Crossword Solving Algorithm

*Thomas Laverghetta (*[*tlave002@odu.edu*](mailto:tlave002@odu.edu)*)*,   
Old Dominion University – Computational Modeling and Simulation Engineering

Please read over Assignment 3 System Requirements document (which accompanies this report) first to gain understanding of program overview.

# Algorithms

In this section, I will discuss the algorithms and data structures used to solve crossword puzzles. The algorithm used to solve the crossword is backtracking. Backtracking is a general algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions (adding words to crossword), and abandons a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution. Therefore, allowing me to add words to crossword puzzle, test if word does not fail crossword rules, and either adding another word or trying another word. Figure 1 illustrates backtracking process – i.e., starting initial node then adding additional values until reaching leaf node (potential solution) before going back-up and trying a different combination. Although, before deploying backtracking, pre-word processing and data structures are needed to allow backtracking to word efficiently.

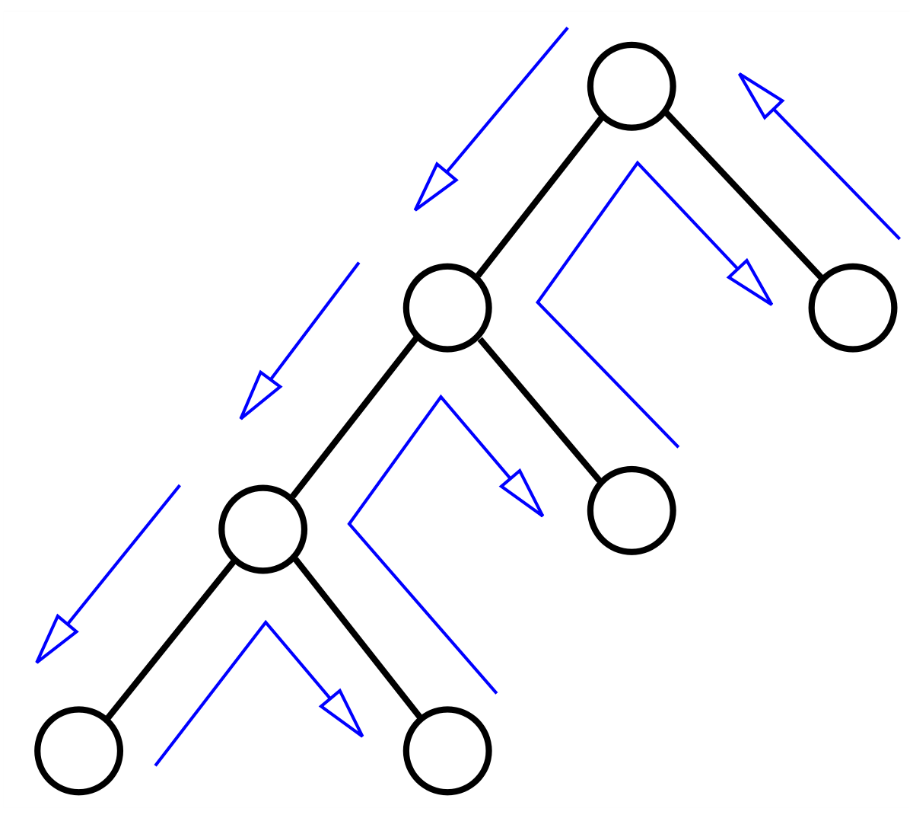


Figure 1. Backtracking Algorithm

There are three pre-word processes done before backtracking and two data structures to encapsulate pre-word processing. I will start with the data-structures then processes.

There are two data-structures used: crossword-element and crossword-element-set. A crossword-element represents a word within crossword which include the word itself and identifier (where the word belongs in crossword). A crossword-element-set represents all possible words that can be placed within crossword-element (word can fit in crossword-element). Hence, a crossword-element-set encapsulates a set of possible words and element identifier. Figure 2 illustrates the data-structures.

Table

Description automatically generated

Figure 2. Data structures

The three processes: crossword element requirements loader, intersection finding, and dictionary filtering.

* Crossword element requirements process loads crossword requirements foreach element. A requirement for an element includes size of element (word size constraint), element identifier, and direction of element (vertical or horizontal).
* Intersection finding uses the information from crossword element requirement loader to find element intersections. The element intersections are a set of elements that intersect a specified element, and where elements intersect with specified element. The intersections are used in the backtracking’s selecting algorithm (explained in next section).
* Dictionary filtering process loads a dictionary and iterates through the dictionary foreach crossword element to produce crossword-element-sets. Before iterating, crossword-element-sets are created based on crossword element requirement data. Where each crossword-element-set just has element identifier associated with crossword element requirement element identifier and empty word set. Within each iteration, dictionary word will be compared with crossword element requirement length to determine if word will fit within crossword-element. If word conforms, then word will be appended to crossword-element-set word set. Else, continue to next word.

Once dictionary filtering has finished and returned crossword-element-sets, the crossword-element-sets will be sorted in descending order by number of words in each set then back will be popped off and used as initial crossword-element-set to iterate across for backtracking. The pseudocode for main process (includes three processes) is shown below:

*main (void)*

*1. Load and save crossword requirements (i.e., elements and their respected sizes)*

*2. Find and save crossword element intersections (i.e., where elements intersect)*

*3. Load word dictionary and filter words into crossword element sets (creating crossword element sets foreach crossword element, and foreach element set, save words from the dictionary that meet crossword element requirements (size))*

*4. sort crossword element sets in descending order by number of words*

*5. select back crossword element set from sorted sets (init element set) and pop sorted sets*

*6. foreach crossword element in init element set*

*7. if Backtracking (crossword element, sorted set)*

*8. break*

*9. end if*

*10. end for*

*11. if sorted set length == 0*

*12. solution found*

*13. else*

*14. no solution found*

*15. end if*

Note, *&* in parameter input is pass by reference.

## Backtracking and Selecting Algorithms for Crossword Puzzle

*Backtracking (& curr words (CW)[], remaining word set (RWS)[])*

*1. Get next crossword element set {SelectNextElementSet(CW.end,RWS)}*

*2. if getting next crossword element returns true (element found):*

*3. foreach element in next crossword element set:*

*4. Append element to CW*

*5. if RWS length equals 0:*

*6. save CW as solution set*

*7. return true*

*8. end if*

*9. if Backtracking (CW, RWS)*

*10. return true*

*11. end if*

*12. Pop element from CW*

*13. end for*

*14. return false*

*SelectNextElementSet (const& newest element added (newE), & RWS)*

*1. Allocate temporary list called tmp*

*2. foreach intersection element (IE) in newE:*

*3. if IE in RWS:*

*4. Allocate new crossword element*

*5. foreach word in IE:*

*6. if word does not conflict with newE’s word*

*7. Append word to new crossword element word set*

*8. end if*

*9. end for*

*10. if new crossword element word set length > 0:*

*11. Append new crossword element to tmp*

*12. else:*

*13. return false (no solution with given words)*

*14. end if*

*15. end if*

*16. if tmp length equals zero (either no intersections or all intersections are in CW):*

*17. sort RWS by size of word set in descending order*

*18. set next crossword element set to RWS.back*

*19. Pop RWS*

*20. return next crossword element*

*21. end if*

*22. sort tmp by size of word set in ascending order*

*23. set next crossword element to tmp.front*

*24. foreach element in tmp[1:]:*

*25 replace RWS’s element with element*

*26. end for*

*27. return next crossword element*

# Results