**23**

AC: Given a graph G, and an integer k, decide if G contains a set V' of k vertices such that, every vertex in V' has no neighbor in V' (in other words, if u and v belong to V' then there cannot be an edge between them).

ACX: Given a graph G, and an integer k, decide if G contains a set V' of at least k vertices such that, every vertex in V' has at most one neighbor in V'.

**AC** is a known NP-Hard problem: Given a graph G(V, E), return 1 if and only if, it contains a set V' of k vertices such that, every vertex in V' has no neighbor in V' (in other words, if u and v belong to V' then there cannot be an edge between them).

To prove: **ACX** is **NP-Hard**

**AIM:** AC <= ACX

**Prove the Problem is NP-Hard i.e HEMPATH <= ThinMST**

Consider the following algorithm.

def **reduce(G, k)**:

// V represents the vertices of graph G

H = copy of G

for every vertex v in graph G:

Create new vertex u in H

Add edge (u, v) in H

knew = k + | V |

return H, knew

**Complexity analysis Reduce:**

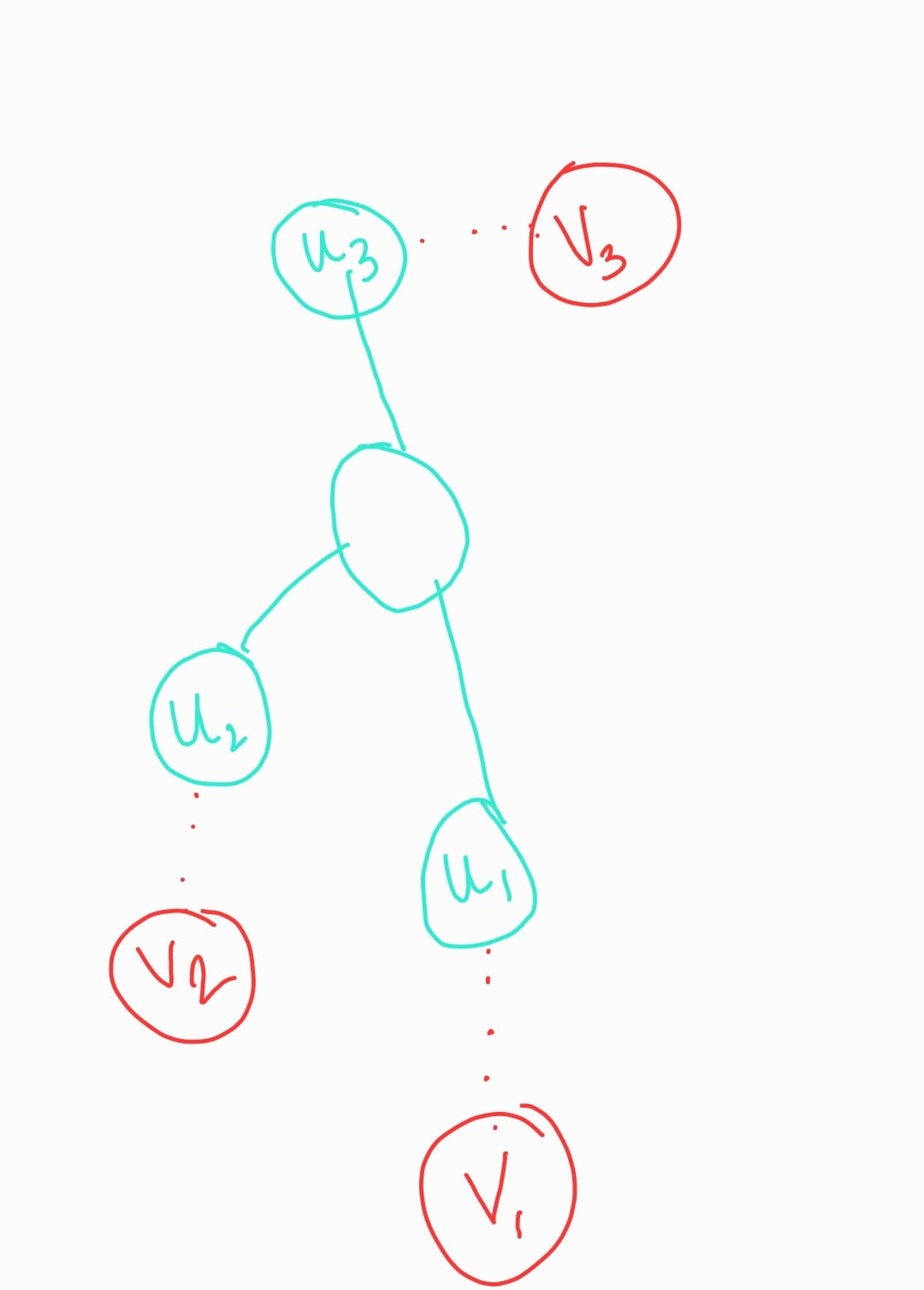
The reduction algo first creates a copy of G in linear time of G i.e. takes **O(V+E)** time.

Adding an additional vertex in H for every vertex in G takes **O(V+E) \* O(1)** time.

Hence, the entire reduction algorithm takes ***O(V+E)*** time i.e. it's a poly-time reduction.

**Statement of correctness lemma for the reduction:**

*Given an undirected graph (****G,k), AC*** *returns* ***TRUE*** *for the original graph* ***G*** *(i.e. there exists a set V' of k vertices in G such that, every vertex in V' has no neighbor in V'), iff* **ACX** *returns* ***TRUE*** *for the reduced graph (****H, knew)*** *(i.e. there is set V'’ of at least k vertices in H such that, every vertex in V'’ has at most one neighbor in V'’)****. [H,knew=reduce(G,k)]***



|V’| = {u1, u2, .. , un}

{v1, v2, .. , vn} are added to respective {u1, u2, .. , un} as shown above.

**Forward proof(=>):**

Say graph **G** has set V' of k vertices such that every vertex in V' has no neighbor in V'**.** The reduction step will set knew=k + | V |. Let this reduced graph be **H, knew i.e. H,knew=reduce(G, k). & |V'| = k.**

When **H** is given as input to **ACX,** it will consider all the earlier elements in V' for V'' (as they have no neighbor between them) & also the new |V| vertices that were added in the reduction step as they are connected through exactly 1 edge to the vertices in V'. Hence, the newly added vertices can have at most one neighbor in V''.

So at max |V''| = |V'| + |V|, Where |V’| = {u1, u2, .. , un} & |V’’| = {u1, u2, .. , un} + {v1, v2, .. , vn}

Since **|V'| = k , then** |V''| = |V'| + |V| = k + |V| = knew

Hence there exists a set V’’ with cardinal number = knew such that each every vertex in V'’ has at most one neighbor in V'’.

Hence ACX will also return TRUE.

**Backward proof(<=):**

If y is a “TRUE” instance of **ACX**, then x is a “TRUE” instance of ***AC.***

***OR,*** *we can prove its contrapositive statement:*

If x is a “FALSE” instance of ***AC***, then y will also be a “FALSE” instance of **ACX*.***

Say graph **G** does not have any independent set of size k **i.e. G is a FALSE** instance of ***AC.***. It is only possible that it has an independent set whose size is less than k. (otherwise **AC** would have returned **TRUE.)**

**Hence,**  |V’| < k i.e. the independent set in G has less number of elements than k. It is trivial to observe that if nothing else, there will always be an independent set for k=1.

Now the reduction step will reduce the graph G into H & will set knew = k + |V|.

When **H** is given as input to **ACX,** it will consider all the earlier elements in V' for V'' (as they have no neighbor between them) & also the new |V| vertices that were added in the reduction step as they are connected through exactly 1 edge to the vertices in V'. Hence, the newly added vertices can have at most one neighbor in V''.

So at max |V''| = |V'| + |V|

|V’| = {u1, u2, .. , un} & |V’’| = {u1, u2, .. , un} + {v1, v2, .. , vn}

But since |V'| < k, so |V''| will also be less than knew. Thus we can clearly notice that in **H,** there will not be such a set with cardinality knew.

Hence, ***ACX*** will also return **FALSE**.

Hence, the above reduction algorithm is a polynomial-time reduction from **AC** to **ACX.**

From the above statements we can conclude that NP-hard problem( **AC** ) can be reduced into ACX in polynomial time, hence the given problem **ACX** is **NP HARD.**