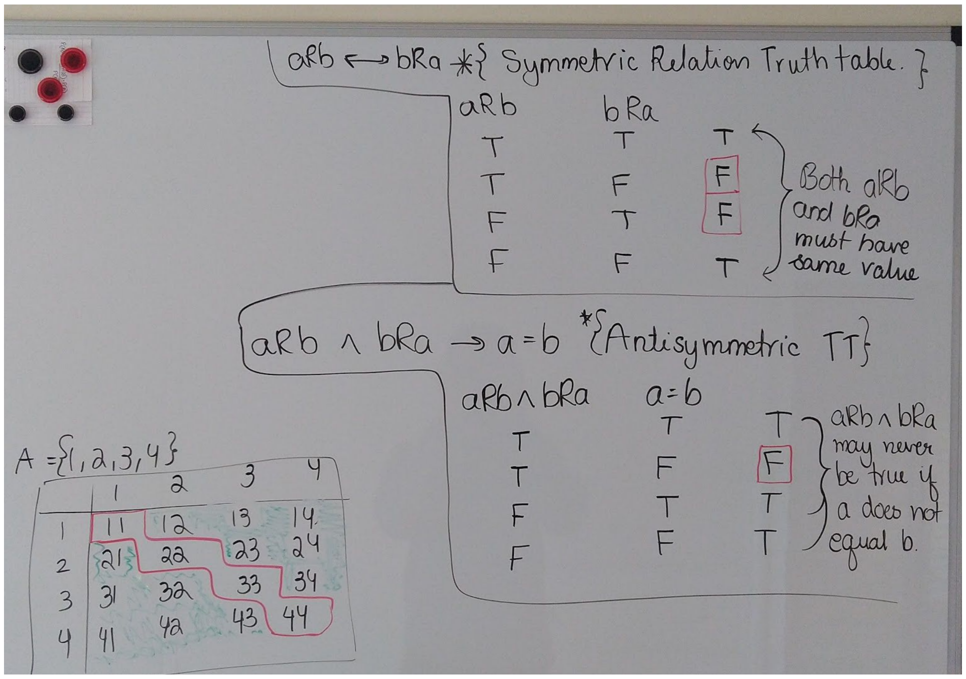
Kev Sharma
Professor Ames
Fall 2020



aRb ⟷ bRa *{ Symmetric Relation Truth table. }

aRb	bRa	
T	T	T
T	F	F
T	T	F
F	F	T

Both aRb and bRa must have same value

aRb ∧ bRa → a=b *{Antisymmetric TT}

aRb∧bRa	a=b	
T	T	T
T	F	F
F	T	T
F	F	T

aRb ∧ bRa may never be true if a does not equal b.

A ={1,2,3,4}

	1	2	3	4
1	11	12	13	14
2	21	22	23	24
3	31	32	33	34
4	41	42	43	44

1.

Let a and b be two values in the set D (I wrote A in picture but it's D). Then the truth tables for symmetry and antisymmetry pose restrictions on how many binary relations we can construct.

- The truth table for Symmetry tells us that for a binary relation to be symmetric, we must ensure that both aRb and bRa must either evaluate to true or evaluate to false {biconditional}.
- The truth table for antisymmetry tells us that for a binary relation to be antisymmetric, if a is equal to b, then aRb = bRa may take on any value. That is, if aRb is false, then if a=b, we know that bRa is also false. This means aRb ^ bRa is false. But because of the implication we know that since a=b, this would still be an antisymmetric binary relation.

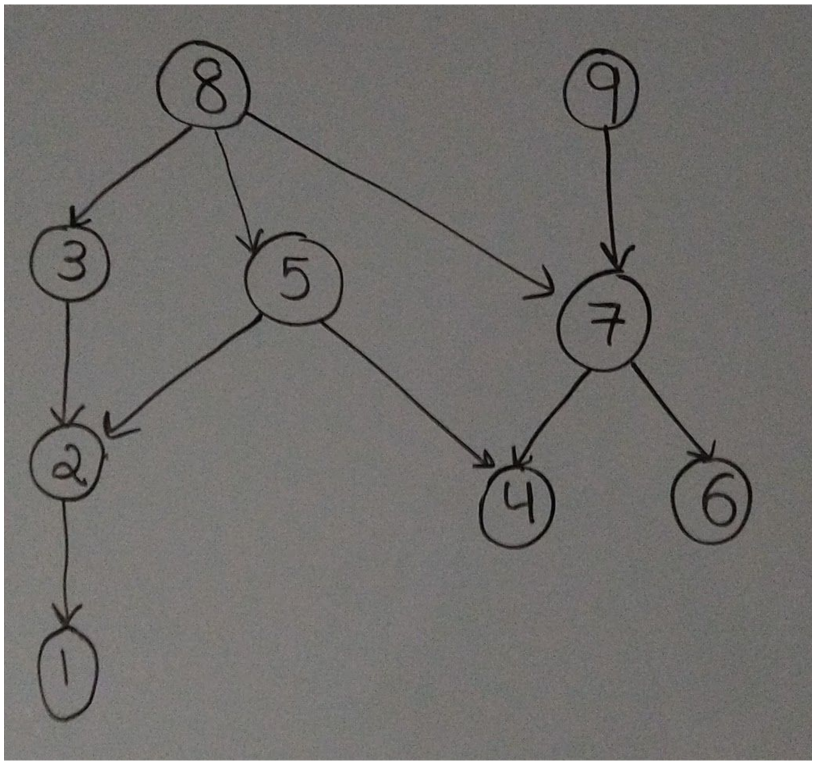
- There are n cells on the diagonal (colored in red), where n = |D| (the image says A, but take it to be D). Note that in each cell, aRb and bRa will evaluate to the same value (either to T or to F but not both) {on a diagonal a = b}.
- The number of symmetric binary relations on the diagonal is exactly 2^n because we can choose to give aRb the value of T or F and its corresponding bRa will have the same value since a = b. This satisfied the condition for a symmetrical binary relation {biconditional}.
- The number of antisymmetric binary relations on the diagonal is also exactly 2^n because given a = b for each cell in the diagonal, aRb ^ bRa may evaluate to either false or true, based on what aRb evaluates to (T^T = T or F^F = F). However due to the implication, since a =b is true for the cells on the diagonal, we satisfy the condition for an asymmetrical binary relation.

2. a) Maximum length increasing subsequences of S: (1,2,3,8), (1,2,5,8)
Maximum length dec. subseq of S: (6,4,1), (6,4,2), (6,4,3), (6,5,3), (7,5,3), (9,5,3)

b) Note that the partial order graph we have drawn is a directed acyclic graph. A minimum node in a DAG is one that can reach every other node. There are no minimum nodes in this DAG (8 cannot reach 9 because < and 9 cannot reach 8 because $<_s$). A minimal node is one that is not reachable by any other node (Ames week12-slides.pdf). In this graph the 2 minimal nodes are 8 and 9.

Maximal elements: no element q such that for p p < q and p$<_s$q. Such elements(p) are 8, 9.
Minimal element:no element q such that for p q < p and q $<_s$ p. Such elements (p) are 1, 4, 6.

Please ignore arrow signs.



c) A chain is a sequence of connected nodes (Ames). For example 8,3,2,1 is a chain (left most nodes). By our definition of this partial order, a chain happens to be an increasing subsequence of S. And the chain with the most nodes happens to be the maximum-length increasing subsequence of S. For example, 8,3,2,1 is the longest chain, as is, 8,5,2,1. These chains have 4 nodes. And it turns out that these sequences (1,2,3,8) and (1,2,5,8) are in fact the maximum-length increasing subsequences of S (which this Hasse diagram has modelled).

An antichain is a set of nodes with no connections between them (Ames). An example is (7, 5, and 3) {what looks to be the middle level}. This anti chain can be ordered (7,5,3) (in decreasing order) to represent a decreasing subsequence of S.

I have done my best to draw the diagram in a horizontally aligned fashion to make it evident that these sets of nodes have no connections.

It is very possible that antichains exist which may not be a decreasing subsequence of S: (3 and 6) is an antichain yet it is not a decreasing subsequence of S based on the way it is written. If we were to write it as (6,3) then it would be a decreasing subsequence of S. So for this particular partial order, if we find an antichain of any given length and write the elements in descending order, we have found a decreasing subsequence of S.

Notice that longest anti-chains *may* form the maximum-length decreasing subsequence of S provided the nodes in the anti-chain are written in descending order. Thus anti-chains *may* form decreasing subsequences of S provided the nodes in the anti-chain are written in descending order.

2d. Prove that every sequence S of length n has an increasing subsequence of length greater than √ n or a decreasing subsequence of length at least √ n.

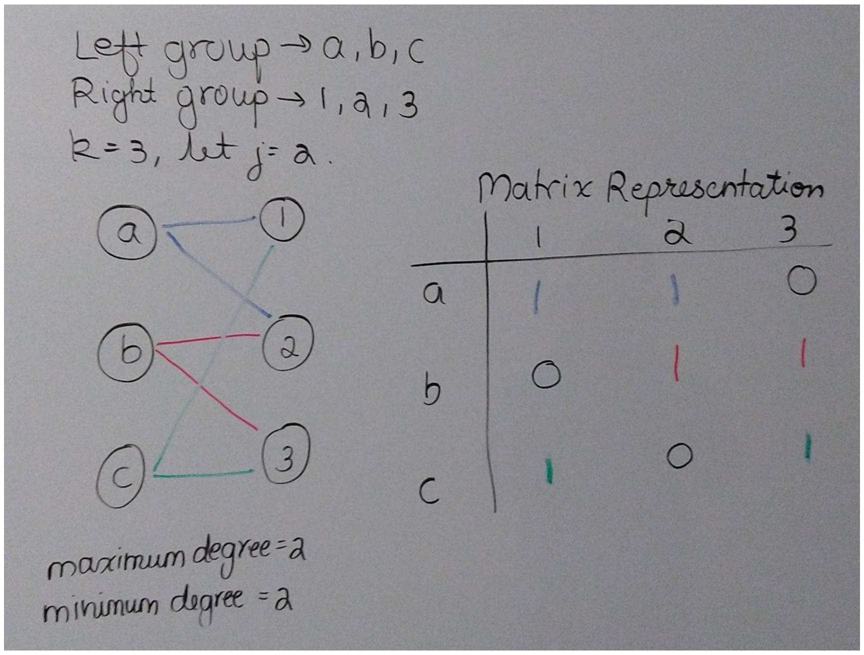
Let me preface this proof with a lemme of a theorem about the set S which in our case is a partially ordered DAG. Dilworth's Lemma tells us that every n-vertex DAG has a chain of size greater than √ n or an antichain of size >= √ n.

Suppose for any sequence of length n, we did not have an increasing subsequence of length greater than √ n. This would mean that we do not have a chain of size greater than √ n. However if this is the case, the Lemma assures us that there would be an antichain in the graph for S of size at least √n. This would directly give us a decreasing subsequence of length at least √ n (defined earlier) and vice-versa.

Aside: I did not want to steal the proof for the theorem, tweak it for my problem, and claim credit for it. This is a topic we did not cover and I was not able to formulate a proof during these trying times. I would like to acknowledge the Lemma I used: https://youtu.be/cUYTlKA8jaw?t=339. I hope this is acceptable use as our Textbook would provide the same material as covered in MIT's OCW lectures.

3. Prove that if G is a balanced graph, then the edges of G can be partitioned into blocks such that each block is a perfect matching.

- We know that G is balanced
    - Premise 1: Hence we know that there are an equal number of left and right vertices.
    - Premise 2: Each vertex, no matter on the left or right, has the same degree.
    - Premise 3: A vertex in the left group must map to only one distinct vertex in the right group for each mapping (that is no two vertices in the left group can map to the same vertex in the right group).
- We must prove that for 2k vertices (k in the left group and k in the right group) and where each vertex has a degree of j, that there are j perfect mappings. Hence we must prove that 'there exists j perfect mappings' .
- First let me define some terminology.
- Consider a simple mapping and its matrix representation where a 1 represents that row vertex (left group) is connected to the column vertex (right group) by an edge in G. There can only be a maximum of k rows and columns and there can only be j number of 1s in each row and in each column.

- 

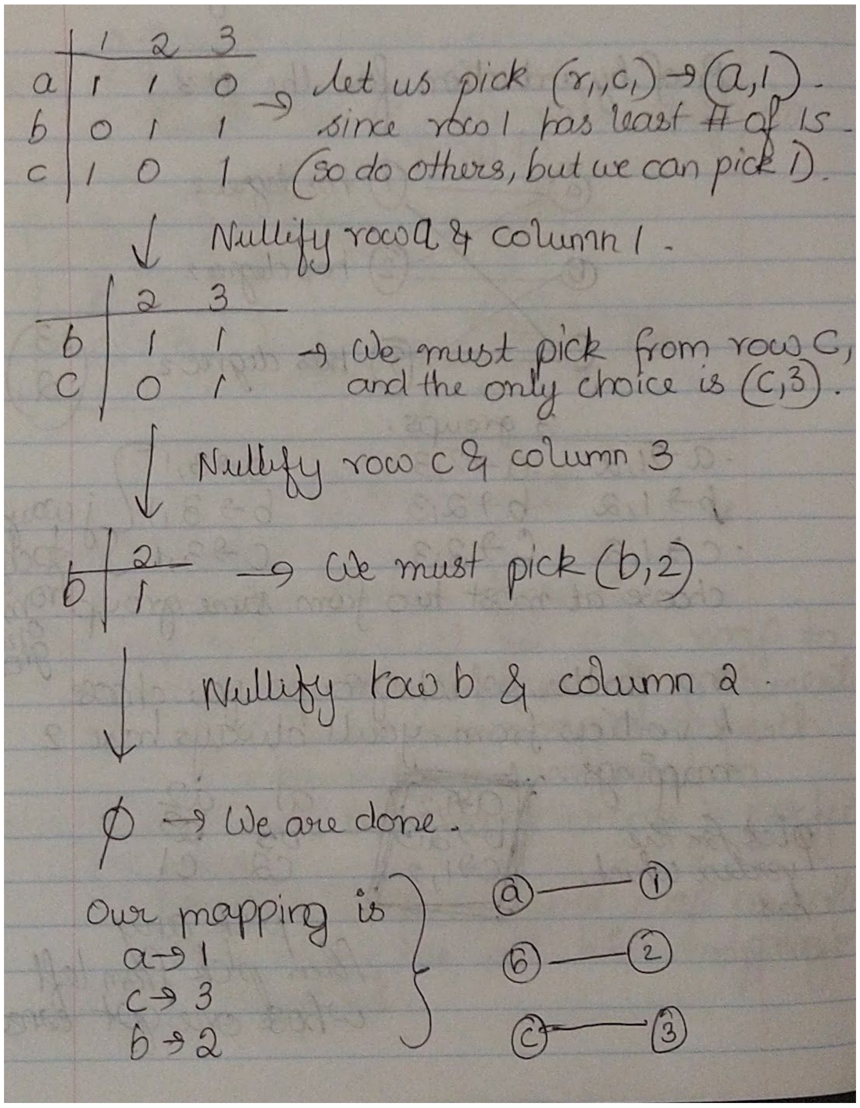
  Left group → a, b, c
  Right group → 1, 2, 3
  k = 3, let j = 2.

  Matrix Representation

  $$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline a & 1 & 1 & 0 \\ b & 0 & 1 & 1 \\ c & 1 & 0 & 1 \end{array}$$

  maximum degree = 2
  minimum degree = 2

- The steps to map are as follows:
    - 0) Pick the first row and choose the first available valid column (a r,c pair that is 1 in the matrix) that was not used as the first column in a previous iteration to get a perfect matching previously.
    - 1) Pick a (row, column) pair such that the cell (row,column) is 1 from the row which has the last least number of 1s. If there are multiple rows which have the least number of 1s, then pick either row.
    - 2) Limit the next (r,c) mapping to include r and c only from the rows and columns which have not been chosen yet (nullify the row and column of a matrix which has been picked). This maintains premise 3 because our next picks may not map an already paired vertex (in left group (row)), or map to an already mapped vertex (in right group (column)).
    - 3) Repeat steps 1 and 2 until no row and column are left.
- Let us see these steps in action:

- 

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline a & 1 & 1 & 0 \\ b & 0 & 1 & 1 \\ c & 1 & 0 & 1 \end{array}$$

→ Let us pick $(r_1, c_1) \rightarrow (a,1)$. since row 1 has least # of 1s. (so do others, but we can pick 1).

↓ Nullify row a & column 1.

$$\begin{array}{c|cc} & 2 & 3 \\ \hline b & 1 & 1 \\ c & 0 & 1 \end{array}$$

→ We must pick from row c, and the only choice is $(c,3)$.

↓ Nullify row c & column 3

$$\begin{array}{c|c} & 2 \\ \hline b & 1 \end{array}$$

→ We must pick $(b,2)$

↓ Nullify row b & column 2.

ϕ → We are done.

Our mapping is
$a \rightarrow 1$
$c \rightarrow 3$
$b \rightarrow 2$

$$\begin{tikzcd} a \arrow[r, dash] & 1 \\ b \arrow[r, dash] & 2 \\ c \arrow[r, dash] & 3 \end{tikzcd}$$

- We could have picked a different r,c to start with and gotten the bipartite graph in the first image (left).

  Continued below...

- Let us define some rules we can come up with using this method:
    - We have a square matrix (Premise 1).
    - Each row and each column start off with exactly j number of 1s (Premise 2).
    - After a (row,column) pairing occurs, we must reduce our matrix's dimension and exclude that row and dimension (Premise 3).

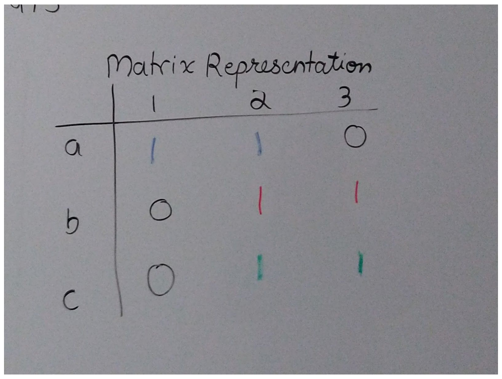
So given these steps and assumptions let us prove that there exist j perfect mappings.

- We will do a proof by contradiction on the statement that "There do not exist j perfect mappings' given our assumptions and premises in the bullet point above.
    - The proposition "there do not exist j perfect mappings" is equal to the proposition "we cannot go through the matrix method j different times" {doing the method j times using different columns starting off implies that we are creating a new block}.

    - Let us list the cases when we cannot go through the matrix method j different times:
        - Case 1: The matrix is not square and thus we cannot complete the steps.
        - Case 2: While completing the method, we get a row which does not have any valid columns to pair with (no 1s to be found for any column for that row).
        - Case 3: There do not exist j different ways to complete this method.

    - Let us prove why each case is a contradiction for any valid and sound matrix representation.
        - Case 1 is a contradiction as the matrix must be square when starting off {Premise 1}. If we did not have a square matrix there would be an unequal number of vertices in the left group as compared with the number of vertices in the right group. This is a contradiction.

        - Case 2 is a contradiction since at no time can we have a row for which each column is a 0. There is at least one 1 in that row's columns always since at any given time, that row becomes the row which has the least 1s and we must pick from that row. Additionally no two rows can, at the same time, have only one 1 for their columns. If this were true, then the previous row,column elimination would have eliminated the 1s in both those second two rows' columns. But by definition this would mean the degree of that column now exceeds the maximum degree assigned to it. For example:

        - 

          Matrix Representation

	1	2	3
a	1	1	0
b	0	1	1
c	0	1	1

          Here we see that we have a square matrix and each row has j number of 1s (as j and k are defined in the first graph). But upon picking a,2, now both b and c only are left with one 1s. However notice that column 2 in this matrix has 3 edges connecting to it. So straight from the start the degree of vertex 2 in the right group exceeded j. Hence this case is a contradiction because it would never happen.

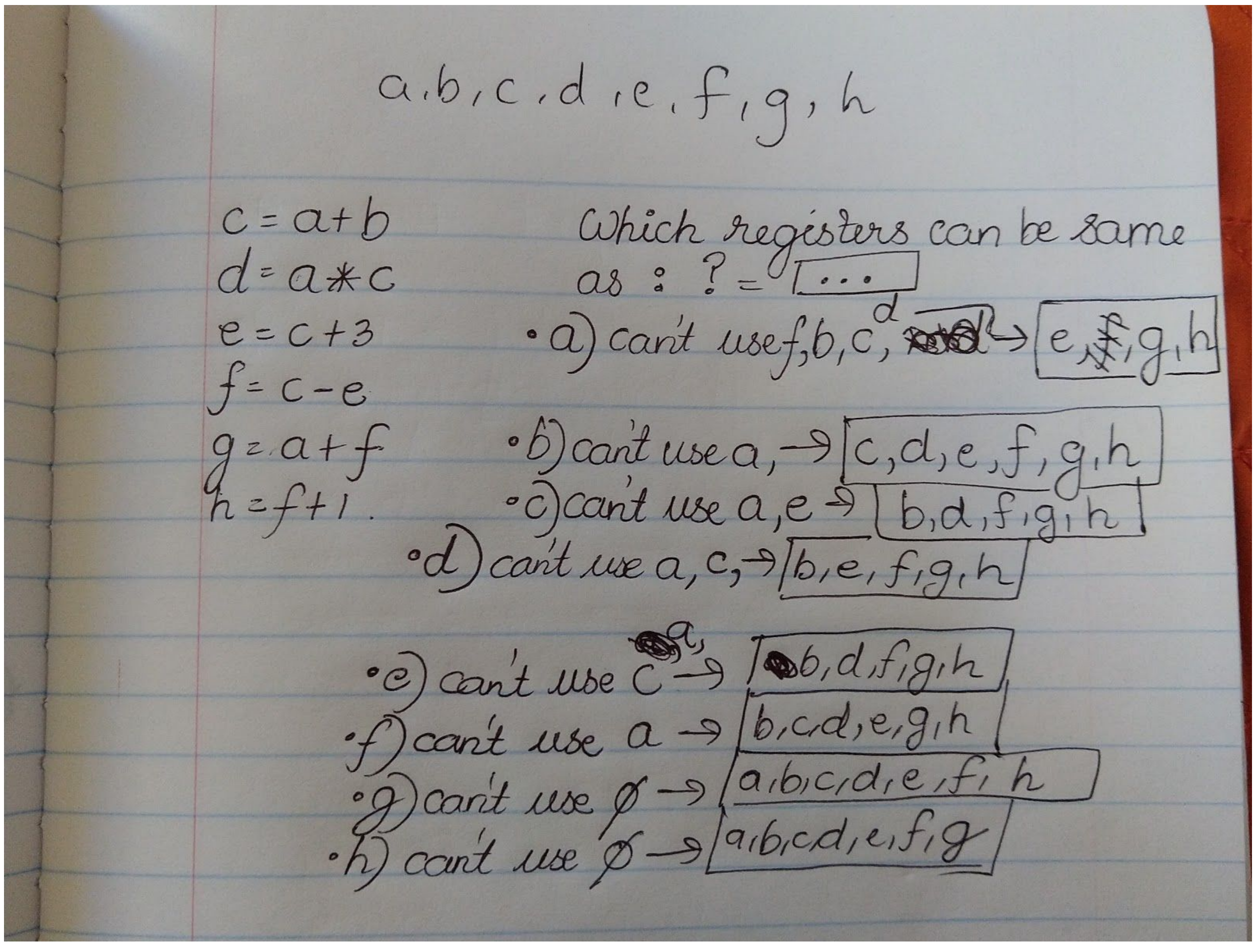
    - Case 3: There do not exist j different ways to complete this method.
        - Suppose this was true. Then condition 0 would be useless. Recall that the matrix must have an equal number of rows and columns and each row must have exactly j columns which are valid mappings. That is, there are exactly j columns in each row such that that row,column cell in the matrix is 1.
        - Every iteration would start with the leftmost 1, then work its way over to the right most 1. Each iteration would have the first row (first vertex v in the left group) map to different columns (v maps to different vertex in right group in each iteration).
        - Since we know each row has exactly j 1s and we know each iteration is different since at least 1 vertex (namely the first) is mapped to something different on each iteration matching, we know that we have j different perfect matches after the j iterations we complete. Hence this case is a contradiction.

- We have proved that the statement "There do not exist j perfect mappings' is contradictory. Hence through a proof by contradiction, we have proved that there exists j perfect mappings.

4a. We know that we wish to use as few registers as possible while avoiding flaws by overwriting a register incorrectly.

Thus what we really would like to do is to model a Graph G using vertices to represent registers (as determined by the number of different values in the equations) and using edges to represent to connect registers that are interdependent. So an edge is not drawn if one of the values in the equation corresponding to a vertex is never used again in those sets of instructions. An edge between two vertices A and B means that we will use B later so if we change A, don't change B.

Let me list each register and for that register, when we may not overwrite that register. The image shows that for any vertex ?, the vertices in the box may not be connected to this one. So the vertices which we may be connected to are connected later on. Recall that connected edges imply that the vertex values on either end are used later in the set of instructions, so we must use different registers for them (i.e different colors in the graph).



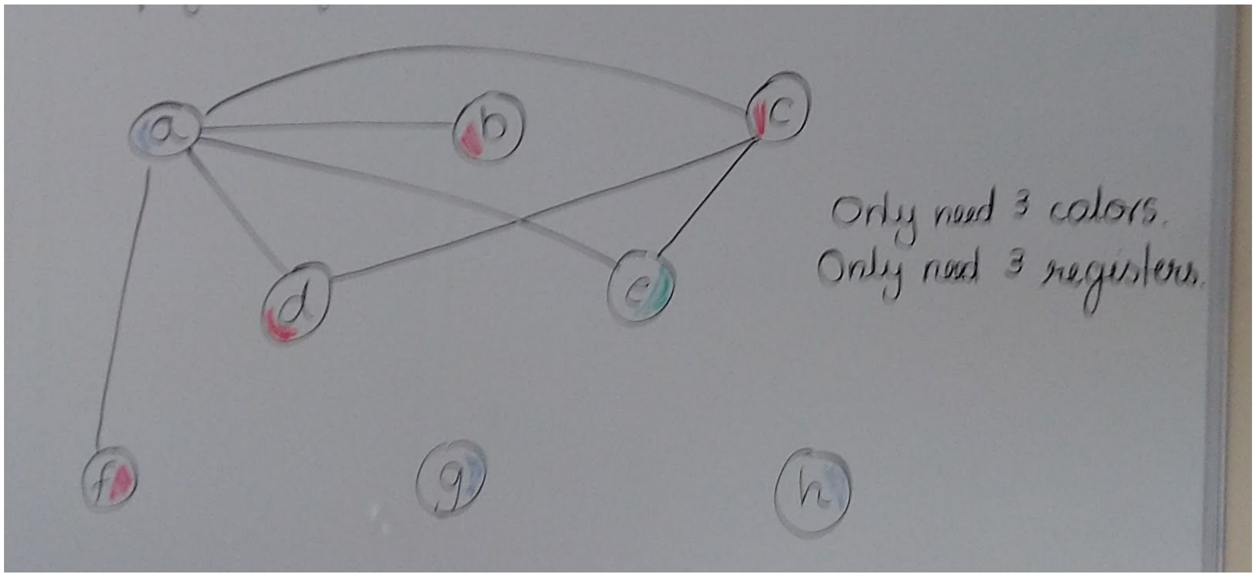
a, b, c, d, e, f, g, h

$c = a+b$
$d = a*c$
$e = c+3$
$f = c-e$
$g = a+f$
$h = f+1.$

Which registers can be same as : ? = $\boxed{\ldots}$

- a) can't use f, b, c, d $\rightarrow \boxed{e, \cancel{f}, g, h}$
- b) can't use a, $\rightarrow \boxed{c, d, e, f, g, h}$
- c) can't use a, e $\rightarrow \boxed{b, d, f, g, h}$
- d) can't use a, c, $\rightarrow \boxed{b, e, f, g, h}$
- e) can't use a, c $\rightarrow \boxed{b, d, f, g, h}$
- f) can't use a $\rightarrow \boxed{b, c, d, e, g, h}$
- g) can't use $\emptyset \rightarrow \boxed{a, b, c, d, e, f, h}$
- h) can't use $\emptyset \rightarrow \boxed{a, b, c, d, e, f, g}$

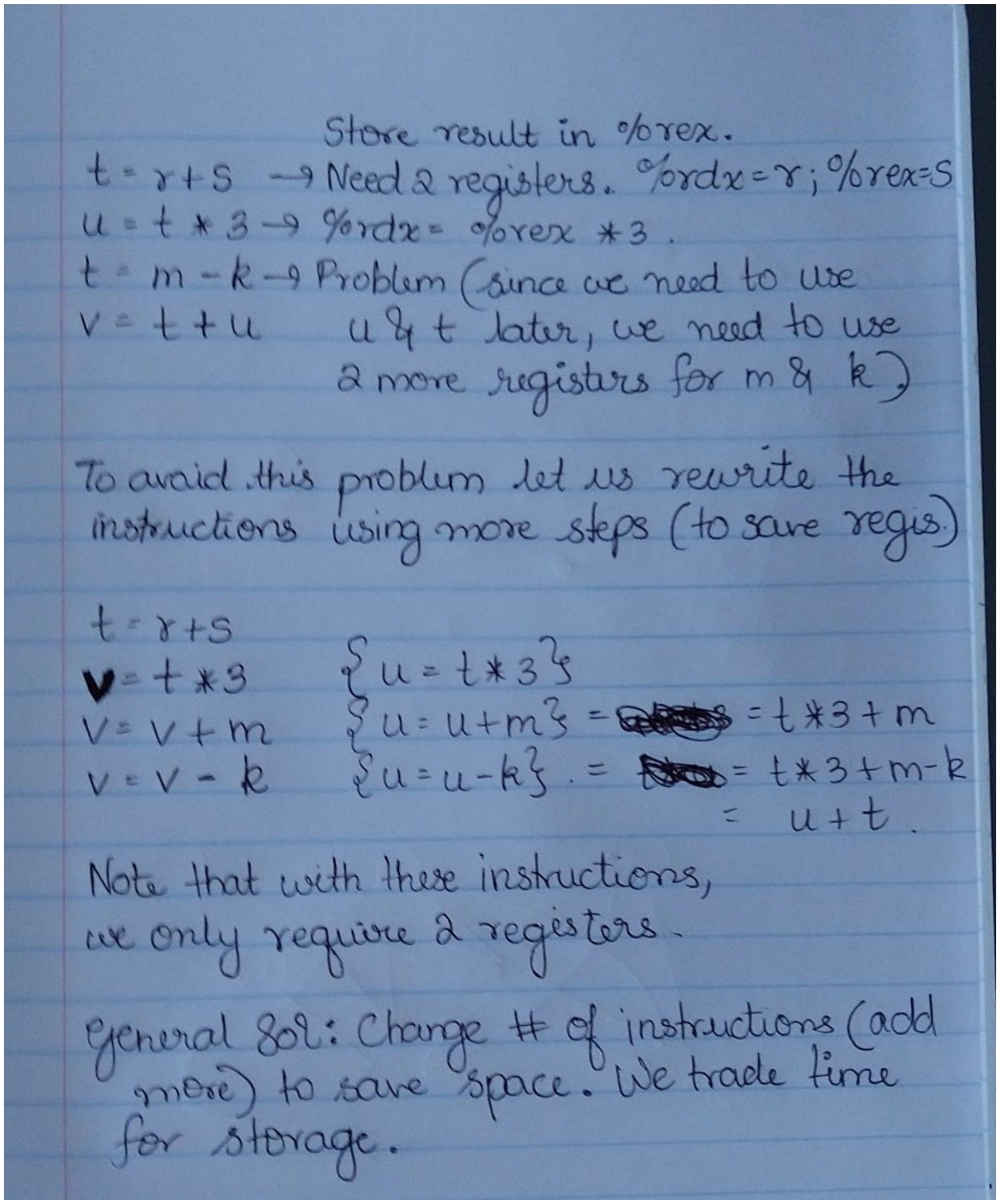
Then we would color G with as few colors as possible (no adj vertices can be the same color) to mimic the compiler optimizing the use of registers (using as few registers as possible by reusing).



4b) Let's take the edge connecting a and c. We know that c will be used later (in instructions 3 and We know that a will be used later in instruction 5. So we must use different registers for them and they must be colored differently.

The maximum number of registers we need are 3 because we have 3 colors. This is evident by observing instruction 2,3,4,5. In instruction 2, we need 2 registers to store a and c. In instruction 3, we need to add a third register to store e, because we must maintain registers for a and c as they are used in instruction 4 and 5. So we may not store value for e in either register a or in c.

Let R1 be the red register. Let R2 be the blue register. Let R3 be the green register. Since vertex with value e is adjacent to both a and c, it needs to be given its own color. Since vertex a and c are adjacent to each other, both must have a separate color. Hence the color assigned to vertex e must be distinct from the colors assigned to vertex a and c.



Store result in %rex.

t = r+s → Need 2 registers. %rdx = r; %rex = S

u = t * 3 → %rdx = %rex * 3.

t = m - k → Problem (since we need to use

v = t + u u & t later, we need to use

2 more registers for m & k)

To avoid this problem let us rewrite the instructions using more steps (to save regis.)

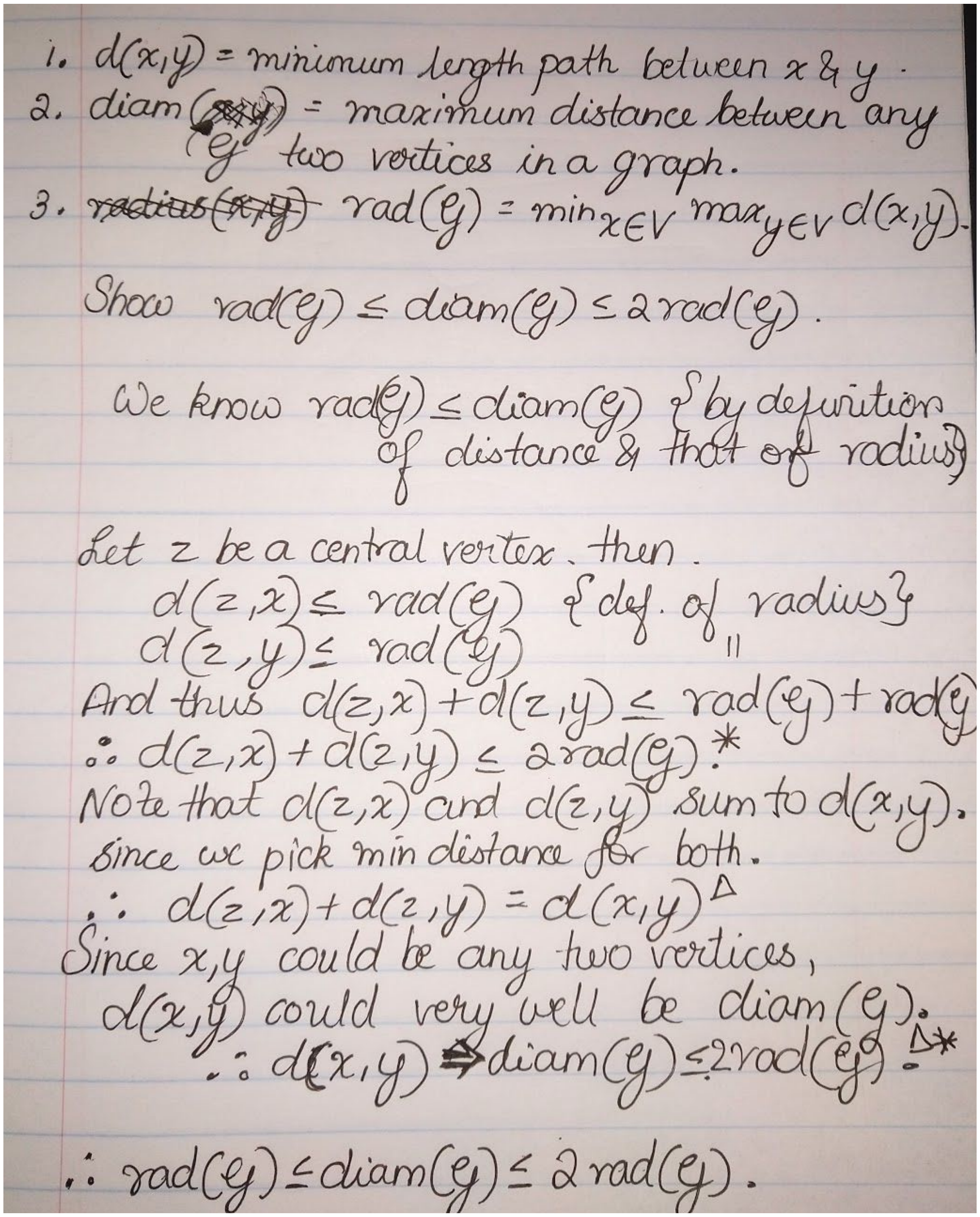
```
t = r+s
v = t*3      {u = t*3}
v = v+m      {u = u+m} = [illegible] = t*3 + m
v = v - k    {u = u-k}. = [illegible] = t*3+m-k
                                      = u + t.
```

Note that with these instructions, we only require 2 registers.

General sol: Change # of instructions (add more) to save space. We trade time for storage.

4c.

Since we know that addition is commutative, we can do it one step at a time. For example x = y + (r + s). Is the same as, x = (y + r) + s. Similarly we can change the instructions to exploit commutativity and associativity. By doing this we can save ourselves from having to use additional registers or god forbid, reading from the cache. In the fix above, we changed instructions to exploit the fact that u was only used once afterwards and we were not concerned with the value of t after setting u. So u + t became (u + m) - k and we were able to do this using just two registers.



1. $d(x,y)$ = minimum length path between x & y.
2. diam(~~x,y~~ $\mathcal{G}$) = maximum distance between any two vertices in a graph.
3. ~~radius(x,y)~~ $rad(\mathcal{G}) = \min_{x \in V} \max_{y \in V} d(x,y)$.

Show $rad(\mathcal{G}) \leq diam(\mathcal{G}) \leq 2\,rad(\mathcal{G})$.

We know $rad(\mathcal{G}) \leq diam(\mathcal{G})$ {by definition of distance & that of radius}

Let z be a central vertex. then.

$d(z,x) \leq rad(\mathcal{G})$ {def. of radius}

$d(z,y) \leq rad(\mathcal{G})$ ″

And thus $d(z,x) + d(z,y) \leq rad(\mathcal{G}) + rad(\mathcal{G})$

$\therefore d(z,x) + d(z,y) \leq 2\,rad(\mathcal{G})^{*}$

Note that $d(z,x)$ and $d(z,y)$ sum to $d(x,y)$. Since we pick min distance for both.

$\therefore d(z,x) + d(z,y) = d(x,y)^{\Delta}$

Since x,y could be any two vertices, $d(x,y)$ could very well be $diam(\mathcal{G})$.

$\therefore d(x,y) \Rightarrow diam(\mathcal{G}) \leq 2\,rad(\mathcal{G})$ $\Delta *$

$\therefore rad(\mathcal{G}) \leq diam(\mathcal{G}) \leq 2\,rad(\mathcal{G})$.

5.