SNS COLLEGE OF ENGINEERING

Coimbatore-107

COURSE NAME: ANALYSIS OF ALGORITHM

II YEAR/ IV SEMESTER

UNIT – IV

ITERATIVE IMPROVEMENT

Topic

Bipartite Graph

UNIT - IV ITERATIVE IMPROVEMENT & STRING MATCHING ALGORITHM

Flow N/w – Ford Fulkerson method - Maxim matching in Bipartite Graphs - string matching : Naive String matching Algorith Knuth Morris Pratt Algorithm - Rabin Karp Algorithm.

(From the mean Video)

Topic 3: Maximum Bipartite Matching graph problem.

Definition:-

Bipartite graph is a graph $G = (U \cup V, E)$ where the vertex set is divided into two disjoint sets U and V, and every edge connects a vertex in U to one in V.

Matching:-

A set of edges with no two edges sharing a common ve

Maximum Matching:-

A matching that contains the largest possible number of edges. (No two edge share a vertex).

Applications:-

⇒ Job Assignment (workers & Jobs)
⇒ Matching students to schools.
⇒ N/w flow problems.

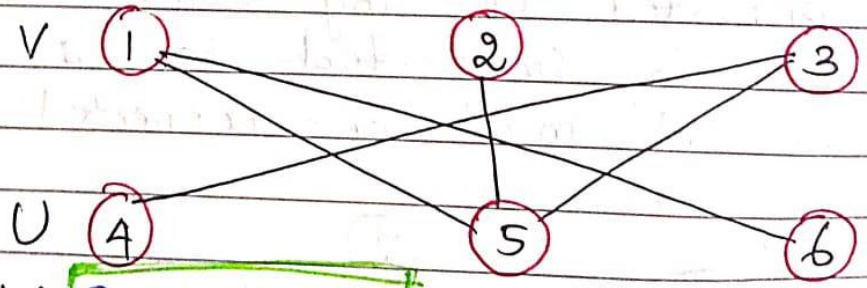
Example:

Given 2 set $V = \{1, 2, 3\}$
$U = \{4, 5, 6\}$

Condition:

Each vertex in 'U' set must be connected only with the one vertex in 'V' set.

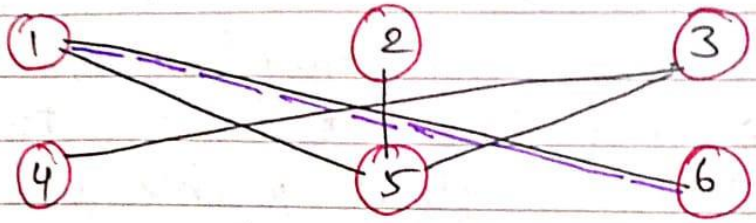
Graph:



Step 1: Queue 1, 2

It is "not necessary" that number of vertices in 'U' set and number of vertices in 'V' set must be equal.

Connect any vertex in set V with any one vertex in set U. Also we must check if that vertex is only connected to this vertex

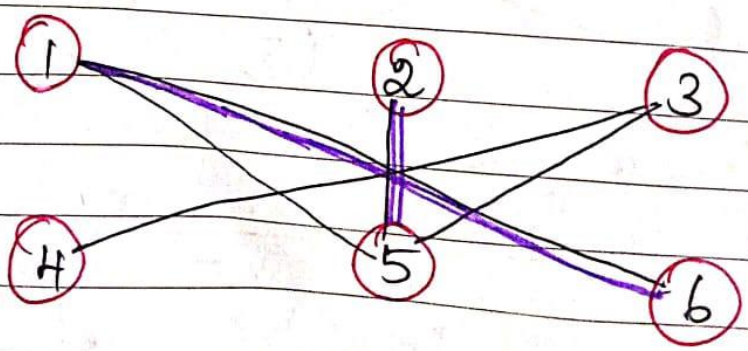


Now '1' vertex is connected to 2 & 6. But 5 is also connected to '2'. So we must consider it only.

Step 2: Queue 2, 3

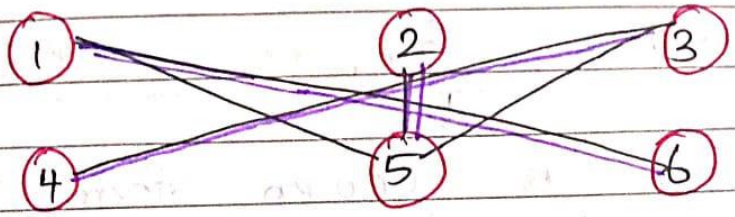
Now '2' vertex is connected to only. But '5' vertex is connected to 3 also. So check vertex '3' connected to any other vertex in It is connected to '4'.

So '2' must be connected to '5'



Step 3: Queue 3

Vertex '3' is connected to '4' & '5' vertex. But '5' is already augmented with '2'. So we have connect 3 and 4 vertices.




```
Algorithm maxBipartiteMatch()
{
result = 0;
memset (match, -1, sizeof (match));
for (v=0; v<n; v++)
{ memset (seen, 0, sizeof (seen));
if (findaug (v))
result ++;
}
return result;
}
```

Time complexity:
O(N * m)
Where N = Nodes in U
M = Nodes in V

Space Complexity
O (N X m)
(for Adjacency list)
O (N + m + E)
(for Sparse graph)