ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (296-300), Month: April - June 2015, Available at: www.researchpublish.com

Algorithm for Keyword Search on an Execution Path

Adarsh Pandiri[1], Tarik Eltaeib[2]

Abstract: **This paper introduces a code-seek technique, which incorporates a calculation of catchphrase code-pursuit and a model execution. In this paper, a question is a situated of magic words and a Query item is a situated of execution ways satisfying the question, that is, each of the execution ways incorporates the greater part of the essential words. Here, an execution way speaks to one of all levels of strategy calls of all conceivable element dispatches in an OO program; in this way, numerous execution ways can be created even from a little program. The calculation chips away at an information structure named an And/Or/Call chart, which is a smaller representation of execution ways. The model execution quests names of routines or sorts, alternately words in string literals from Java source code.**

Keywords: **code-seek technique, Algorithm for Keyword Search on an Execution Path.**

I. INTRODUCTION

Different sorts of seeking have been presented in programming advancement. These incorporate looking a part which actualizes a requested capacity, looking code sections like a code piece which a designer is wanting to alter, or looking a specimen code to comprehend the utilization of a class or a part. Then again, the development of programming dialects have been acknowledging fine-grained modules, for example, system, conclusion, object, legacy, angle, or decorator. A source code can be part into little pieces coupling to one another with these dialect characteristics, and therefore, in the seeking said over, a query item is liable to turn into a set of various modules or source documents coupling to one another The Java Pathfinder is a static, formal model checking instrument to discover the execution ways that damage some declarations or reason halt. Likewise, an analysis suggests a situation where a sort of engineer's undertaking may get to be more troublesome by actualizing the same project with more fine-grained modules.

To defeat the fine-grained module (or split code) issue in code looking, this paper shows a seeking calculation to distinguish an arrangement of code parts which are part in source code, however are on the same execution way (hence, coupling to one another with some dialect characteristic). The proposed calculation likewise permits an engineer to determine words in string literals as a question, notwithstanding naming routines or sorts.

II . AND/OR/CALL GRAPH

An And/Or/Call diagram is a chart based information representation, which develops And/or diagram, utilized broadly as a part of a space of programming building, e.g., displaying highlights in item.[1]

A. Definition:

An And/Or/Call chart is a DAG (coordinated non-cyclic diagram) where every hub is one of an And hub , Or hub or Call hub. These three sorts of hub exhibits the accompanying structures in a system. Figure 4 demonstrates an And/Or/Call chart extricated from a Java program in Figure 3, which was utilized as an inquiry focus in Figure 1.

An and hub speaks to a succession structure of Structured Programming [2]. E.g, A piece of a project executes sentences Formula, and Formula in a specific order. Such a structure is mapped onto an and hub having three kid hubs Formula, and Formula. An Or hub speaks to a determination structure of Structured Programming. E.g, An if-proclamation executes either sentence st or se. Such a structure is mapped onto an Or hub having two youngster hubs st and se. The dreary

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (296-300), Month: April - June 2015, Available at: www.researchpublish.com

structure of Structured Programming (i.e., a circle) is deteriorated into a choice of the arrangements that contain (preferably) discretionary times redundancy of sentences inside the circle in an And/Or/Call chart. Then again, practically speaking, such reiteration is cut into 0 or 1 time, which is sufficient to code looking as its motivation. The Call hub speaks to a system (method) call with element dispatch, that is, a strategy body is chosen in runtime from a percentage of the applicant strategies. E.g. A piece of a project incorporates a technique call of interface Formula's method Formula and a system Formula has two strategy bodies, one of class Formula and the other of class Formula. Such a strategy call is mapped onto a Call hub having two tyke hubs Formula and Formula. Here, the hub name, for example, Formula remains for a class (or an interface) Formula's technique Formula. A Call hub is an augmentation of an Or hub, as in a Call hub speaks to a choice (in runtime) of the system bodies to be executed.

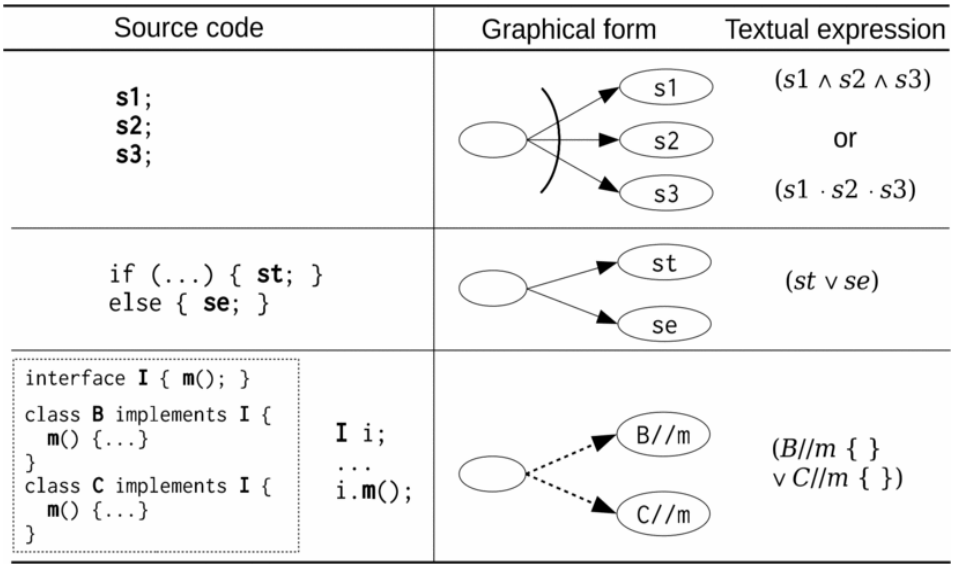

Fig. 1 Mapping from Source Code Structures to And/Or/Call Graph Nodes

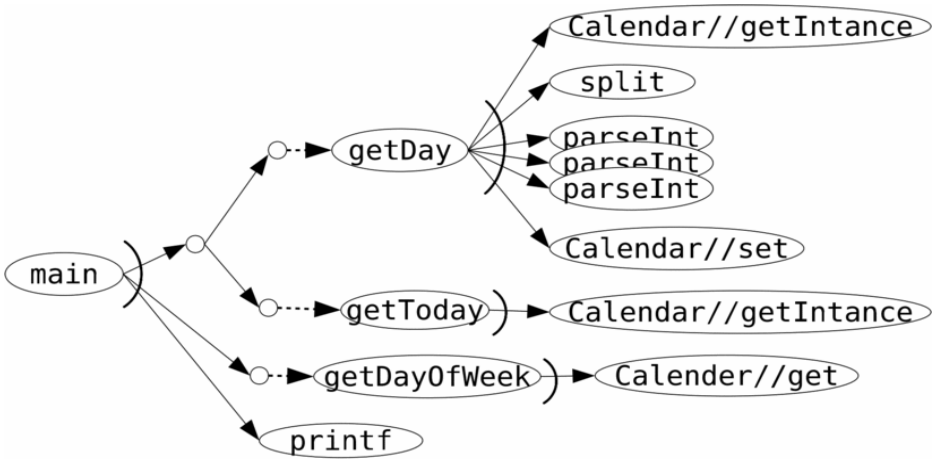

Fig. 2 An And/Or/Call Graph

A diagram is produced from each of the root hubs, that is, section purposes of a system. On account of a Java program, a common section point is a primary system. Amid a diagram era, when a system incorporates recursive calls, such a chain of recursive calls is recognized and cut onto single (profundity) call, the same as circles, to evade the chart having an unending profundity.

Note that in the above mappings, the request/duplication of tyke hubs is kept for every hub. Something else, if the design is just to figure out if there are the execution ways including a given arrangement of strategies (no compelling reason to find where they are called or in which arrange they are called), then youngster hubs are uninhibitedly reordered and blended when copied. SAT(satisfiability issue) calculations, for example, [3] are pertinent for this reason, by supplanting every Call hub with an Or hub and considering a sub-tree of an And/Or/Call chart as a Boolean recipe.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (296-300), Month: April - June 2015, Available at: www.researchpublish.com

III. KEYWORD- SEARCH ALGORITHM

This area depicts an algorithm to recognize the subgraphs of an And/Or/Call diagram that incorporate magic words in a given question and are persistent as far as execution way depicted already. In fact, when an And/Or/Call diagram is a tree, such a sub-chart is a tree cut of a sub-tree of the tree.

A. Label and Node Rundown:

Every immediate tyke node of a Call node has a label, which speaks to a called technique body or estimations of contentions. (Here, to improve the dialog without relinquishing consensus, we need a suspicion; to be specific, that every system has an one of a kind name in a project, and is hence recognized just by its name from alternate techniques.) An inquiry question is a situated of labels. Every node of an And/Or/Call diagram has a rundown.[4] A node's rundown is a situated of labels of tyke and relative nodes of the node. As such, a rundown of node n incorporates node c's label, iff c is joined by a way beginning at n furthermore the way's length 1. E.g., on account of Figure 4, a rundown of the node getDay is the accompanying: f getInstance, part, parseInt, set g. For any node n and any of its relative nodes d, an outline of n is a super situated of the outline of d.

B. Algorithm:

A catchphrase question seek on an And/Or/Call diagram is characterized utilizing a label presented already as takes after: For a given And/Or/Call chart G produced from a system and a given question Q = fl1; l2; :::g as a set of labels, discover the associated sub-diagrams R of G, where an arrangement of labels of R's nodes is a super situated Q, at the end of the day, every sub-chart in a query output R satisfies the inquiry Q.

```
▷ G is an And/Or/Call graph.
▷ S(n) is a mapping from node n to n's summary.
▷ Q is a set of labels of query.

proc keyword_search(Q, G, S) {
  R := an empty set; ▷ search result, a set of treecuts
  if S(root node of G) ⊉ Q { stop ; }
  else {
    STs := get_local_deepest_subtrees(G, Q); ▷ S1
    for each ST in STs {
      TC := get_shallowest_treecut(ST, Q); ▷ S2
      remove uncontributing leaves from TC; ▷ S3
      update R by adding TC;
    }
  }
  return R;
}

proc get_local_deepest_subtrees(ST, Q) {
  DSTs := an empty set; ▷ deepest subtrees
  r := a root node of ST;
  if S(r) ⊇ Q {
    for each subnode sn of r {
      if S(sn) ⊇ Q {
        SST := subgraph of ST where the root is sn;
        update DSTs by adding get_local_deepest_subtrees(SST, Q);
      }
    }
    if DSTs is empty { update DSTs by adding G; }
  }
  return DSTs;
}

proc get_shallowest_treecut(ST, Q) {
  r := a root node of ST;
  for d := 1, 2, ... {
    Ns := a set of nodes in ST of which distance from r ≤ d;
    TC := a subgraph of ST spanned by nodes Ns;
    if S(root node of TC) ⊇ Q { return TC; }
  }
}
```

Fig. 3 Pseudo Code for Keyword Search Algorithm

Fig 3 shows Algoithm search. This calculation first (S1) discovers question satisfying sub-trees having the profundities, then (S2) makes a treecut in each of the sub-trees by uprooting the more profound leaf (hubs further from the root hub of the sub-tree) over and over until the treecut no more satisfies the inquiry, and finally (S3) uproots the leaf hubs that don't have names coordinating one of the question watchwords. Note that step (S2) evacuates the leaf hubs coordinating the question, while step (S3) won't. Thus every leaf hub of the subsequent tree cuts matches one of the inquiry decisive words.

C. Search output as an Execution Path:

Everything of a query output has the capacity grow to execution ways in the same path as II-B, on the grounds that such a thing is a tree cut of a sub-tree of Formula, along these lines is an And/Or/Call diagram. Be that as it may, all in all, a situation where the majority of the created ways from a question satisfying sub-diagram mostly fill the inquiry however don't satisfy the inquiry happens. E.g., a pivotal word Formula of a question is incorporated just by some youngster hub of an Or hub and the other catchphrase Formula is incorporated just by one of the other kid hub of the same Or hub, so that one execution way can exclude both Formula and Formula.

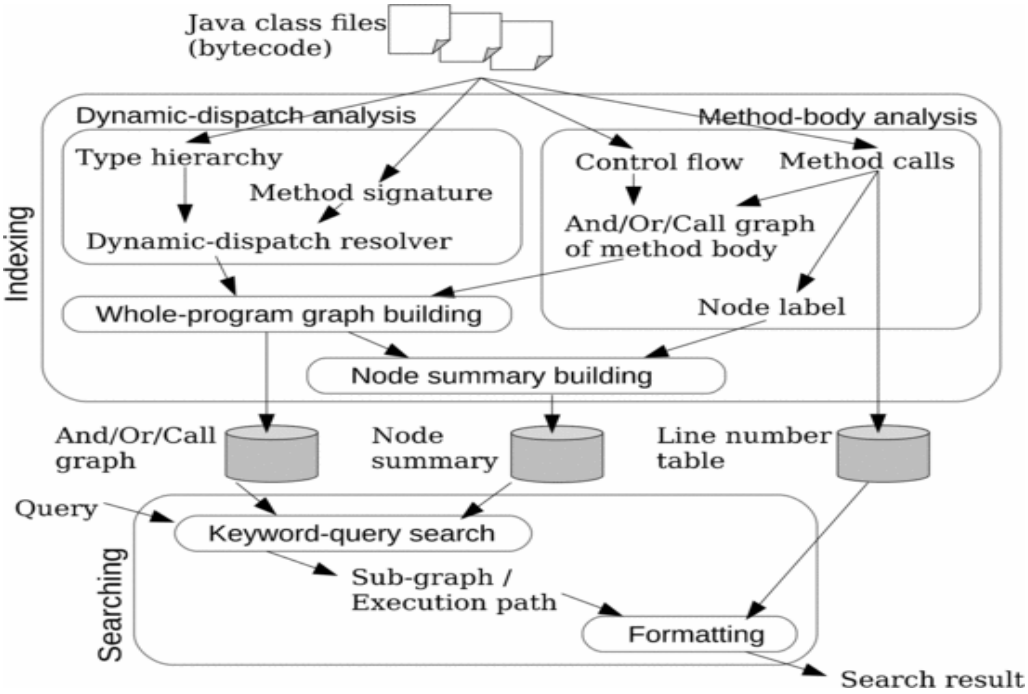

Fig 4 Tool Architecture

D. Result:

The indexing step took 48.8 sec. in slipped by time and its top memory utilization was 644 MiB. The Strategy defs revealed the system bodies that were not found in the tree, that is, will never be demonstrated in query items for any inquiry. The proportion is roughly 73% (= 6174/8466). This proportion demonstrates low scope as a hunt device. The purpose behind this low scope was not completely researched, notwithstanding, halfway due to (Y2) in Sec. IV-A; GUI-related entrance focuses, were not investigated.

For a basic assessment of the instrument, it ought to be run it to discover execution ways with inquiries of a couple of class names. The classes were chosen arbitrarily from external (not inward) classes showing up in any event once in the chart as a recipient of some system call. Every run of looking step took 3.09 ~ 72.2 (st. 5.71) sec. in slipped by time and crest memory use was dependent upon 1412 MiB. Table II demonstrates the outcome. The low rate (16.4% = 311/1,891) of the discovered sub-charts is not astonishing in light of the fact that the And/Or/Call diagram had 241 root hubs (section focuses) and a few classes were in a tree under a root hub, while alternate classes were found in an alternate tree. The issue is the drop of a discovered execution way. In 57 (= 311 − 254) cases, the device was not ready to concentrate any execution way satisfying the given question. In some of these cases no such way existed in the And/Or/Call diagram; be that as it may, alternate cases were false negatives on the grounds that the calculation is not yet wrapped.

IV. CONCLUSIONS

We proposed an algorithm to discover sub-diagrams of an And/Or/ - Call diagram, which is created from a Java program (bytecode then again source document) and stores all execution ways of the program. The information structure and its transformation to execution ways were characterized and clarified with a little example. The look algorithm was indicated as a pseudo code. The model usage was connected to an open-source item,

This study is at ahead of schedule stage and the algorithm (execution) is juvenile and needs further refinements, particularly in the accompanying ways:

1. The algorithm must be done to guarantee execution ways,
2. There are impediment of the magic words in the inquiry, as of now just the names of routines and sorts, and string literals in contentions. A client cannot utilize alternate elements, for example, numerical/Boolean literals, remark, and variable names in a question.
3. The element dispatch of techniques are determined just with the sorts (of recipient, return quality, and contentions). No dead code disposal (location of inaccessible code) has been presented yet.

REFERENCES

[1] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, C.Lucena,"Refactoring Product Lines," Proc. 5th int'l Conf. Generative Programming and Component Eng. (GPCE'06), p.201-210, 2006.

[2] E. W. Dijkstra, "Structured Programming," In Softw. Eng. Techniques,B. Randell, J.N. Buxton, (Eds.), NATO Scientific Affairs Division,pp.84-88, 1970

[3] N. E´en and N. S¨orensson, "An Extensible SAT-Solver," Proc. 6th Int'l.Conf. Theory and Applications of Satisfiability Testing (SAT'03), LNCS2919, pp.502-518, 2004.

[4] J. Feigenspan, S. Apel, J. Liebig, C. K¨astner, "Exploring Software Measures to Assess Program Comprehension," Proc. the 5th ACM/IEEE Int'l Sympo. Empirical Software Eng. and Measurement (ESEM'11), pp.127-136, 2011.

[5] K. Havelund, T. Pressburger, "Model Checking Java Programs Using Java PathFinder," Int'l Journal on Softw. Tools for Technology Transfer,Vol.2, No.4, pp.366-381, 2000.

[6] Toshihiro Kamiya "An Algorithm for keyword search on an Execution path" IEEE Int'l Conf. Program Comprehension (ICPC' 14). pp 328-332, 2014.