CYK algorithm

From Wikipedia, the free encyclopedia

In computer science, the **Cocke–Younger–Kasami algorithm** (alternatively called **CYK**, or **CKY**) is a parsing algorithm for context-free grammars, named after its inventors, John Cocke, Daniel Younger and Tadao Kasami. It employs bottom-up parsing and dynamic programming.

The standard version of CYK operates only on context-free grammars given in Chomsky normal form (CNF). However any context-free grammar may be transformed to a CNF grammar expressing the same language (Sipser 1997).

The importance of the CYK algorithm stems from its high efficiency in certain situations. Using Landau symbols, the worst case running time of CYK is $\Theta(n^3 \cdot |G|)$, where n is the length of the parsed string and $|G|$ is the size of the CNF grammar G (Hopcroft & Ullman 1979, p. 140). This makes it one of the most efficient parsing algorithms in terms of worst-case asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.

Contents

Standard form

The algorithm requires the context-free grammar to be rendered into Chomsky normal form (CNF), because it tests for possibilities to split the current sequence in half. Any context-free grammar that does not generate the empty string can be represented in CNF using only production rules of the forms $A \to \alpha$ and $A \to BC$.

Algorithm

As pseudocode

The algorithm in pseudocode is as follows:

```
let the input be a string S consisting of n characters: a_1 ... a_n.
let the grammar contain r nonterminal symbols R_1 ... R_r.
This grammar contains the subset R_s which is the set of start symbols.
let P[n,n,r] be an array of booleans. Initialize all elements of P to false.
for each i = 1 to n
  for each unit production R_j -> a_i
    set P[1,i,j] = true
for each i = 2 to n -- Length of span
  for each j = 1 to n-i+1 -- Start of span
    for each k = 1 to i-1 -- Partition of span
      for each production R_A -> R_B R_C
        if P[k,j,B] and P[i-k,j+k,C] then set P[i,j,A] = true
if any of P[n,1,s] is true (s is iterated over the set s, where s are all the indices for R_s) then
  S is member of language
else
  S is not member of language
```

As prose

In informal terms, this algorithm considers every possible subsequence of the sequence of words and sets $P[i, j, k]$ to be true if the subsequence of words of length i starting from j can be generated from R_k. Once it has considered subsequences of length 1, it goes on to subsequences of length 2, and so on. For subsequences of length 2 and greater, it considers every possible partition of the subsequence into two parts, and checks to see if there is some production $P \to Q\ R$ such that Q matches the first part and R matches the second part. If so, it records P as matching the whole subsequence. Once this process is completed, the sentence is recognized by the grammar if the subsequence containing the entire sentence is matched by the start symbol.

Example

This is an example grammar:

$$
\begin{aligned}
S &\to NP\ VP \\
VP &\to VP\ PP \\
VP &\to V\ NP \\
VP &\to eats \\
PP &\to P\ NP \\
NP &\to Det\ N \\
NP &\to she \\
V &\to eats \\
P &\to with \\
N &\to fish \\
N &\to fork \\
Det &\to a
\end{aligned}
$$

Now the sentence *she eats a fish with a fork* is analyzed using the CYK algorithm. In the following table, in $P[i, j, k]$, i is the number of the row (starting at the bottom at 1), and j is the number of the column (starting at the left at 1).

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

For readability, the CYK table for P is represented here as a 2-dimensional matrix M containing a set of non-terminal symbols, such that R_k is in $M[i,j]$ if, and only if, $P[i,j,k]$. In the above example, since a start symbol S is in $M[7,1]$, the sentence can be generated by the grammar.

Extensions

Generating a parse tree

The above algorithm is a recognizer that will only determine if a sentence is in the language. It is simple to extend it into a parser that also construct a parse tree, by storing parse tree nodes as elements of the array, instead of the boolean 1. The node is linked to the array elements that were used to produce it, so as to build the tree structure. Only one such node in each array element is needed if only one parse tree is to be produced. However, if all parse trees of an ambiguous sentence are to be kept, it is necessary to store in the array element a list of all the ways the corresponding node can be obtained in the parsing process. This is sometimes done with a second table B[n,n,r] of so-called *backpointers*. The end result is then a shared-forest of possible parse trees, where common trees parts are factored between the various parses. This shared forest can conveniently be read as an ambiguous grammar generating only the sentence parsed, but with the same ambiguity as the original grammar, and the same parse trees up to a very simple renaming of non-terminals, as shown by Lang (1994).

Parsing non-CNF context-free grammars

As pointed out by Lange & Leiß (2009), the drawback of all known transformations into Chomsky normal form is that they can lead to an undesirable bloat in grammar size. The size of a grammar is the sum of the sizes of its production rules, where the size of a rule is one plus the length of its right-hand side. Using g to denote the size of the original grammar, the size blow-up in the worst case may range from g^2 to 2^{2g}, depending on the transformation algorithm used. For the use in teaching, Lange and Leiß propose a slight generalization of the CYK algorithm, "without compromising efficiency of the algorithm, clarity of its presentation, or simplicity of proofs" (Lange & Leiß 2009).

Parsing weighted context-free grammars

It is also possible to extend the CYK algorithm to parse strings using weighted and stochastic context-free grammars. Weights (probabilities) are then stored in the table P instead of booleans, so P[i,j,A] will contain the minimum weight (maximum probability) that the substring from i to j can be derived from A. Further extensions of the algorithm allow all parses of a string to be enumerated from lowest to highest weight (highest to lowest probability).

Valiant's algorithm

The worst case running time of CYK is $\Theta(n^3 \cdot |G|)$, where n is the length of the parsed string and $|G|$ is the size of the CNF grammar G. This makes it one of the most efficient algorithms for recognizing general context-free languages in practice. Valiant (1975) gave an extension of the CYK algorithm. His algorithm computes the same parsing table as the CYK algorithm; yet he showed that algorithms for efficient multiplication of matrices with 0-1-entries can be utilized for performing this computation.

Using the Coppersmith–Winograd algorithm for multiplying these matrices, this gives an asymptotic worst-case running time of $O(n^{2.38} \cdot |G|)$. However, the constant term hidden by the Big O Notation is so large that the Coppersmith–Winograd algorithm is only worthwhile for matrices that are too large to handle on present-day computers (Knuth 1997), and this approach requires subtraction and so is only suitable for recognition. The dependence on efficient matrix multiplication cannot be avoided altogether: Lee (2002) has proved that any parser for context-free grammars working in time $O(n^{3-\epsilon} \cdot |G|)$ can be effectively converted into an algorithm computing the product of $(n \times n)$-matrices with 0-1-entries in time $O(n^{3-\epsilon/3})$.

See also

- GLR parser
- Earley parser
- Packrat parser

References

- Cocke, John; Schwartz, Jacob T. (April 1970). *Programming languages and their compilers: Preliminary notes* (PDF) (Technical report) (2nd revised ed.). CIMS, NYU.
- Hopcroft, John E.; Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading/MA: Addison-Wesley. ISBN 0-201-02988-X.
- Kasami, T. (1965). *An efficient recognition and syntax-analysis algorithm for context-free languages* (Technical report). AFCRL. 65-758.
- Knuth, Donald E. (November 14, 1997). *The Art of Computer Programming Volume 2: Seminumerical Algorithms* (3rd ed.). Addison-Wesley Professional. p. 501. ISBN 0-201-89684-2.
- Lang, Bernard (1994). "Recognition can be harder than parsing". *Comput. Intell.* **10** (4): 486–494. doi:10.1111/j.1467-8640.1994.tb00011.x. CiteSeerX: 10.1.1.50.6982.
- Lange, Martin; Leiß, Hans (2009). "To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm". *Informatica Didactica*. **8**.
- Lee, Lillian (2002). "Fast context-free grammar parsing requires fast Boolean matrix multiplication". *J. ACM*. **49** (1): 1–15. doi:10.1145/505241.505242.
- Sipser, Michael (1997). *Introduction to the Theory of Computation* (1st ed.). IPS. p. 99. ISBN 0-534-94728-X.
- Valiant, Leslie G. (1975). "General context-free recognition in less than cubic time". *J. Comput. Syst. Sci.* **10** (2): 308–314. doi:10.1016/s0022-0000(75)80046-8.
- Younger, Daniel H. (February 1967). "Recognition and parsing of context-free languages in time n^3". *Inform. Control*. **10** (2): 189–208. doi:10.1016/s0019-9958(67)80007-x.

External links

- CYK parsing demo in JavaScript (http://martinlaz.github.io/demos/cky.html)
- Interactive Applet from the University of Leipzig to demonstrate the CYK-Algorithm (Site is in german) (http://www.informatik.uni-leipzig.de/alg/lehre/ss08/AUTO-SPRACHEN/Java-Applets/CYK-Algorithmus.html)
- Exorciser is a Java application to generate exercises in the CYK algorithm as well as Finite State Machines, Markov algorithms etc (http://www.swisseduc.ch/compscience/exorciser/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=CYK_algorithm&oldid=728704140"

Categories: Parsing algorithms

- This page was last modified on 7 July 2016, at 02:40.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.