Unit-2

Paths, path products and Regular Expressions

Overview

* Path expressions are introduced as an algebraic representations of sets of paths in a graph.

* With suitable arithmetic laws and weights, path expressions are converted into algebraic functions or regular expressions that can be used to examine structural properties of flowgraphs such as number of paths, Processing time.

3.1 path products and path expressions

Basic concepts

→ Every link of a graph can be given a name. The link name will be denoted by lowercase italic letter

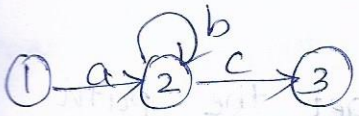
→ The pathname (Link names along the path) or path segment that corresponds to these links is expressed naturally by concatenating those link names.

→ If we traverse links a, b, c and d along some path, the path name for that path segment is abcd

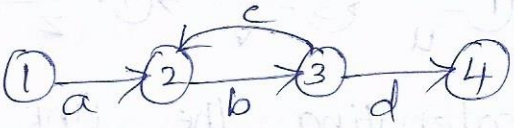
This path name is called path product.

* There are some examples of paths are shown below

eacf, eadf, ebdf, ebcf



ac, abc, abbc, abbbc



abd, abcbd, abcbcbd

Eg: Examples of paths

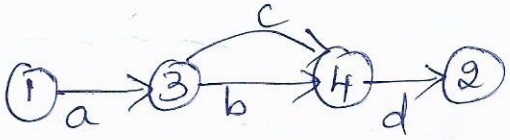
* Any expression that consists of path names and "ORs" which denotes a set of paths between two nodes is called a <u>path expression.</u>

Paths are : abdf, acef, abef, acdf

Path expression of above control flow graph is

abdf + acef + abef + acdf

Path products :- The name of a path that consists of two successive path segments is done by Path Product concatenation of two or more links, two or more path segments or path names, two or more path expression are called path expression.



Link names = abd
= acd

→ concatenating the link names, we get abd the path segments as abd and acd

→ we denote abd as X1 and acd as X2

→ concatenating the path segments X1 and X2 we

path expression of above flowgraph is

abd + acd ⟶ pathexpression 1 (PE1)

→ concatenating the link names, we get the path segments as uvz, uvy. we denote uvz as R1 and uvy as R2

concatenating the path segments R1 and R2 we get as

$R1 \cdot R2 = uvzuvy$

path expression of the above flowgraph is $\boxed{uvz + uvy}$ ↓ Path expression (PE2)

concatenating of path expression 1 and 2, we get

$(abd + acd)(uvz + uvy)$

$= abduvz + abduvy + acduvz + acduvy$

* If a link or segment name is repeated then that fact is denoted by an exponent. The exponent's value denotes the number of repetitions.

$a^1 = a$; $a^2 = aa$; $a^3 = aaa$; $a^n = aaa, \ldots$ ntimes

Similarly if

$X = abcde$

then

$X^1 = abcde$; $X^2 = abcde\,abcde = (abcde)^2$

$X^3 = abcde\,abcde\,abcde = (abcde)^2 abcde$

* The path product is not commutative i.e, $(XY \neq YX)$ but expressions derived from it may be commutative

* The path product is associative, but expression derived from it may not be that is

Rule 1 : $A(BC) = (AB)C = ABC$

where A, B and C are path names, sets of path names or path expressions.

* The zeroth power of a link name, path product or path expression is also needed for completeness. It is denoted by the numeral '1' and denotes the path whose length is zero

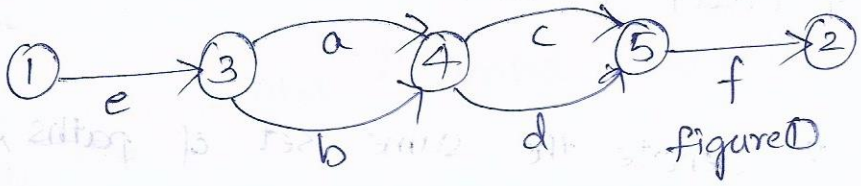
1 is a multiplicative identity element

i.e $a^0 = 1$, $x^0 = 1$

Path sums

* The path sums denotes paths in parallel between two nodes

* The "+" sign was used to ~~create~~ denote the fact that path names were part of the same set of paths

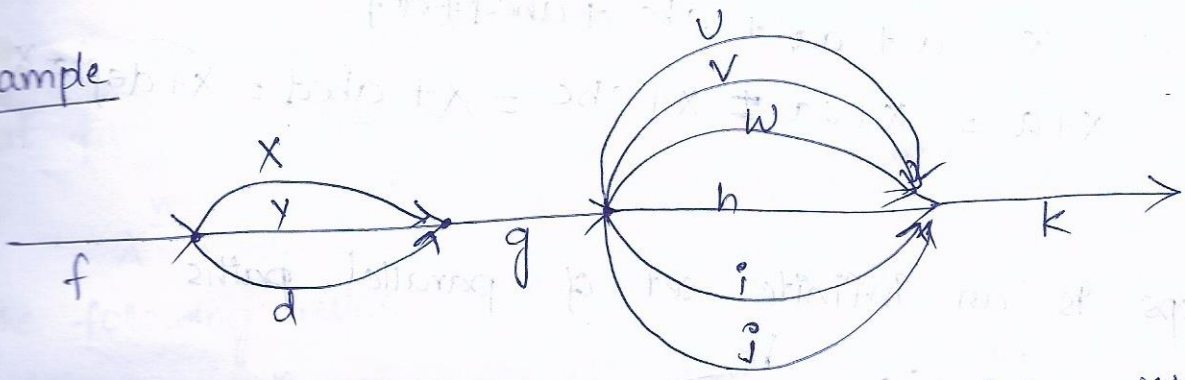


figure①

$e(a+b)(c+d)f$

Links a and b in the above figure are parallel paths and are denoted by $a+b$

Example



* The first set of parallel paths is denoted by $x+y+d$ and the second set by $u+v+w+h+i+j$.

The set of all possible paths in the above

$= f(x+y+d) + g(v+v+w+h+i+j)k$

* path sum is commutative and associative

Rule 2: $x+y = y+x$

Rule 3: $(x+y)+z = x+(y+z) = x+y+z$

Distributive laws

The product and sum operations are distributi and the ordinary rules of multiplication apply, that is

Rule 4: $A(B+C) = AB+AC$

$B(C+D) = BC+BD$

Apply these rules on path segment of figure1 yield

$e(a+b)(c+d)f$

$= e(ac+ad+bc+bd)f$

$= eacf + eadf + ebcf + ebdf$

Absorption Rule

If x and y denote the same set of paths, the the union (+) of these sets is unchanged

Rule 5; $x+x = x$

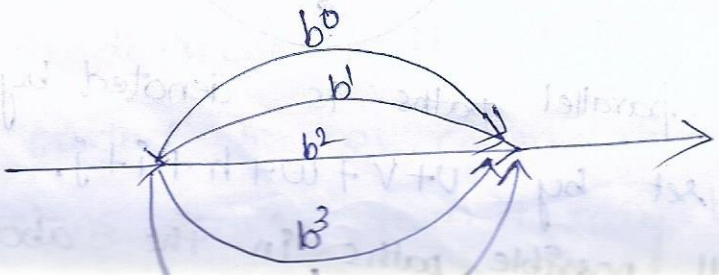
for example

$x = a + aa + abc + abcd + def$

$x+a = x+aa \neq x+abc = x+abcd = x+def = x$

Loops

Loops is an infinite set of parallel paths



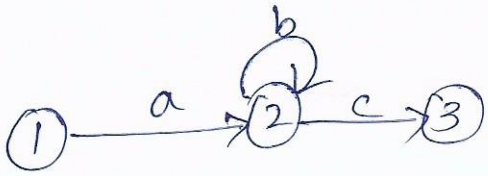
The loop consists of a single link b. Then set of paths through that loop point is

$$b^0 + b^1 + b^2 + b^3 + \cdots$$

This potentially infinite sum is denoted by b^* for an individual link and by x^* when x is a path expression

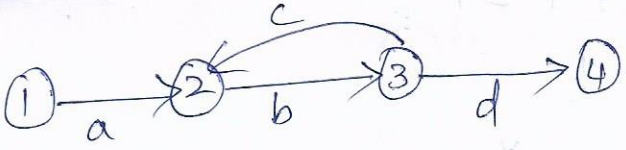
If loop is taken at least once then it is denoted by a^+ or x^+

Example



Path expression is

$$ab^*c = ac + abc + abbc + abbbc + \cdots$$



$$a(bc)^* bd = abd + abcbd + abcbcbd + abcbcbcbd + \cdots$$

$$= abd + abcbd + abcbcbd + a(bc)^3 bd$$

Fridentally

$$a a^* = a^* a = a^+$$

and

$$x x^* = x^* x = x^+$$

* A bar is used under the exponent to denote that fact as follows

$$x^{\underline{n}} = x^0 + x^1 + x^2 + x^3 + \cdots + x^{n-1} + x^n$$

The following rules can be derived

Rule 6: $x^{\underline{n}} + x^{\underline{m}} = x^{\underline{n}}$ (if n is bigger than m)

$= x^{\underline{m}}$ (if m is bigger than n)

Rule 7: $x^{\underline{n}} x^{\underline{m}} = x^{\underline{n+m}}$

Rule 8: $x^n x^*$

Rule 8: $x^n x^* = x^* x^n = x^*$

Rule 9: $x^n x^+ = x^+ x^n = x^+$

Rule 10: $x^* x^+ = x^+ x^* = x^+$

Identity Elements

The zeroth power of a link name, path product or path expression is denoted by the numeral 1 and denotes the path whose length is zero

i.e, $a^0 = 1$, $x^0 = 1$

Rule 11: $1 + 1 = 1$

Rule 12: $1X = X1 = X$

Rule 13: $1^n = 1^{\underline{n}} = 1^* = 1^+ = 1$

Rule 14: $1^+ + 1 = 1^* = 1$

Rule 15: $X + 0 = 0 + X = X$

Rule 16: $X0 = 0X = 0$

Rule 17: $0^* = 1 + 0 + 0^2 + \cdots = 1$

The meaning and behavior of zero and one (The identity elements) given above apply to only path names

3.2 Reduction procedure

* A Reduction procedure for converting a flowgraph whose links are labelled with names into path expression that denotes all the set of entry/exit paths

* The procedure is a node-by-node removal algorithm

80 steps are followed: They are

Step1: Select any node for removal other than the initial or final node.

Replace it with set of equivalent links whose path expression correspond to all the ways you can form a product of the set of inlinks with the set of outlinks of that node.

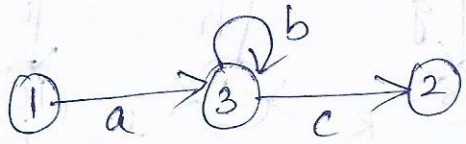
(8)

Step2: Combine all the serial links by multiplying their path expressions.

step3: Combine all the parallel links by adding their path expression.

step4: Remove all the self-loops by replacing them with a link of the form x^*, where x is the path expression of the link

Example:-



After removing loop :

Step5:- Does the graph consist of a single link between the entry node and the exit node?

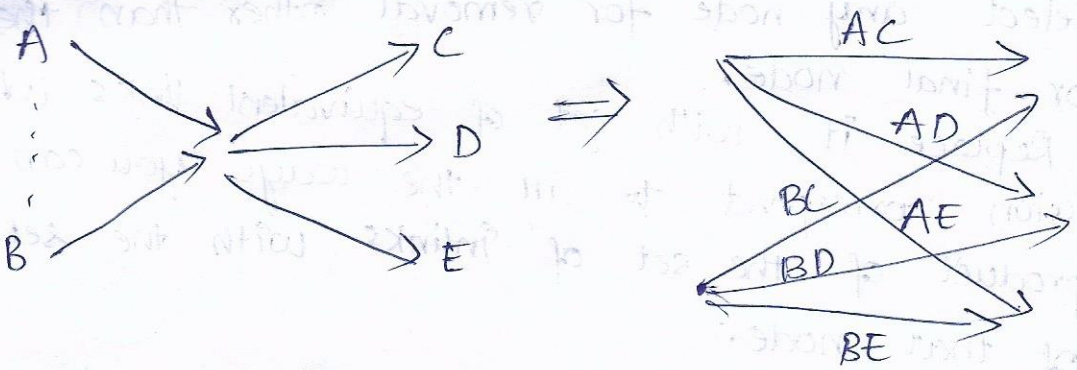
If yes then the path expression for that link is a path expression for the original flowgraph

Otherwise return to step1.

Cross-Term Step

The cross term step is the fundamental step of the reduction algorithm. It removes a node there by reducing the number of nodes by one.

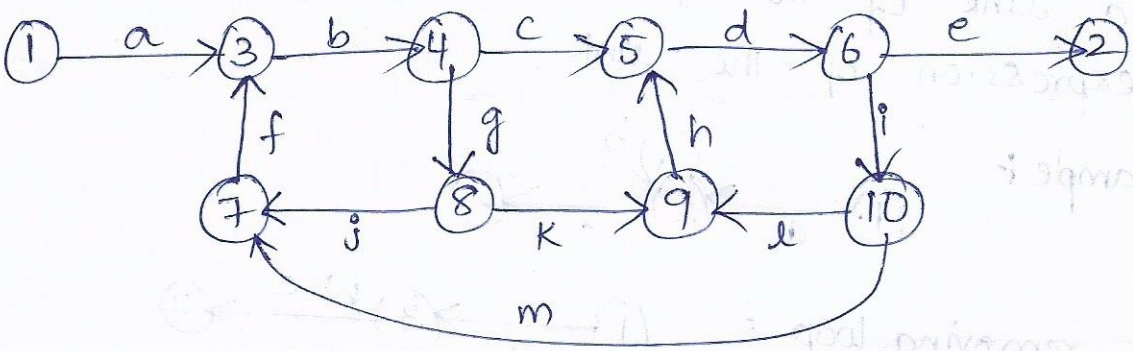
The following diagram shows the situation at an arbitrary node that has been selected for removal.



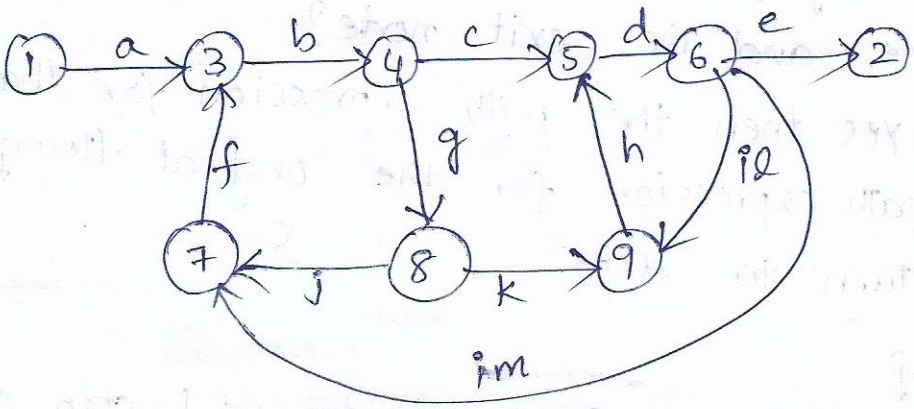
The combination of incoming and outgoing path segments as in

$$(a+b)(c+d) = ac + ad + bc + bd$$

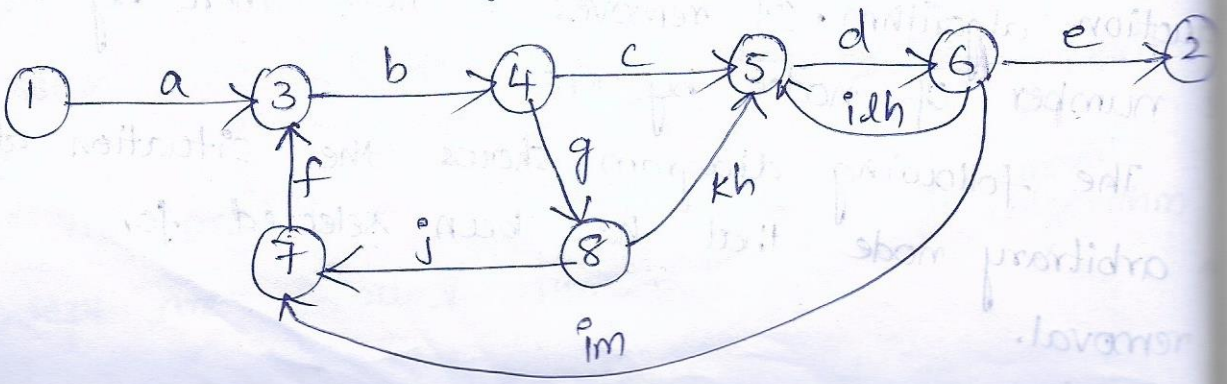
Applying the steps and remove several nodes for the following example flowgraph



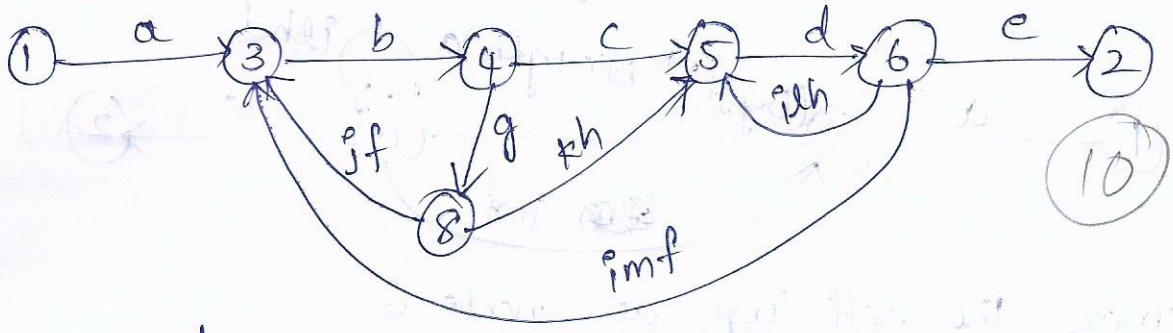
Remove node 10, we get



Remove node 9, we get

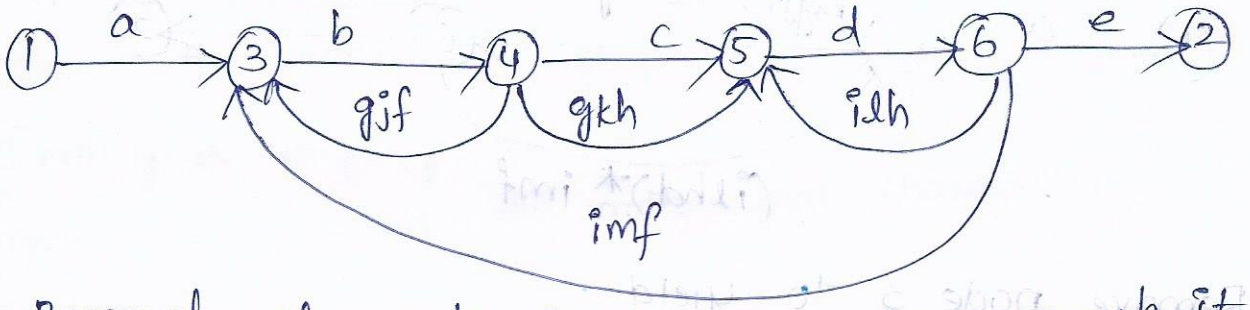


Remove node 7, we get

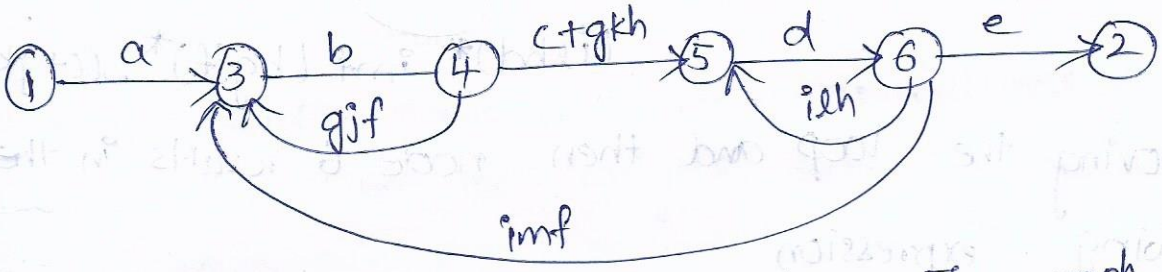


10

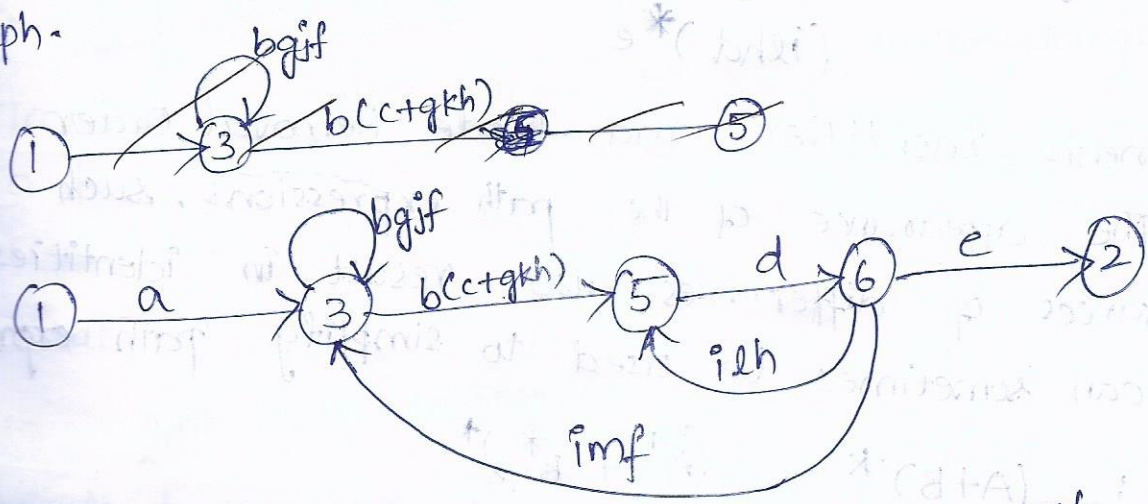
Remove node 8, we get



* Removal of node 8 in the above graph it leads to a pair of parallel ~~paths~~ links between nodes 4 and 5

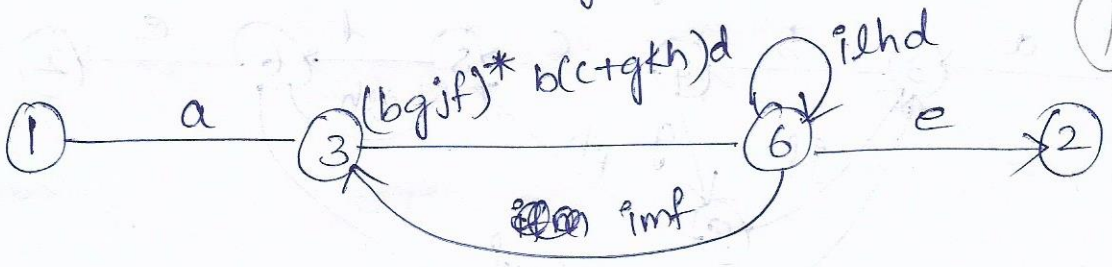


* Remove node 4 leads to a loop term. The graph has now been replaced with following equivalent, simpler graph.

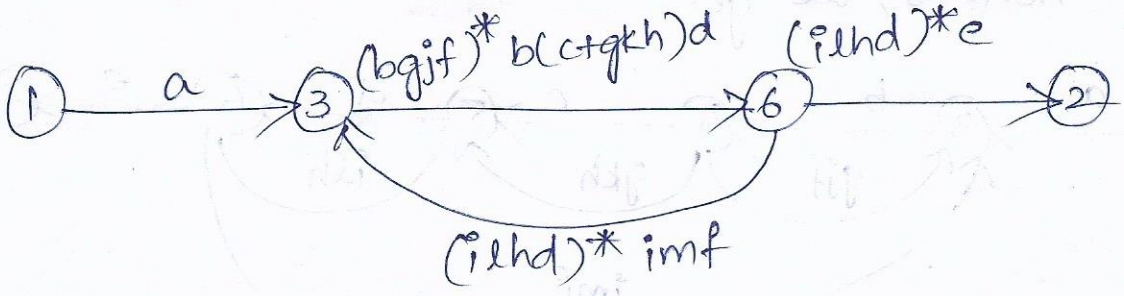


* After removing self loop at node 3, we get

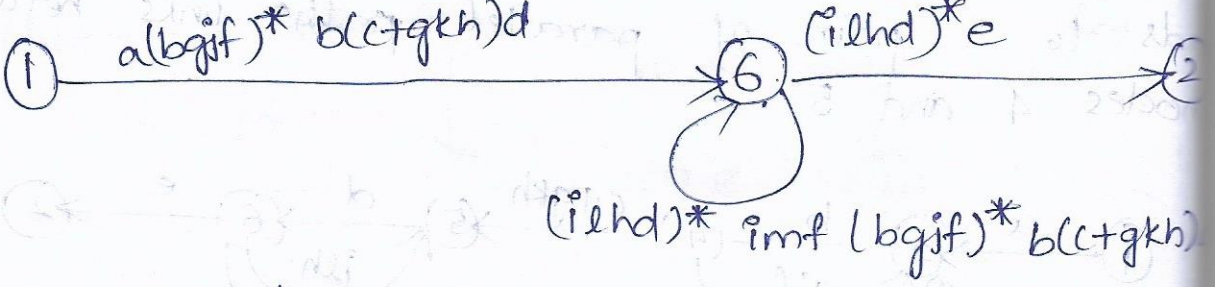
Remove node 5, we get



* Remove the self loop at node 6



Remove node 3 to yield



Removing the loop and then node 6 results in the following expression

$$a(bgjf)^* b(c+gkh)d((ilhd)^* imf (bgjf)^* b(c+gkh)d)^* (ilhd)^* e$$

Comments, Identities and Node Removal Order

The appearance of the path expressions, such appearances of differences also result in identities that can sometimes be used to simplify path expres

I1 : $(A+B)^* = (A^* + B^*)^*$

I2 : $= (A^* B^*)^*$

I3 : $= (A^* B^*) A^*$

I4 : $= (B^* A^*)^*$

I5: $= (A^* B + A)^*$

I6: $= (B^* A + B)^*$

I7: $(A + B + C + \cdots)^* = (A^* + B^* + C^* + \cdots)^*$

I8: $= (A^* B^* C^*)^*$

3.3. Applications

Every application follows this common pattern

1. convert the program or graph into a path expression
2. Identify a property of interest and derive an appropriate set of arithmetic rules that characterizes the property.
3. Replace the link names by the link weights. The path expression in some algebra
4. Simplify or evaluate the resulting algebraic expression to answer the question

* In practice, we don't do it as outlined above. Rather we substitute the weights first and simplify as you develop the path expression, using the right kind of arithmetic as you remove the nodes

How many paths in a Flowgraph

→ To find Maximum number of paths in the flowgraph
→ To find Minimum number of paths in the flowgraph
→ How many different paths are there really?
→ what is the average number of paths?

Maximum path count arithmetic

* to find Maximum number of paths in the flowgraph
* Label each link with a link weight that corresponds to the number of paths that the link represents
* Mark each loop with the maximum number of times

* If the answer is yes infinite, you might as well stop the analysis because it is clear that maximum no of paths will be infinite

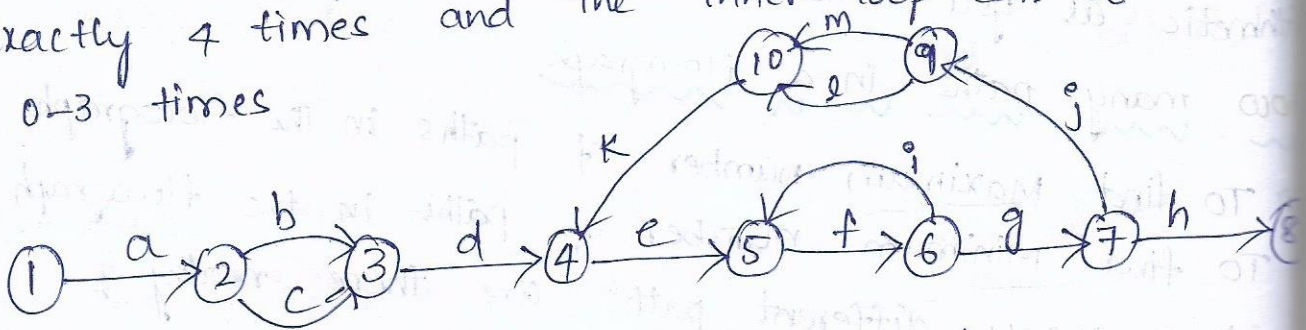
* There are 3 cases i.e, parallel links, serial link and loops

CASE	path expressions	weight expressions
Parallel	A+B	$W_A + W_B$
series	AB	$W_A W_B$
Loop	A^n	$\sum_{j=0}^{n} W_A^j$

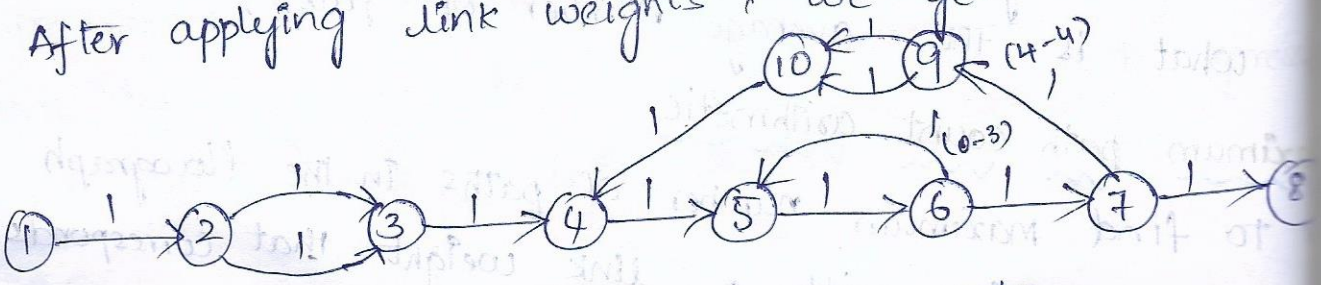
Where A, B are path expressions and W_A, W_B are algebraic expressions in the weights

A Concrete Example

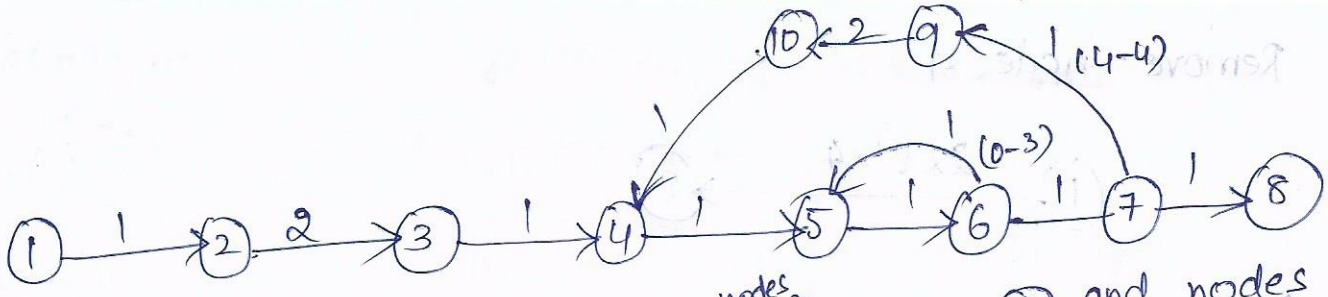
* Each link represents a single link and consequently is given a weight of "1" to start.
* Let's say that the outer loop will be taken exactly 4 times and the inner loop can be taken 0-3 times



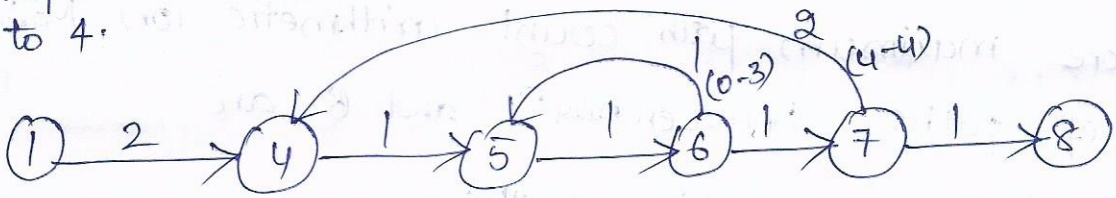
After applying link weights, we get



Add parallel nodes 2 and 3 and nodes 10 and 9

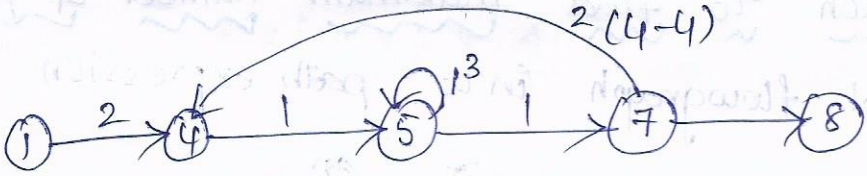


Multiple the serial links from nodes 1 to 4 and nodes 4 to 4.



(14)

Remove node 6, we get



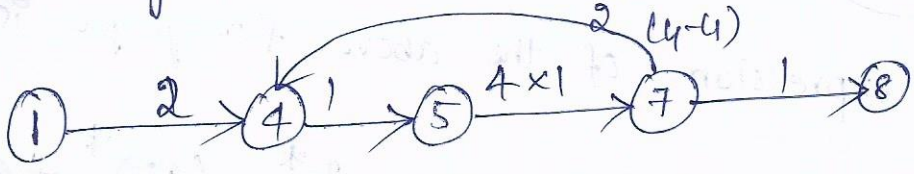
From Arithmetic rules

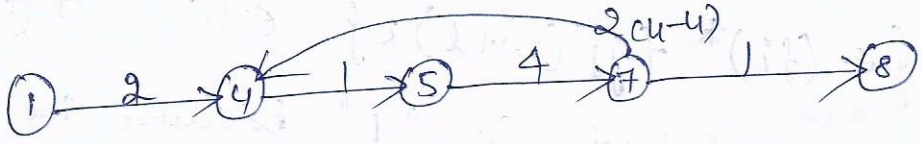
loop = $A^n = \sum_{j=0}^{n} W_A^j$

$1^3 = \sum_{j=0}^{3} 1^3$

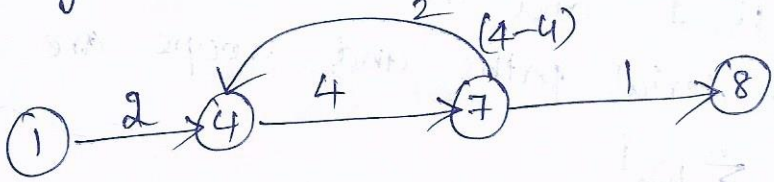
$= 1^0 + 1^1 + 1^2 + 1^3 = 4$

After removing self loop, we get

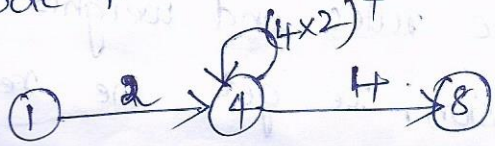


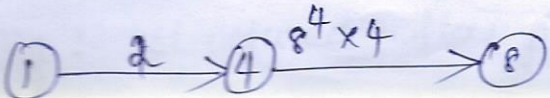


After removing node 5 we get



Remove node 7



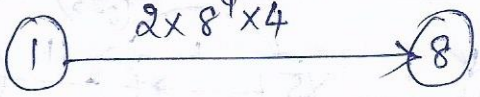


$A^n = \sum_{j=0}^{n} W_A^j$

$8^4 = \sum_{i=4}^{4} 8^1$

$= 8^4$

Remove node 4



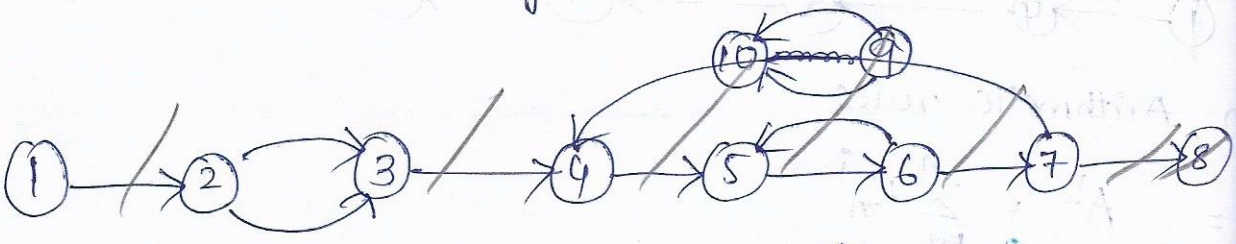
i.e $2 \times 8^4 \times 4 = 32,768$

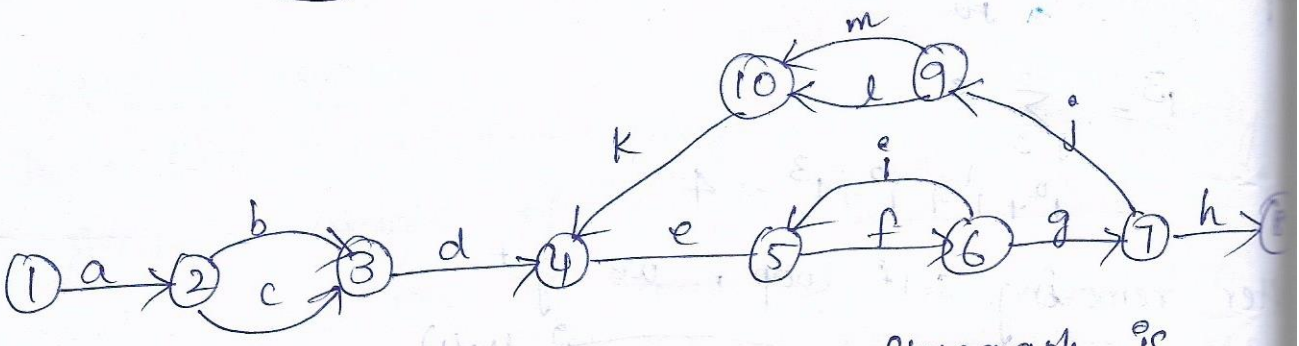
Therefore maximum path count arithmetic (or) Maximum number of paths between nodes ① and ⑧ are

$$2 \times 8^4 \times 4 = 32,768 \text{ paths.}$$

Alternate approach to find maximum number of paths

1. convert control flowgraph into path expression





The path expression of the above flowgraph is

$$a(b+c)d\{e(fi)^* fgj(m+l)k\}^* e(fi)^* fgh$$

② The property of interest is "1" because the length of each link is 1 and the arithmetic rules for parallel paths, serial paths and loops are $W_A + W_B$ $W_A W_B$ and $\sum_{j=0}^{n} W_A^j$

③ After applying arithmetic rules and weights specified above on the path expression, we get the regular expression as

$a(b+c)d\{e(fi)^* fgj(m+l)k\}^* e(fi)^* fgh$

$1(1+1)1\{1(1*1)^* 1*1*1(1+1)1\}^* 1*(1*1)^* 1*1*1$

(F) After evaluating the above expression we get maximum number of paths between nodes (1) and (8)

i.e $1(1+1)\{1(1*1)^3 1*1*1(1+1)\}^4 1*(1)^3 1*1*1$

$= 1(2)*\{1^3 * 1(2)\}^4 1^3$

$1^3 = 1^0+1^1+1^2+1^3$
$= 4$

$= 2*\{4*2\}^4 4$

$8^4 = 4096$

$= 2*8^4*4$

$= 32,768$

Approximate Minimum Number of paths

Structured code

* The node-by-node reduction procedure can also be used as a test for structured code.
* structured code can be defined in several different ways

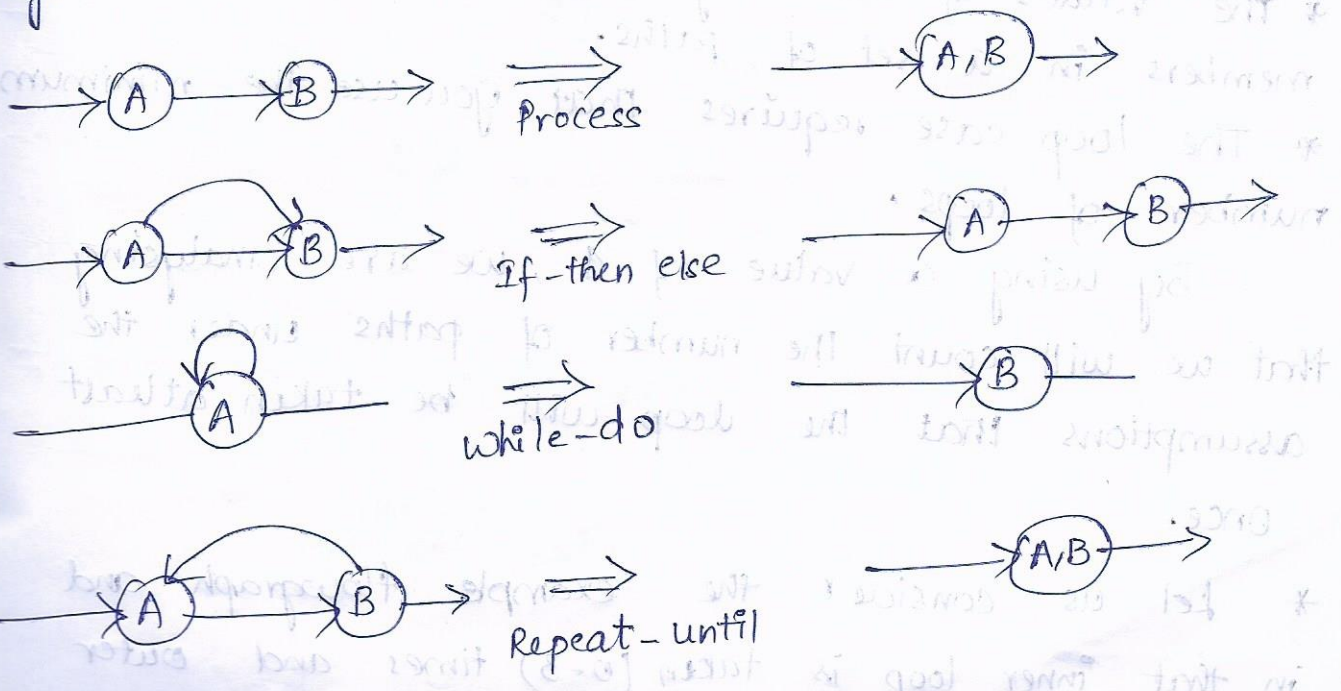


Fig: structured flowgraphs transformations

* A structured flowgraph is one that can be reduced to a single link by successive application of the transformations.

Lower path count Arithmetic (or) Minimum no of paths

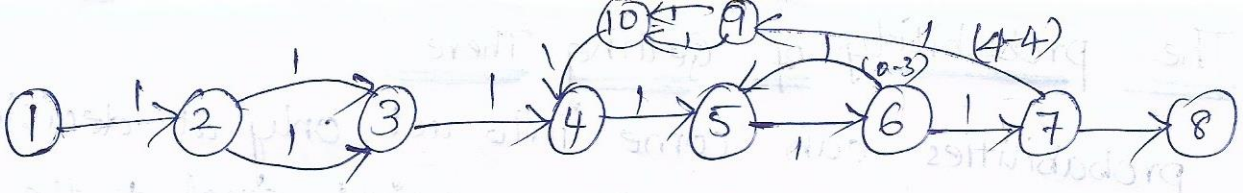
* Label each link with a link weight that corresponds to the number of paths that the link represents
* Mark each loop with the minimum number of times that the loop can be taken
* The arithmetic rules are as follows

case	path expression	Weight expression
Parallel	A+B	$W_A + W_B$
Series	AB	MAX (W_A, W_B)
Loop	A^n	1, W_1

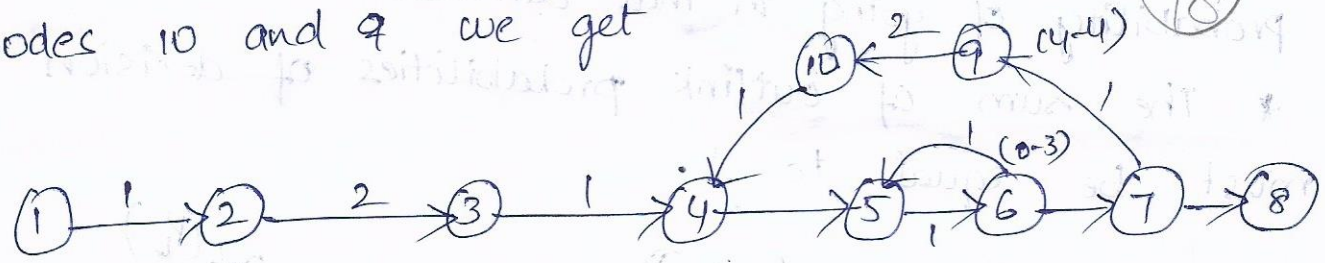
* The parallel case is same as before
* The values of the weights are the number of members in a set of paths.
* The loop case requires that you use the minimum number of loops.

By using a value of 1, we are analysing that we will count the number of paths under the assumptions that the loop will be taken atleast once.

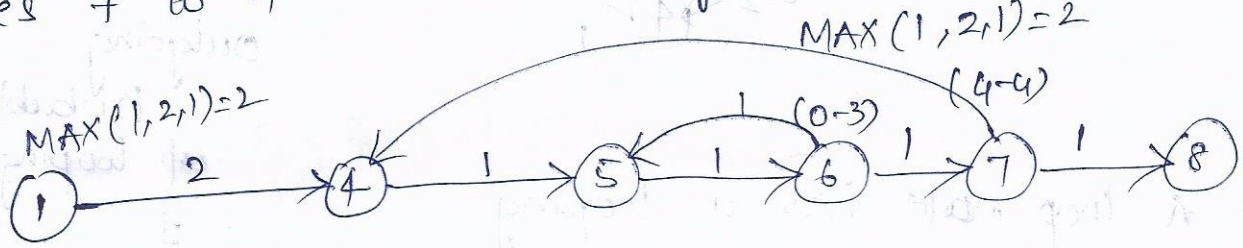
* Let us consider the example flowgraph and in that inner loop is taken (0-3) times and outer loop taken exactly (4-4) times

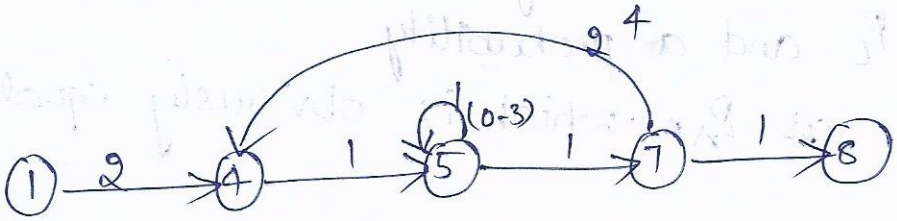


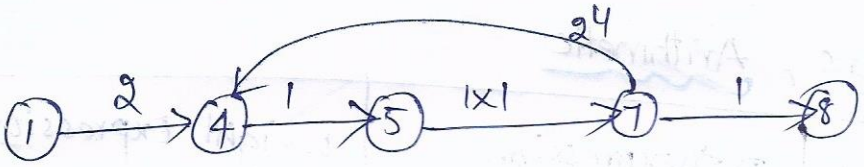
Add the parallel links between nodes 2 and 3, nodes 10 and 9 we get

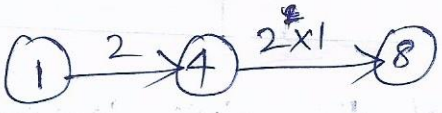


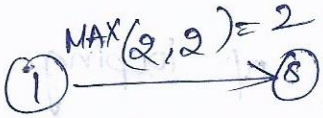
Series link between nodes 1 and 4 and nodes 7 to 4 are solved by formula $MAX(W_A, W_B)$











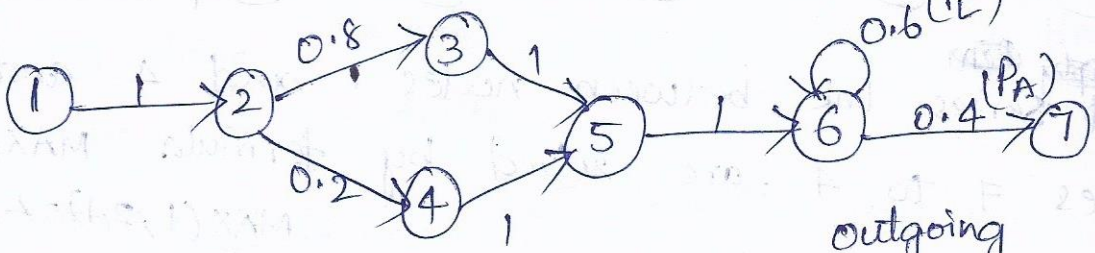
$= 2$

For the above flowgraph the minimum no of

The probability of Getting There

* probabilities can come into act only at decisio
* Annotate each outlink with a weight equal to the probability of going in that direction
* The sum of outlink probabilities of decision must be equal to 1.

Ex:-



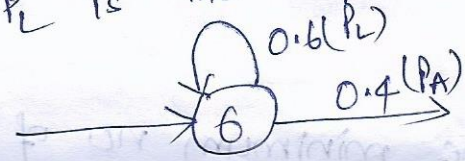
outgoing Probabilit of looping is 1 -

* A loop node has a looping probability of P_L and a probability of not looping as P_A which is obviously equal $1 - P_L$.

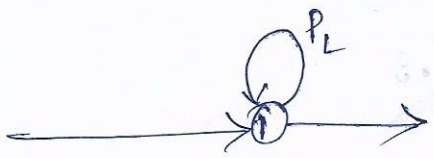
Weights, Notations, Arithmetic

CASE	pathexpression	weight Expression
Parallel	A + B	$P_A + P_B$
Series	AB	$P_A P_B$
Loop	A^*	$P_A / (1 - P_L)$

P_A is the probability of the link leaving the loop and P_L is the probability of looping.

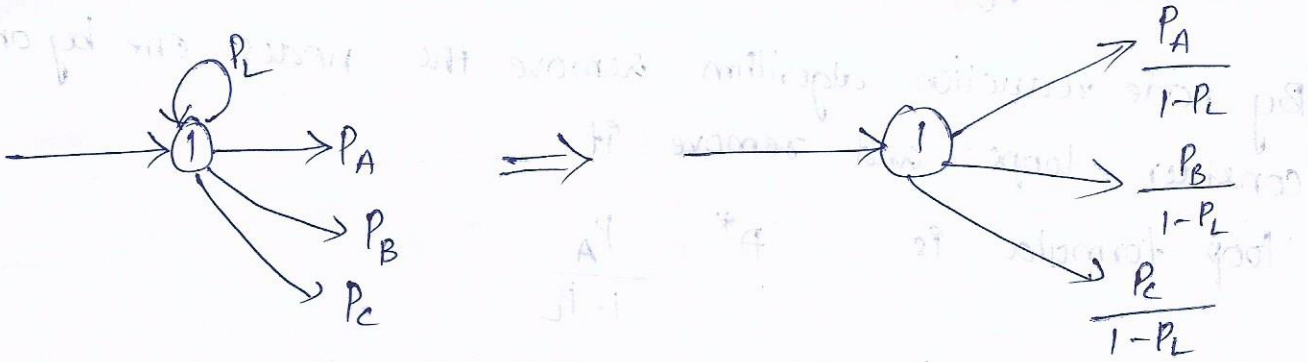


i.e, $P_A = \frac{0.4}{} = \frac{0.4}{} = 1$



$$P_A = 1 - P_L$$

$$P_{new} = \frac{P_A}{1-P_L} = \frac{1-P_L}{1-P_L} = 1$$

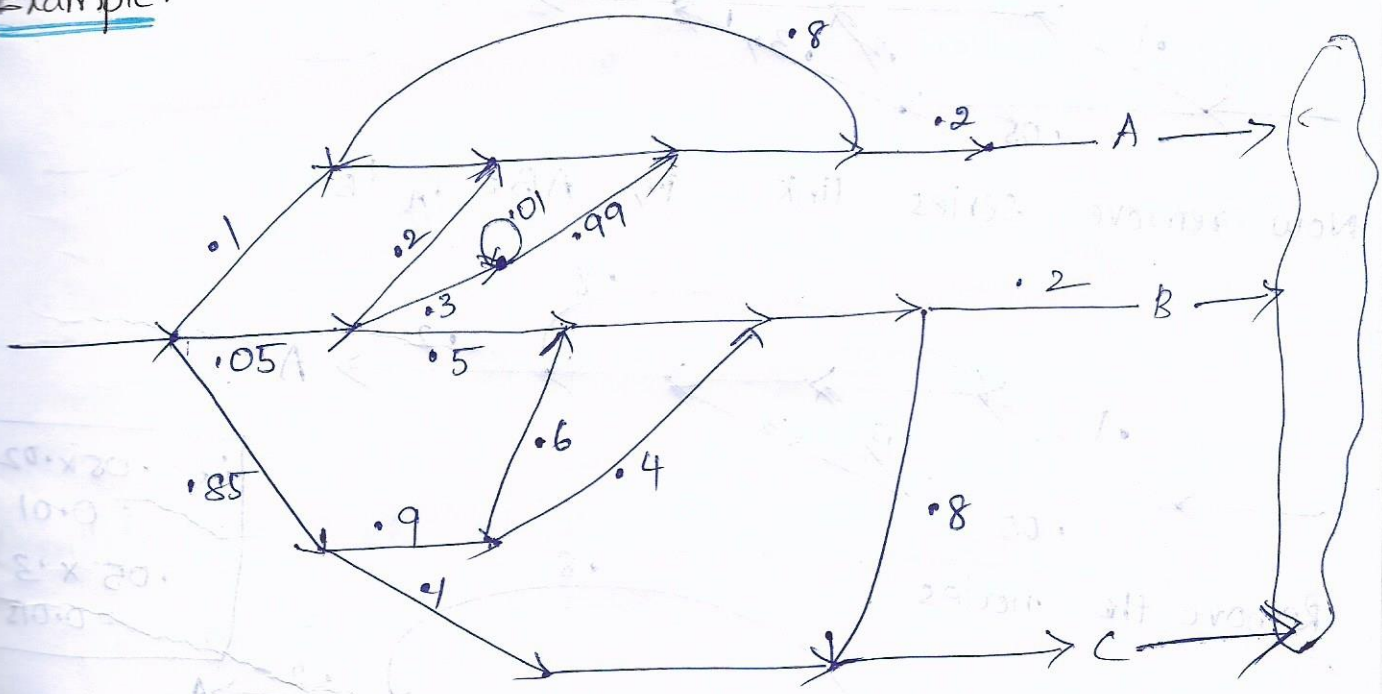


because $P_L + P_A + P_B + P_C = 1$

$$1 - P_L = P_A + P_B + P_C$$

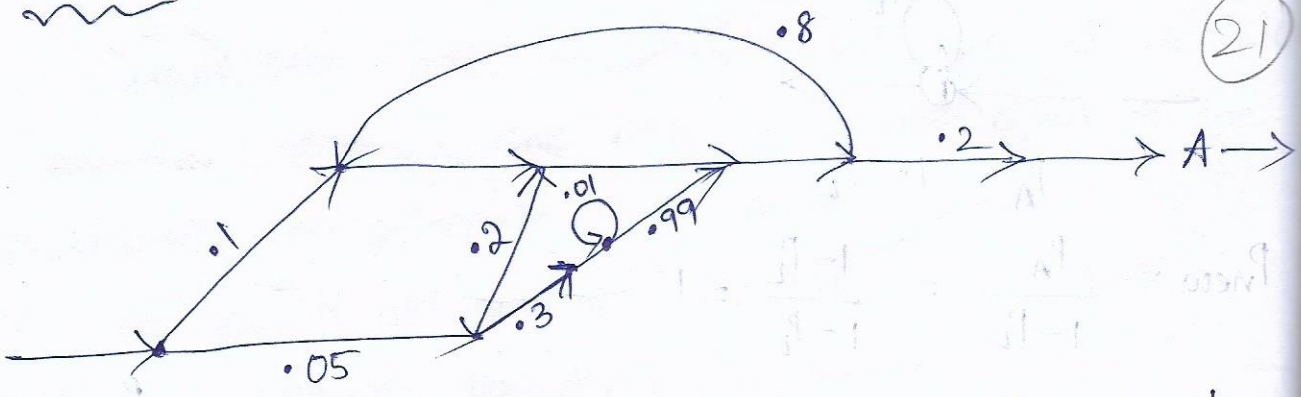
and $$\frac{P_A}{1-P_L} + \frac{P_B}{1-P_L} + \frac{P_C}{1-P_L} = \frac{P_A + P_B + P_C}{1-P_L} = 1$$

Example :-



The above control flow graph is splitted into three cases

Case A



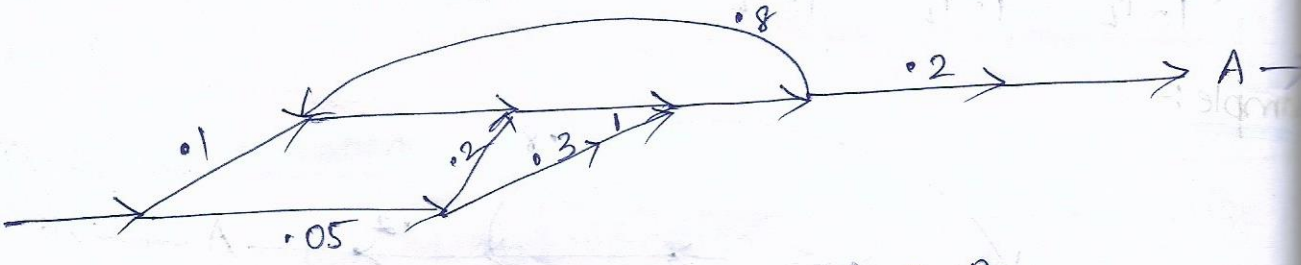
By node reduction algorithm remove the nodes one by on consider loops and remove it

loop formula is $A^* = \frac{P_A}{1-P_L}$

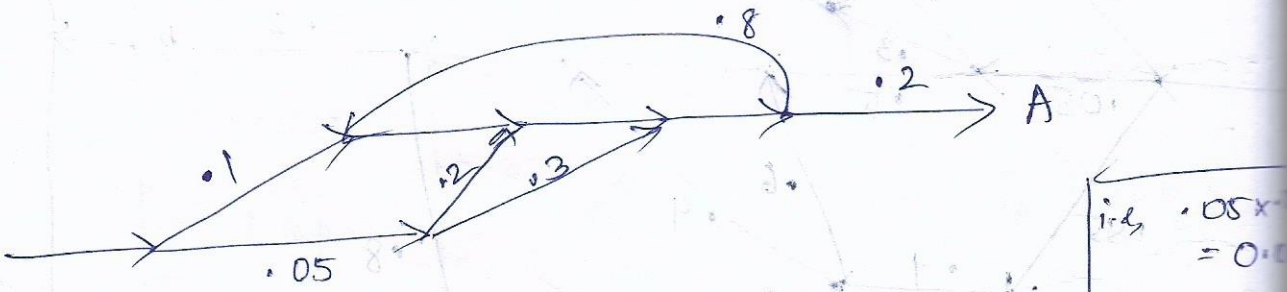
In the above flowgraph

P_L is .01 and P_A is .99

i.e $\frac{P_A}{1-P_L} = \frac{.99}{(1-.01)} = 1$

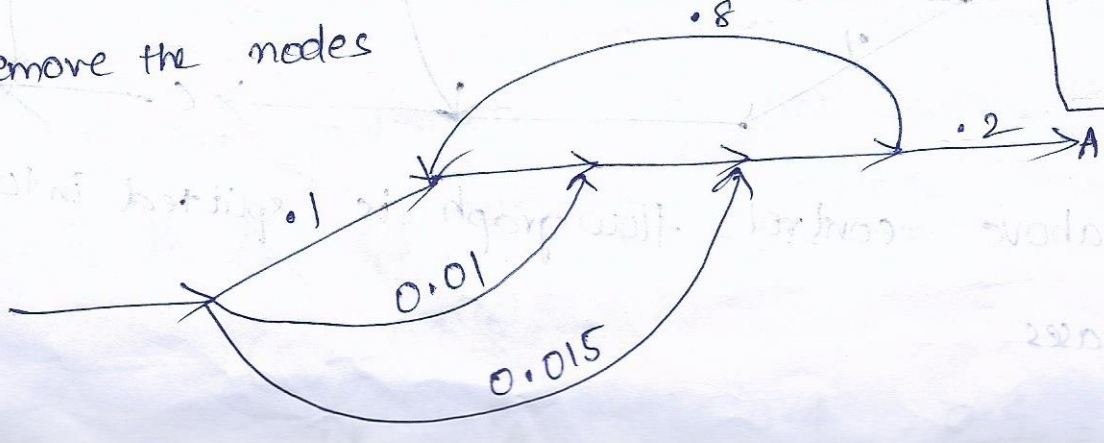


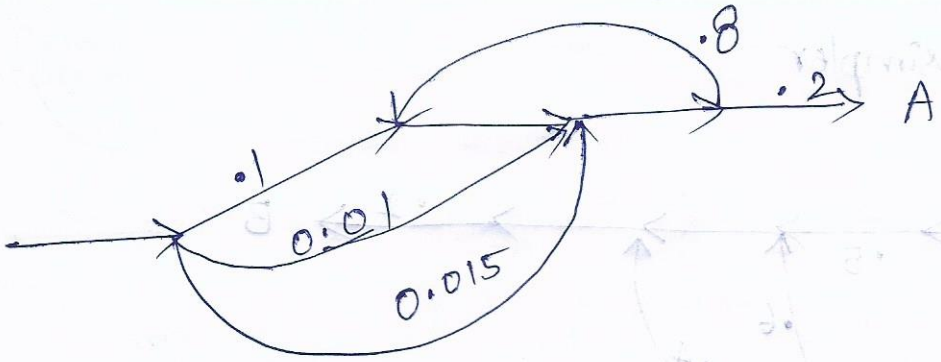
Now remove series link i.e $AB = P_A P_B$



i.e, .05 x
= 0.0
.05 x
= 0.0

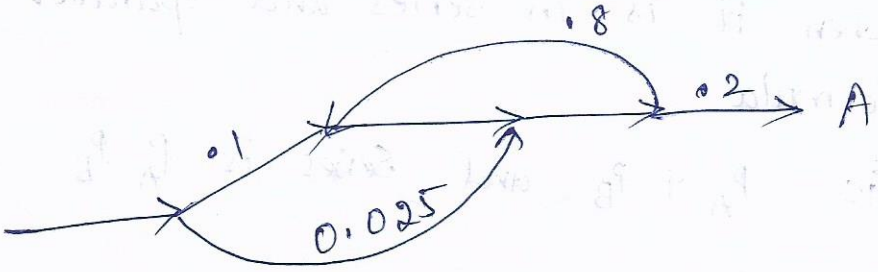
Remove the nodes

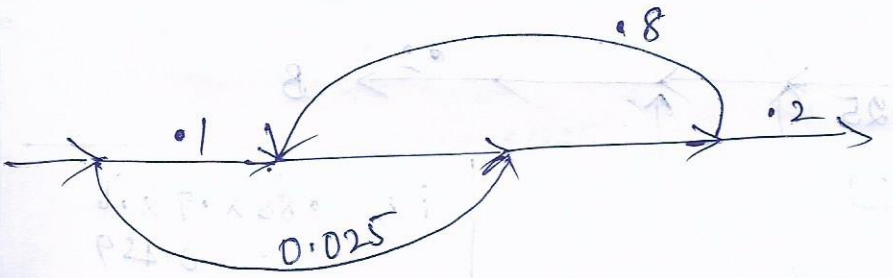


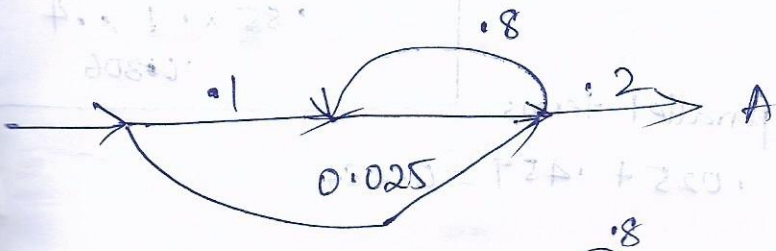


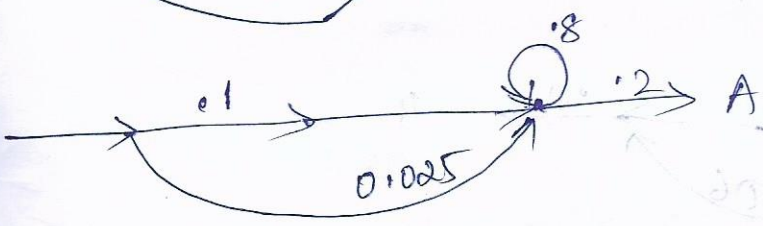
Add parallel terms i.e $P_A + P_B$

$0.01 + 0.015$
$= 0.025$

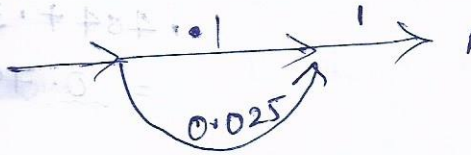








$$\frac{P_A}{1-P_L} = \frac{.2}{1-.8}$$
$$= 1$$

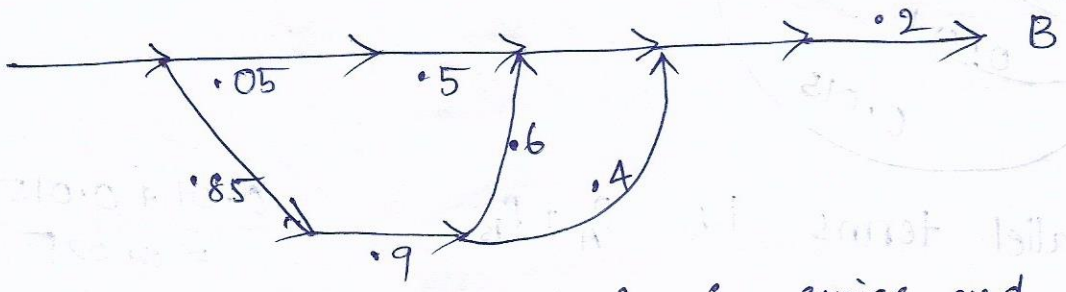


Parallel links i.e, .1 and 0.025

$= 0.125$

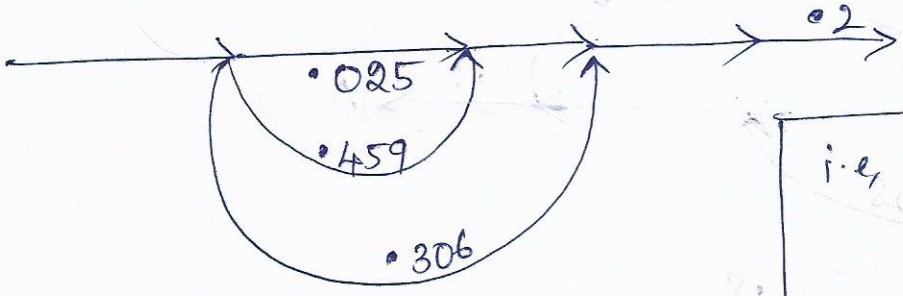
Probability of reaching Node A is 0.125

case B is simpler



Removes nodes when it is in series and parallel links by using formula

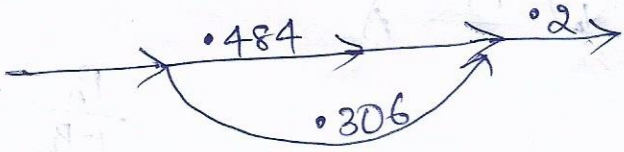
i.e, parallel is $P_A + P_B$ and series is $P_A P_B$

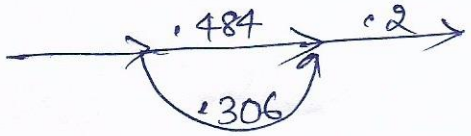


i.e, .85 x .9 x .6
= 0.459
.85 x .9 x .
=0.306

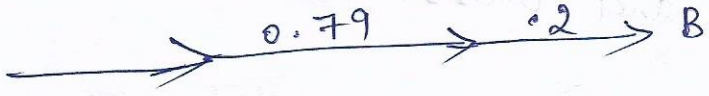
Add parallel terms

i.e, .025 + .459 = 0.484





.484 + .3
= 0.79

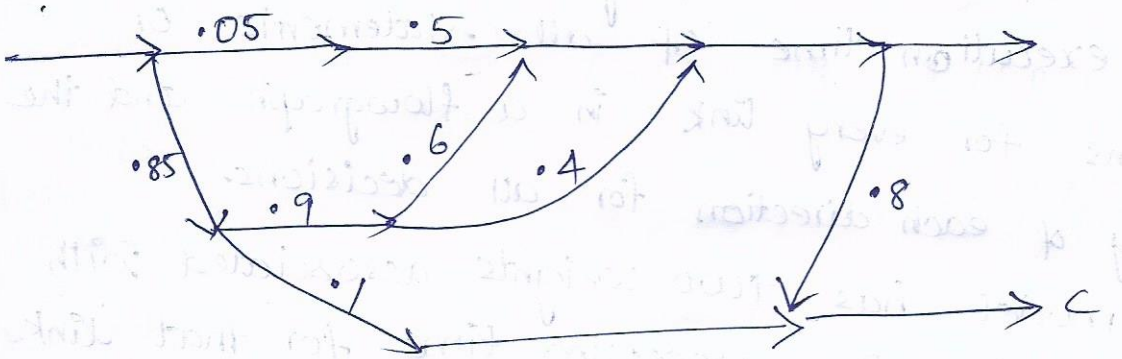


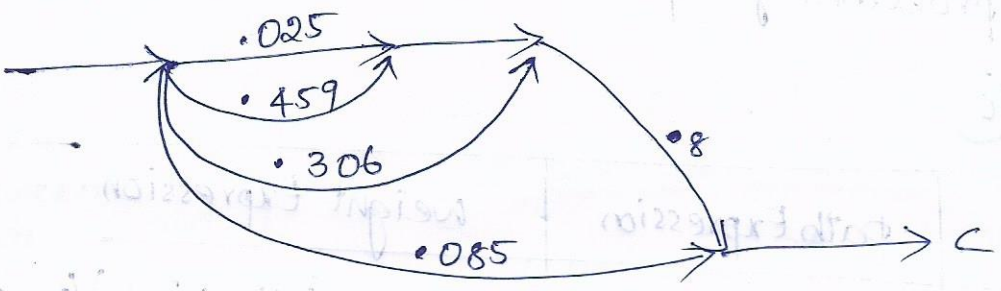
0.79 x .2
= 0.158

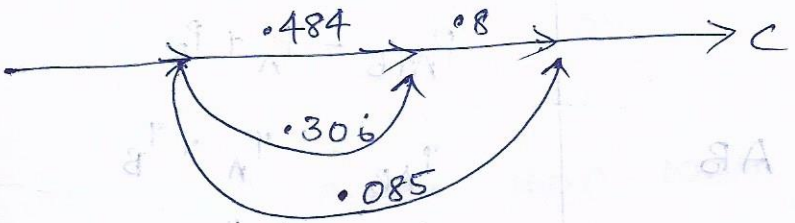
= 0.158

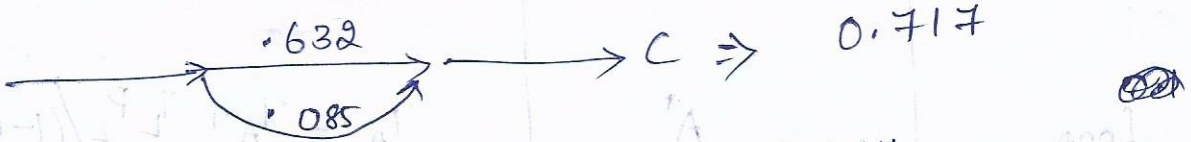
Probability of reaching node B is 0.158

case c









Probability of reaching node c is 0.717

Probability of Reaching node A is 0.125
" " B Is 0.158
" " C is 0.717

The sum of probabilities is 1 (i.e. 0.125 + 0.158 + 0.717 = 1)

If the sum is equals to 1 then our calculated Probabilities are correct.

The Mean processing Time of a Routine

To find the mean processing time of a routine give the execution time of all statements or instructions for every link in a flowgraph and the probability of each direction for all decisions.

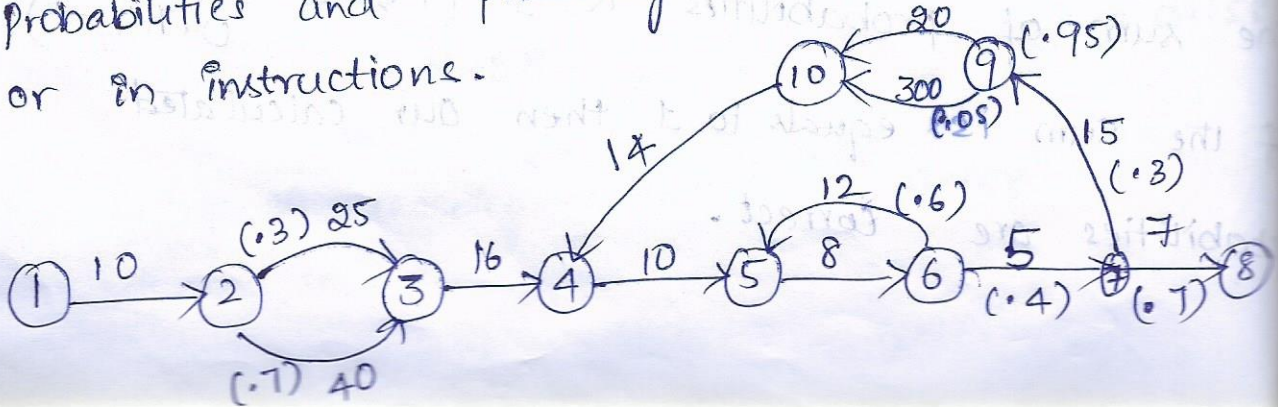
→ The model has two weights associated with every link i.e, the processing time for that link and the probability of that link

The Arithmetic

CASE	path Expression	weight Expression
Parallel	A+B	$T_{A+B} = (P_A T_A + P_B T_B)/(P_A + P_B)$ $P_{A+B} = P_A + P_B$
Series	AB	$T_{AB} = T_A + T_B$ $P_{AB} = P_A P_B$
Loop	A^*	$T_A = T_L P_L/(1-P_L)$ $P_A = P_A/(1-P_L)$

Example

Start with original flowgraph annotated with probabilities and processing time in microseconds or in instructions.



Parallel links are between node 2 and 3 and node 10 and 9

Parallel links between node 2 and 3 reduced as

i.e $T_{A+B} = (P_A T_A + P_B T_B) / (P_A + P_B)$

$= (.3 \times 25 + .7 \times 40) / (.3 + .7)$

$= 35.5/1$

$= 35.5$

Parallel links between node 10 and 9

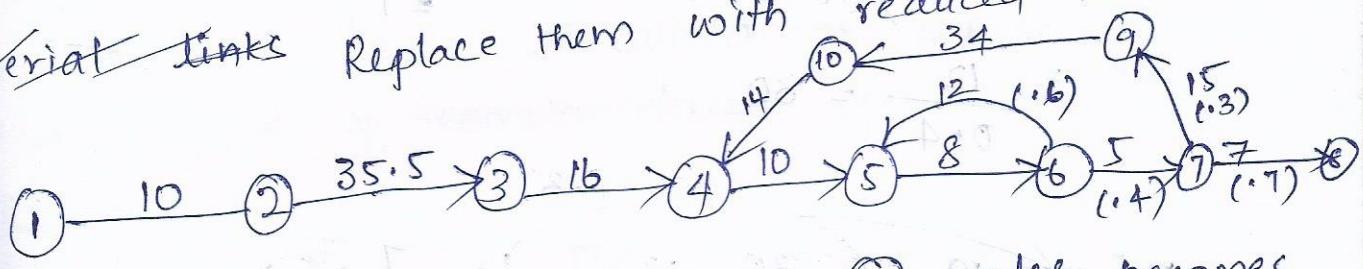
i.e $T_{A+B} = (P_A T_A + P_B T_B) / (P_A + P_B)$

$= (.95) \times 20 + (.05) 300 / (.95 + .05)$

$= 34/1$

$= 34$

~~Serial links~~ Replace them with reduced numbers i.e



Now links between series ① and ④ nodes becomes

series $T_{ABC} = T_A + T_B + T_C$

$= 10 + 35.5 + 16$

$= 61.5$

$P_{ABC} = P_A P_B P_C$

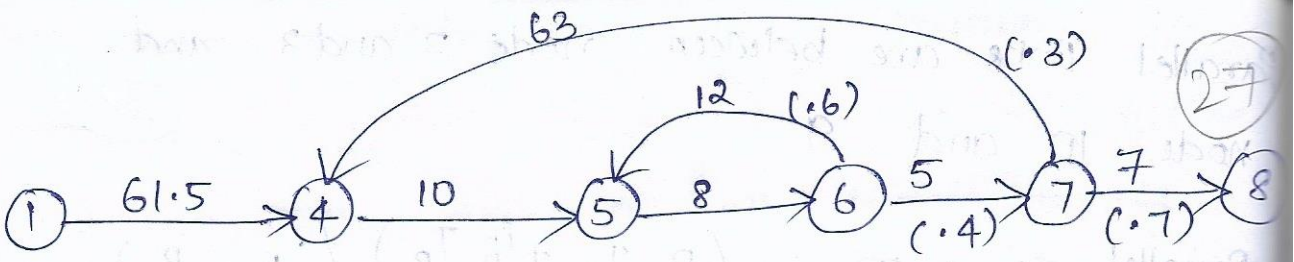
$= 1 \times 1 \times 1$

$= 1$

Nodes between 7 and 4 becomes

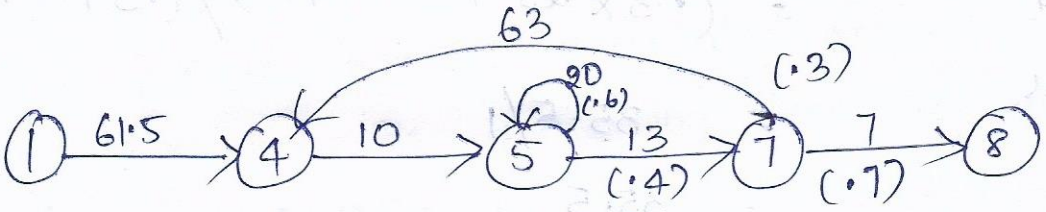
$T_{ABC} = T_A + T_B + T_C = 15 + 34 + 14 = 63$

Now the flowgraph becomes

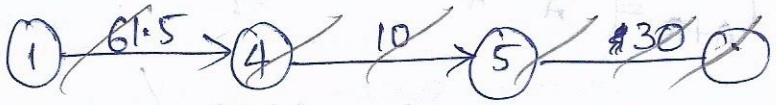


Remove Node 6 we get

12+8 =20



8.



To remove self loop, use

$$T_A = T_L P_L / (1 - P_L)$$

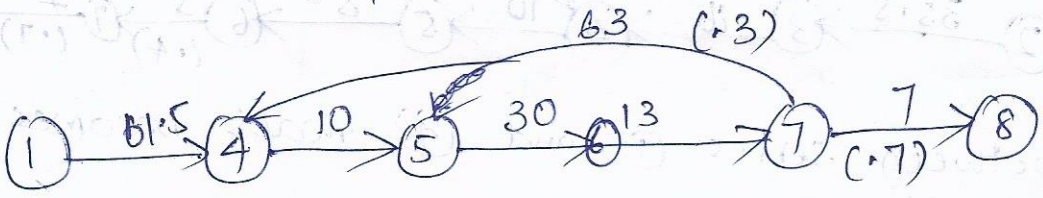
$$= 20 \times 0.6 / (1 - 0.6)$$

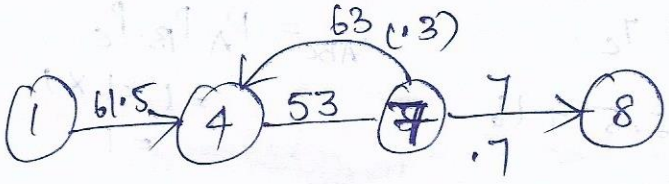
$$= \frac{12}{0.4} = 30$$

$$P_A = P_A / (1 - P_L)$$

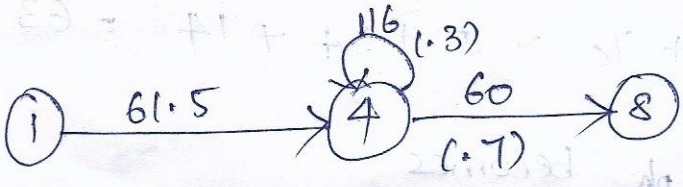
$$= 0.4 / (1 - 0.6)$$

$$= 1$$





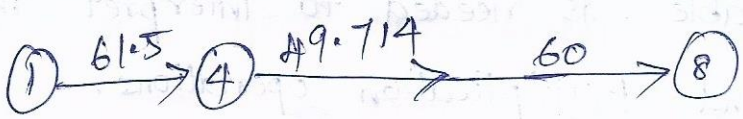
Remove node 7, we get



To remove self loop, use

$$T_A = T_L P_L / (1 - P_L)$$

$$P_A = P_A / (1 - P_L)$$



$= 61.5 + 49.7 + 60$

$= 171.214$

So the mean processing time of above routine is 171.214 microseconds

Push/pop, Get/ Return

* In a routine, which contains many complementary operations such as push and pop operations.
* we can find the net effect of the routine how many push's and pop's performed under what conditions push operations are performed and under what conditions pop operations are performed, using Node-Reduction procedure.
* The above specified calculations can be done to other example of complementary operations like

GET/ RETURN of a resource block,
OPEN/ CLOSE of a file
START / STOP a device or process.

push / pop Arithmetic

The Arithmetic rules of complementary operations are

CASE	PATH Expression	Weight Expression
Parallel	$A+B$	$W_A + W_B$
Series	AB	$W_A W_B$
Loops	A^*	W_A^*

→ An Arithmetic table is needed to interpret the weight addition and Multiplication operations.

→ 'H' denotes "push" and 'P' denotes "pop" and the numeral '1' is used to indicate that nothing i.e (neither push nor pop) occurs on a given link.

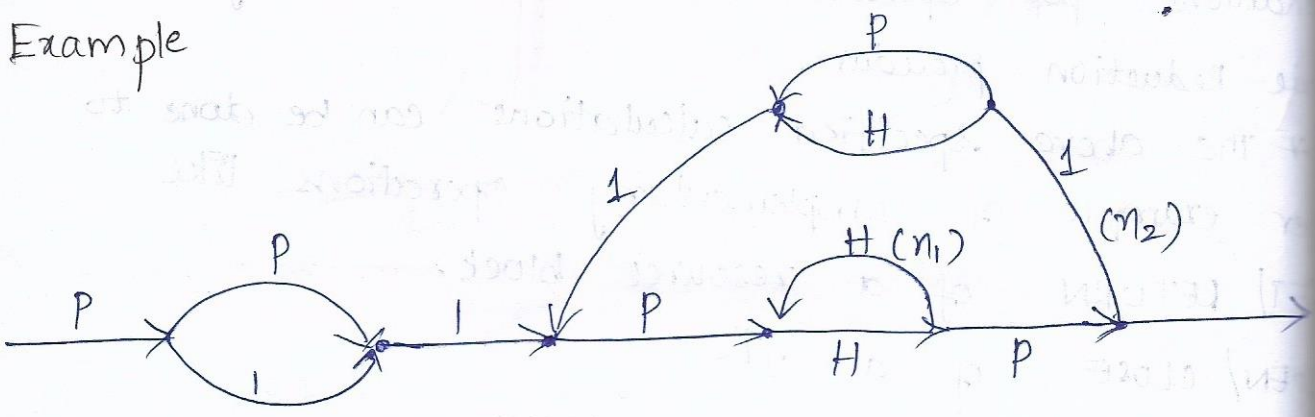
PUSH/pop Multiplication table

X	H PUSH	P Pop	1 NONE
H	H^2	1	H
P	1	P^2	P
1	H	P	1

PUSH/pop Addition Table

+	H PUSH	P POP	1 NONE
H	H	P+H	H+1
P	P+H	P	P+1
1	H+1	P+1	1

Example



The path expression of above flowgraph is

$$P(p+1)1\{P(HH)^{n_1}HP1(P+H)1\}^{n_2}P(HH)^{n_1}HPH$$

Simplifying by using arithmetic tables

$$(P^2+P)\{P(H^2)^{n_1}HP(P+H)\}^{n_2}P(H^2)^{n_1}HPH$$

$$\Rightarrow (P^2+P)\{P(H^2)^{n_1}1(P+H)\}^{n_2}(H^2)^{n_1}$$

$$\Rightarrow (P^2+P)\{(H^2)^{n_1}P(P+H)\}^{n_2}(H^2)^{n_1}$$

∴ H.P = 1
P.H =

$$\Rightarrow (p^2+p)\{H^{2n_1}(p^2+(p\cdot H))\}^{n_2} H^{2n_1}$$

$$\Rightarrow (p^2+p)\{H^{2n_1}(p^2+1)\}^{n_2} H^{2n_1}$$

→ The circumstances under which the stack will be pushed, popped or left alone by the routine can now be determined

→ The below table shows several combinations of values for the two looping terms

→ 'n_1' is the number of times the inner loop will be taken.

→ 'n_2' is the number of times the outer loop will be taken.

Table: Result of push/pop Graph Analysis

N_1	N_2	push/pop
0	0	$p+p^2$
0	1	$p+p^2+p^3+p^4$
0	2	$\sum_{i=1}^{6} p^i$
0	3	$\sum_{i=1}^{8} p^i$
1	0	$1+H$
1	1	$\sum_{i=0}^{3} H^i$
1	2	$\sum_{i=0}^{5} H^i$
1	3	$\sum_{i=0}^{7} H^i$
2	0	H^2+H^3
2	1	$\sum_{i=4}^{7} H^i$
2	2	$\sum_{i=6}^{11} H^i$

→ These expressions states that the stack will be popped only if the inner loop is not taken.

The stack will be left alone only if the inner loop is iterated once, but it may also be pushed.

→ For all other values of the inner loop, the stack will only be pushed.

* Exactly the same arithmetic tables are used for GET/RETURN a buffer block or resource.

The arithmetic tables for GET/RETURN are

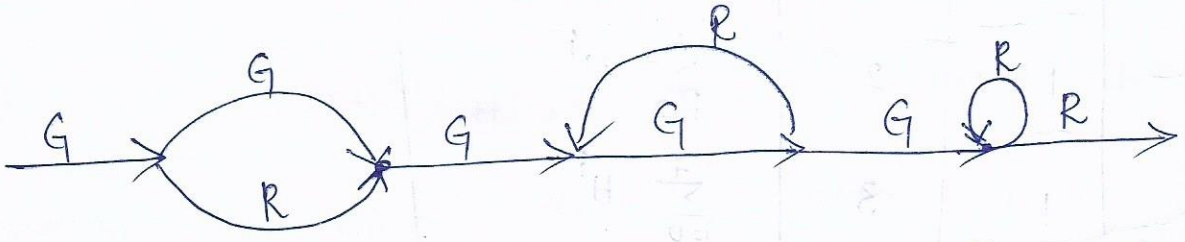
X	G	R	1
G	G^2	1	G
R	1	R^2	R
1	G	R	1

GET/RETURN multiplication table

+	G	R	1
G	G	G+R	G+1
R	G+R	R	R+1
1	G+1	R+1	1

GET/RETURN Addition Table.

→ Let us consider example



The path expression for the above flowgraph is

$$G(G+R)\ G(GR)^*\ G(GR^*R)$$

→ By using arithmetic rules and tables, we get

$\Rightarrow G(G+R)\ G\ (GR)^*\ G\ (GR^*R)$

$\Rightarrow G(G+R)\ G^2R\ G(1)^*\ G(GR^*R)$ $(\because G \cdot R = 1)$

$\Rightarrow G(G+R)\ G\ G\ (GR^*\ R)$

$\Rightarrow G(G+R)\ G^2(GR^*\ R)$

$\Rightarrow G(G+R)\ G^3\ R^*\ R$

$\Rightarrow (G+R)\ G^3\ R^*$ $(\because G \cdot R = 1)$

$\Rightarrow (G^4\ R^*) + (G^3\ R\ R^*)$

$\Rightarrow (G^4 R^*) + (G^2 \cdot G \cdot R \cdot R^*)$

$\Rightarrow G^4 R^* + G^2 R^*$

$\Rightarrow (G^4 + G^2) R^*$

→ This expression specifies the condition under which the resources will be balanced on leaving the routine.

Limitations and solutions

→ The Node-by-Node reduction procedure, and most graph theory based algorithms work well when all paths are possible, but may provide misleading results when some paths are unachievable.

→ The approach to handling unachievable paths is to partition the graph into subgraphs so that all paths in each of the subgraph are achievable.

↓ Each predicate truth functional value to potentially splits the graph into two subgraphs.

For n predicates there could be as many as 2^n subgraphs.

Here is the algorithm for one predicate:-

1. Set the value of the predicate to TRUE and strike out all FALSE links for that predicate.

2. Discard any node, other than entry or exit node that has no incoming links.

3. Repeat step 2 until there are no more links or nodes to discard.

The Resulting graph is the subgraph corresponding to a TRUE predicate value.

4. change "TRUE" to "FALSE" in the above steps and repeat.

* The Resulting graph is the subgraph that corresponds to a FALSE predicate value.

Regular Expression and flow-anamoly Detection

→ The Generic flow-anamoly detection problem is that of looking for a specific sequence of operations considering all possible paths through a routine.

* For example let's say the operations are SET and RESET denoted by s and r respectively and if we want to check if SET followed by a SET or a RESET (i.e, ss or rr sequence)

→ we can easily detect flow-anomaly through regular expressions i.e, to check specific sequence of operations are occured, no need to simplify the expression using arithmetic tables.

→ Here are some more application examples

1. A file can be opened (o), closed (c), read (r) or written (w)

if the file is read or written to after it's been closed, the sequence is non sensical. Therefore

Furthermore oo and cc, though not actual bugs are a waste of time and therefore should also be examined.

2. A tape transport can do rewind (d), fast-forward (f), read (r), write (w), stop (P) and skip (k).

The following sequences are anamalous df, dr, dw, fd, fr.

3. Regular expressions help to detect dataflow anomalies like dd, dk, kk .etc.,

The Method

→ Annotate each link in the graph, with the appropriate or the null operator 1.

→ Simplify things using laws i-e $a+a=1$, $1^2=1$

A useful theorem by Huang helps simplify things

→ After simplifying, we get a regular expression that denotes all the possible sequences of operators in that graph

Let us consider A, B, C be non empty sets of character strings.

Let T be a two-character string of characters

As an example

$A = pp$, $B = srr$, $c = rp$, $T = ss$

$AB^n c = pp\,(Srr)^n\,rp$

The theorem states that ss will appear in $pp(Srr)^n rp$ if it appears in $pp(srr)^2 rp$. we don't need theorem to see that ss does not appear in the given string.

Let

$A = p + pp + ps$, $B = psr + ps(r+ps)$

→ It is obvious that there is a p^4 sequence in AB^nC?

The theorem states that we have only look at

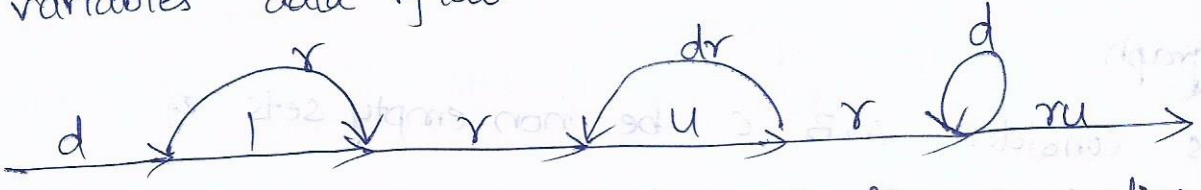
$$(p+pp+ps)\left[psr + ps(r+ps)\right]^2 rp$$

multiplying out the expression and simplifying shows that there is no p^4 sequence.

A further point of Huang theorem is direct useful in test design. If you substitute $1+x^2$ for every expression of the form x^* the paths result from this substitution are sufficient to determine whether a given two character sequen exists or not.

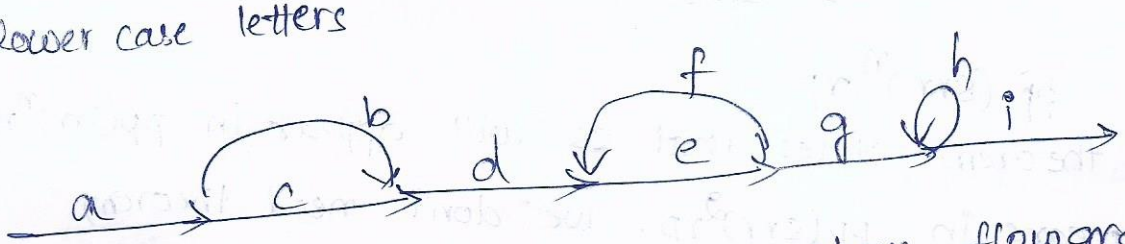
A Data flow Testing Example

Consider a flowgraph annotated with one variables data flow



It is difficult to find data flow anomalies of all variables in a routine.

So annotate the flowgraph with general ital lower case letters



The path expression for the above flowgraph is

$$a(b+c)\,d\,(ef)^*\,egh^*\,i$$

* After substituting the data flow of a particular variable we get the regular expression is

$$d(r+1)r\,(udr)^*\,urd^*\,ru$$

→ Here we are simplifying the regular expression using "Huang theorem" we get

$$d(r+1)r\,(udr)^*\,urd^*\,ru$$

$$\Rightarrow (dr+d)r\,(udr)^*\,urd^*\,ru$$

$$\Rightarrow (drr+dr)\left[1+(udr)^2\right]ur[1+d^2]ru$$

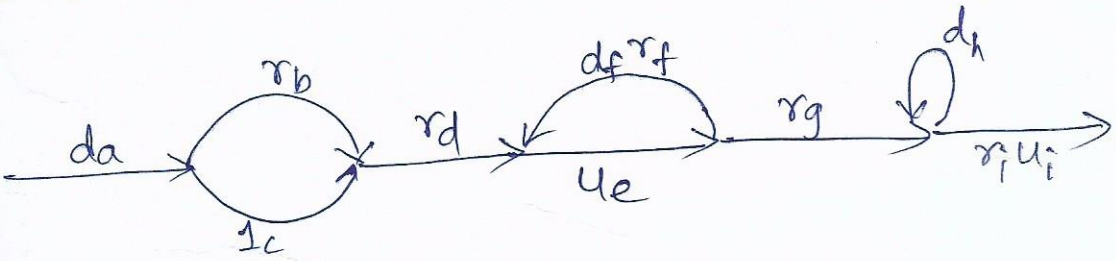
$$\Rightarrow (drr+dr)\left[1+(udr)^2\right]\left[ur+urd^2\right]ru$$

$$\Rightarrow (drr+dr)\left[1+udr\,udr\right]\left[urru+urd^2ru\right]$$

i-1 Huang theorem
$X^* = 1+X^2$

→ From this, we can detect the data flow anomalies of the variable of our interest but we can not find the path where the anomaly has occured

→ A better way to do this is to subscript the operator with the link name. Doing it this way you don't lose the valuable path information i.e,



Regular expression is

$$da(r_b+1_c)\,r_d\,(u_e d_f r_f)^*\,u_e r_g d_h^*\,r_i u_i$$

Apply Huang theorem

$$da(r_b+1_c)\,r_d\,(1+(u_e d_f r_f)^2)\,u_e r_g\,(1+d_h^2)\,r_i u_i$$

$$\Rightarrow (da\,r_b r_d + da\,1_c r_d)\,(u_e r_g + u_e r_g\,u_e d_f r_f\,u_e d_f r_f)$$

The resulting expression tells us the same thing and preserves the pathnames so that we can work back to see what paths are potentially responsible for the bug.

Generalization, Limitations and comments.

1. Huang's Theorem can be easily generalized to cover sequences of greater length than two characters. Beyond 3 characters, though things get complex and this method has probably reached.

2. There are some nice theorems for finding sequences that occur at the beginnings and ends of strings but no nice algorithms for finding strings buried in an expression.

3. The flow-anomaly application for example, does not tell us that there will be a flow anomaly - it tells us whether path is achievable, then there will be flow anomaly.