \mid x \in X, Y \in Y, \ell(x) + \ell(y) = w(x,y)\}$
- Why is it important?

[Kuhn-Munkres Theorem]: Let ℓ be a feasible labeling of G. If M is a perfect matching in G_ℓ, then M is a maximum weighted matching in G.

Hungarian Algorithm: Basic Idea

- Start with any feasible labeling ℓ and $M = \emptyset$
- While M is not a perfect matching repeat

1. Find an augmenting path for M in E_ℓ and augment M
2. If no augmenting path exists,

Improve ℓ to ℓ' such that at least one new edge is added to the equality subgraph

Go to Step 1

Initial Feasible Labeling

- Start with this feasible labelling
  - $\ell(x) = \max\{w(x,y) \mid y \in Y\}$ for all $x \in X$
  - $\ell(y) = 0$
- Guarantees that in the equality subgraph G_ℓ
  - E_ℓ has at least one edge from every vertex $x \in X$

Some Definitions

- Let ℓ be a feasible labeling
- Neighbor of $u \in V$
  - $N_\ell(u) = \{v : (u, v) \in E_\ell\}$
- For any set $S \subseteq V$, neighborhood of S
  - $N_\ell(S) = \bigcup_{u \in S} N_\ell(u)$

- We will maintain two sets, S and T
- At any time, S and T will keep information about the alternating/augmenting paths
  - S will have a subset of vertices in X
  - T will have a subset of vertices in $N_{\ell}(S)$
  - S and T together will keep track of a tree of alternating paths rooted at some free vertex in X (which will be in S)

How to find the matching

- Find a free vertex $x \in X$
  - Must exist unless you have reached the perfect matching
- Create a tree rooted at X such that all paths in the tree from x are alternating
  - Vertices at even levels (0, 2, …) = vertices in S
    - These will be in X
  - Vertices at odd levels (1, 3, …)= vertices in T
    - These will be in Y
  - If we encounter a free vertex at odd level, we have found an augmenting path
    - Augment and continue

How to improve the labeling

- Let $S \subseteq X$ and $T = N_{\ell}(S) \neq Y$
- Let

$$\alpha_{\ell} = \min\{\ell(x) + \ell(y) - w(x, y) \mid x \in S, y \text{ not in } T\}$$

  - Note that $\alpha_{\ell} > 0$
- Then set

$$\begin{aligned} \ell'(v) &= \ell(v) - \alpha_{\ell} && \text{if } v \in S \\ &= \ell(v) + \alpha_{\ell} && \text{if } v \in T \\ &= \ell(v) && \text{otherwise} \end{aligned}$$

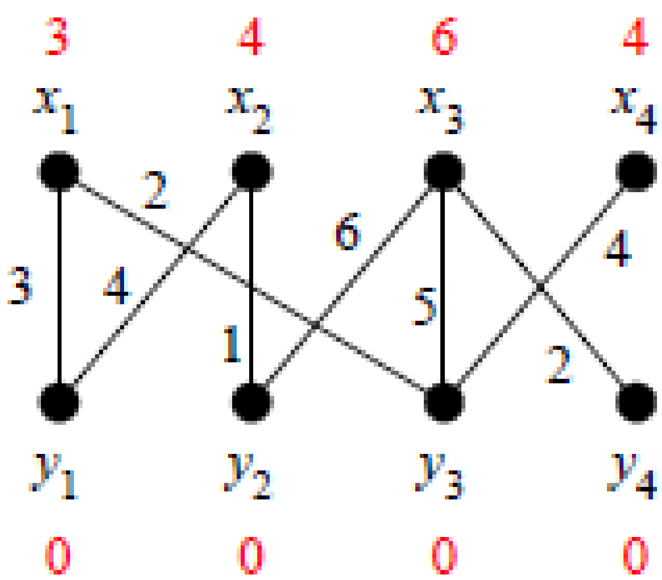
- The updated labeling ℓ' is a feasible labeling with the following properties:
  - If $(x, y) \in E_\ell$ for $x \in S$, $y \in T$ then $(x, y) \in E_{\ell'}$
    - Decrease in $\ell(x)$ is same as increase in $\ell(y)$
  - If $(x, y) \in E_\ell$ for x not in S, y not in T then $(x, y) \in E_{\ell'}$
    - Labels remain the same for them
  - There is some edge $(x, y) \in E_{\ell'}$ for $x \in S$, y not in T
    - At least for one edge ((the one with the minimum in α_ℓ computation), $\ell(x)$ is decreased by the excess, $\ell(y)$ is unchanged, so brings in the edge into the new equality graph
- This shows that the new labelling increases the neighborhood of S

The Algorithm

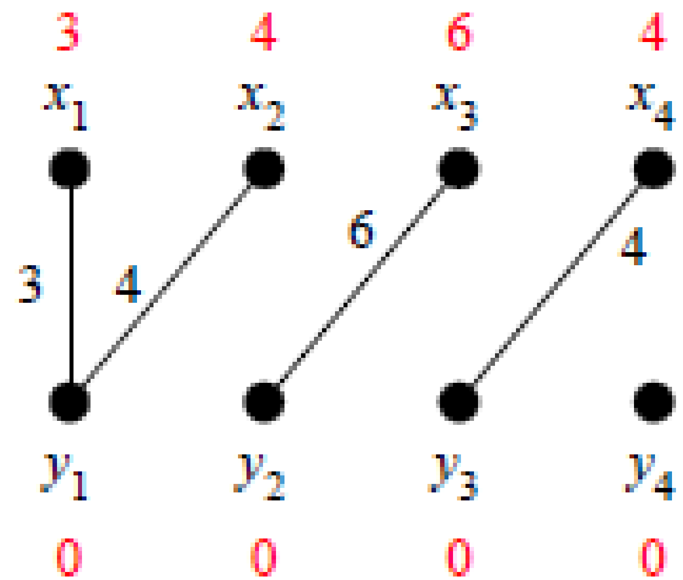
1. Start with the initial feasible mapping l, and the corresponding equality graph. Set $M = \emptyset$.
2. While M is not a perfect matching in the equality graph, repeat:
   (a) Pick a free vertex $x \in X$.
   (b) Set $S = \{x\}$ and $T = \emptyset$.
   (c) While $N_l(S) \neq T$, repeat:
      (i) Pick $y \in N_l(S) \setminus T$.
      (ii) If y is free, augment M, and go to Step 2.
      (iii) Otherwise, y is matched, say, to $z \in X$. Set $T = T \cup \{y\}$ and $S = S \cup \{z\}$.
   (d) Here we have $N_l(S) = T$. Do the following two steps.
      (iv) Compute α.
      (v) Decrement $l(x)$ by α for all $x \in S$, and increment $l(y)$ by α for all $y \in T$.
   (e) The condition $N_l(S) \neq T$ is again restored, so go to the top of the loop (c).

Example

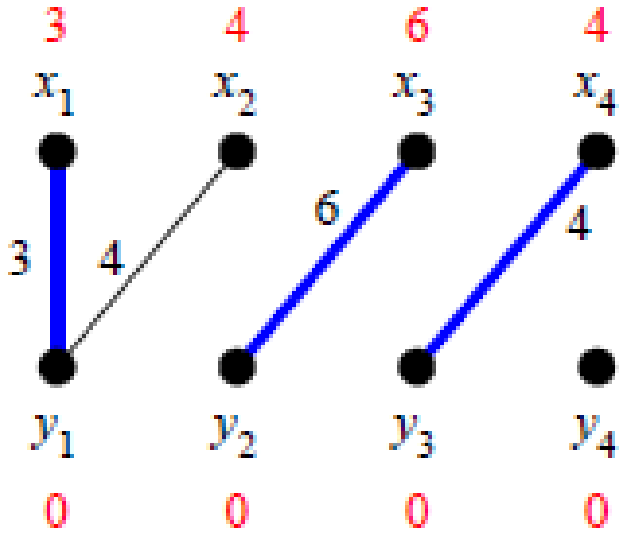
- We do not show the dummy 0-weight edges added, though they are there, and you include them in all calculations of the steps of the algorithm



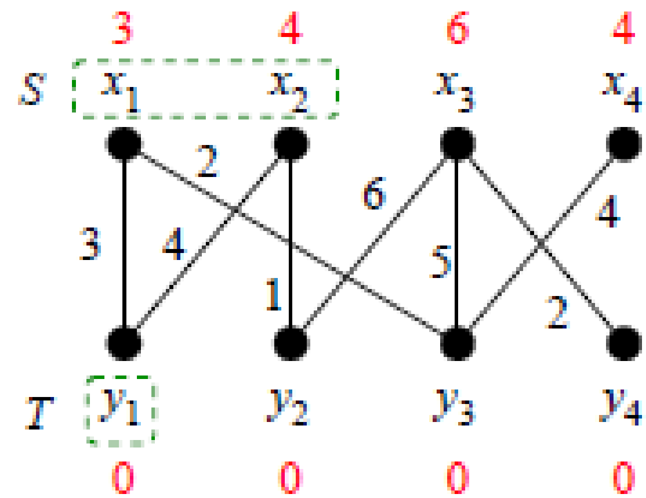
Initial labeling



Initial equality graph

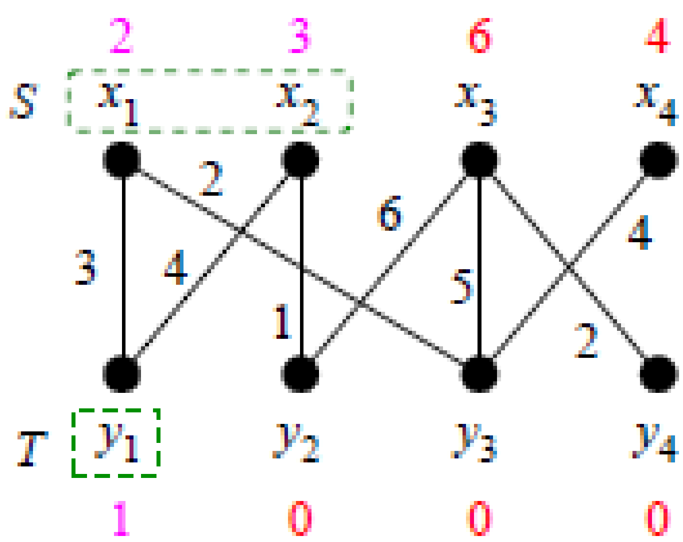


Intermediate matching

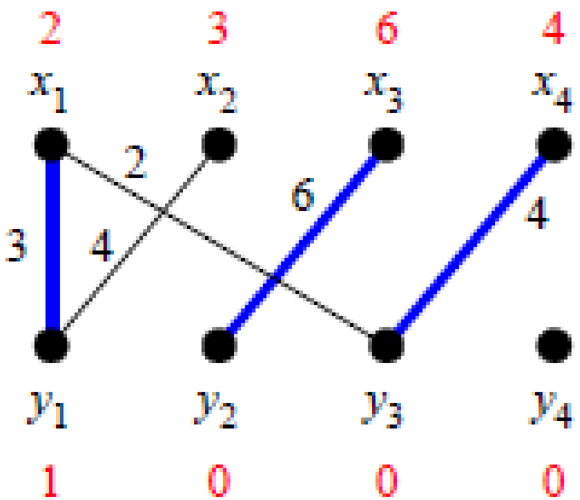


$$\alpha_l = \min\begin{pmatrix} 3+0-0,\ 3+0-2,\ 3+0-0, \\ 4+0-1,\ 4+0-0,\ 4+0-0 \end{pmatrix}$$

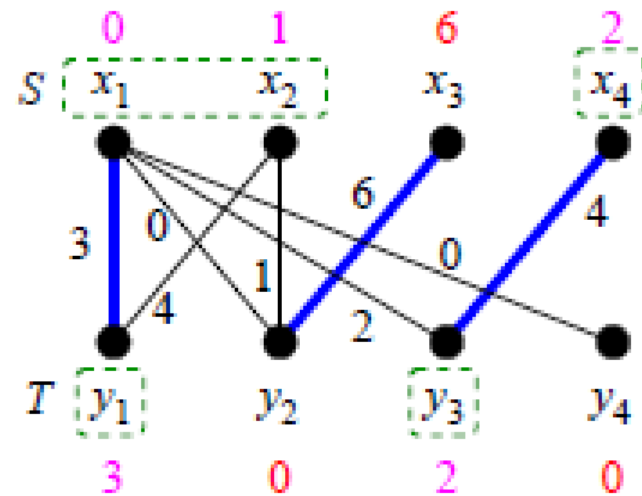
$$= 1$$



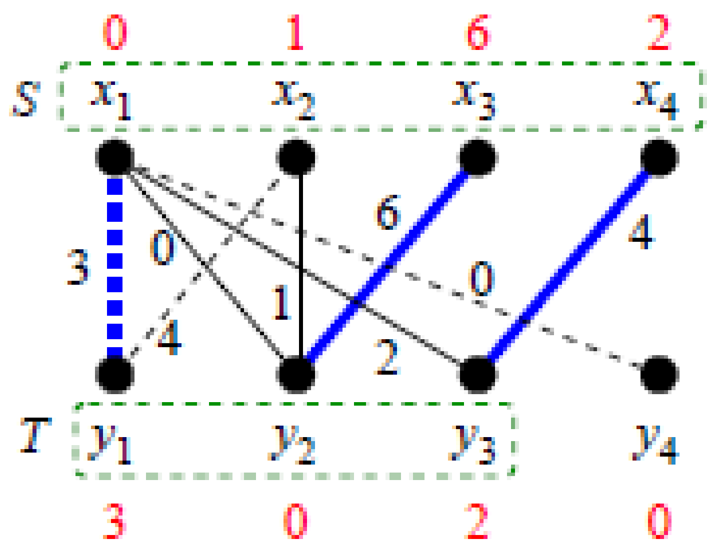
Update labels



New equality graph



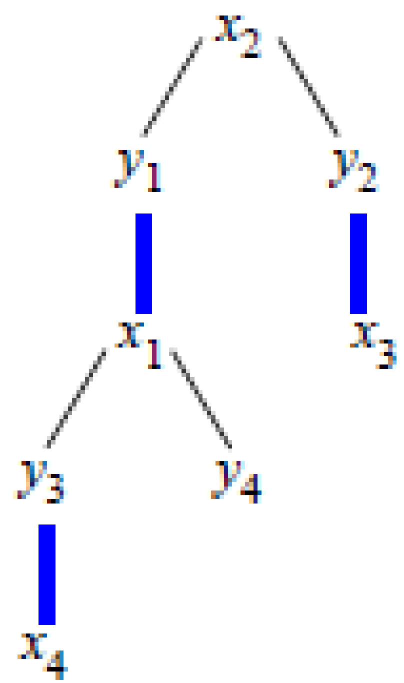
Another label change



Discovery of an augmenting path

Alternating Tree Generated

- Tree generated while discovering the augmenting path in the last equality graph
  - From free vertex x_2, first go to y_2, then to x_3, cannot grow this anymore
  - Come back to x_2, explore y_1, then x_1, then y_3, then x_4, cannot grow this anymore
  - Come back to x_1, go to y_4, found an augmenting path, so stop
- Note that the tree depends on order of visit
  - May have gone to y_1 first from x_2, then the augmenting path would have been found before y_2 is explored (so y_2 and x_3 would not have been in the tree)
  - Similar possibility at x_1 if we had visited y_4 first



The alternating tree

- The final maximum weighted matching found by the Hungarian algorithm for the complete bipartite graph is $\{(x_1, y_4), (x_2, y_1), (x_3, y_2), (x_4, y_3)\}$ with weight 14 (= 0 + 4 + 6 + 4)
- But (x_1, y_4) is a dummy edge (not in the original graph)
- So drop it
- Final maximum weighted matching M for the input graph is $\{(x_2, y_1), (x_3, y_2), (x_4, y_3)\}$ with weight 14
  - x_1 remains a free vertex as it cannot be matched
  - Dropping dummy edge does not affect weight as its weight is 0

Time Complexity

- Let $|X| = |Y| = n$
- The outer while loop in Step 2 is executed once when the size of the matching increases by 1
  - So max. no of iterations = size of perfect matching = n
- What is the time for one iteration of the outer while loop?
  - Step 2(a) and 2(b) take O(1) time
  - The while loop in step 2(c) can run O(n) times
    - It can run when $N_\ell(S) \neq T$ until $N_\ell(S) = T$
    - After coming out of the loop when $N_\ell(S) = T$, it can then run again from step 2(e) after the relabeling is done which makes $N_\ell(S) \neq T$ again
    - Irrespective of where it runs from, every time the loop runs, it will either augment M and break to go to while loop in Step 2, or add one new vertex to S and T
    - Since only O(n) vertices can be added before an augmenting path is found, max. no. of iterations is O(n)
  - Time per iteration of the while loop in 2(c) = O(n)
    - If augmenting M, any path has maximum length O(n)
    - If not, picking y and finding x takes O(n) time
  - Total time for the while loop in Step 2(c) = $O(n^2)$

- At Step 2(d)
  - Computing α_ℓ takes $O(n^2)$ time (in naïve approach)
  - Updating the labels take $O(n)$ time
  - In the worst case, relabeling can be done $O(n)$ times
    - Each time adding exactly one new node to $N_\ell(S)$
  - Total $O(n^3)$ time
- So total time for one iteration of the outer while loop = $O(1) + O(n^2) + O(n^3) = O(n^3)$
- So total time for the algorithm = no. of iterations of Step 2 × time for one iteration = $O(n) \times O(n^3) = O(n^4) = O(|V|^4)$
- However, this uses a naïve approach that computes α_ℓ from scratch every time, not efficient

- Time for step 2(d) can be reduced to $O(n^2)$ instead of $O(n^3)$ per iteration of the outer while loop
  - At any relabeling step, note that you have to consider (x,y) pairs such that $x \in S$, y not in T
  - $\forall$y not in T keep track of

    $$\text{slack}(y) = \min_{x \in S}\{\ell(x)+\ell(y) - w(x, y)\}$$
  - Initialize slack at beginning of outer while loop (Step 2) iteration in O(n) time as only one node in S
  - When a node goes from X-S to S (inside inner while loop in step 2(c)), update slacks
    - O(n) time as only one vertex moved in S each time, so does not change the time for one iteration of the inner while loop we computed
    - So total $O(n^2)$ time over all iterations of the while loop in step 2(c), same as before
  - During relabeling, compute α_ℓ as $\min_{y \in T}$ slack(y) in O(n) time
    - So total $O(n^2)$ time as relabeling can be done at most O(n) times as we have seen
  - After computing α_ℓ update slacks: $\forall$y not in T, slack(y) = slack(y) $-$ α_ℓ
    - O(n) time for each update, total $O(n^2)$ time over all relabeling

- Final Time complexity of Hungarian algorithm

  $$O(n) \times O(n^2) = O(n^3) = O(|V|^3)$$

- Note:
  - There is an equivalent matrix based description of Hungarian algorithm that manipulates matrices instead of bipartite graphs
  - The algorithm is the same, just the representation is different
  - We will not do it in this class, but useful to know from a practical implementation point of view