Module 4
PATHS, PATH PRODUCTS AND REGULAR EXPRESSIONS

Dr. Sandip Mal

SCSE, CSE-Core

Syllabus

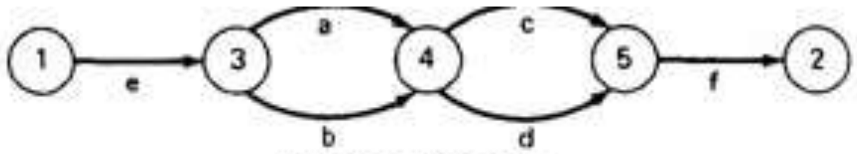
Paths, Path products and Regular expressions: Path products & path expression, reduction procedure, applications, regular expressions & flow anomaly detection. Enumerated data type – Union.

PATHS, PATH PRODUCTS AND REGULAR EXPRESSIONS

- Interpret the control flow graph and identify the path products, path sums and path expressions.
- Identify how the mathematical laws (distributive, associative, commutative etc) holds for the paths.
- Apply reduction procedure algorithm to a control flow graph and simplify it into a single path expression.
- Find the all possible paths (Max. Path Count) of a given flow graph.
- Find the minimum paths required to cover a given flow graph.
- Calculate the probability of paths and understand the need for finding the probabilities.
- Differentiate between Structured and Un-structured flow graphs.
- Calculate the mean processing time of a routine of a given flow graph.
- Understand how complimentary operations such as PUSH / POP or GET / RETURN are interpreted in a flow graph.
- Identify the limitations of the above approaches.
- Understand the problems due to flow-anomalies and identify whether anomalies exist in the given path expression.

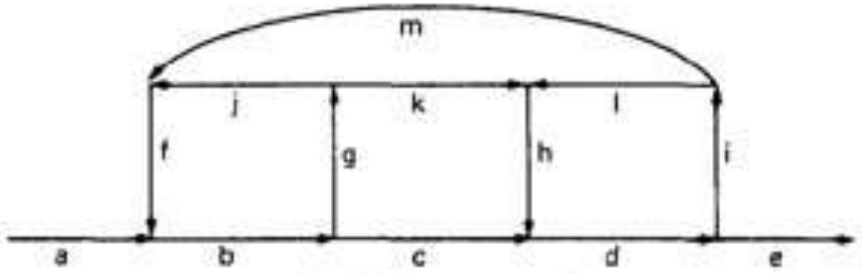
PATH PRODUCTS

- Normally flow graphs used to denote only control flow connectivity.
- The simplest weight we can give to a link is a name.
- Using link names as weights, we then convert the graphical flow graph into an equivalent algebraic like expressions which denotes the set of all possible paths from entry to exit for the flow graph.
- Every link of a graph can be given a name.
- The link name will be denoted by lower case italic letters.
- In tracing a path or path segment through a flow graph, you traverse a succession of link names.
- The name of the path or path segment that corresponds to those links is expressed naturally by concatenating those link names.
- For example, if you traverse links a,b,c and d along some path, the name for that path segment is abcd. This path name is also called a **path product.**



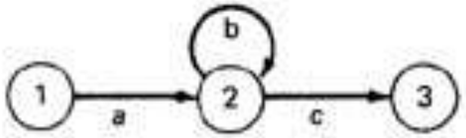
eacf, eadf, ebcf, ebdf

(a)



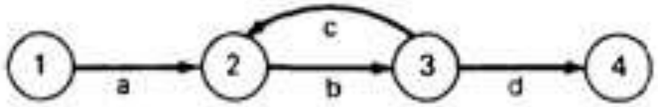
abcde, abgjfbcde, abcdimfbcde

(b)



ac, abc, abbc, abbbe, abbbbc

(c)



abd, abcbd, abcbcbd, abcbcbcbd

(d)

PATH EXPRESSION

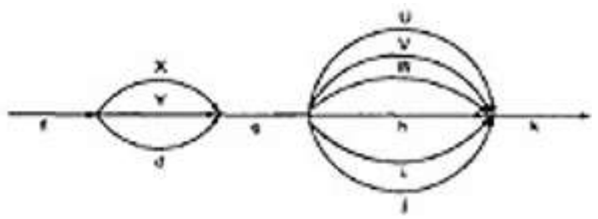
- Consider a pair of nodes in a graph and the set of paths between those node.
- Denote that set of paths by Upper case letter such as X,Y. the members of the path set can be listed as follows:
  - ac, abc, abbc, abbbc, abbbbc.............
- Alternatively, the same set of paths can be denoted by: ac+abc+abbc+abbbc+abbbbc+...........
- The + sign is understood to mean "or" between the two nodes of interest, paths ac, or abc, or abbc, and so on can be taken.
- Any expression that consists of path names and "OR"s and which denotes a set of paths between two nodes is called a "**Path Expression.**".

PATH PRODUCTS

- The name of a path that consists of two successive path segments is conveniently expressed by the concatenation or **Path Product** of the segment names.
- For example, if X and Y are defined as X=abcde,Y=fghij, then the path corresponding to X followed by Y is denoted by XY=abcdefghij,
- Similarly,YX=fghijabcde aX=aabcde Xa=abcdea XaX=abcdeaabcde
- If X and Y represent sets of paths or path expressions, their product represents the set of paths that can be obtained by following every element of X by any element of Y in all possible ways. For example, X = abc + def + ghi Y = uvw + z Then, XY = abcuvw + defuvw + ghiuvw + abcz + defz + ghiz
- If a link or segment name is repeated, that fact is denoted by an exponent. The exponent's value denotes the number of repetitions:$a^1 = a$; $a^2 = aa$; $a^3 = aaa$; $a^n = aaaa \ldots$ n times. Similarly, if X = abcde then X^1 = abcde X^2 = abcdeabcde = $(abcde)^2$ X^3 = abcdeabcdeabcde = $(abcde)^2$ abcde = abcde$(abcde)^2$ = $(abcde)^3$
- The path product is not commutative (that is XY!=YX).
- The path product is Associative.
  RULE 1: A(BC)=(AB)C=ABCwhere A,B,C are path names, set 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 - that is, the path that doesn't have any links. $a^0 = 1$ $X^0 = 1$

PATH SUMS

- The "+" sign was used to denote the fact that path names were part of the same set of paths.
- The "PATH SUM" denotes paths in parallel between nodes.
- Links a and b in figure1 are parallel paths and are denoted by a + b. Similarly, links c and d are parallel paths between the next two nodes and are denoted by c + d.
- The set of all paths between nodes 1 and 2 can be thought of as a set of parallel paths and denoted by eacf+eadf+ebcf+ebdf.
- If X and Y are sets of paths that lie between the same pair of nodes, then X+Y denotes the UNION of those set of paths. For 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 paths in this flow graph is f(X + Y + d)g(U + V + W + h + i + j)k
- The path is a set union operation, it is clearly 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 distributive, and the ordinary rules of multiplication apply; that is
  - **RULE 4: A(B+C)=AB+AC and (B+C)D=BD+CD**
  - Applying these rules to the below
  - 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, then the union of these sets is unchanged; consequently,
  - **RULE 5: X+X=X (Absorption Rule)**
  - If a set consists of paths names and a member of that set is added to it, the "new" name, which is already in that set of names, contributes nothing and can be ignored.
  - For example,if X=a+aa+abc+abcd+def then
  - X+a = X+aa = X+abc = X+abcd = X+def = X
  - It follows that any arbitrary sum of identical path expressions reduces to the same path expression.

RULES 6 - 16:

The following rules can be derived from the previous rules:

RULE 6: $X^{\underline{n}} + X^{\underline{m}} = X^{\underline{n}}$ if n>m **RULE 6:** $X^{\underline{n}} + X^{\underline{m}} = X^{\underline{m}}$ if m>n

RULE 7: $X^nX^m = X^{\underline{n+m}}$

RULE 8: $X^nX^* = X^*X^n = X^*$

RULE 9: $X^nX^+ = X^+X^n = X^+$

RULE 10: $X^*X^+ = X^+X^* = X^+$

RULE 11: $1 + 1 = 1$

RULE 12: $1X = X1 = X$

Following or preceding a set of paths by a path of zero length does not change the set.

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

No matter how often you traverse a path of zero length, It is a path of zero length.

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

The null set of paths is denoted by the numeral 0. it obeys the following rules

RULE 15: $X+0=0+X=X$

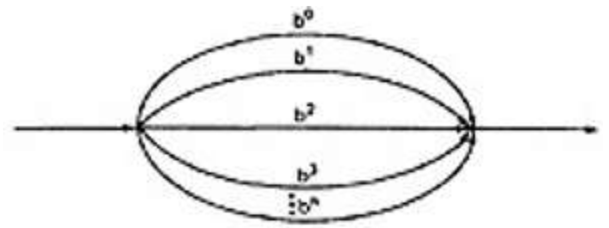
RULE 16: $0X=X0=0$

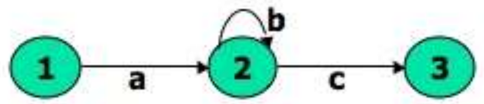
If you block the paths of a graph for or aft by a graph that has no paths , there wont be any paths.

LOOPS

- Loops can be understood as an infinite set of parallel paths. Say that the loop consists of a single link b. then the set of all paths through that loop point is b0+b1+b2+b3+b4+b5+.............

- This potentially infinite sum is denoted by b* for an individual link and by X* when X is a path expression.





- The path expression for the above figure is denoted by the notation: ab*c=ac+abc+abbc+abbbc+................
- aa*=a*a=a+ and XX*=X*X=X+
- It is more convenient to denote the fact that a loop cannot be taken more than a certain, say n, number of times.

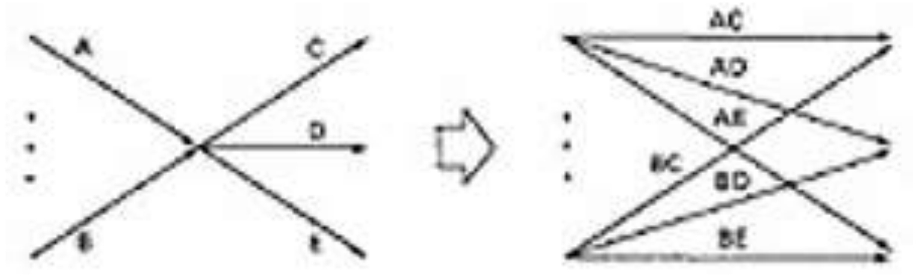
REDUCTION PROCEDURE

- This section presents a reduction procedure for converting a flow graph whose links are labelled with names into a path expression that denotes the set of all entry/exit paths in that flow graph. The procedure is a node-by-node removal algorithm.
- The steps in Reduction Algorithm are as follows:
  1. Combine all serial links by multiplying their path expressions.
  2. Combine all parallel links by adding their path expressions.
  3. Remove all self-loops (from any node to itself) by replacing them with a link of the form X*, where X is the path expression of the link in that loop.

     STEPS 4 - 8 ARE IN THE ALGORIHTM'S LOOP:
  4. Select any node for removal other than the initial or final node. Replace it with a set of equivalent links whose path expressions correspond to all the ways you can form a product of the set of in links with the set of out links of that node.
  5. Combine any remaining serial links by multiplying their path expressions.
  6. Combine all parallel links by adding their path expressions.
  7. Remove all self-loops as in step 3.
  8. 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 flow graph; otherwise, return to step 4.
- A flow graph can have many equivalent path expressions between a given pair of nodes; that is, there are many different ways to generate the set of all paths between two nodes without affecting the content of that set.
- The appearance of the path expression depends, in general, on the order in which nodes are removed.

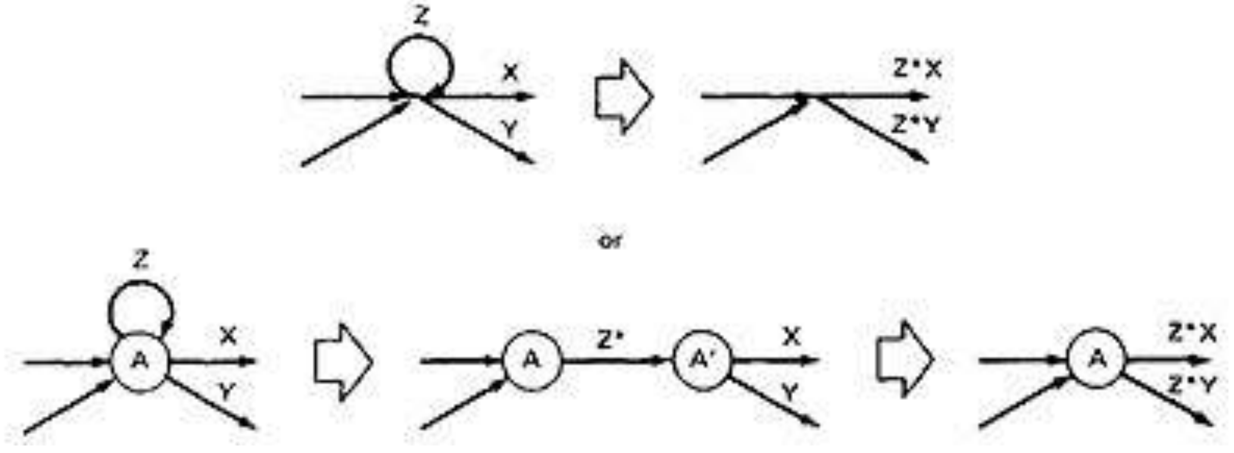
CROSS-TERM STEP

- The cross - term step is the fundamental step of the reduction algorithm.
- It removes a node, thereby reducing the number of nodes by one.
- Successive applications of this step eventually get you down to one entry and one exit node. The following diagram shows the situation at an arbitrary node that has been selected for removal:
- From the diagram, one can infer: (a + b)(c + d + e) = ac + ad + + ae + bc + bd + be



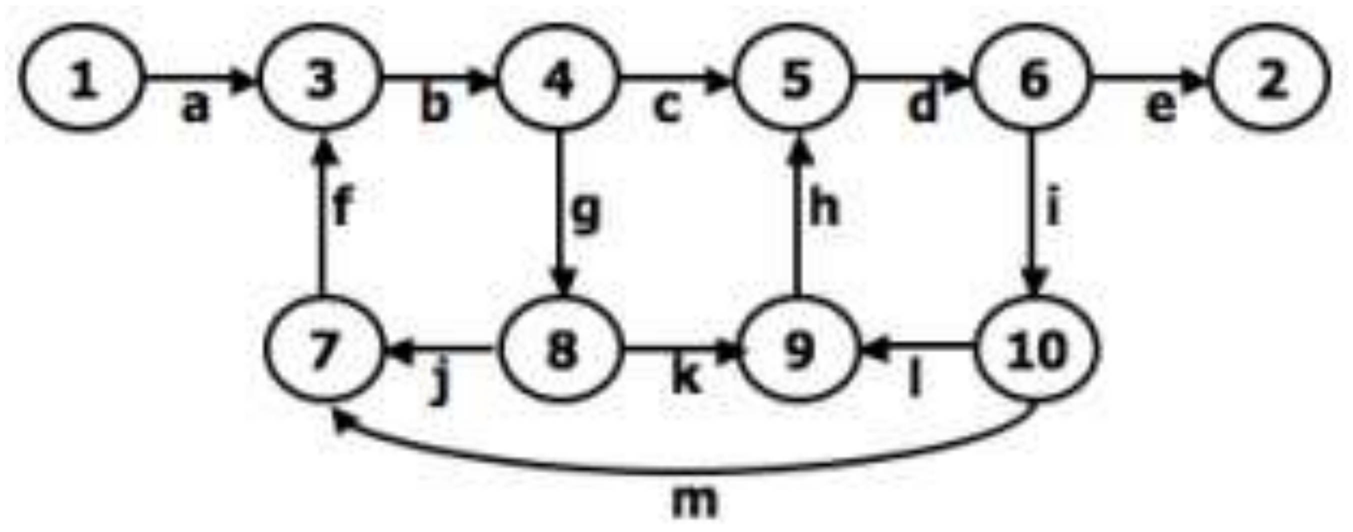
LOOP REMOVAL OPERATIONS

- There are two ways of looking at the loop-removal operation:
- In the first way, we remove the self-loop and then multiply all outgoing links by Z*.
- In the second way, we split the node into two equivalent nodes, call them A and A' and put in a link between them whose path expression is Z*. Then we remove node A' using steps 4 and 5 to yield outgoing links whose path expressions are Z*X and Z*Y.

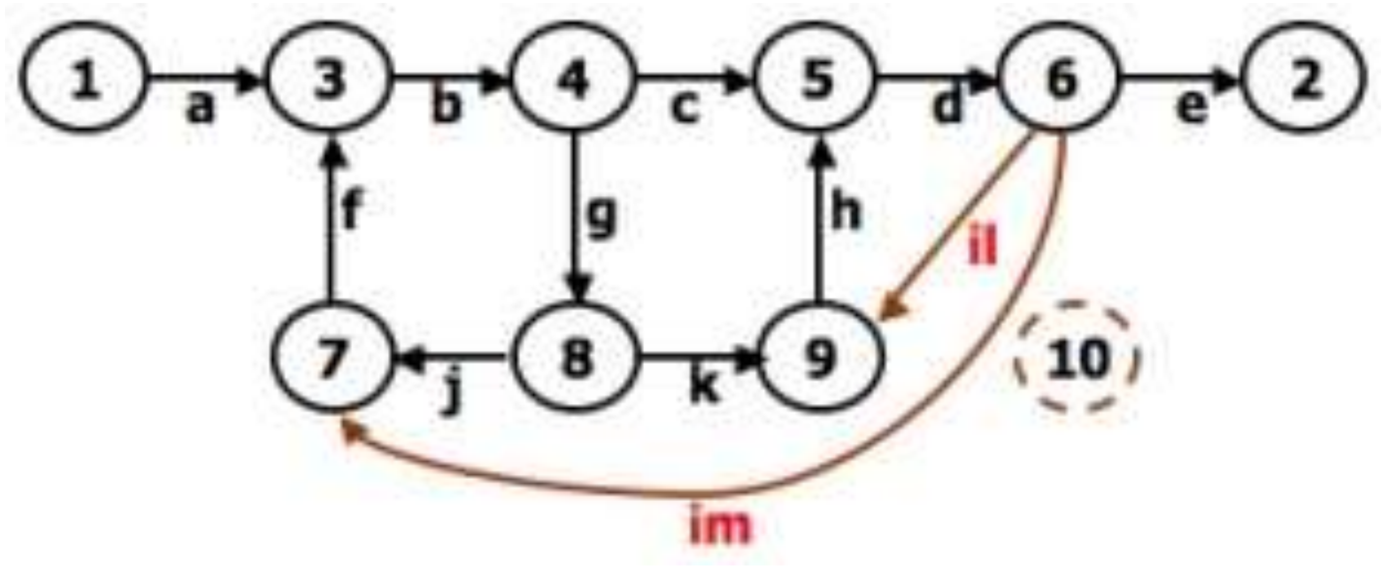


A REDUCTION PROCEDURE - EXAMPLE

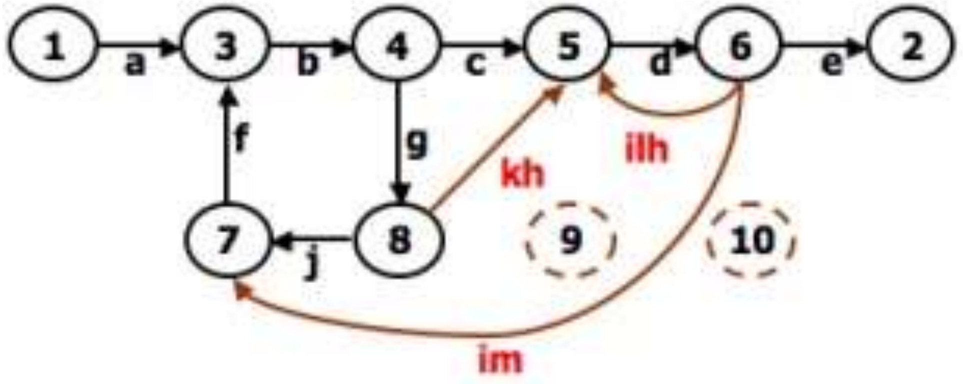
- Let us see by applying this algorithm to the following graph where we remove several nodes in order; that is



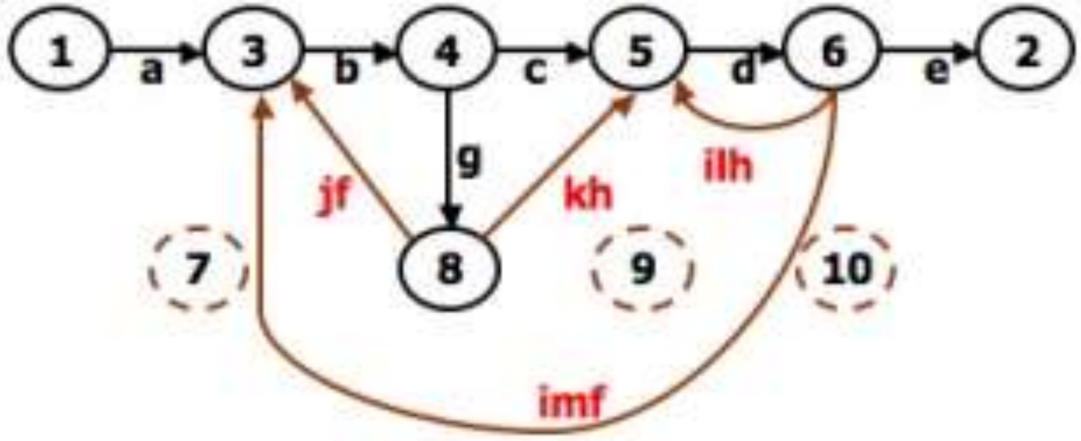
- Remove node 10 by applying step 4 and combine by step 5 to yield



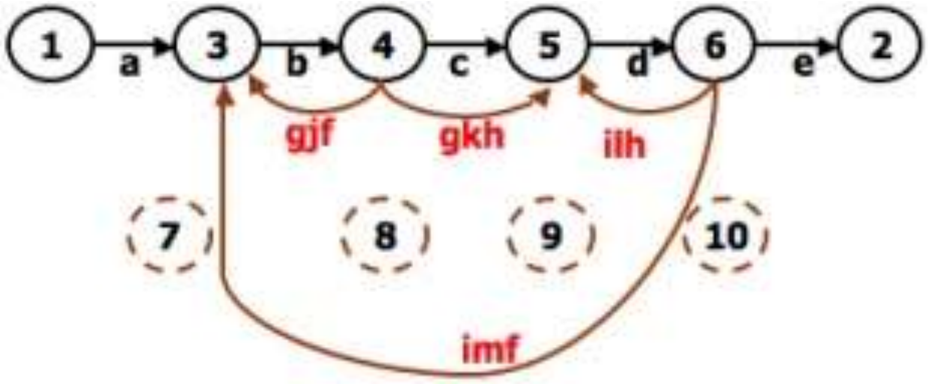
- Remove node 9 by applying step4 and 5 to yield



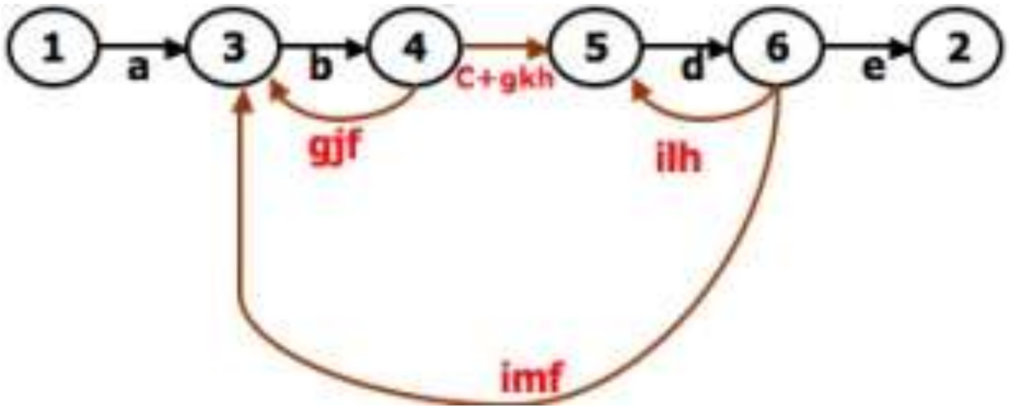
- Remove node 7 by steps 4 and 5, as follows:



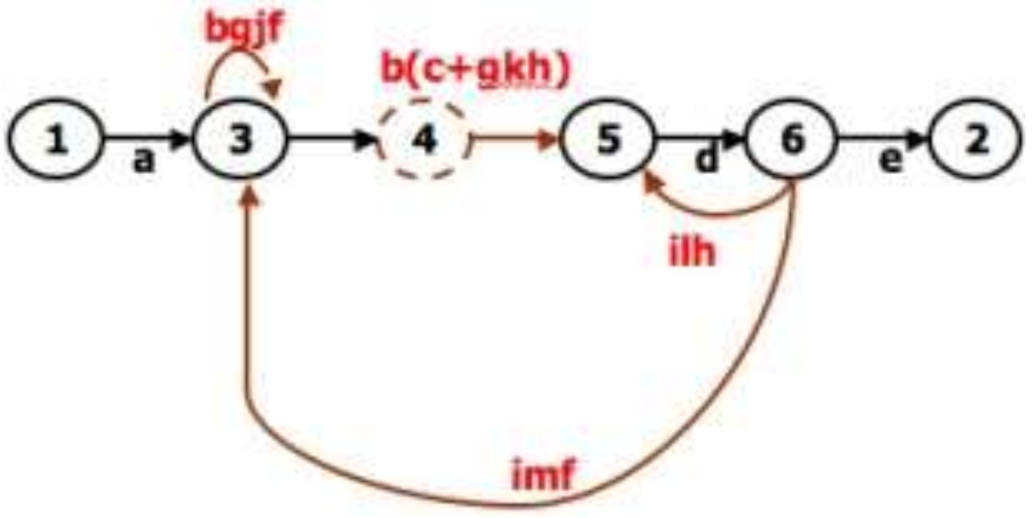
- Remove node 8 by steps 4 and 5, to obtain:



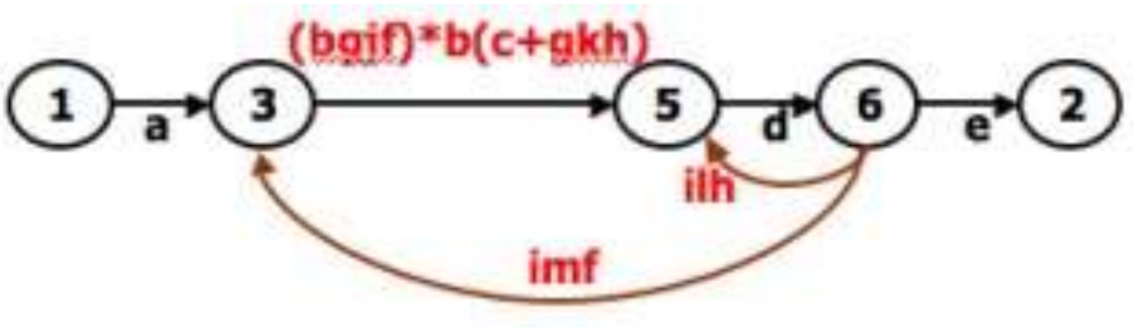
- **PARALLEL TERM (STEP 6):**
- Removal of node 8 above led to a pair of parallel links between nodes 4 and 5. combine them to create a path expression for an equivalent link whose path expression is c+gkh; that is



- **LOOP TERM (STEP 7):**
- Removing node 4 leads to a loop term. The graph has now been replaced with the following equivalent simpler graph:



- Continue the process by applying the loop-removal step as follows:



- Removing node 5 produces:

$$\begin{tikzcd}
1 \arrow[r, "a"'] & 3 \arrow[rr, "(bgif)^*b(c+gkh)d"] & & 6 \arrow[ll, bend left, "imf"] \arrow[loop above, "ilhd"] \arrow[r, "e"'] & 2
\end{tikzcd}$$

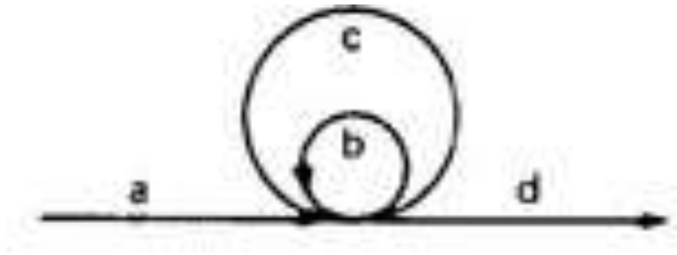
- Remove the loop at node 6 to yield:

$$\begin{tikzcd}
1 \arrow[r, "a"'] & 3 \arrow[rr, "(bgif)^*b(c+gkh)d"] & & 6 \arrow[ll, bend left, "(ilhd)^*\ imf"] \arrow[r, "(ilhd)^*e"] & 2
\end{tikzcd}$$

- Remove node 3 to yield:

$$\begin{tikzcd}
1 \arrow[rr, "a(bgjf)^*b(c+gkh)d"] & & 6 \arrow[loop below, "(ilhd)^*\ imf\ (bgif)^*b(c+gkh)d"] \arrow[r, "(ilhd)^*e"] & 2
\end{tikzcd}$$

- Removing the loop and then node 6 result in the following expression: a(bgjf)*b(c+gkh)d((ilhd)*imf(bjgf)*b(c+gkh)d)*(ilhd)*e



APPLICATIONS

- The purpose of the node removal algorithm is to present one very generalized concept- the path expression and way of getting it.
- Every application follows this common pattern:
  - Convert the program or graph into a path expression.
  - Identify a property of interest and derive an appropriate set of "arithmetic" rules that characterizes the property.
  - Replace the link names by the link weights for the property of interest. The path expression has now been converted to an expression in some algebra, such as ordinary algebra, regular expressions, or boolean algebra. This algebraic expression summarizes the property of interest over the set of all paths.
  - Simplify or evaluate the resulting "algebraic" expression to answer the question you asked.

HOW MANY PATHS IN A FLOWGRAPH ?

- The question is not simple. Here are some ways you could ask it:
  - What is the maximum number of different paths possible?
  - What is the fewest number of paths possible?
  - How many different paths are there really?
  - What is the average number of paths?

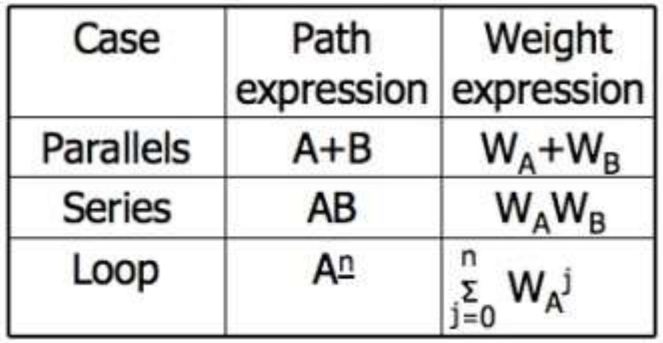
- Determining the actual number of different paths is an inherently difficult problem because there could be unachievable paths resulting from correlated and dependent predicates.

- If we know both of these numbers (maximum and minimum number of possible paths) we have a good idea of how complete our testing is.

- Asking for "the average number of paths" is meaningless.

MAXIMUM PATH COUNT ARITHMETIC

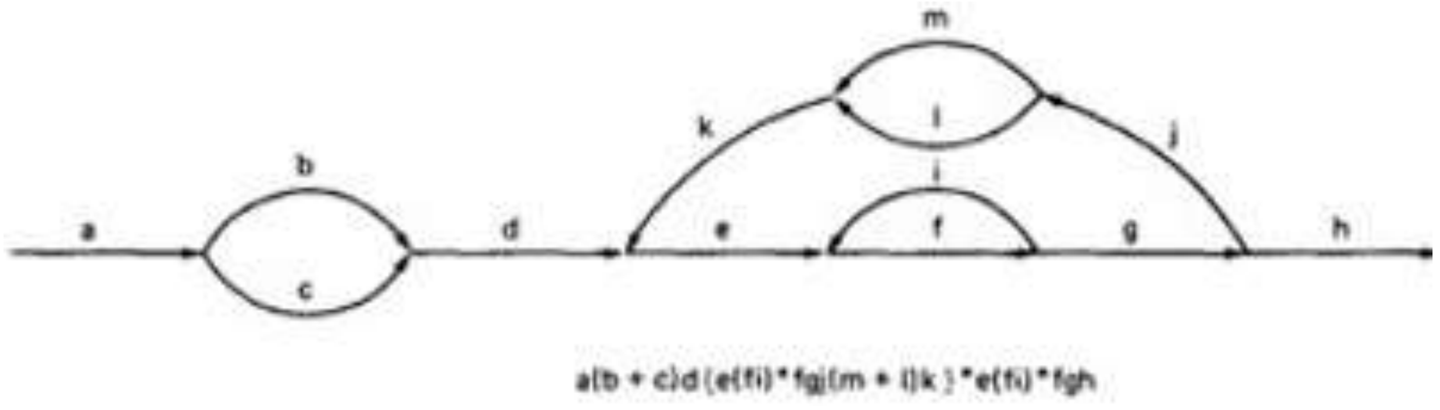
- Label each link with a link weight that corresponds to the number of paths that link represents.
- Also mark each loop with the maximum number of times that loop can be taken. If the answer is infinite, you might as well stop the analysis because it is clear that the maximum number of paths will be infinite.
- There are three cases of interest: parallel links, serial links, and loops.



Case	Path expression	Weight expression
Parallels	A+B	W_A+W_B
Series	AB	W_AW_B
Loop	$A^{\underline{n}}$	$\sum_{j=0}^{n} W_A^j$

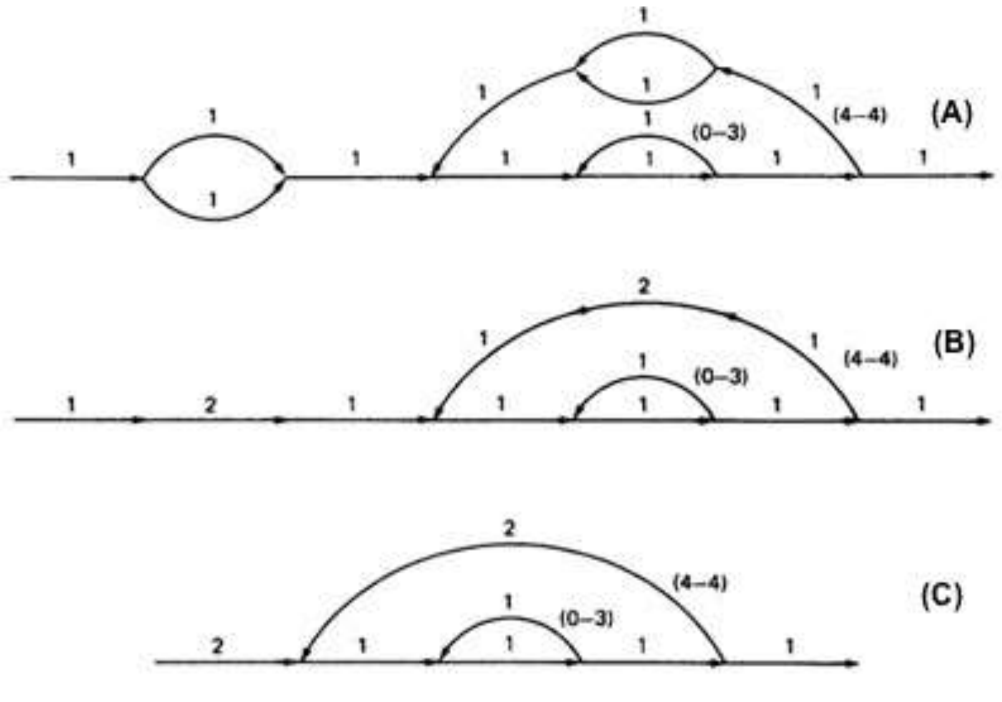
- This arithmetic is an ordinary algebra. The weight is the number of paths in each set.

EXAMPLE

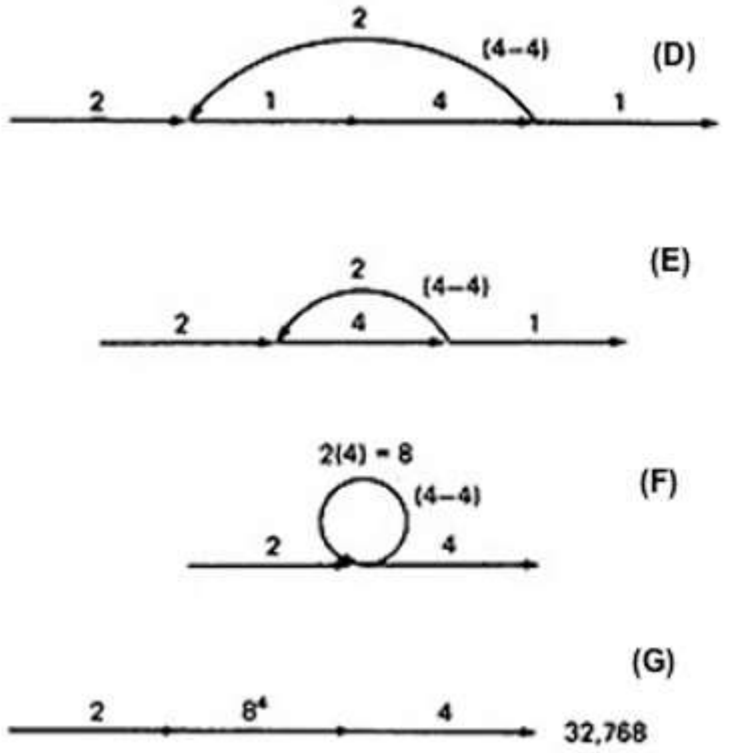


- Each link represents a single link and consequently is given a weight of "1" to start. Lets say the outer loop will be taken exactly four times and inner Loop Can be taken zero or three times Its path expression, with a little work, is:
- Path expression: a(b+c)d{e(fi)*fgj(m+l)k}*e(fi)*fgh

- **A:** The flow graph should be annotated by replacing the link name with the maximum of paths through that link (1) and also note the number of times for looping.
- **B:** Combine the first pair of parallel loops outside the loop and also the pair in the outer loop.
- **C:** Multiply the things out and remove nodes to clear the clutter.



- **For the Inner Loop:**
- **D:**Calculate the total weight of inner loop, which can execute a min. of 0 times and max. of 3 times. So, it inner loop can be evaluated as follows: $1^{\underline{3}} = 1^0 + 1^1 + 1^2 + 1^3 = 1 + 1 + 1 + 1 = 4$
- **E:** Multiply the link weights inside the loop: 1 X 4 = 4
- **F:** Evaluate the loop by multiplying the link weights: 2 X 4 = 8.
- **G:** Simplifying the loop further results in the total maximum number of paths in the flow graph: 2 X 8^4 X 2 = 32,768.



- Alternatively, you could have substituted a "1" for each link in the path expression and then simplified, as follows:

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

$= 1(1 + 1)1(1(1 \text{ x } 1)^{\underline{3}}1 \text{ x } 1 \text{ x } 1(1 + 1)1)^{\underline{4}}1(1 \text{ x } 1)^{\underline{3}}1 \text{ x } 1 \text{ x } 1$

$= 2(1^{\underline{3}}1 \text{ x } (2))^{4}1^{\underline{3}}$

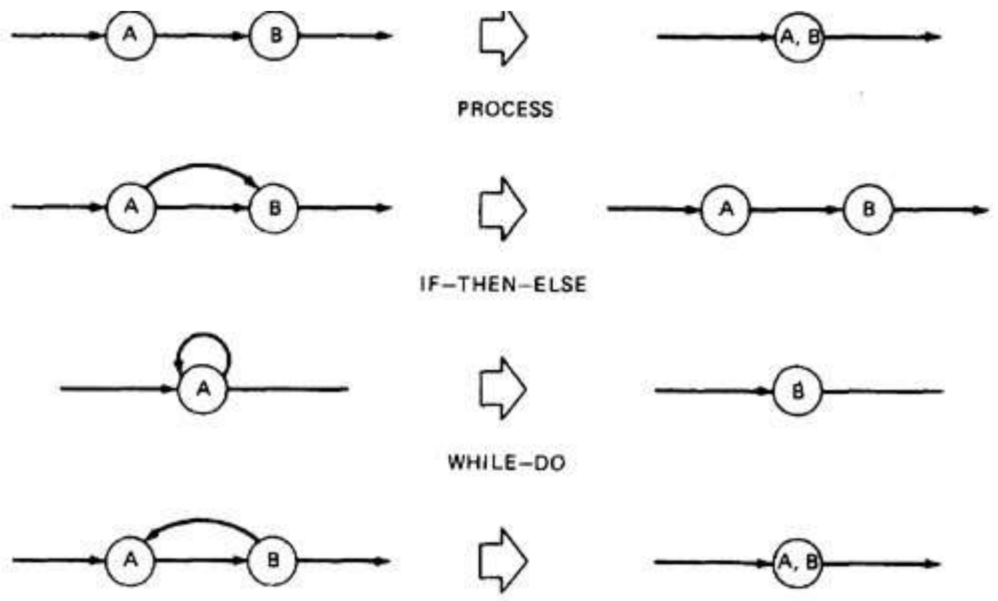
$= 2(4 \text{ x } 2)^{4} \text{ x } 4$

$= 2 \text{ x } 8^{4} \text{ x } 4 = 32{,}768$

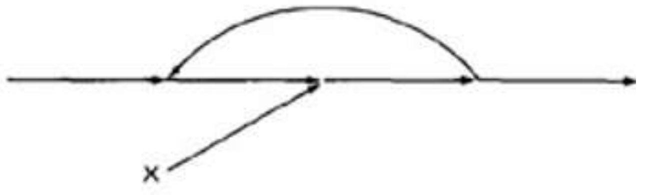
- This is the same result we got graphically.

STRUCTURED FLOWGRAPH

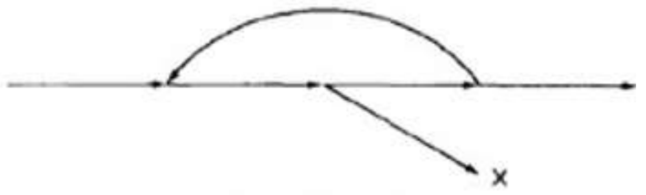
- Structured code can be defined in several different ways that do not involve ad-hoc rules such as not using GOTOs.
- A structured flow graph is one that can be reduced to a single link by successive application of the transformations of Figure.



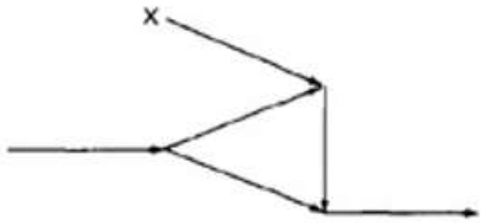
- The node-by-node reduction procedure can also be used as a test for structured code.
- Flow graphs that DO NOT contain one or more of the graphs shown below as subgraphs are structured.
  - Jumping into loops
  - Jumping out of loops
  - Branching into decisions
  - Branching out of decisions



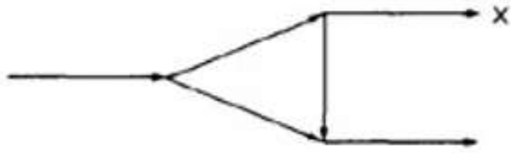
Jumping Into Loops



Jumping Out of Loops



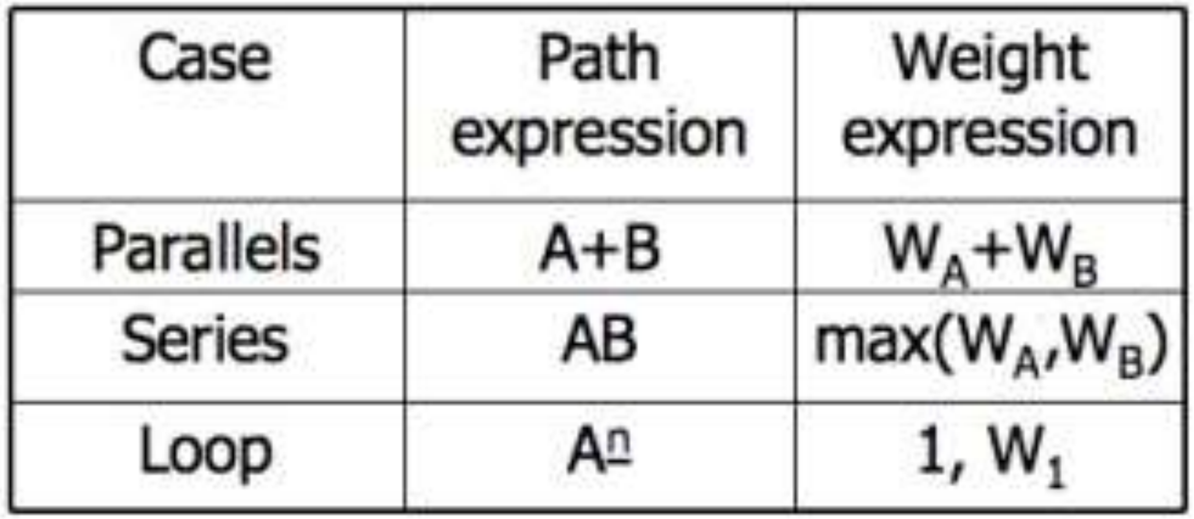
Branching Into Decisions



Branching Out of Decisions

LOWER PATH COUNT ARITHMETIC

- A lower bound on the number of paths in a routine can be approximated for structured flow graphs.
- The arithmetic is as follows:

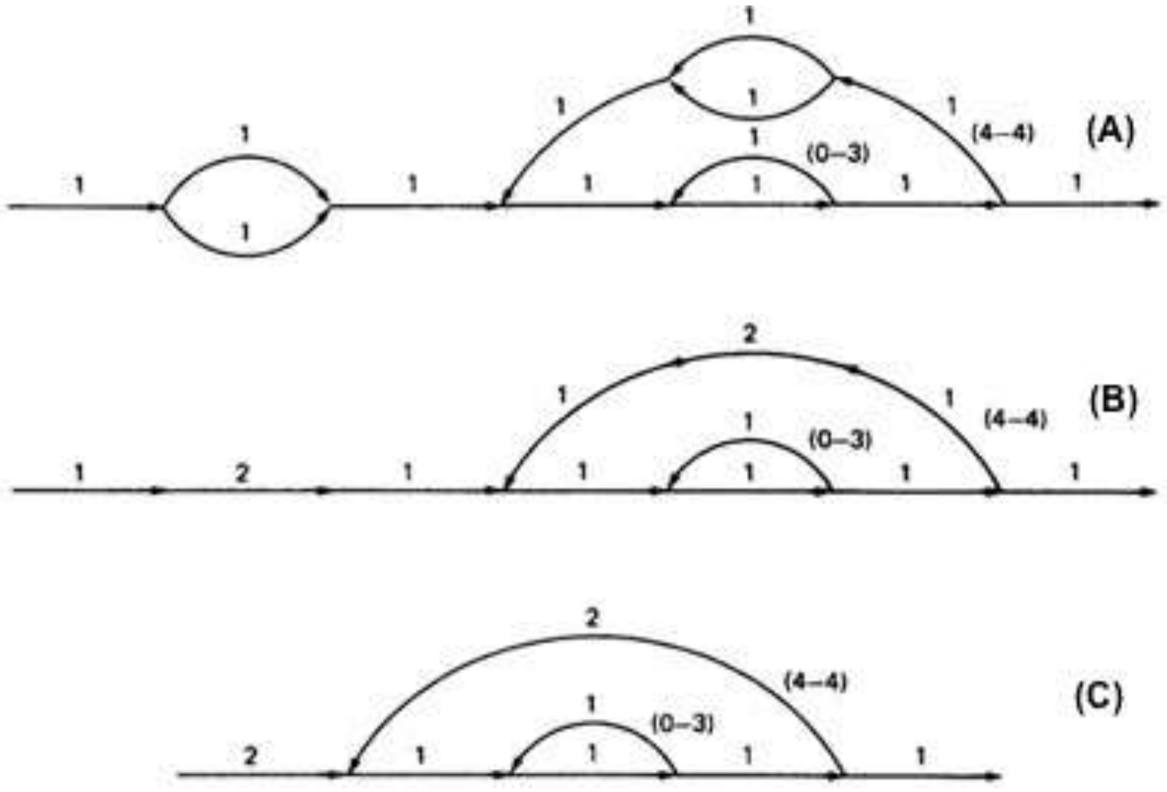


Case	Path expression	Weight expression
Parallels	A+B	W_A+W_B
Series	AB	$\max(W_A,W_B)$
Loop	$A^{\underline{n}}$	$1, W_1$

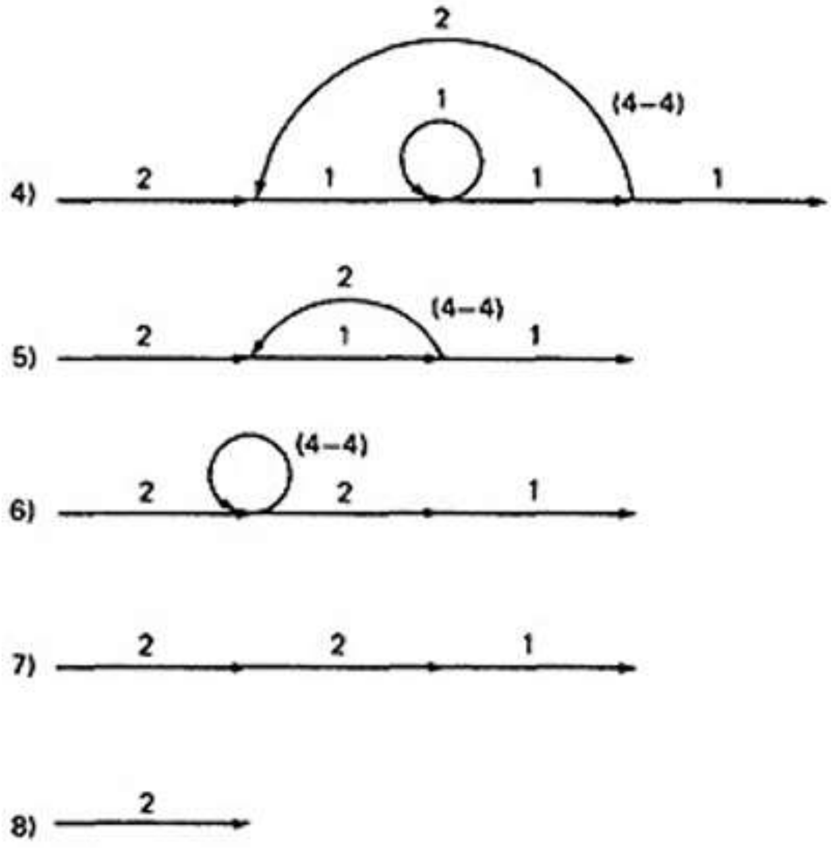
- The values of the weights are the number of members in a set of paths.

EXAMPLE

- Applying the arithmetic to the earlier example gives us the identical steps until step 3 (C) as below:



- From Step 4, the it would be different from the previous example:



- If you observe the original graph, it takes at least two paths to cover and that it can be done in two paths.

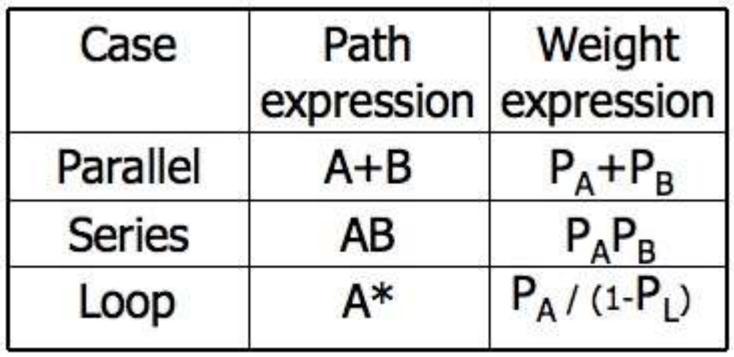
CALCULATING THE PROBABILITY

- Path selection should be biased toward the low - rather than the high-probability paths.

What is the probability of being at a certain point in a routine?

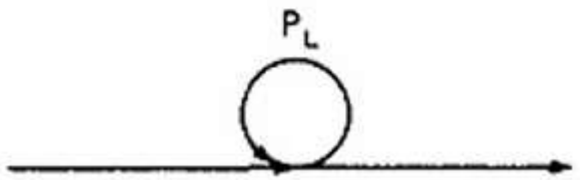
This question can be answered under suitable assumptions, primarily that all probabilities involved are independent, which is to say that all decisions are independent and uncorrelated.

- We use the same algorithm as before : node-by-node removal of uninteresting nodes.
- **Weights, Notations and Arithmetic:**
  - Probabilities can come into the act only at decisions (including decisions associated with loops).
  - Annotate each out link with a weight equal to the probability of going in that direction.
  - Evidently, the sum of the out link probabilities must equal 1
  - For a simple loop, if the loop will be taken a mean of N times, the looping probability is N/(N + 1) and the probability of not looping is 1/(N + 1).
  - A link that is not part of a decision node has a probability of 1.
  - The arithmetic rules are those of ordinary arithmetic.



Case	Path expression	Weight expression
Parallel	A+B	P_A+P_B
Series	AB	P_AP_B
Loop	A*	$P_A / (1-P_L)$

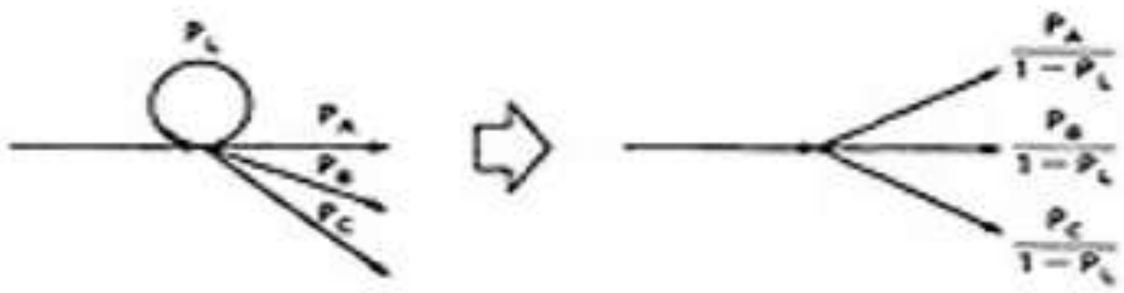
- In this table, in case of a loop, P_A is the probability of the link leaving the loop and P_L is the probability of looping.
- The rules are those of ordinary probability theory.
  - If you can do something either from column A with a probability of P_A or from column B with a probability P_B, then the probability that you do either is $P_A + P_B$.
  - For the series case, if you must do both things, and their probabilities are independent (as assumed), then the probability that you do both is the product of their probabilities.
- For example, a loop node has a looping probability of P_L and a probability of not looping of P_A, which is obviously equal to 1 - PL.



$$P_A = 1 - P_L$$

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

- Following the above rule, all we've done is replace the outgoing probability with 1 - so why the complicated rule? After a few steps in which you've removed nodes, combined parallel terms, removed loops and the like, you might find something like this:



- because $P_L + P_A + P_B + P_C = 1$, $1 - P_L = P_A + P_B + P_C$, and

  which is what we've postulated for any decision. In other words, division by $1 - P_L$ renormalizes the outlink probabilities so that their sum equals unity after the loop is removed.