UNIT-IV

Summarize the Simplex method. Or Write the procedure to initialize simplex which determine if a linear program is feasible or not . Or Use simplex to solve the farmers problem given below.

Simplex method procedure:

Step 0 [Initialization] Present a given LP problem in standard form and set up initial tableau.

Step 1 [Optimality test] If all entries in the objective row are nonnegative then stop: the tableau represents an optimal solution.

Step 2 [Find entering variable] Select the most negative entry in the objective row. Mark its column to indicate the entering variable and the pivot column.

Step 3 [Find departing [leaving] variable] For each positive entry in the pivot column, calculate the θ-ratio by dividing that row's entry in the rightmost column [solution] by its entry in the pivot column. [If there are no positive entries in the pivot column then stop: the problem is unbounded.] Find the row with the smallest θ-ratio, mark this row to indicate the departing variable and the pivot row.

Step 4 [Form the next tableau] Divide all the entries in the pivot row by its entry in the pivot column. Subtract from each of the other rows, including the objective row, the new pivot row multiplied by the entry in the pivot column of the row in question. Replace the label of the pivot row by the variable's name of the pivot column and go back to Step 1.

Standard form of LP problem

- Must be a maximization problem
- All constraints [except the nonnegativity constraints] must be in the form of linear equations
- All the variables must be required to be nonnegative
- Thus, the general linear programming problem in standard form with m constraints and n unknowns [$n \geq m$] is

  Maximize $c1\ x1 + ... + cn\ xn$

  Subject to $ai\ 1x1 + ... + ain\ xn = bi$, $i = 1,...,m, x1 \geq 0, ... , xn \geq 0$

  Example

  maximize $3x + 5y$ maximize $3x + 5y$

  $+ 0u + 0v$ subject to $x + y \leq 4$

  subject to $x + y + u = 4$

  $x + 3y \leq 6$ to $x + 3y + v = 6$

  $x\geq0, y\geq0$ $x\geq0, y\geq0, u\geq0, v\geq0$

  Variables u and v, transforming inequality constraints into equality constrains, are called *slack Variables*

$$\begin{array}{cc|cccc|c|}
 & & x & y & u & v & \\ \hline
 & u & 1 & 1 & 1 & 0 & 4 \\
\leftarrow & v & 1 & 3 & 0 & 1 & 6 \\ \hline
 & & -3 & -5 & 0 & 0 & 0 \\ \hline
 & & & \uparrow & & &
\end{array}$$

$$\begin{array}{cc|cccc|c|}
 & & x & y & u & v & \\ \hline
\leftarrow & u & \frac{2}{3} & 0 & 1 & -\frac{1}{3} & 2 \\
 & y & \frac{1}{3} & 1 & 0 & \frac{1}{3} & 2 \\ \hline
 & & -\frac{4}{3} & 0 & 0 & \frac{5}{3} & 10 \\ \hline
 & & \uparrow & & & &
\end{array}$$

$$\begin{array}{c|cccc|c|}
 & x & y & u & v & \\ \hline
x & 1 & 0 & 3/2 & -1/3 & 3 \\
y & 0 & 1 & -1/2 & 1/2 & 1 \\ \hline
 & 0 & 0 & 2 & 1 & 14 \\ \hline
\end{array}$$

basic feasible sol. (3, 1, 0, 0) $z = 14$

Example 1:

Use Simplex method to solve the formers problem given below.

A farmer has a 320 acre farm on which she plants two crops: corn and soybeans. For each acre of corn planted, her expenses are \$50 and for each acre of soybeans planted, her expenses are \$100. Each acre of corn requires 100 bushels of storage and yields a profit of \$60; each acre of soybeans requires 40 bushels of storage and yields a profit of \$90. If the total amount of storage space available is 19,200 bushels and the farmer has only \$20,000 on hand, how many acres of each crop should she plant in order to maximize her profit? What will her profit be if she follows this strategy?

Solution

Linear Programming Problem Formulation

	Corn	Soybean	**Total**
Expenses	\$50	\$100	\$20,000
Storage(bushels)	100	40	19,200
Profit	60	90	Maximize profit

A farmer has a 320 acre farm is unwanted data but c+s<=320.

c = corn planted acres and s = soybean planted acres

$50c + 100s \leq 20,000$

$100c + 40s \leq 19,200$

Maximize: $60c + 90s = P$

Canonical form of LPP

Maximize: $60c + 90s$

Subject to $50c + 100s = 20000$

$100c + 40s = 19200$

$c \geq 0, s \geq 0$

Solving by algebra (Intersection of lines)

Maximize: $60c + 90s$

Subject to $50c + 100s = 20000$ (1)

$100c + 40s = 19200$ (2)

(1)/50 => $c + 2s = 400$

(2)/20 => $5c + 2s = 960$

(2) – (1) => $4c = 560$

$c = 140$

Substitute c = 140 in (1) then s = 130

Profit: $p = 60c + 90s = 60(140) + 90(130) = \$20,100$

She should plant 140 acres corn and 130 acres of soybean for \$20,100.

State and Prove Maximum Flow Min cut Theorem. Or Explain Max-Flow Problem Or How do you compute maximum flow for the following graph using ford-Fulkerson method?

Maximum Flow Problem

Problem of maximizing the flow of a material through a transportation network [e.g., pipeline system, communications or transportation networks]

Formally represented by a connected weighted digraph with *n* vertices numbered from 1 to *n* with the following properties:

- Contains exactly one vertex with no entering edges, called the ***source*** [numbered 1]
- Contains exactly one vertex with no leaving edges, called the ***sink*** [numbered *n*]
- Has positive integer weight *uij* on each directed edge [*i.j*], called the ***edge capacity,*** indicating the upper bound on the amount of the material that can be sent from *i* to *j* through this edge.

A digraph satisfying these properties is called a **flow network** or simply a network

Flow value and Maximum Flow Problem

Since no material can be lost or added to by going through intermediate vertices of the network, the total amount of the material leaving the source must end up at the sink:

$\Sigma\, x1j = \Sigma\, xjn$

j: $[1,j] \in E$ j: $[j,n] \in E$

The *value* of the flow is defined as the total outflow from the source [= the total inflow into the sink].

The *maximum flow problem* is to find a flow of the largest value [maximum flow] for a given network.

Max-Flow Min-Cut Theorem

1. The value of maximum flow in a network is equal to the capacity of its minimum cut
2. The shortest augmenting path algorithm yields both a maximum flow and a minimum cut:
   - Maximum flow is the final flow produced by the algorithm
   - Minimum cut is formed by all the edges from the labeled vertices to unlabeled vertices on the last iteration of the algorithm.
   - All the edges from the labeled to unlabeled vertices are full, i.e., their flow amounts are equal to the edge capacities, while all the edges from the unlabeled to labeled vertices, if any, have zero flow
     amounts on them.

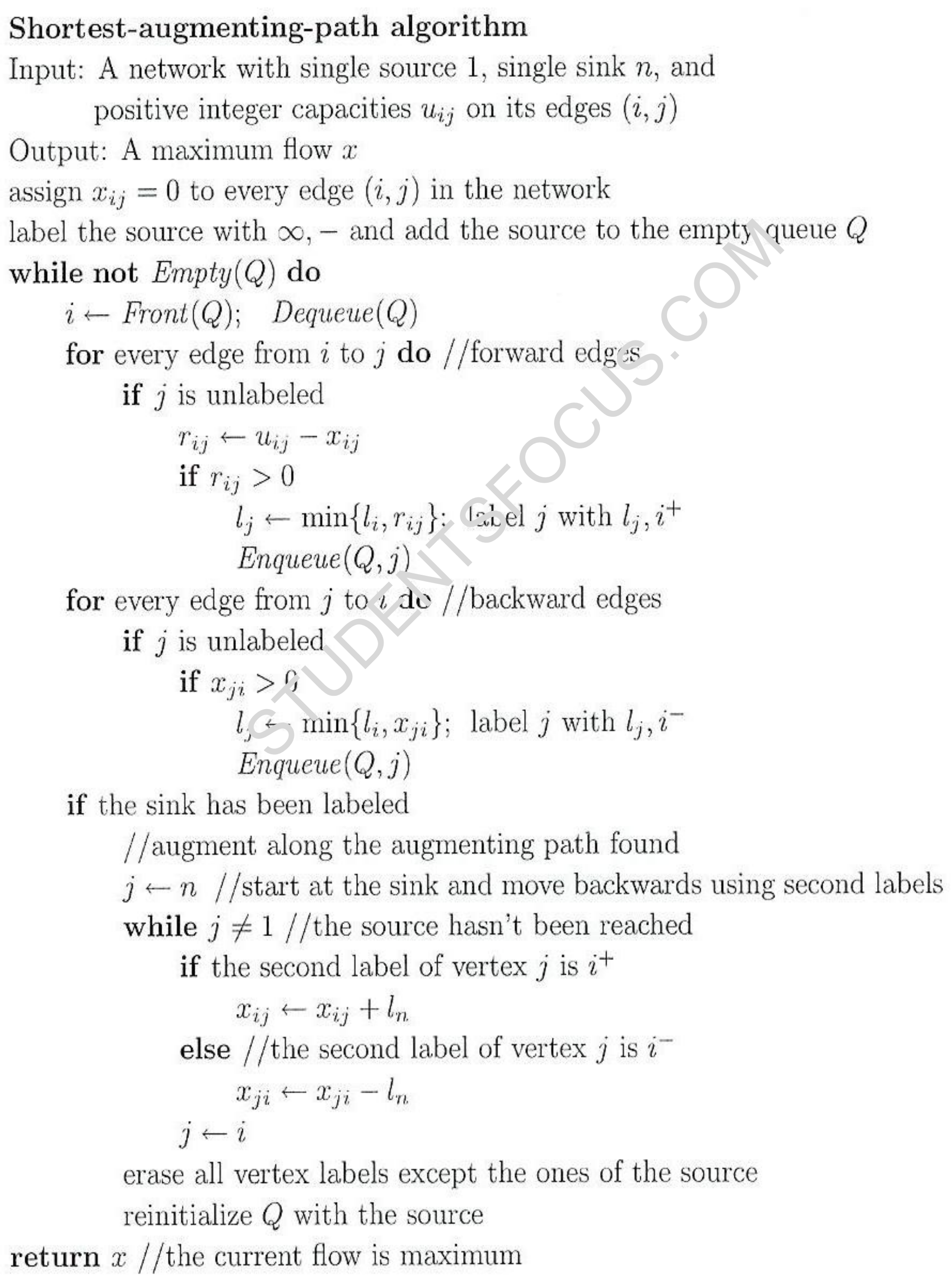
Flow augmenting path algorithm or Ford Fulkerson method to solve max-flow problem

Thus, to find a flow-augmenting path for a flow x, we need to consider paths from source to sink in the underlying undirected graph in which any two consecutive vertices i, j are either

i. connected by a directed edge from i to j with some positive unused capacity rij = uij − xij (so that we can increase the flow through that edge by up to rij units), or
ii. connected by a directed edge from j to i with some positive flow xji (so that we can decrease the flow through that edge by up to xji units)

For example,

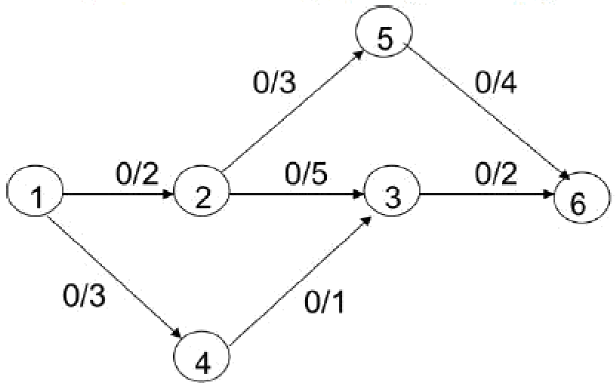
Edges of the first kind are called forward edges because their tail is listed before their head in th vertex list 1→. . . i →j . . .→n defining the path; edges of the second kind are called backward edges because their tail is listed after their head in the path list 1→. . . i ←j . . .→n. To illustrate, for the path 1→4→3←2→5→6 of the last example, (1, 4), (4, 3), (2, 5), and (5, 6) are the forward edges, and (3, 2) is the backward edge.

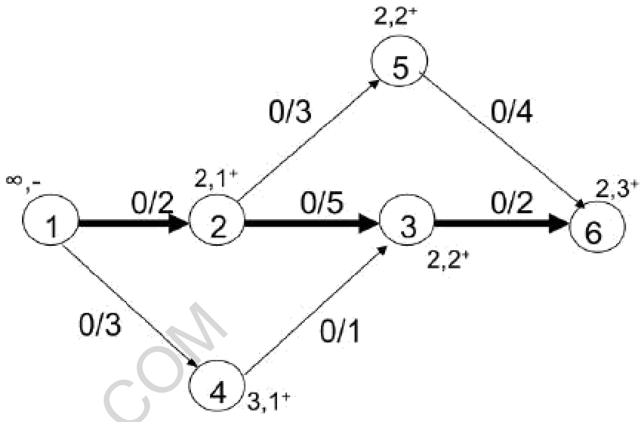


Shortest-augmenting-path algorithm

```
Input: A network with single source 1, single sink $n$, and
       positive integer capacities $u_{ij}$ on its edges $(i, j)$
Output: A maximum flow $x$
assign $x_{ij} = 0$ to every edge $(i, j)$ in the network
label the source with $\infty, -$ and add the source to the empty queue $Q$
while not $Empty(Q)$ do
    $i \leftarrow Front(Q)$;   $Dequeue(Q)$
    for every edge from $i$ to $j$ do //forward edges
        if $j$ is unlabeled
            $r_{ij} \leftarrow u_{ij} - x_{ij}$
            if $r_{ij} > 0$
                $l_j \leftarrow \min\{l_i, r_{ij}\}$;  label $j$ with $l_j, i^+$
                $Enqueue(Q, j)$
    for every edge from $j$ to $i$ do //backward edges
        if $j$ is unlabeled
            if $x_{ji} > 0$
                $l_j \leftarrow \min\{l_i, x_{ji}\}$;  label $j$ with $l_j, i^-$
                $Enqueue(Q, j)$
    if the sink has been labeled
        //augment along the augmenting path found
        $j \leftarrow n$  //start at the sink and move backwards using second labels
        while $j \neq 1$ //the source hasn't been reached
            if the second label of vertex $j$ is $i^+$
                $x_{ij} \leftarrow x_{ij} + l_n$
            else //the second label of vertex $j$ is $i^-$
                $x_{ji} \leftarrow x_{ji} - l_n$
            $j \leftarrow i$
        erase all vertex labels except the ones of the source
        reinitialize $Q$ with the source
return $x$ //the current flow is maximum
```

Example: Shortest-Augmenting-Path Algorithm

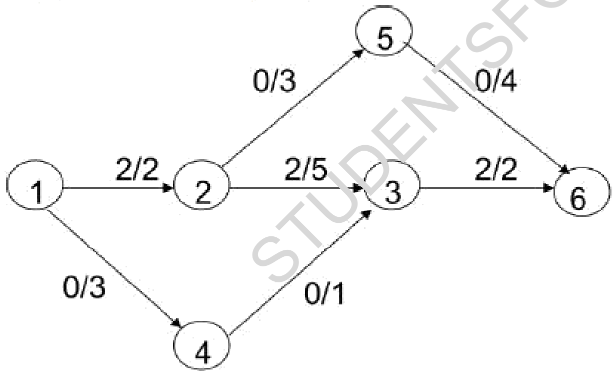


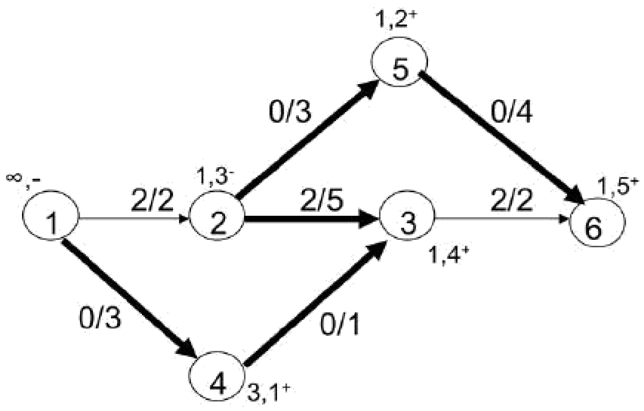


Queue: 1 2 4 3 5 6

↑↑↑↑

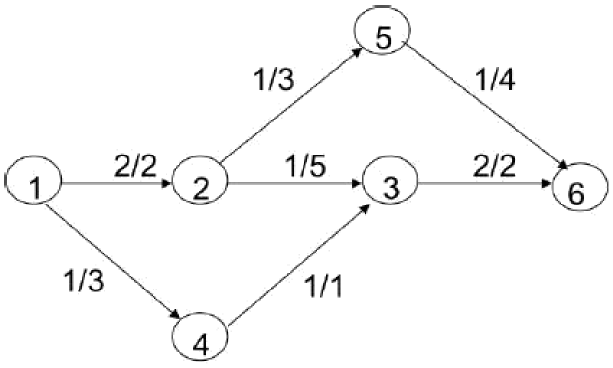
Augment the flow by 2 (the sink's first label) along the path 1→2→3→6

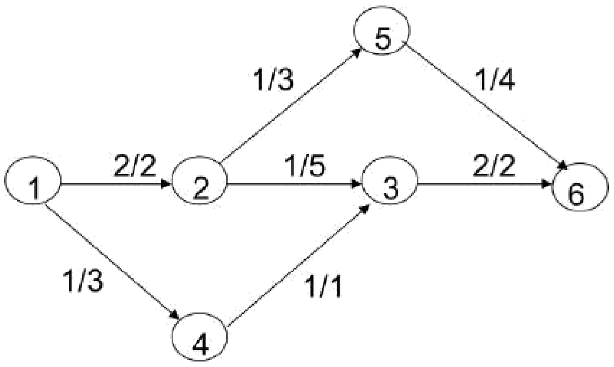




Queue: 1 4 3 2 5 6
↑ ↑ ↑ ↑ ↑

Augment the flow by 1 (the sink's first label) along the path 1→4→3←2→5→6





Queue: 1 4
↑ ↑

No augmenting path (the sink is unlabeled) the current flow is maximum

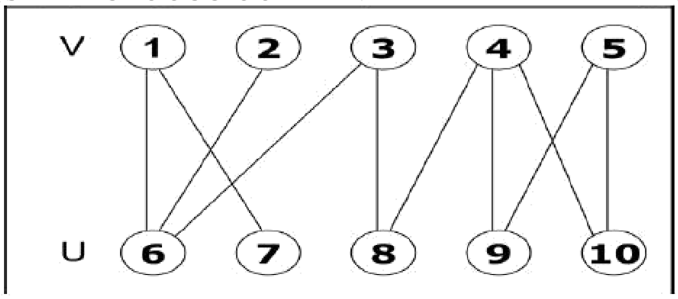
Write down the optimality condition and algorithmic implementation for finding M-augumenting paths in bipartite graphs? Or Illustrate the workings of the maximum matching algorithm on the following weighted trees.

A matching in a graph is a subset of its edges with the property that no two edges share a vertex. A maximum matching—more precisely, a maximum cardinality matching—is a matching with the largest number of edges.

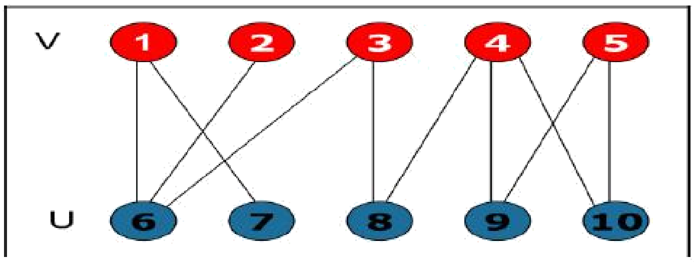
Case 1 (the queue's front vertex w is in V) If u is a free vertex adjacent to w, it is used as the other endpoint of an augmenting path; so the labeling stops and augmentation of the matching commences. The augmenting path in question is obtained by moving backward along the vertex labels (see below) to alternately add and delete its edges to and from the current matching. If u is not free and connected to w by an edge not in M, label u with w unless it has been already labeled.

Case 2 (the front vertex w is in U) In this case, w must be matched and we label its mate in V with w.

A graph is bipartite if and only if it does not have a cycle of an odd length.



A bipartite graph is *2-colorable*: the vertices can be colored in two colors so that every edge has its vertices colored differently



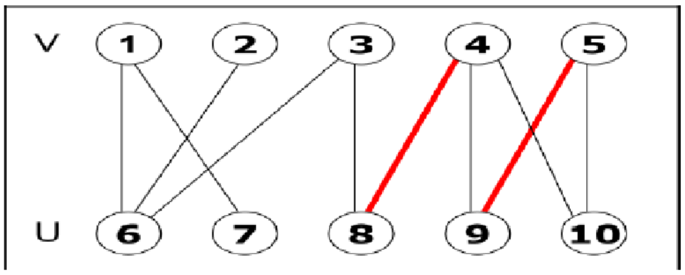
ALGORITHM Maximum Bipartite Matching(G)

```
//Finds a maximum matching in a bipartite graph by a BFS-
like traversal //Input: A bipartite graph G = V, U, E
//Output: A maximum-cardinality matching M in the input
graph initialize set M of edges with some valid matching
(e.g., the empty set) initialize queue Q with all the free
vertices in V (in any order) while not Empty(Q) do
w←Front(Q); Dequeue(Q)
if w ∈ V
for every vertex u adjacent to w do
if u is free
//augment
M ←M ∪ (w, u)
v←w
while v is labeled do
u←vertex indicated by v‘s label; M ←M − (v, u)
v←vertex indicated by u‘s label; M ←M ∪ (v, u)
remove all vertex labels
reinitialize Q with all free vertices in V
break //exit the for loop
else //u is matched
if (w, u) ∈ M and u is unlabeled
label u with w
Enqueue(Q, u)
else //w ∈ U (and matched)
label the mate v of w with w
```

```
Enqueue(Q, v)
    return M //current matching is maximum
```

For Example,

A *matching* in a graph is a subset of its edges with the property that no two edges share a vertex

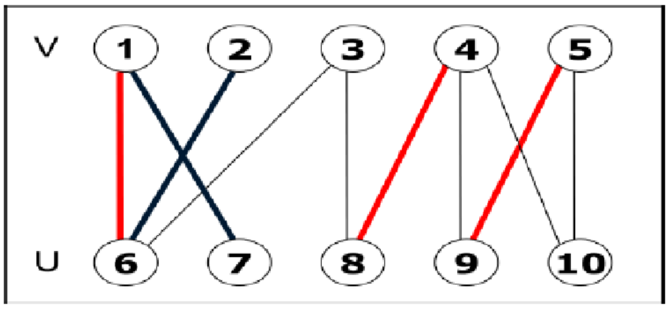


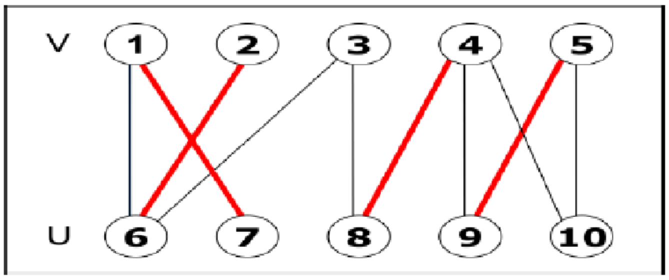
a matching in this graph M = {(4,8), (5,9)}

Augmenting Paths and Augmentation

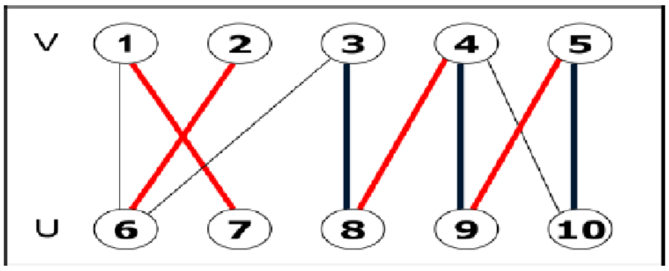
An *augmenting path* for a matching M is a path from a free vertex in V to a free vertex in U whose edges alternate between edges not in M and edges in M

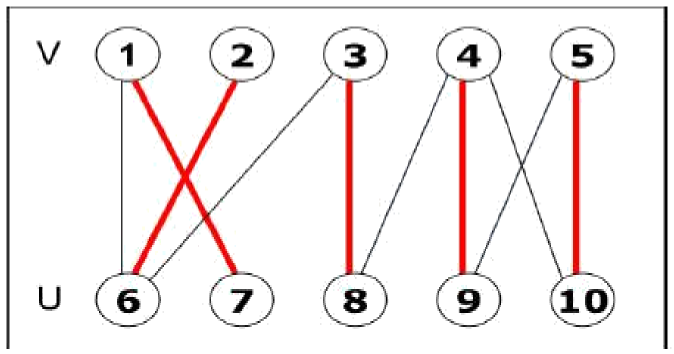
- The length of an augmenting path is always odd
- Adding to M the odd numbered path edges and deleting from it the even numbered path edges increases the matching size by 1 (*augmentation*)
- One-edge path between two free vertices is special case of augmenting path





Augmentation along path 2,6,1,7





Augmentation along 3, 8, 4, 9, 5, 10

Matching on the right is maximum (*perfect* matching).

Briefly describe on the stable marriage problem. (or)Gale shaply algorithm.

Stable marriage algorithm

A marriage matching M is a set of n (m, w) pairs whose members are selected from disjoint n-element sets Y and X in a one-one fashion, i.e., each man m from Y is paired with exactly one woman w from X and vice versa

Consider an interesting version of bipartite matching called the stable marriage problem.

Consider a set Y = {m1, m2, . . . , mn } of n men and a set X = {w1, w2, . . . , wn } of n women. Each man has a preference list ordering the women as potential marriage partners with no ties allowed. Similarly, each woman has a preference list of the men, also with no ties.

Input: A set of n men and a set of n women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings

Output: A stable marriage matching

Step 0 Start with all the men and women being free.

Step 1 While there are free men, arbitrarily select one of them and do the following:

Proposal The selected free man m proposes to w, the next woman on his preference list (who is the highest-ranked woman who has not rejected him before).

Response If w is free, she accepts the proposal to be matched with m. If she is not free, she compares m with her current mate. If she prefersm to him, she accepts m‘s proposal, making her former mate free; otherwise, she simply rejects m‘s proposal, leaving m free.

Step 2 Return the set of n matched pairs.

For Example,

Instance of the Stable Marriage Problem

An instance of the stable marriage problem can be specified either by two sets of preference lists or by a ranking matrix, as in the example below.

<u>men's preferences</u>

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

<u>women's preferences</u>

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

<u>ranking matrix</u>

	Ann	Lea	Sue
Bob	$\boxed{2,3}$	1,2	3,3
Jim	3,1	$\boxed{1,3}$	2,1
Tom	3,2	2,1	$\boxed{1,2}$

Example

Free men: Bob, Jim, Tom

	Ann	Lea	Sue
Bob	2,3	$\boxed{\underline{1,2}}$	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Lea, Lea accepted Bob

Free men: Jim, Tom

	Ann	Lea	Sue
Bob	2,3	$\boxed{1,2}$	3,3
Jim	3,1	<u>1,3</u>	2,1
Tom	3,2	2,1	1,2

Jim proposed to Lea, Lea rejected

Free men: Jim, Tom

	Ann	Lea	Sue
Bob	2,3	$\boxed{1,2}$	3,3
Jim	3,1	1,3	$\boxed{2,1}$
Tom	3,2	2,1	1,2

Jim proposed to Sue, Sue accepted

Free men: Tom

	Ann	Lea	Sue
Bob	2,3	$\boxed{\mathbf{1,2}}$	3,3
Jim	3,1	1,3	$\boxed{\mathbf{2,1}}$
Tom	3,2	2,1	$\underline{1,2}$

Tom proposed to Sue, Sue rejected

Free men: Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	$\boxed{\underline{\mathbf{2,1}}}$
Tom	3,2	$\boxed{\underline{\mathbf{2,1}}}$	1,2

Tom proposed to Lea, Lea replaced Bob with Tom

Free men: Bob

	Ann	Lea	Sue
Bob	$\boxed{2,3}$	1,2	3,3
Jim	3,1	1,3	$\boxed{2,1}$
Tom	3,2	$\boxed{2,1}$	1,2

Bob proposed to Ann, Ann accepted