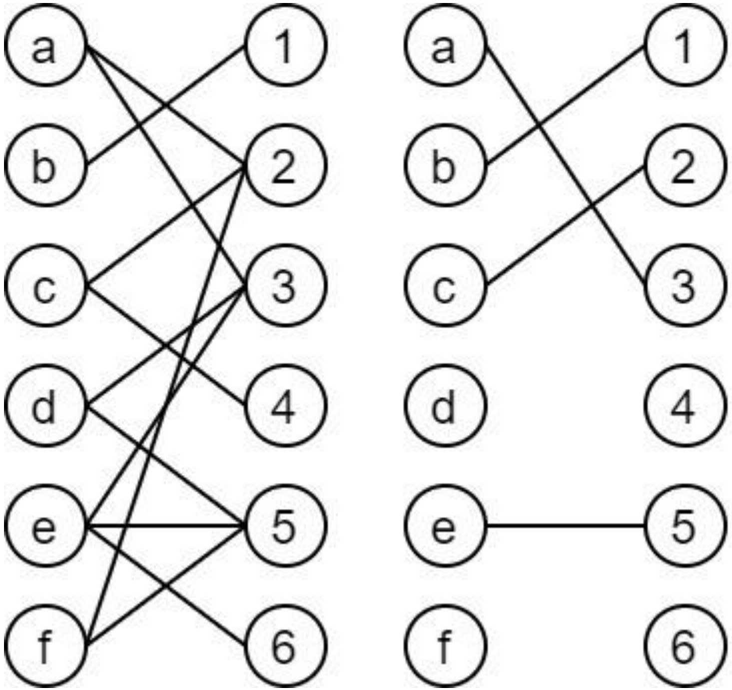
Maximum Matchings

Bhargey Mehta
201701074

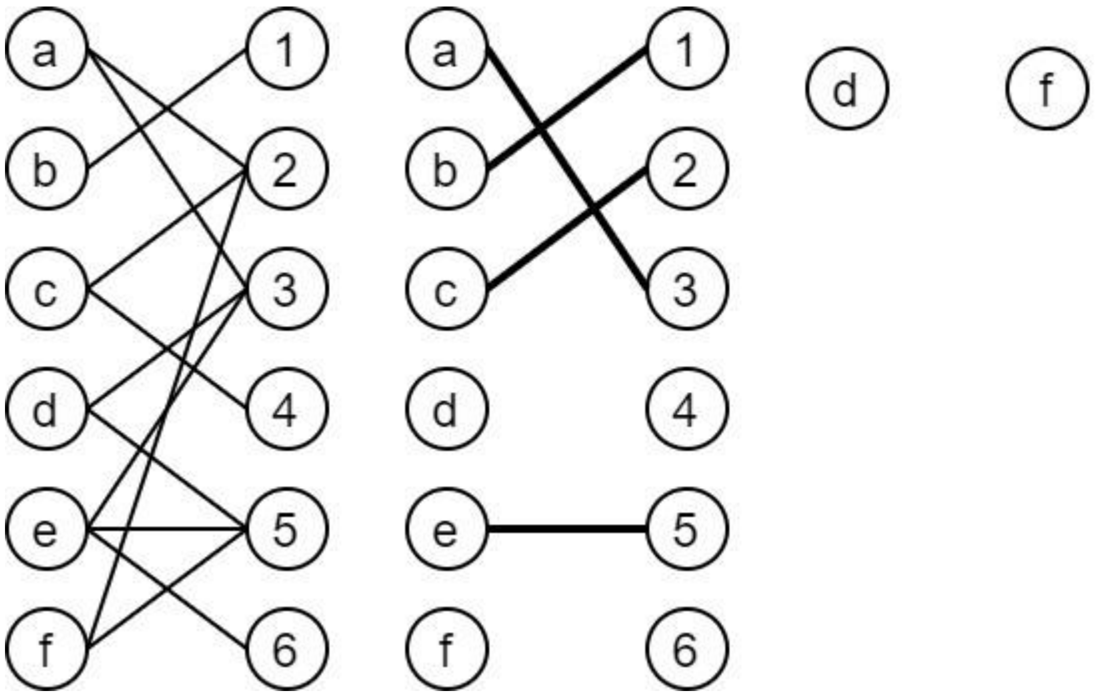
Hopcroft-Karp Algorithm

John Hopcroft, Richard Karp

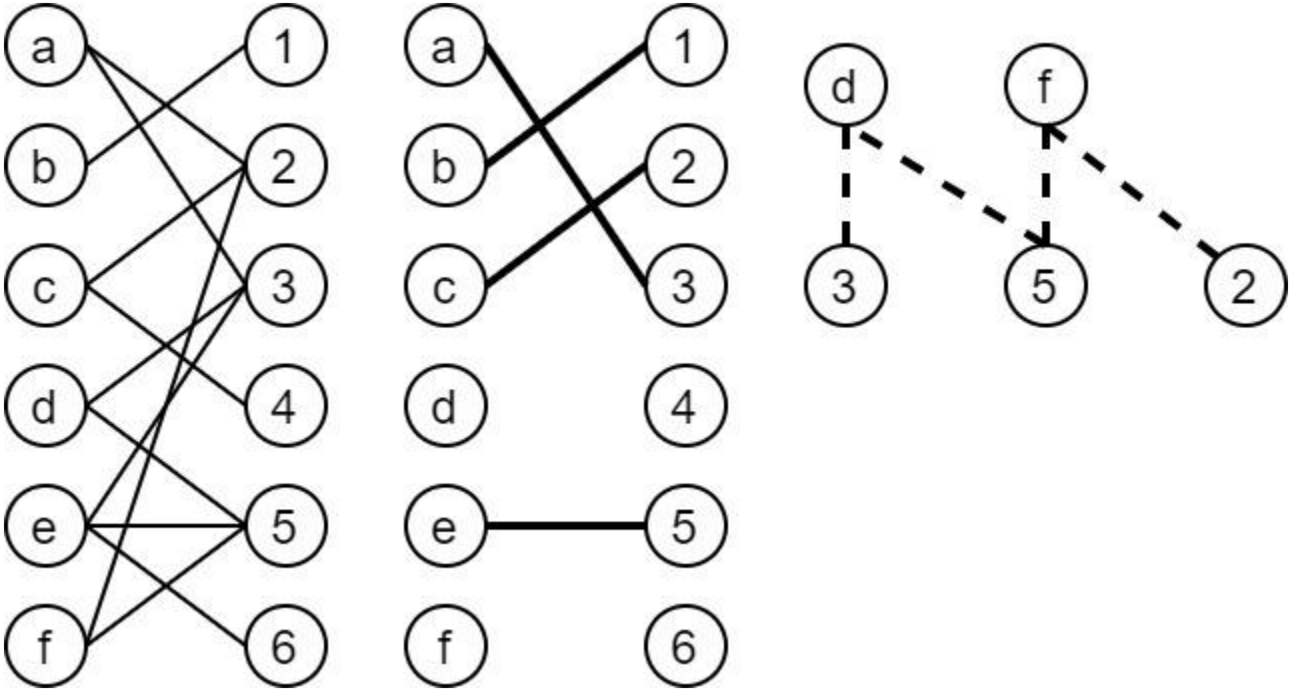
Hopcroft Karp



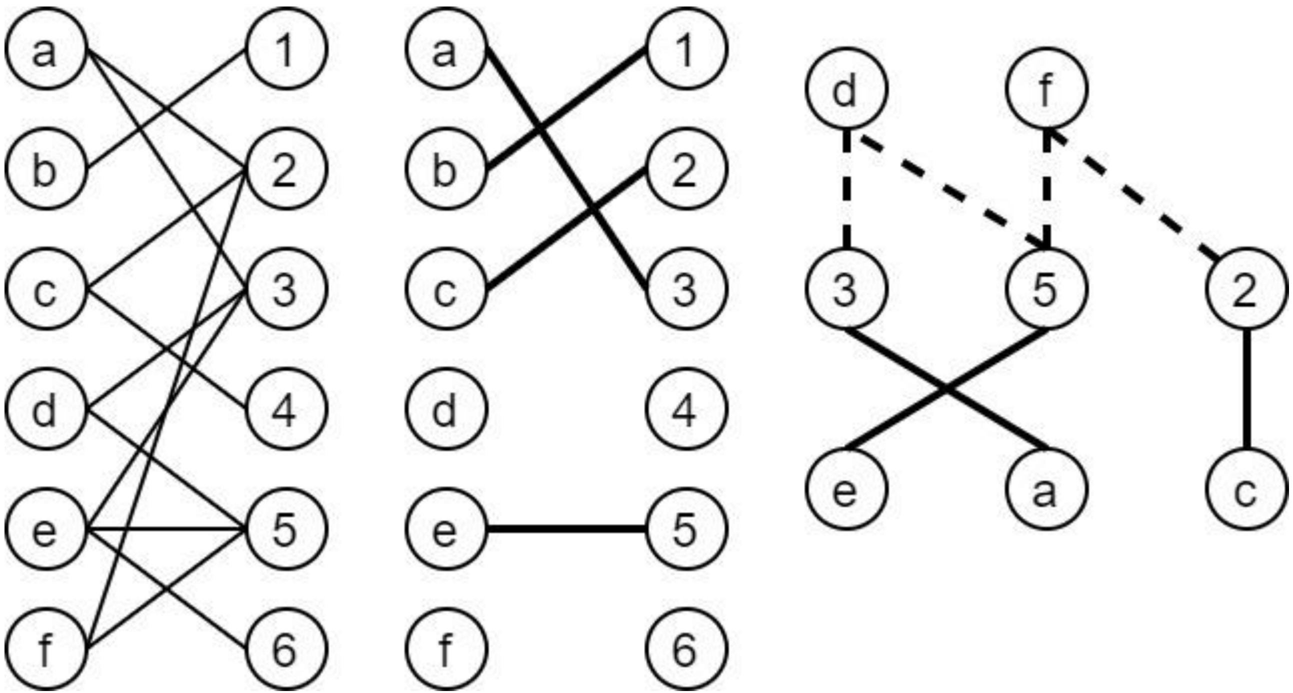
Hopcroft Karp



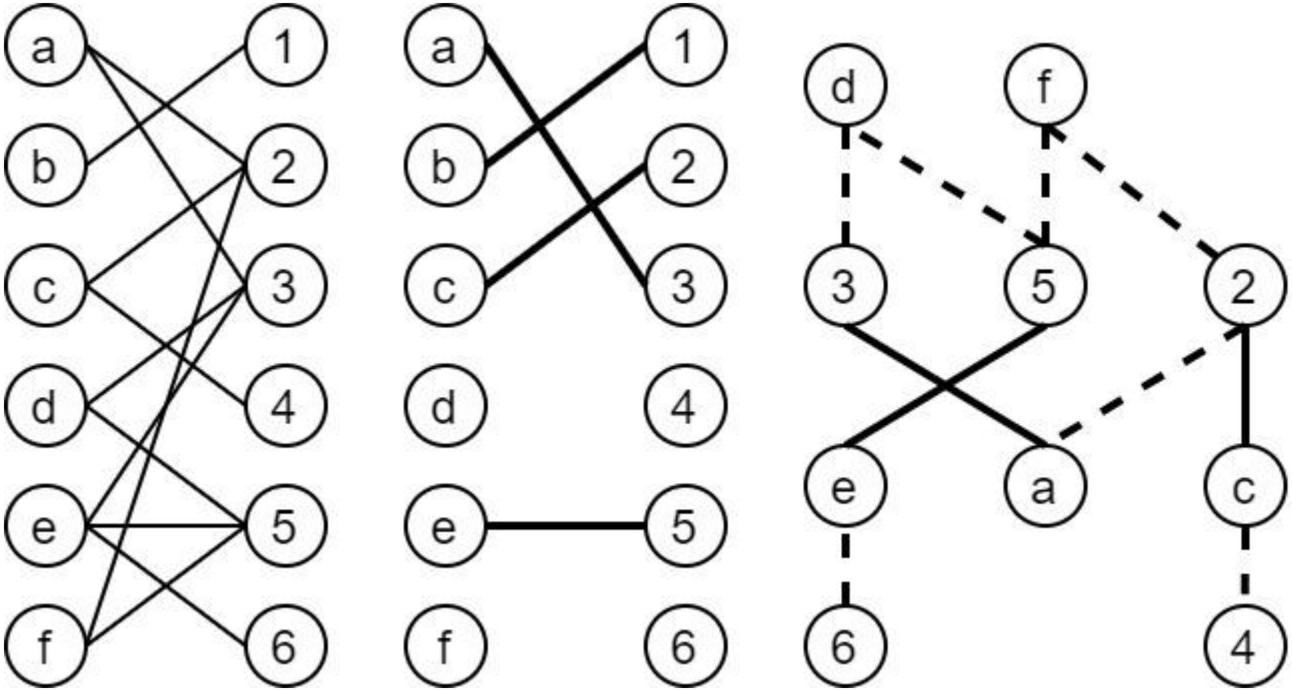
Hopcroft Karp



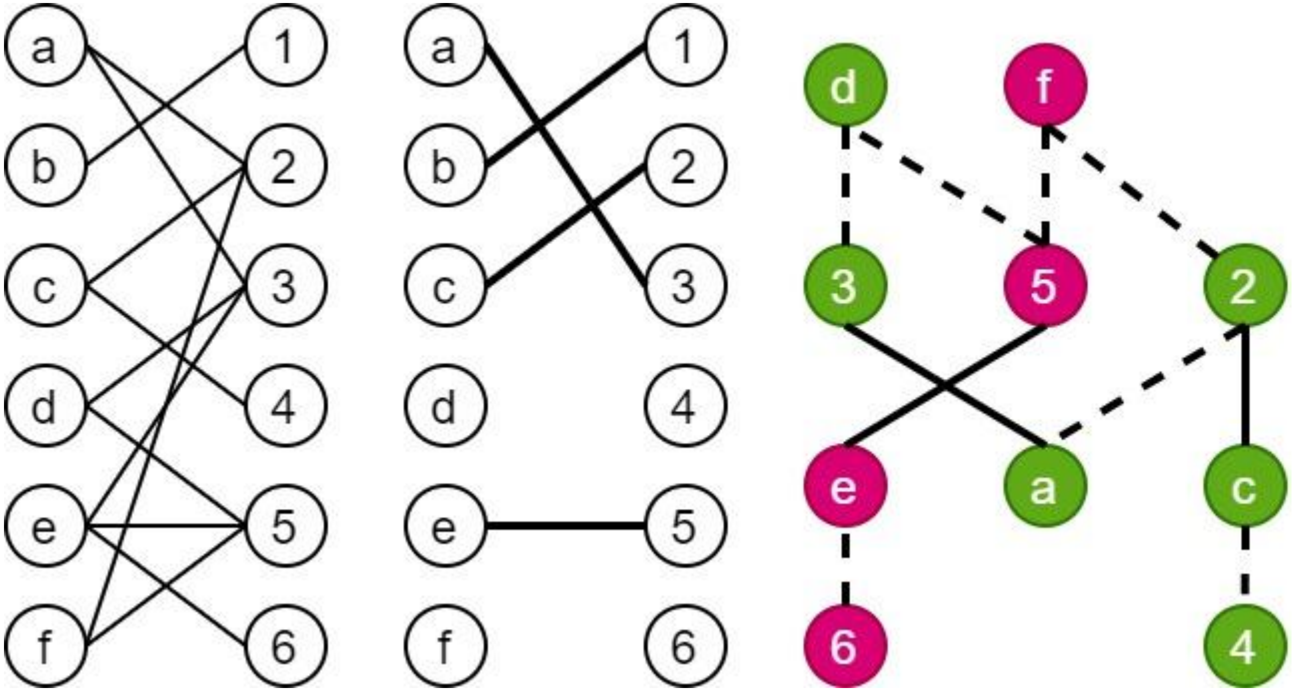
Hopcroft Karp



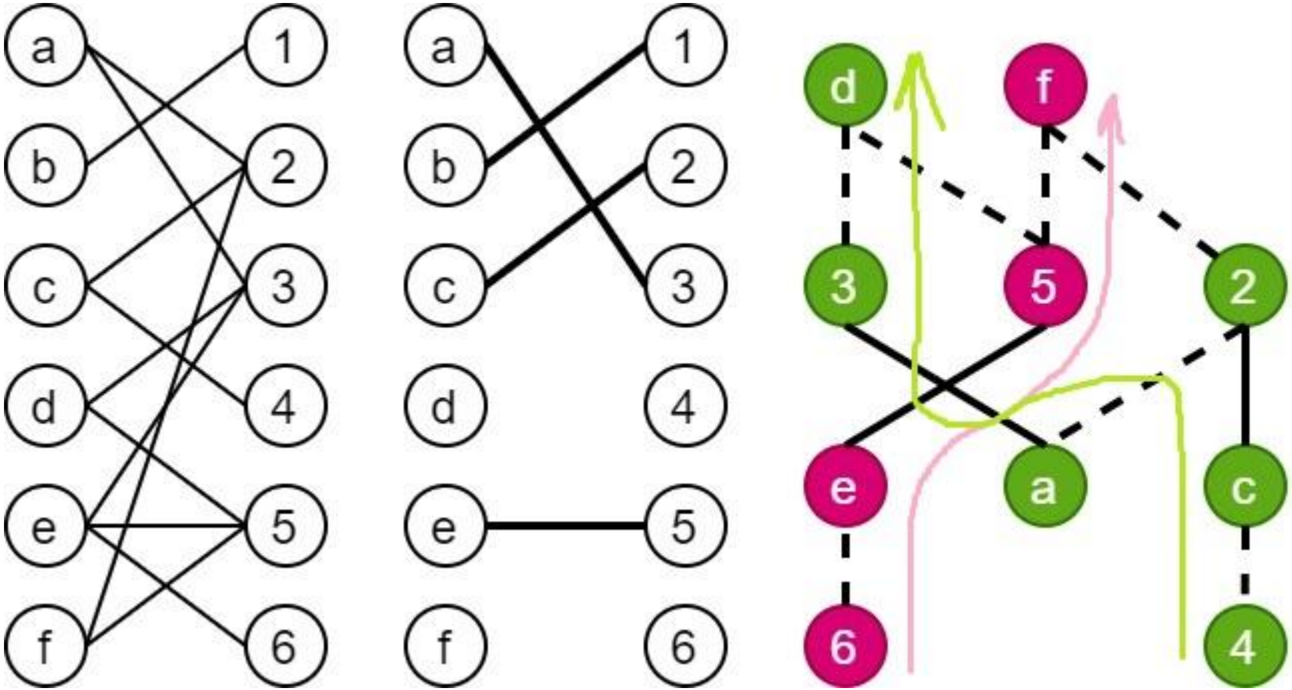
Hopcroft Karp



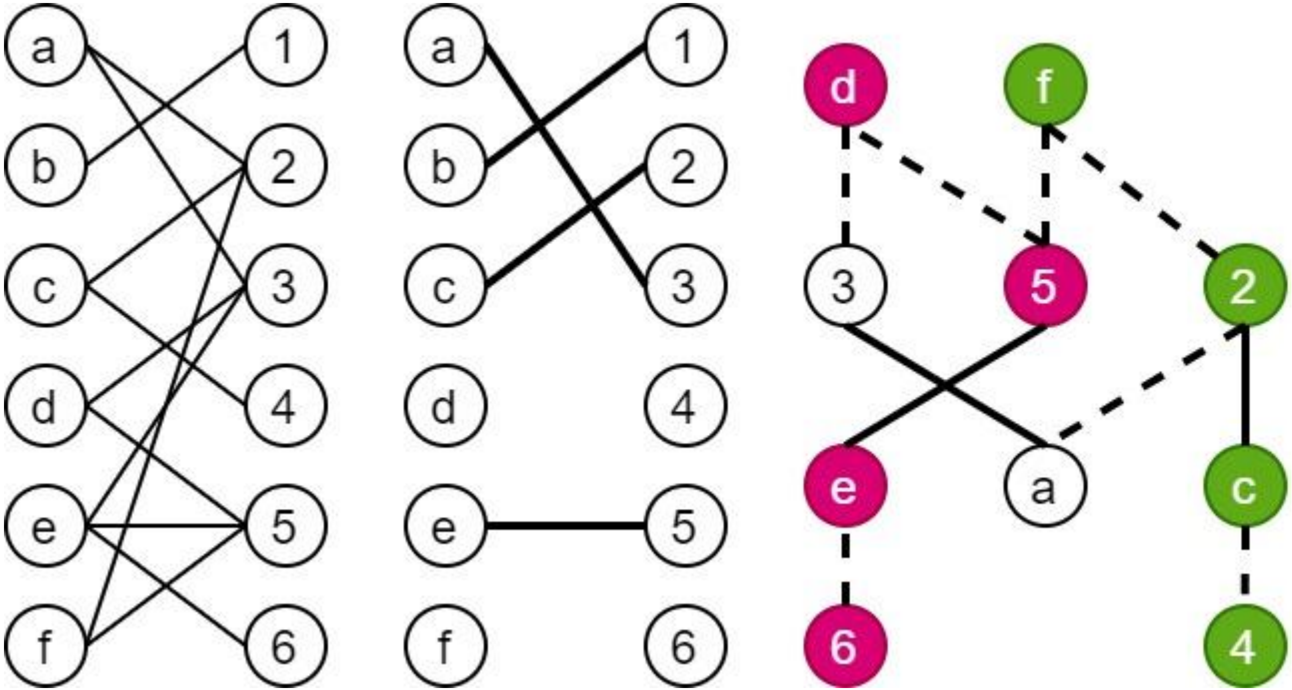
Hopcroft Karp



Hopcroft Karp



Hopcroft Karp



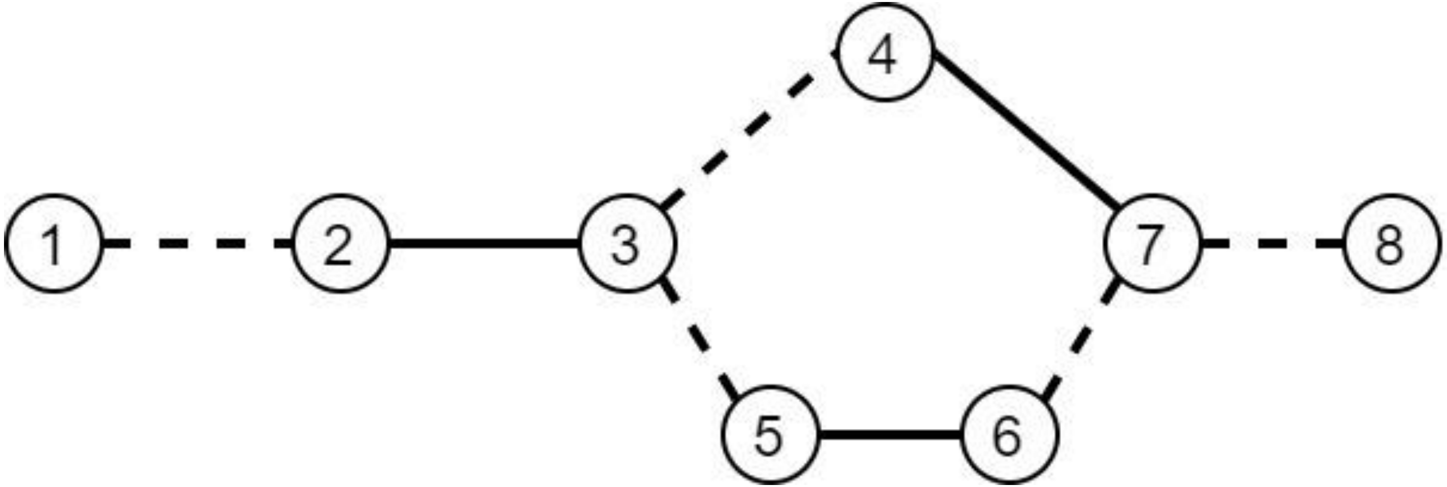
Hopcroft Karp

- Each phase of the algorithm augments the graph using a maximal set of augmenting paths.
- This implies that after each phase, length of shortest path increases by 1.
- So after $\sqrt{V}$ iterations, all paths are at least $\sqrt{V}$. Let this matching be M
- $M \Delta M^*$ contains vertex disjoint augmenting paths. By pigeonhole principle no more than $\sqrt{V}$ augmenting paths can be present.
- Each phase also increases the matching by at least 1. Hence no more than $\sqrt{V}$ phases are required.
- Each phase takes O(E) time so overall time is no more than $2E\sqrt{V}$

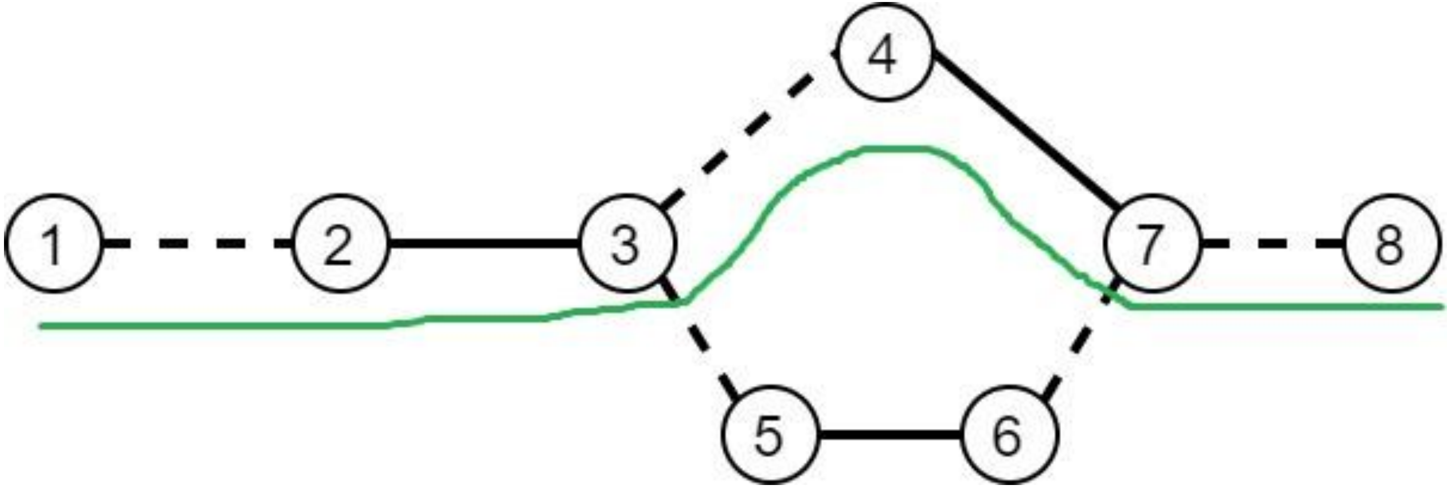
Blossom Algorithm

Jack Edmonds

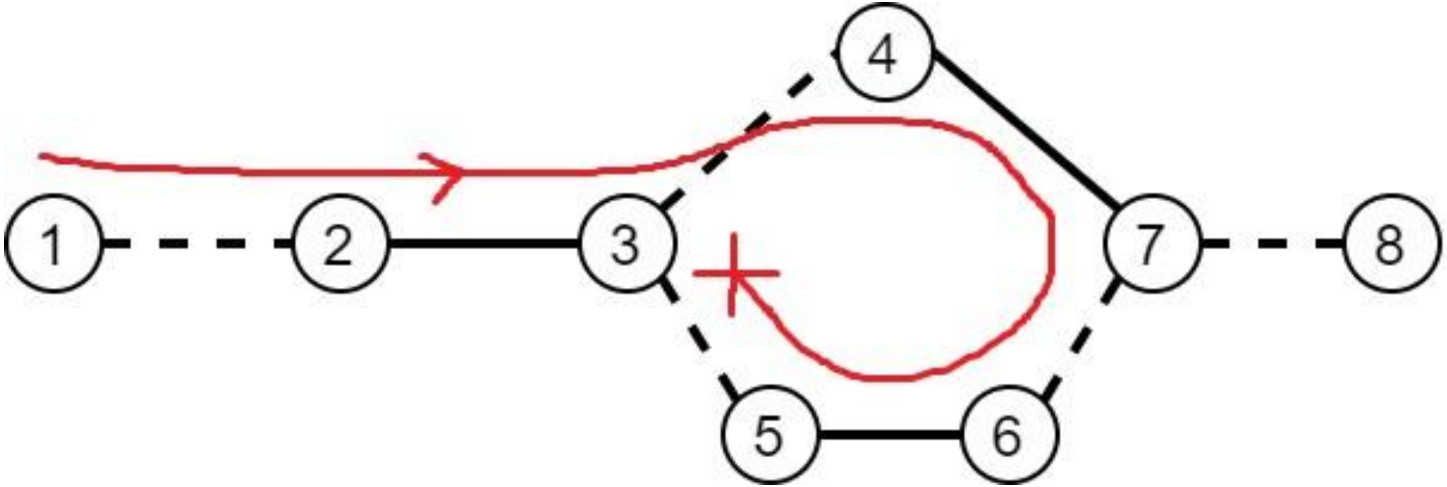
Blossom



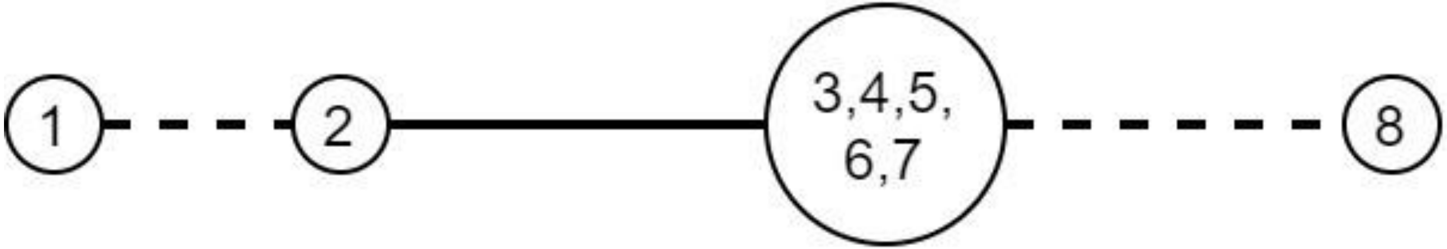
Blossom



Blossom



Blossom

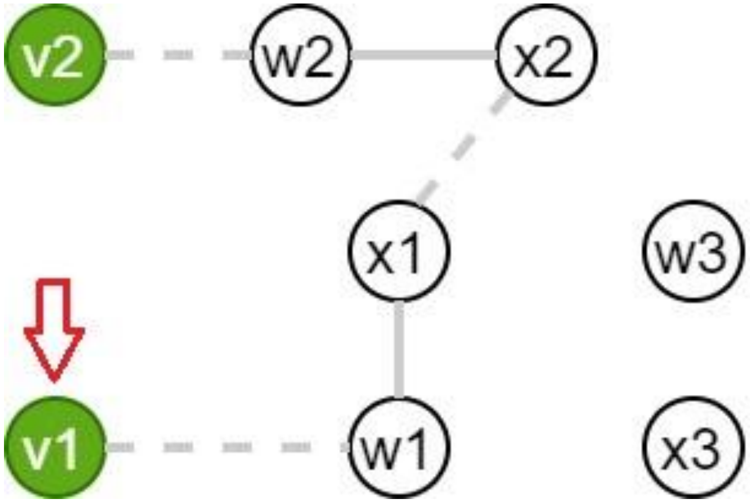


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
  Add v to F
  rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
  for *each unmarked edge (v, w) adjacent to v* **do**
    if $w \notin F$ **then**
      AddToForest(M, F, v, w, terminals, rootOfNode)
    else
      $p_w \leftarrow$ rootOfNode(w)
      if *$d(w, p_w)$ is even* **then**
        $p_v \leftarrow$ rootOfNode(v)
        if $p_v == p_w$ **then**
          $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
        else
          $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
        return P
    mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

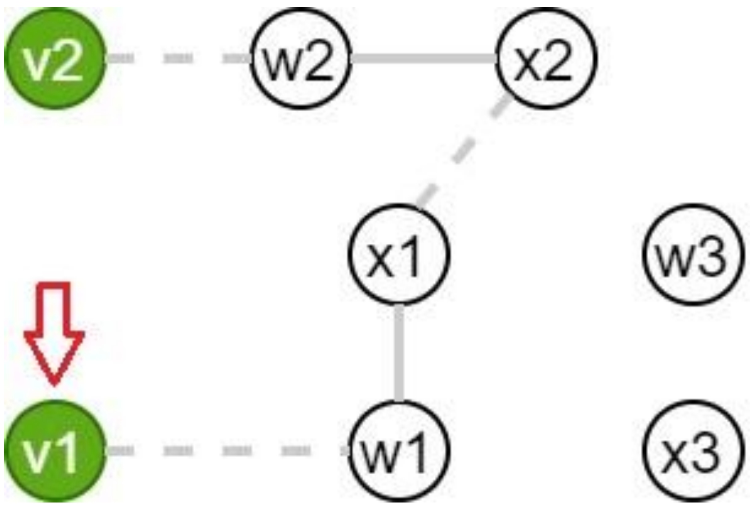


Blossom

```
F ← empty forest
rootOfNode ← empty node-to-node mapping
terminals ← unsaturated vertices of G
for each node v in terminals do
    Add v to F
    rootOfNode(v) ← v
mark all edges of M in G
for each node v in terminals do
    for each unmarked edge (v, w) adjacent to v do
        if w ∉ F then
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            p_w ← rootOfNode(w)
            if d(w, p_w) is even then
                p_v ← rootOfNode(v)
                if p_v == p_w then
                    P ← BlossomRecursion(G, M, F, v, w)
                else
                    P ← ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path
```

Algorithm 4: AddToForest

```
Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
x ← node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) ← rootOfNode(v)
rootOfNode(x) ← rootOfNode(v)
```

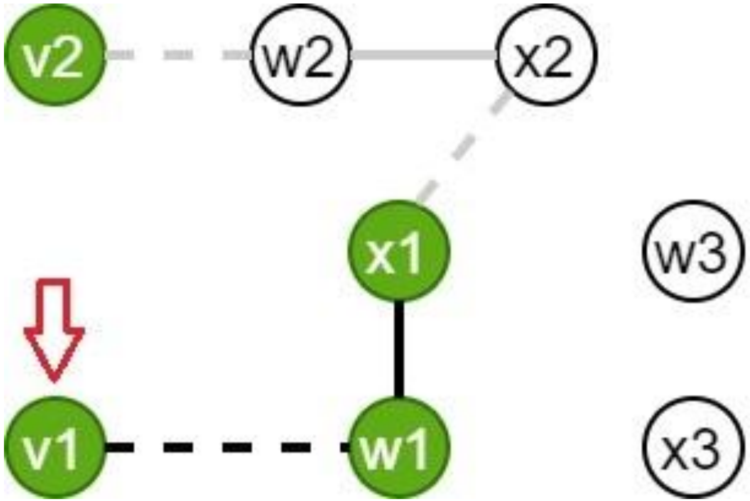


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
    Add v to F
    rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
    for *each unmarked edge (v, w) adjacent to v* **do**
        if $w \notin F$ **then**
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            $p_w \leftarrow$ rootOfNode(w)
            if *$d(w, p_w)$ is even* **then**
                $p_v \leftarrow$ rootOfNode(v)
                if $p_v == p_w$ **then**
                    $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
                else
                    $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

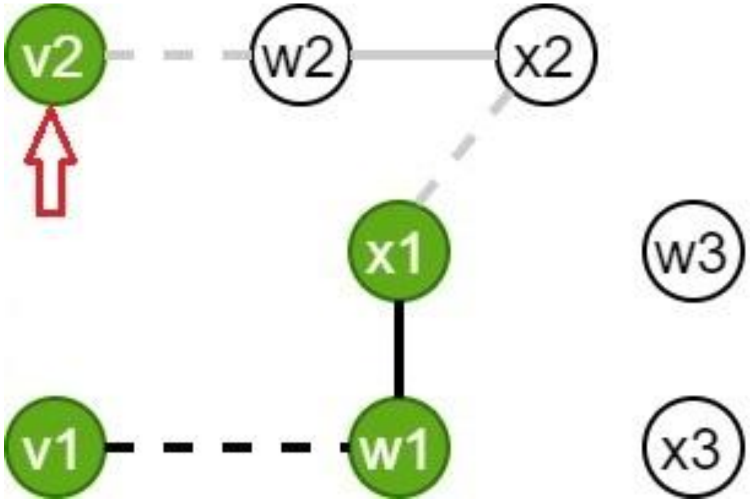


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
    Add v to F
    rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
    for *each unmarked edge (v, w) adjacent to v* **do**
        if $w \notin F$ **then**
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            $p_w \leftarrow$ rootOfNode(w)
            if $d(w, p_w)$ *is even* **then**
                $p_v \leftarrow$ rootOfNode(v)
                if $p_v == p_w$ **then**
                    $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
                else
                    $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

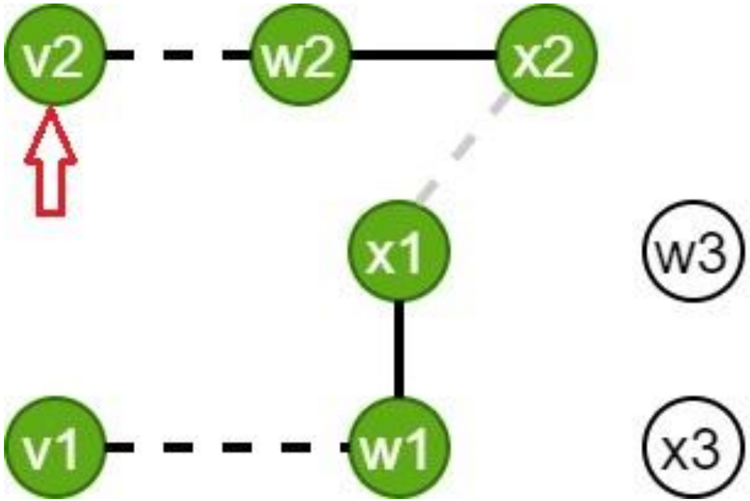


Blossom

```
F ← empty forest
rootOfNode ← empty node-to-node mapping
terminals ← unsaturated vertices of G
for each node v in terminals do
    Add v to F
    rootOfNode(v) ← v
mark all edges of M in G
for each node v in terminals do
    for each unmarked edge (v, w) adjacent to v do
        if w ∉ F then
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            p_w ← rootOfNode(w)
            if d(w, p_w) is even then
                p_v ← rootOfNode(v)
                if p_v == p_w then
                    P ← BlossomRecursion(G, M, F, v, w)
                else
                    P ← ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path
```

Algorithm 4: AddToForest

```
Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
x ← node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) ← rootOfNode(v)
rootOfNode(x) ← rootOfNode(v)
```

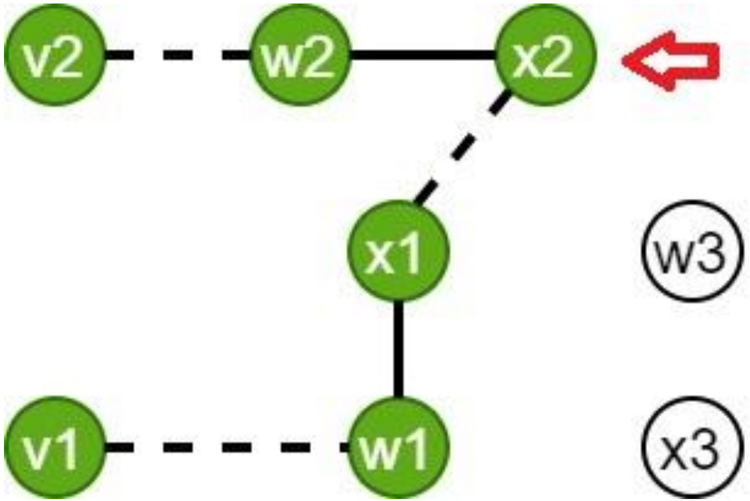


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
    Add v to F
    rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
    for *each unmarked edge (v, w) adjacent to v* **do**
        if $w \notin F$ **then**
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            $p_w \leftarrow$ rootOfNode(w)
            if *$d(w, p_w)$ is even* **then**
                $p_v \leftarrow$ rootOfNode(v)
                if $p_v == p_w$ **then**
                    $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
                else
                    $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

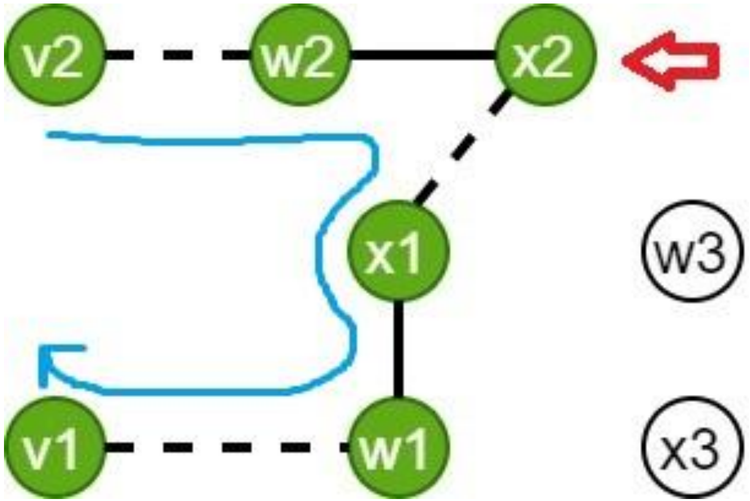


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
  Add v to F
  rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
  for *each unmarked edge (v, w) adjacent to v* **do**
    if $w \notin F$ **then**
      AddToForest(M, F, v, w, terminals, rootOfNode)
    else
      $p_w \leftarrow$ rootOfNode(w)
      if $d(w, p_w)$ *is even* **then**
        $p_v \leftarrow$ rootOfNode(v)
        if $p_v == p_w$ **then**
          $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
        else
          $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
        return P
    mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

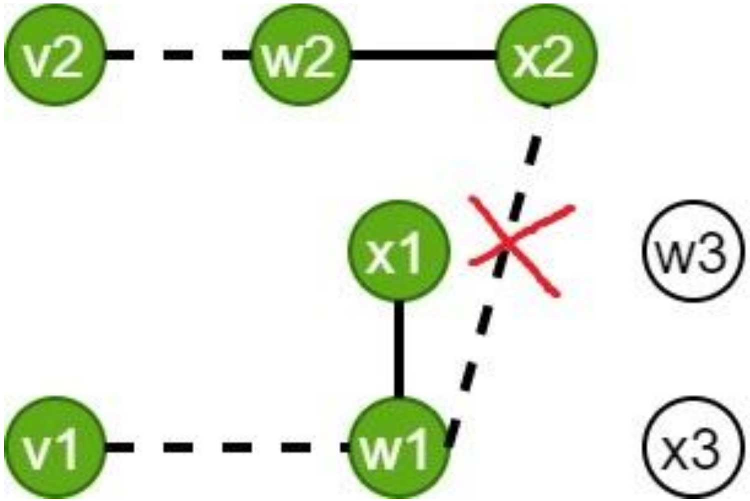


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
    Add v to F
    rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
    for *each unmarked edge (v, w) adjacent to v* **do**
        if $w \notin F$ **then**
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            $p_w \leftarrow$ rootOfNode(w)
            if $d(w, p_w)$ *is even* **then**
                $p_v \leftarrow$ rootOfNode(v)
                if $p_v == p_w$ **then**
                    $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
                else
                    $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

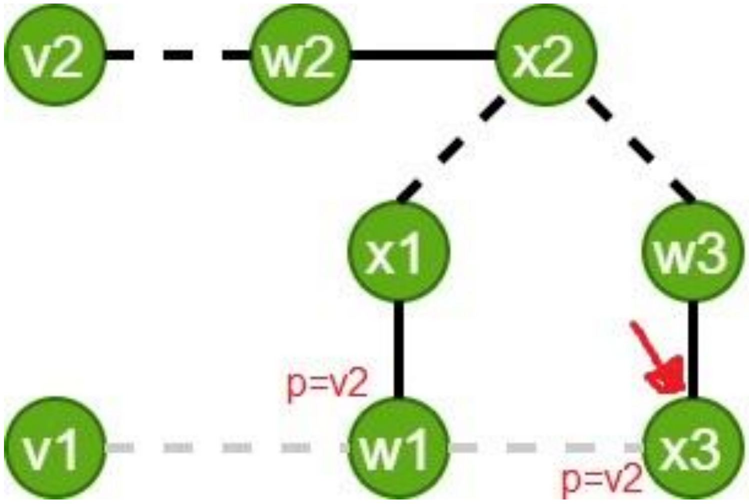


Blossom

$F \leftarrow$ empty forest
rootOfNode $\leftarrow$ empty node-to-node mapping
terminals $\leftarrow$ unsaturated vertices of G
for *each node v in terminals* **do**
    Add v to F
    rootOfNode(v) $\leftarrow v$
mark all edges of M in G
for *each node v in terminals* **do**
    for *each unmarked edge (v, w) adjacent to v* **do**
        if $w \notin F$ **then**
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            $p_w \leftarrow$ rootOfNode(w)
            if $d(w, p_w)$ *is even* **then**
                $p_v \leftarrow$ rootOfNode(v)
                if $p_v == p_w$ **then**
                    $P \leftarrow$ BlossomRecursion(G, M, F, v, w)
                else
                    $P \leftarrow$ ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path

Algorithm 4: AddToForest

Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
$x \leftarrow$ node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) $\leftarrow$ rootOfNode(v)
rootOfNode(x) $\leftarrow$ rootOfNode(v)

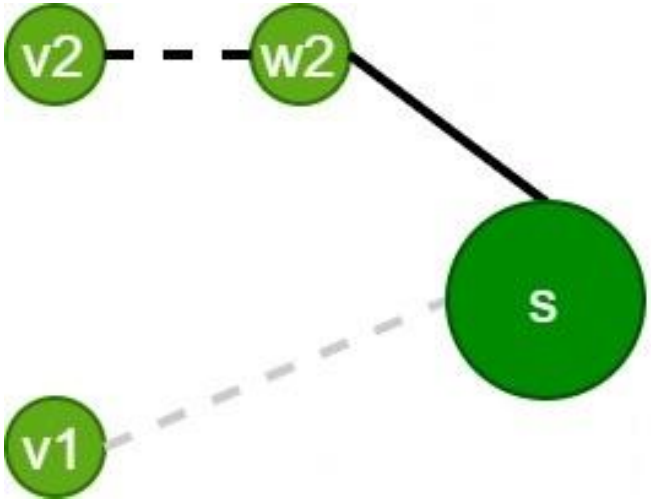


Blossom

```
F ← empty forest
rootOfNode ← empty node-to-node mapping
terminals ← unsaturated vertices of G
for each node v in terminals do
    Add v to F
    rootOfNode(v) ← v
mark all edges of M in G
for each node v in terminals do
    for each unmarked edge (v, w) adjacent to v do
        if w ∉ F then
            AddToForest(M, F, v, w, terminals, rootOfNode)
        else
            p_w ← rootOfNode(w)
            if d(w, p_w) is even then
                p_v ← rootOfNode(v)
                if p_v == p_w then
                    P ← BlossomRecursion(G, M, F, v, w)
                else
                    P ← ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
return empty path
```

Algorithm 4: AddToForest

```
Data: M, F, v, w, terminals, rootOfNode
Result: Null (Utility Function)
x ← node adjacent to w in M
add edges (v, w) and (w, x) in F
add node x in terminals
rootOfNode(w) ← rootOfNode(v)
rootOfNode(x) ← rootOfNode(v)
```



Blossom

- Algorithm proceeds in phases increasing matching by 1. So at max O(V) phases.
- In the algorithm to find augmenting path, if no blossom is encountered we do work of O(E).
- If blossom is found then we contract the blossom. On returning from the recursion we will have to reopen the blossom. Call the work done upto this step as a sub phase.
- Each sub phase take O(E) work due to E edges in the outer loop and blossom contraction and expansion. The size of graph reduces by size of blossom so at maximum O(V) such sub phases.
- Hence total time complexity O(VxExV) = $O(V^2E)$

Blossom

- At most O(V) phases
- Each sub phase takes $O(V^2/p+V+E/p)$ time
- At most O(V) sub phases
- Total time for augmenting path algorithm $O(V^3/p+V^2+VE/p)$
- Total time $O(V^3+V^4/p+V^2E/p2)$

```
mark all edges of M in G
for each node v in terminals do
    temp ← empty map
    for each unmarked edge (v, w) adjacent to v in PARALLEL do
        if w ∉ F then
            x ← node adjacent to w in M
            temp(w) ← x
        else
            p_w ← rootOfNode(w)
            if d(w, p_w) is even then
                p_v ← rootOfNode(v)
                if p_v == p_w then
                    P ← BlossomRecursion(G, M, F, v, w)
                else
                    P ← ConstructPath(F, v, w, rootOfNode)
                return P
        mark edge (v, w)
    for each node w in temp in PARALLEL do
        if w == temp(temp(w)) then
            P ← BlossomRecursion(G, M, F, v, w)
            return P
        else
            add edges (v, w) and (w, temp(w) = x) in F
            add node x = temp(w) to terminals
return empty path
```

Hungarian Algorithm

Harold Kuhn

Hungarian

- Labelling Function L: each vertex is assigned an integer label.
- Feasible labeling: $l(x) + l(y) \geq w(x, y)$
- Equality graph G_L : consider only edges $l(x) + l(y) = w(x, y)$
- KM Lemma: A perfect matching in G_L is maximum in G: since for some perfect matching (not maximum in G) $w(M) = \Sigma w(x, y) \leq \Sigma l(x)+l(y)$ and for perfect matching in G_L $w(M^*) = \Sigma w(x, y) = \Sigma l(x)+l(y)$ and hence $w(M^*) > w(M)$
- $S \subseteq X$, $T = N_G(S)$

$$\alpha_l = \min(\{l(x) + l(y) - w(x,y) : x \in S, y \notin T\})$$

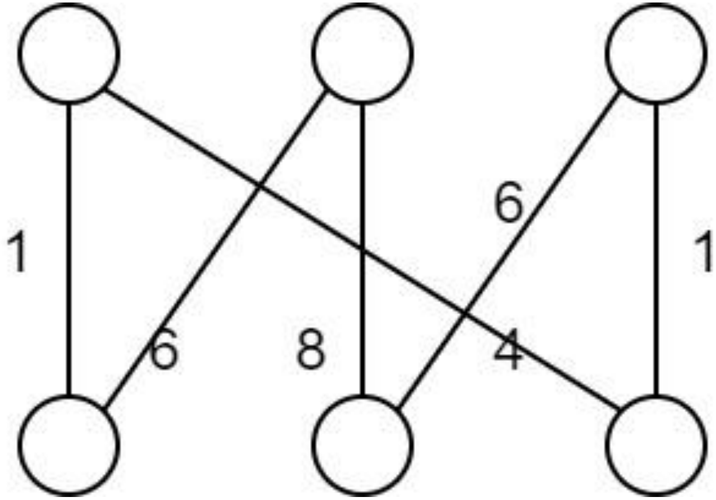
$$l'(v) = \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

Hungarian

pick unsaturated vertex $u \in X$
$S \leftarrow \{u\}$ and $T \leftarrow \phi$
$P \leftarrow$ NULL
while *P is NULL (path not found)* **do**
  if $N_l(S) = T$ **then**
    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x,y) : x \in S, y \notin T\})$

$$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

  $l \leftarrow l^*$
  pick one v from $\in N_l(S) - T$
  if *v is unsaturated* **then**
    $P \leftarrow$ path from v to root u
  else
    $z \leftarrow$ the matched vertex of v by edge $e(v,z) \in M$
    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$
return P, l^*



Hungarian

pick unsaturated vertex $u \in X$

$S \leftarrow \{u\}$ and $T \leftarrow \phi$

$P \leftarrow$ NULL

while P *is NULL (path not found)* **do**

  if $N_l(S) = T$ **then**

    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x,y) : x \in S, y \notin T\})$

$$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

  $l \leftarrow l^*$

  pick one v from $\in N_l(S) - T$

  if v *is unsaturated* **then**

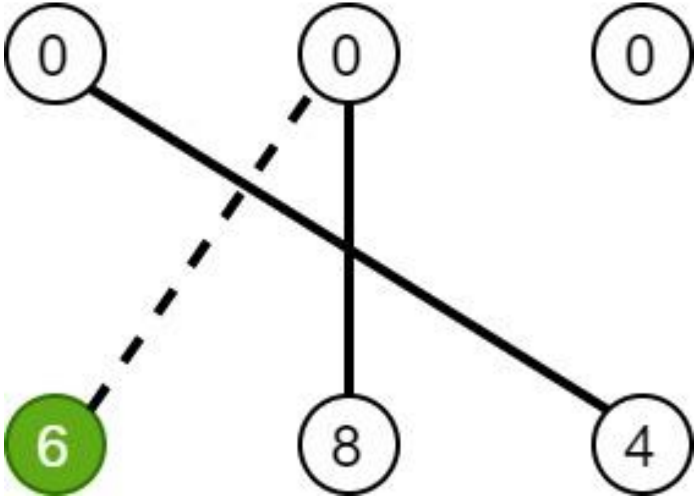
    $P \leftarrow$ path from v to root u

  else

    $z \leftarrow$ the matched vertex of v by edge $e(v,z) \in M$

    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$

return P, l^*



Hungarian

pick unsaturated vertex $u \in X$

$S \leftarrow \{u\}$ and $T \leftarrow \phi$

$P \leftarrow$ NULL

while P *is NULL (path not found)* **do**

  if $N_l(S) = T$ **then**

    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x,y) : x \in S, y \notin T\})$

$$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

  $l \leftarrow l^*$

  pick one v from $\in N_l(S) - T$

  if v *is unsaturated* **then**

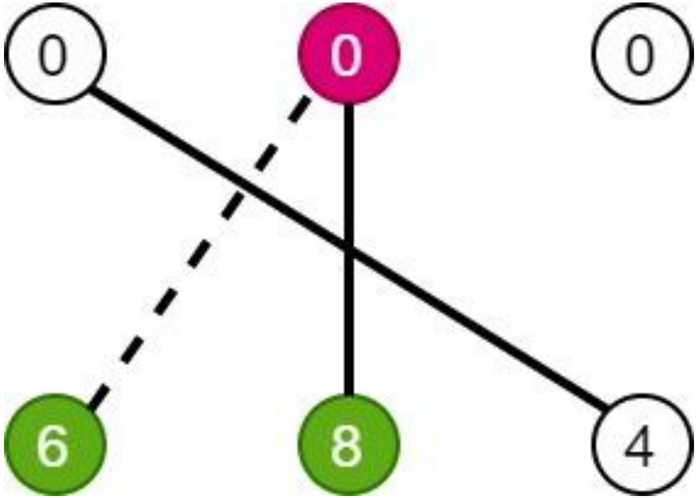
    $P \leftarrow$ path from u to root v

  else

    $z \leftarrow$ the matched vertex of v by edge $e(v,z) \in M$

    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$

return P, l^*

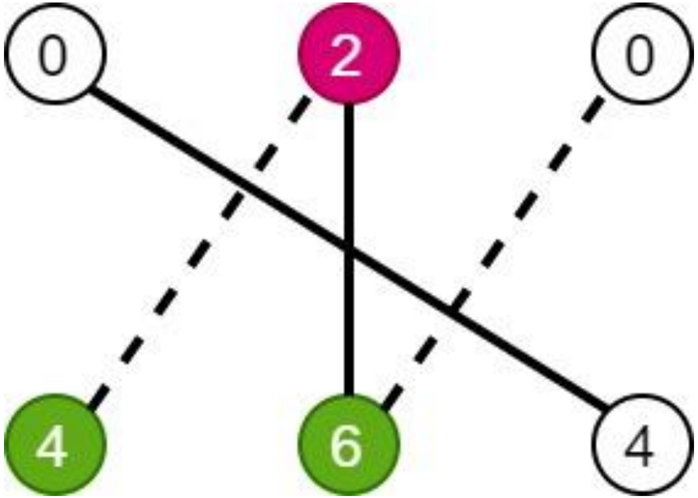


Hungarian

pick unsaturated vertex $u \in X$
$S \leftarrow \{u\}$ and $T \leftarrow \phi$
$P \leftarrow$ NULL
while *P is NULL (path not found)* **do**
  if $N_l(S) = T$ **then**
    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x, y) : x \in S, y \notin T\})$

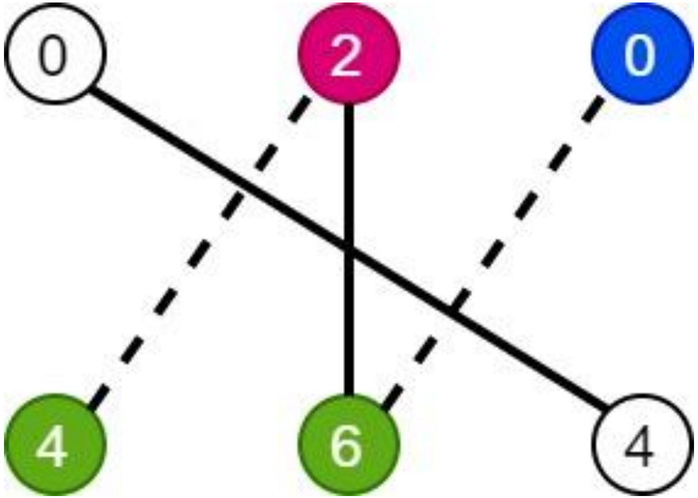
$$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

  $l \leftarrow l^*$
  pick one v from $\in N_l(S) - T$
  if *v is unsaturated* **then**
    $P \leftarrow$ path from v to root u
  else
    $z \leftarrow$ the matched vertex of v by edge $e(v, z) \in M$
    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$
return P, l^*



Hungarian

pick unsaturated vertex $u \in X$
$S \leftarrow \{u\}$ and $T \leftarrow \phi$
$P \leftarrow$ NULL
while P *is NULL (path not found)* **do**
  if $N_l(S) = T$ **then**
    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x,y) : x \in S, y \notin T\})$
    $$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$
  $l \leftarrow l^*$
  pick one v from $\in N_l(S) - T$
  if v *is unsaturated* **then**
    $P \leftarrow$ path from v to root u
  else
    $z \leftarrow$ the matched vertex of v by edge $e(v,z) \in M$
    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$
return P, l^*

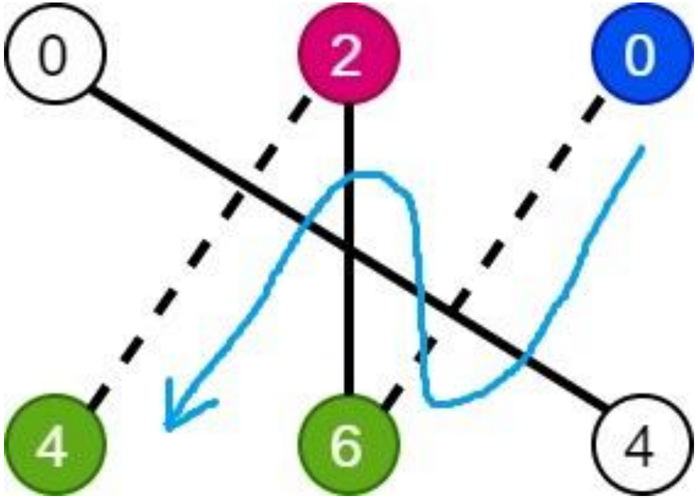


Hungarian

pick unsaturated vertex $u \in X$
$S \leftarrow \{u\}$ and $T \leftarrow \phi$
$P \leftarrow$ NULL
while P *is NULL (path not found)* **do**
  if $N_l(S) = T$ **then**
    $\alpha_l \leftarrow \min(\{l(x) + l(y) - w(x, y) : x \in S, y \notin T\})$

$$l^*(v) \leftarrow \begin{cases} l(v) - \alpha_l & v \in S \\ l(v) + \alpha_l & v \in T \\ l(v) & \text{otherwise} \end{cases}$$

  $l \leftarrow l^*$
  pick one v from $\in N_l(S) - T$
  if v *is unsaturated* **then**
    $P \leftarrow$ path from v to root u
  else
    $z \leftarrow$ the matched vertex of v by edge $e(v, z) \in M$
    $S \leftarrow S \cup \{z\}$ and $T \leftarrow T \cup \{v\}$
return P, l^*



Hungarian

- There will be at most O(V) phases.
- In each iteration of a single phase, we either move a vertex into S or return an augmenting path. Since there are only V vertices, there will be at max O(V) iterations.
- Finding the minimum slack can take $O(V^2)$ time.
- Total time complexity: $O(V^4)$

Thank You