**20(a)**

The **UHAMPATH** problem takes as input an undirected graph and asks if there is a Hamiltonian path in the graph. A Hamiltonian path must visit every vertex exactly once (and have distinct start and end vertices).

The **UHPBUT1** problem takes as input an undirected graph and asks if there is a path that visits every vertex exactly once except one vertex (which is not visited).

Aim: **UHAMPATH** <= **UHPBUT1**

Consider the following algorithm.

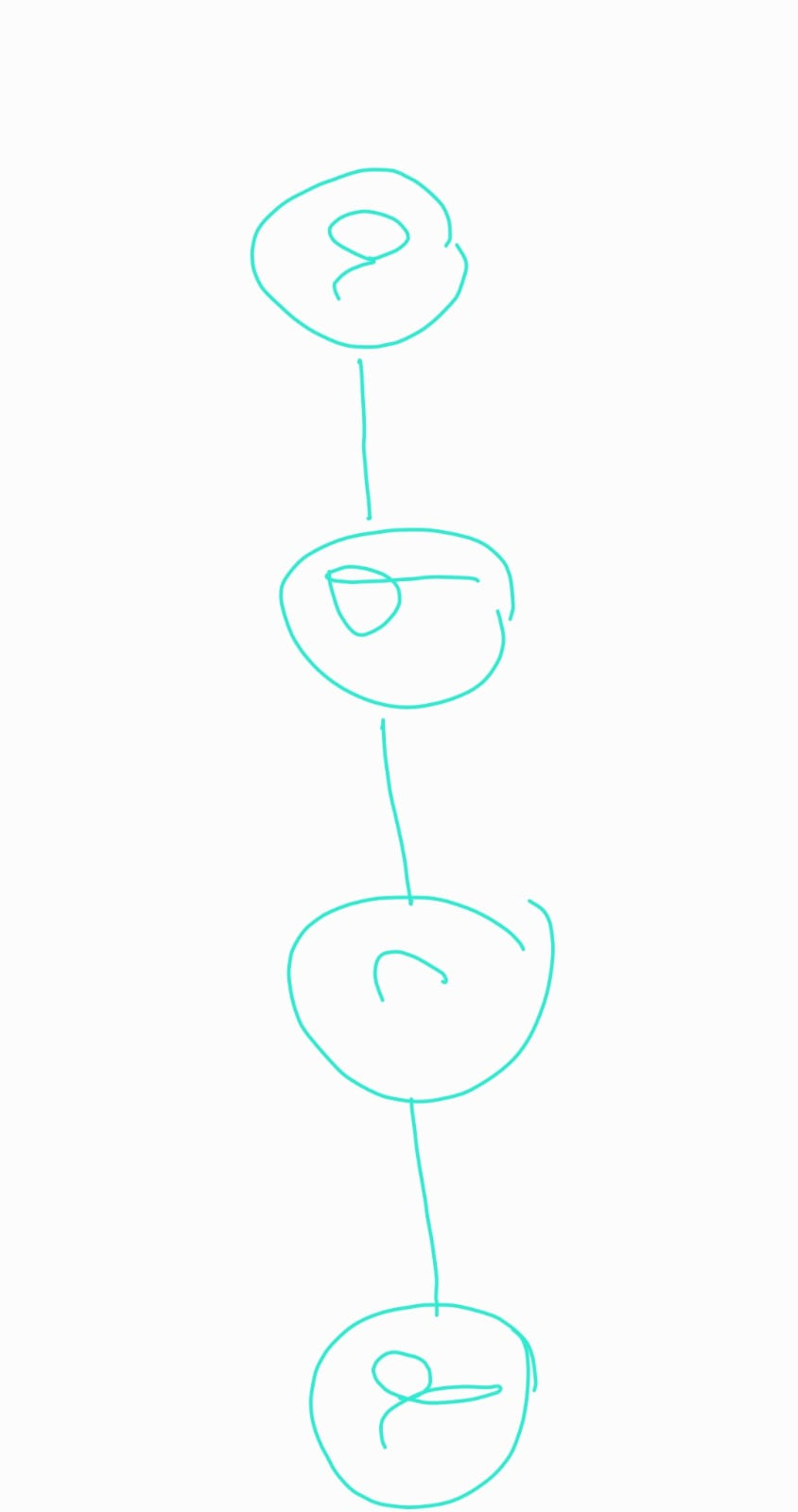
def **reduce(G)**:

1. Find the vertex with the highest number of nodes in G (if there are more than one such vertex, choose anyone.)
2. Let **v** be one such vertex, add another vertex to that graph with the edge **v-x.**
3. Return the transformed graph as **H.**

**Examples:**

* ***YES*** *instance of* ***UHAMPATH*** *and the output of the reduction on this instance :*

Let the input be a graph **G:**

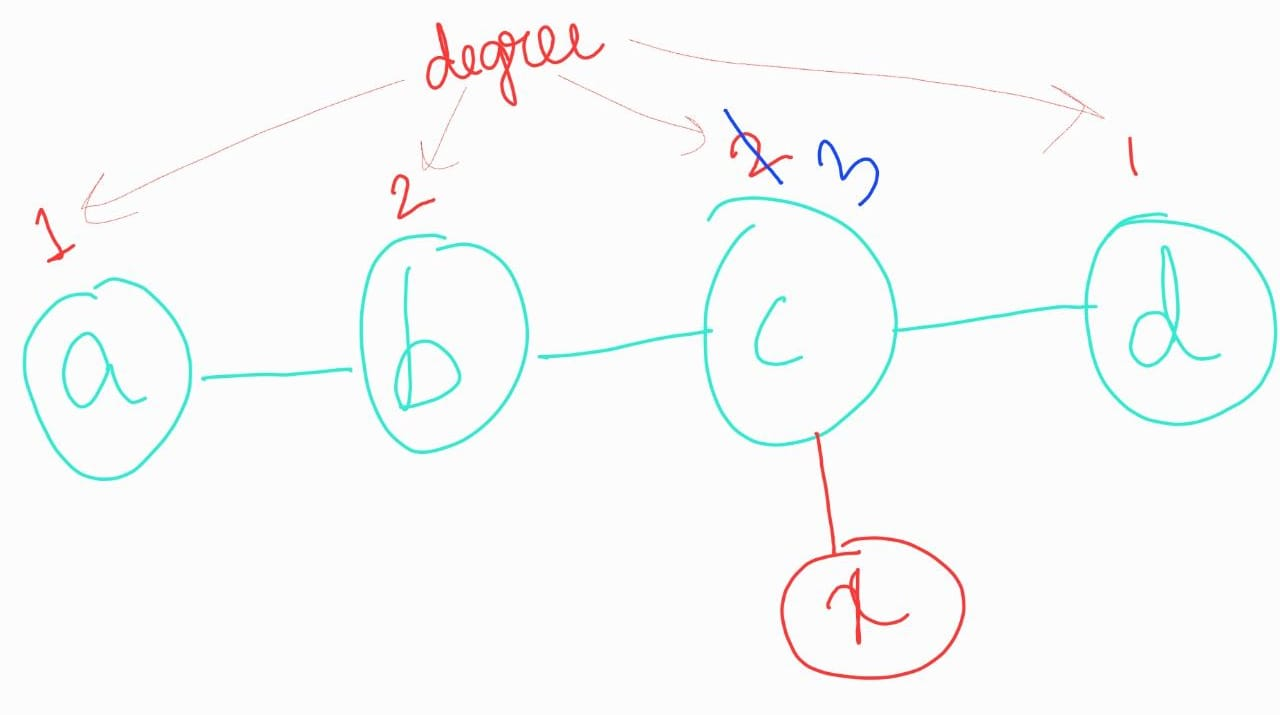


**UHAMPATH** will return TRUE as there is a Hamiltonian path in **G:** (a-b,b-c,c-d)

There are 2 vertices (b & c) with highest degree=2.

Add a vertex x with an edge to any one of them. I have added edge **c-x.**

The output of the reduction will be **H:**

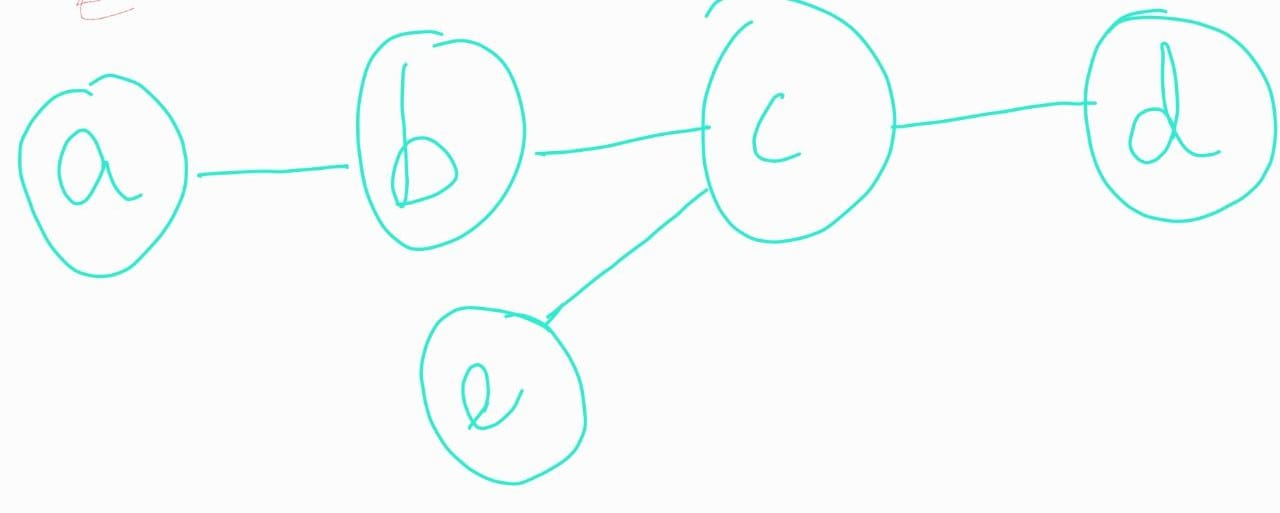
****

**UHPBUT1** will return **TRUE** on **H. ( i.e.** there is a path that visits every vertex

exactly once except one vertex **x** (which is not visited).**)**

* ***NO*** *instance of* ***UHAMPATH*** *and the output of the reduction on this instance :*

Let the input be a graph **G:**

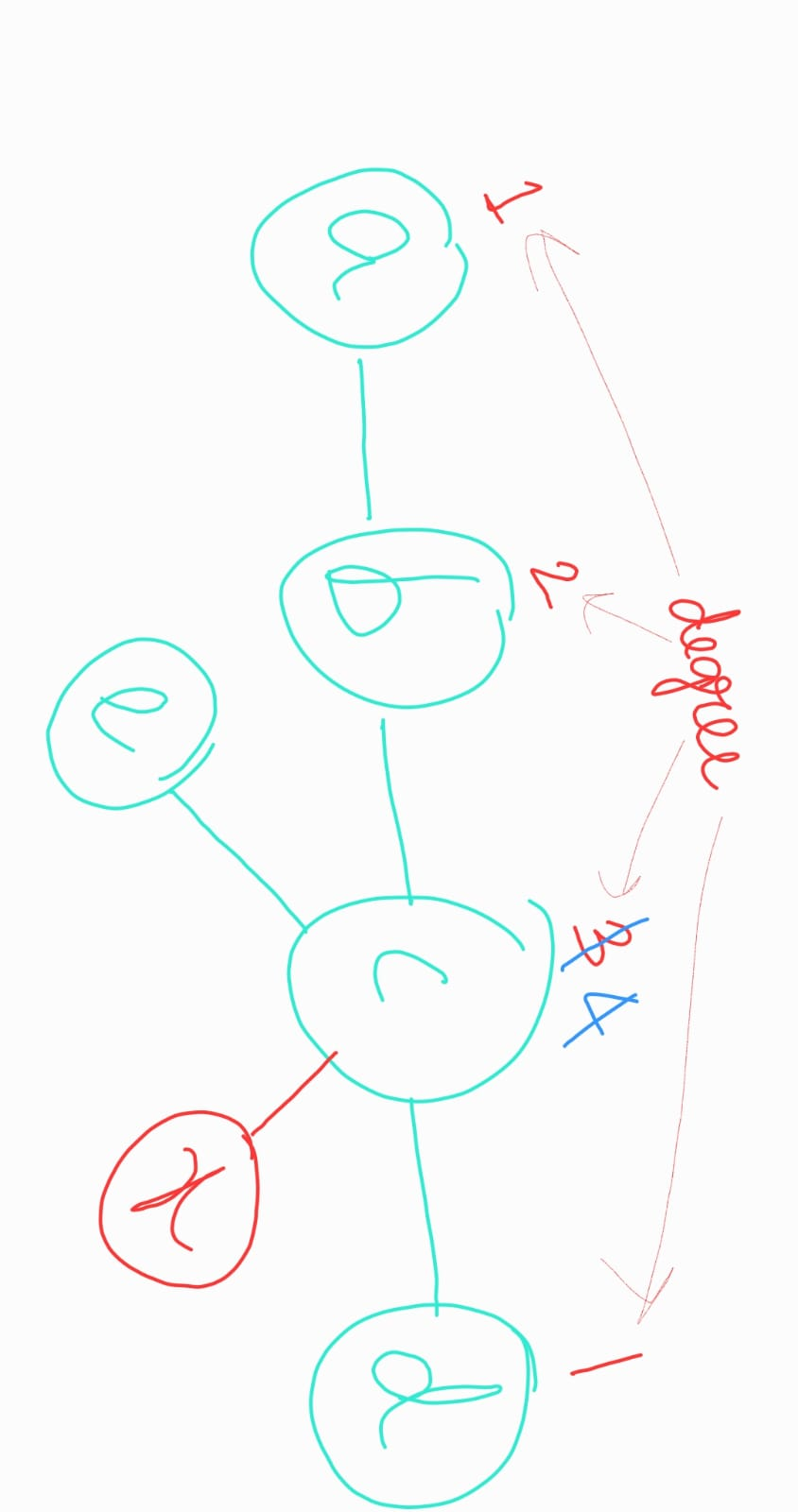


**UHAMPATH** will return **FALSE** as there is no Hamiltonian path in **G.**

There is 1 vertex (c) with highest degree=3.

Add a vertex x with an edge **c-x.**

The output of the reduction will be **H:**

****

**UHPBUT1** will return **FALSE** on **H. ( i.e.** there is no path that visits every vertex

exactly once except one vertex (which is not visited).**)**

**Complexity analysis:**

Line **(1)** in the reduction step takes **O(V2)** time as you need to traverse the entire adjacency matrix to find the vertex with the highest degree. (i.e. traverse the adjacency matrix & return the vertex with the highest row sum or highest column sum).

*[If we assume that number of edges* ***E=O(V2), then*** *this step would take* ***O(V+E)*** *time****.]***

Line **(1) & (2)** takes constant time.

Hence, the reduction takes ***O(V2)*** time i.e. its a poly-time reduction.

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

*Given an undirected graph* ***G, UHAMPATH*** *returns* ***TRUE*** *for the original graph* ***G****(i.e. there is a Hamiltonian path in* ***G****), iff* ***UHPBUT1*** *returns* ***TRUE*** *for the reduced graph* ***H****(i.e. there is a path in* ***H*** *that visits every vertex exactly once except one vertex (which is not visited)****. [H=reduce(G)]***

**Correctness of the reduction:**

In other words, say the reduce algorithm is converting an instance x of ***UHAMPATH*** into a special

instance y of ***UHPBUT1***, then we need to prove:

* (=>) If x is a “TRUE” instance of ***UHAMPATH***, then y is a “TRUE” instance of ***UHPBUT1***.
* (<=) If y is a “TRUE” instance of ***UHPBUT1***, then x is a “TRUE” instance of ***UHAMPATH.***

**Forward proof(=>):**

Say graph **G** has some Hamiltonian path **P.** The reduction step will add one extra vertex (say **x)** to one of the vertex **v** of **G** which has the highest degree. Thus, there is an extra edge **(v-x)**  in **G**. Let this graph be **H i.e. H=reduce(G).**

When **H** is given as input to **UHPBUT1**, we can clearly notice that even if we do not visit vertex **x,** still there would be a hamiltonian path in **H, &** this path would be exactly the same path as the original hamiltonian path **P** in **G.** Thus, **UHPBUT1** will also return **TRUE**.

**Backward proof(<=):**

If y is a “TRUE” instance of ***UHPBUT1***, then x is a “TRUE” instance of ***UHAMPATH.***

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

If x is a “FALSE” instance of ***UHAMPATH***, then y will also be a “FALSE” instance of ***UHPBUT1.***

Say graph **G** does not have any Hamiltonian path**, i.e. G is a FALSE** instance of ***UHAMPATH.*** Then there must be at least one vertex **k** in **G,** which could not be traversed in the pursuit of finding a Hamiltonian path.

Let some vertex **v** of **G** have the highest degree. The reduction step will add another extra vertex (say **x)** to **v.** Thus, there is an extra edge **(v-x)** in **G**. Let this graph be **H i.e. H=reduce(G).**

Thus we can clearly notice that in **H,** now there are at least 2 vertices **(x & k),** which will not be traversed in the pursuit of finding any Hamiltonian path.

When **H** is given as input to **UHPBUT1**, even if it excludes one vertex among **x or k** while finding the Hamiltonian path**,** there would still be another vertex left to traverse. Thus, there will not be any hamiltonian path in **H.** Hence, **UHPBUT1** will also return **FALSE**.

Hence, the above reduction algorithm is a poly-time reduction from **UHAMPATH** to **UHPBUT1.**

**20(b).**

**Correctness of the reduction:**

The given reduce algorithm is incorrect. Had it been a correct reduction, then the following lemma should hold:

*Given an undirected graph* ***G, UHPBUT1*** *returns* ***TRUE*** *for the original graph* ***G****(i.e. there is a path that visits every vertex exactly once except one vertex (which is not visited), iff* ***UHAMPATH*** *returns* ***TRUE*** *for the reduced graph* ***H*** *(i.e. there is a Hamiltonian path in the reduced graph* ***H****).* ***[H=reduce(G)]***

In other words, say the reduce algorithm is converting an instance x of ***UHPBUT1*** into a special

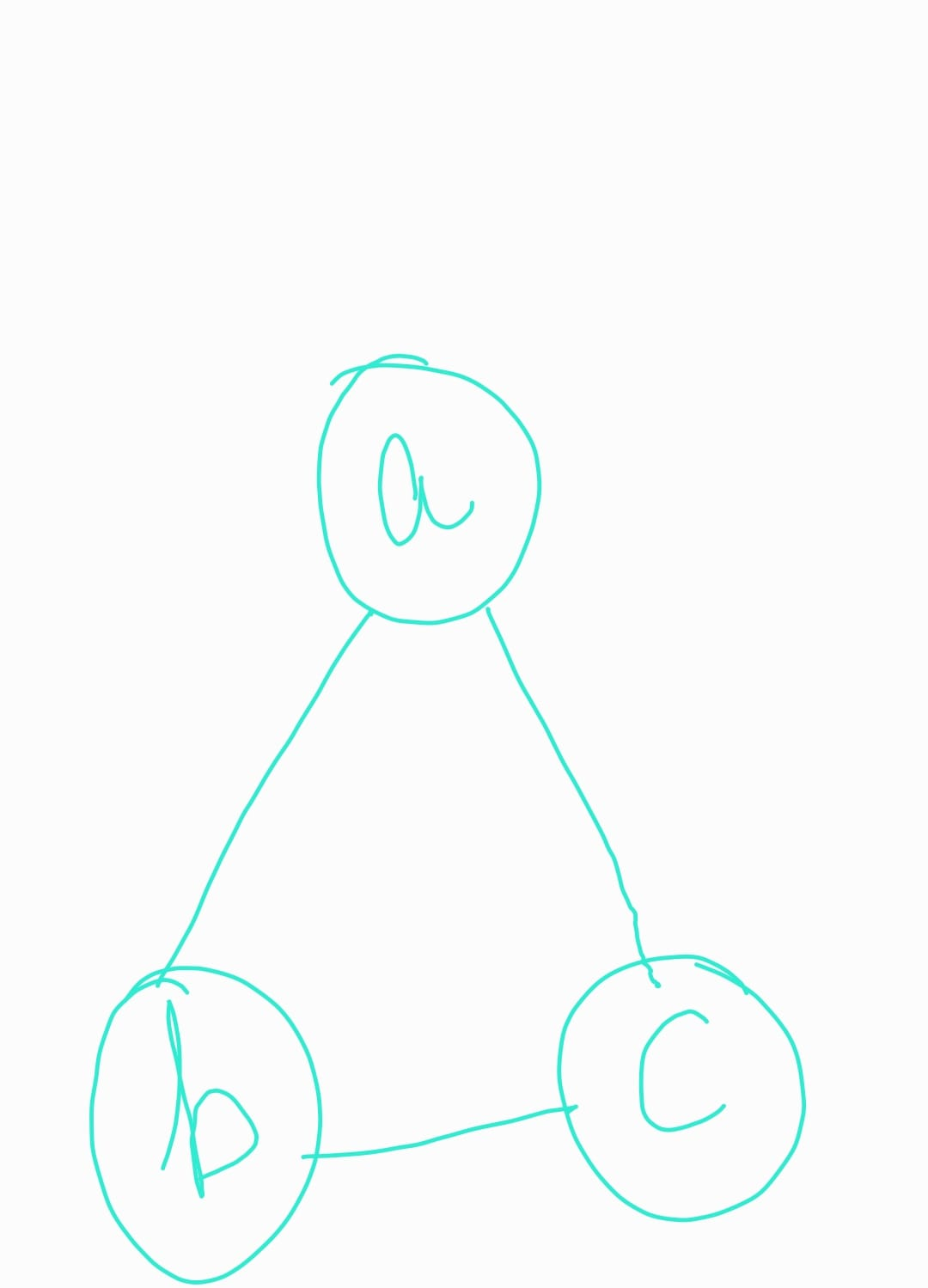
instance y of ***UHAMPATH***, then we need to prove:

* (=>) If x is a “TRUE” instance of ***UHPBUT1***, then y is a “TRUE” instance of ***UHAMPATH***.
* (<=) If y is a “TRUE” instance of ***UHAMPATH***, then x is a “TRUE” instance of ***UHPBUT1.***

**Let's check the backward proof (<=) :**

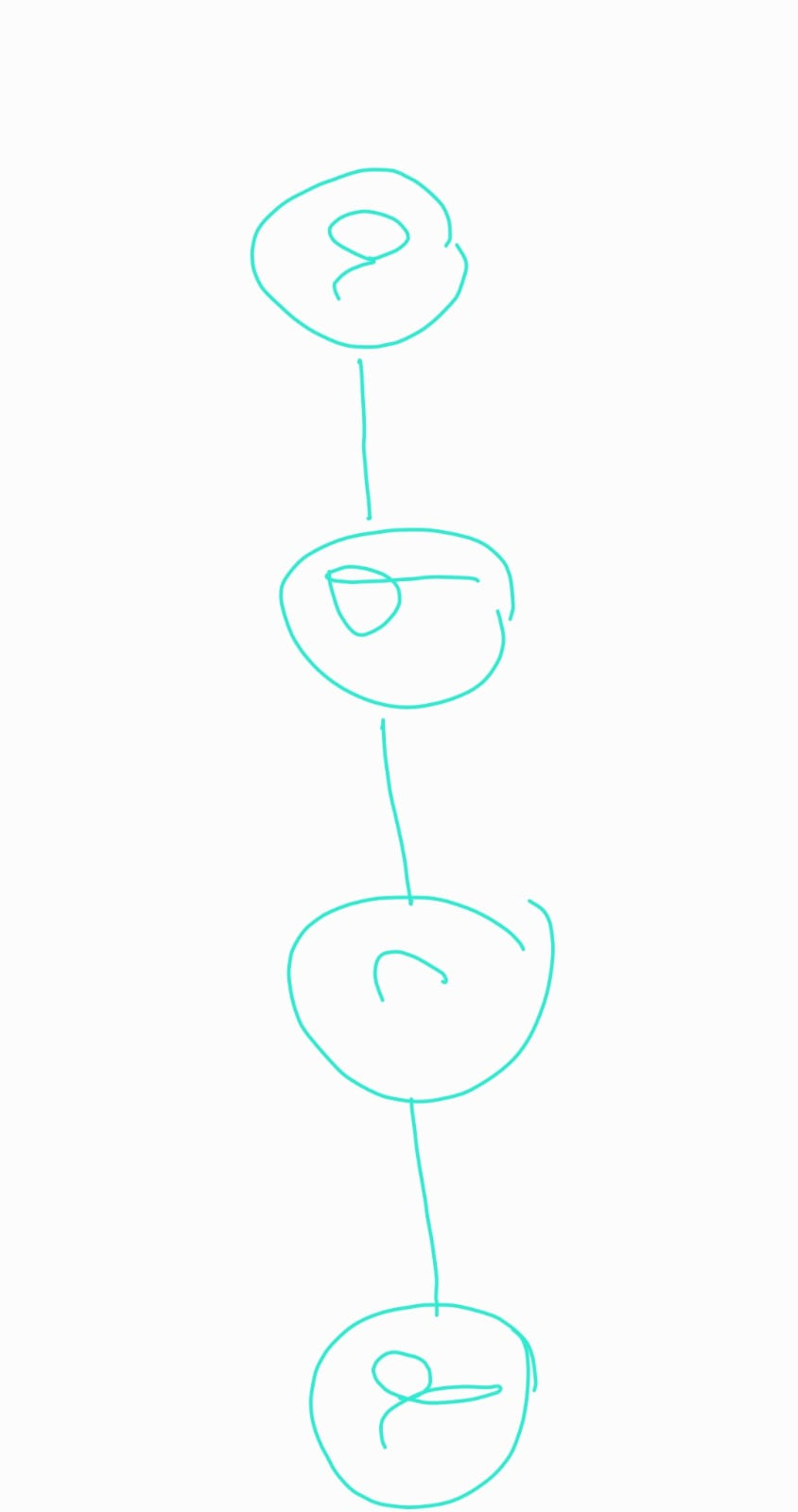
Observe the 2 return statements of the reduction algorithm:

* return a triangle graph (with three vertices)



**Hamiltonian Path:** a-b, b-c

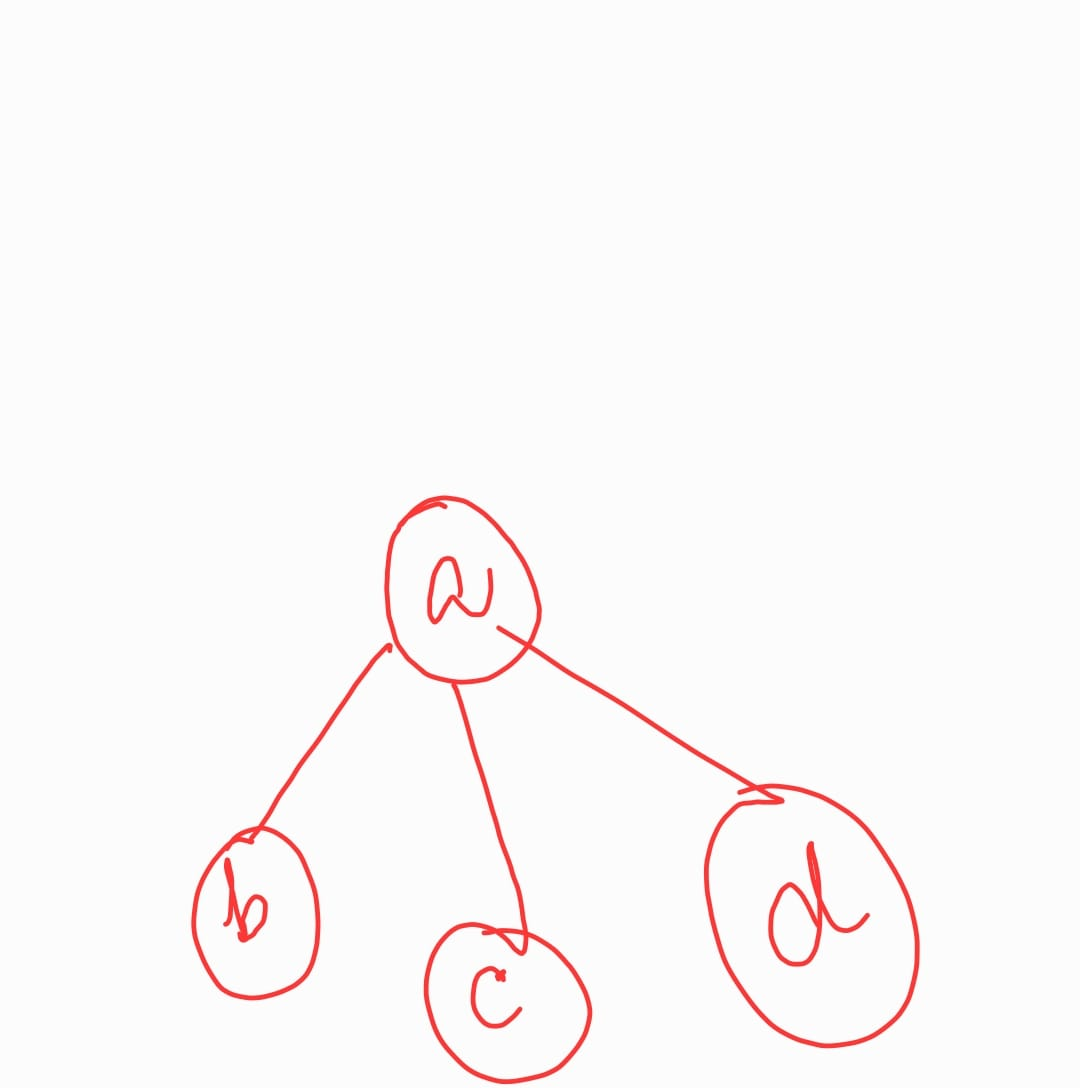
* return the following 4 vertex graph with 3 edges : {a-b, b-c, c-d}.



**Hamiltonian Path:** a-b, b-c, c-d

Both the above cases are **TRUE** instances of ***UHAMPATH*** i.e. in both cases there is a Hamiltonian

path in the graph. Hence the reduction algorithm will always return **TRUE,** irrespective of the input instance x of ***UHPBUT1*.** But there can be **FALSE** instances of ***UHPBUT1,*** *such as:*

****

Althoughthis is a **FALSE** instance of ***UHPBUT1***, yet the reduction will return a **TRUE** instance of ***UHAMPATH*.**

Hence the given reduction is incorrect.

**Complexity analysis:**

Observe the 2 for loops in the reduction algorithm:

**for** every vertex v in G: …(1)

construct G' = copy of G except v and the edges from v …(2)

**for** every permutation P of the vertices in G': …(3)

check if P forms a valid path, i.e., there is an edge between subsequent pairs of

nodes in P …(4)

Line **(1)** will run for **v** times, while the inner for loop in line **(3)** will run for **(v-1)!** times, as there can be (v-1)! permutation of (v-1) vertices. And line **(4)** will take **O(v-1)** time to check if P forms a valid path.

Hence the time complexity for the reduction step will be **O( (v) \* (v-1)! \* (v-1) ),** which is not polynomial.

Hence reduce is not a polynomial time reduction from **UHPBUT1** to **UHAMPATH**.