Knuth's Algorithm X

From Wikipedia, the free encyclopedia

[Donald Knuth](http://en.wikipedia.org/wiki/Donald_Knuth)'s **Algorithm X** is a [recursive](http://en.wikipedia.org/wiki/Recursion_(computer_science)), [nondeterministic](http://en.wikipedia.org/wiki/Nondeterministic_algorithm), [depth-first](http://en.wikipedia.org/wiki/Depth-first), [backtracking](http://en.wikipedia.org/wiki/Backtracking) [algorithm](http://en.wikipedia.org/wiki/Algorithm) that finds all solutions to the [exact cover](http://en.wikipedia.org/wiki/Exact_cover) problem represented by a matrix *A* consisting of 0s and 1s. The goal is to select a subset of the rows so that the digit 1 appears in each column exactly once.

Algorithm X functions as follows:

|  |
| --- |
| 1. If the matrix *A* is empty, the problem is solved; terminate successfully. 2. Otherwise choose a column *c* ([deterministically](http://en.wikipedia.org/wiki/Deterministic_algorithm" \o "Deterministic algorithm)). 3. Choose a row *r* such that *Ar*,*c* = 1 ([nondeterministically](http://en.wikipedia.org/wiki/Nondeterministic_algorithm" \o "Nondeterministic algorithm)). 4. Include row *r* in the partial solution. 5. For each column *j* such that *Ar*,*j* = 1,   for each row *i* such that *Ai*,*j* = 1,  delete row *i* from matrix *A*;  delete column *j* from matrix *A*.   1. Repeat this algorithm recursively on the reduced matrix *A*. |

The nondeterministic choice of *r* means that the algorithm essentially clones itself into independent subalgorithms; each subalgorithm inherits the current matrix *A*, but reduces it with respect to a different row *r*. If column *c* is entirely zero, there are no subalgorithms and the process terminates unsuccessfully.

The subalgorithms form a [search tree](http://en.wikipedia.org/wiki/Search_tree) in a natural way, with the original problem at the root and with level *k* containing each subalgorithm that corresponds to *k* chosen rows. Backtracking is the process of traversing the tree in preorder, depth first.

Any systematic rule for choosing column *c* in this procedure will find all solutions, but some rules work much better than others. To reduce the number of iterations, [Knuth](http://en.wikipedia.org/wiki/Donald_Knuth) suggests that the column choosing algorithm select a column with the lowest number of 1s in it.

**Contents**

  [[hide](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X)]

* [1 Example](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X#Example)
* [2 Implementations](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X#Implementations)
* [3 See also](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X#See_also)
* [4 References](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X#References)
* [5 External links](http://en.wikipedia.org/wiki/Knuth's_Algorithm_X#External_links)

Example[[edit](http://en.wikipedia.org/w/index.php?title=Knuth%27s_Algorithm_X&action=edit&section=1)]

For example, consider the exact cover problem specified by the universe *U* = {1, 2, 3, 4, 5, 6, 7} and the collection of sets \mathcal{S} = {*A*, *B*, *C*, *D*, *E*, *F*}, where:

* *A* = {1, 4, 7};
* *B* = {1, 4};
* *C* = {4, 5, 7};
* *D* = {3, 5, 6};
* *E* = {2, 3, 6, 7}; and
* *F* = {2, 7}.

This problem is represented by the matrix:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| ***B*** | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Algorithm X with Knuth's suggested heuristic for selecting columns solves this problem as follows:

**Level 0**

Step 1—The matrix is not empty, so the algorithm proceeds.

Step 2—The lowest number of 1s in any column is two. Column 1 is the first column with two 1s and thus is selected (deterministically):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | **1** | 0 | 0 | 1 | 0 | 0 | 1 |
| ***B*** | **1** | 0 | 0 | 1 | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Step 3—Rows *A* and *B* each have a 1 in column 1 and thus are selected (nondeterministically).

The algorithm moves to the first branch at level 1…

**Level 1: Select Row *A***

Step 4—Row *A* is included in the partial solution.

Step 5—Row *A* has a 1 in columns 1, 4, and 7:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | **1** | 0 | 0 | **1** | 0 | 0 | **1** |
| ***B*** | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Column 1 has a 1 in rows *A* and *B*; column 4 has a 1 in rows *A*, *B*, and *C*; and column 7 has a 1 in rows *A*, *C*, *E*, and *F*. Thus rows *A*, *B*, *C*, *E*, and *F* are to be removed and columns 1, 4 and 7 are to be removed:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | **1** | 0 | 0 | **1** | 0 | 0 | **1** |
| ***B*** | **1** | 0 | 0 | **1** | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | **1** | 1 | 0 | **1** |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | **1** |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | **1** |

Row *D* remains and columns 2, 3, 5, and 6 remain:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** |
| ***D*** | 0 | 1 | 1 | 1 |

Step 1—The matrix is not empty, so the algorithm proceeds.

Step 2—The lowest number of 1s in any column is zero and column 2 is the first column with zero 1s:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** |
| ***D*** | 0 | 1 | 1 | 1 |

Thus this branch of the algorithm terminates unsuccessfully.

The algorithm moves to the next branch at level 1…

**Level 1: Select Row *B***

Step 4—Row *B* is included in the partial solution.

Row *B* has a 1 in columns 1 and 4:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| ***B*** | **1** | 0 | 0 | **1** | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Column 1 has a 1 in rows *A* and *B*; and column 4 has a 1 in rows *A*, *B*, and *C*. Thus rows *A*, *B*, and *C* are to be removed and columns 1 and 4 are to be removed:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***A*** | **1** | 0 | 0 | **1** | 0 | 0 | 1 |
| ***B*** | **1** | 0 | 0 | **1** | 0 | 0 | 0 |
| ***C*** | 0 | 0 | 0 | **1** | 1 | 0 | 1 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***E*** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Rows *D*, *E*, and *F* remain and columns 2, 3, 5, 6, and 7 remain:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** | **7** |
| ***D*** | 0 | 1 | 1 | 1 | 0 |
| ***E*** | 1 | 1 | 0 | 1 | 1 |
| ***F*** | 1 | 0 | 0 | 0 | 1 |

Step 1—The matrix is not empty, so the algorithm proceeds.

Step 2—The lowest number of 1s in any column is one. Column 5 is the first column with one 1 and thus is selected (deterministically):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** | **7** |
| ***D*** | 0 | 1 | **1** | 1 | 0 |
| ***E*** | 1 | 1 | 0 | 1 | 1 |
| ***F*** | 1 | 0 | 0 | 0 | 1 |

Step 3—Row *D* has a 1 in column 5 and thus is selected (nondeterministically).

The algorithm moves to the first branch at level 2…

**Level 2: Select Row *D***

Step 4—Row *D* is included in the partial solution.

Step 5—Row *D* has a 1 in columns 3, 5, and 6:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** | **7** |
| ***D*** | 0 | **1** | **1** | **1** | 0 |
| ***E*** | 1 | 1 | 0 | 1 | 1 |
| ***F*** | 1 | 0 | 0 | 0 | 1 |

Column 3 has a 1 in rows *D* and *E*; column 5 has a 1 in row *D*; and column 6 has a 1 in rows *D* and *E*. Thus rows *D* and *E* are to be removed and columns 3, 5, and 6 are to be removed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **2** | **3** | **5** | **6** | **7** |
| ***D*** | 0 | **1** | **1** | **1** | 0 |
| ***E*** | 1 | **1** | 0 | **1** | 1 |
| ***F*** | 1 | 0 | 0 | 0 | 1 |

Row *F* remains and columns 2 and 7 remain:

|  |  |  |
| --- | --- | --- |
|  | **2** | **7** |
| ***F*** | 1 | 1 |

Step 1—The matrix is not empty, so the algorithm proceeds.

Step 2—The lowest number of 1s in any column is one. Column 2 is the first column with one 1 and thus is selected (deterministically).

Row *F* has a 1 in column 2 and thus is selected (nondeterministically).

The algorithm moves to the first branch at level 3…

**Level 3: Select Row *F***

Step 4—Row *F* is included in the partial solution.

Row *F* has a 1 in columns 2 and 7:

|  |  |  |
| --- | --- | --- |
|  | **2** | **7** |
| ***F*** | **1** | **1** |

Column 2 has a 1 in row *F*; and column 7 has a 1 in row *F*. Thus row *F* is to be removed and columns 2 and 7 are to be removed:

|  |  |  |
| --- | --- | --- |
|  | **2** | **7** |
| ***F*** | **1** | **1** |

Step 1—The matrix is empty, thus this branch of the algorithm terminates successfully.

As rows *B*, *D*, and *F* are selected, the final solution is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| ***B*** | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| ***D*** | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| ***F*** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

In other words, the subcollection {*B*, *D*, *F*} is an exact cover, since every element is contained in exactly one of the sets *B* = {1, 4}, *D* = {3, 5, 6}, or *F* = {2, 7}.

There are no more selected rows at level 3, thus the algorithm moves to the next branch at level 2…

There are no more selected rows at level 2, thus the algorithm moves to the next branch at level 1…

There are no more selected rows at level 1, thus the algorithm moves to the next branch at level 0…

There are no branches at level 0, thus the algorithm terminates.

In summary, the algorithm determines there is only one exact cover: \mathcal{S}^* = {*B*, *D*, *F*}.

Implementations[[edit](http://en.wikipedia.org/w/index.php?title=Knuth%27s_Algorithm_X&action=edit&section=2)]

[Dancing Links](http://en.wikipedia.org/wiki/Dancing_Links), commonly known as DLX, is the technique suggested by [Knuth](http://en.wikipedia.org/wiki/Donald_Knuth) to efficiently implement his Algorithm X on a computer. Dancing Links implements the matrix using circular [doubly linked lists](http://en.wikipedia.org/wiki/Doubly_linked_list) of the 1s in the matrix. There is a list of 1s for each row and each column. Each 1 in the matrix has a link to the next 1 above, below, to the left, and to the right of itself.

*nnial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare*, Palgrave, pp. 187–214, [arXiv](http://en.wikipedia.org/wiki/ArXiv):[cs/0011047](http://arxiv.org/abs/cs/0011047),[ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-333-92230-9](http://en.wikipedia.org/wiki/Special:BookSources/978-0-333-92230-9).

External links[[edit](http://en.wikipedia.org/w/index.php?title=Knuth%27s_Algorithm_X&action=edit&section=5)]

[Implementation of an Exact Cover solver in C#](http://cheeso.members.winisp.net/srcview.aspx?dir=Sudoku&file=ExactCover.cs) - uses Algorithm X and the Dancing Links optimization.

[Polycube solver](http://github.com/mlepage/polycube-solver) Program (with Lua source code) to fill boxes with polycubes using [Algorithm X](http://en.wikipedia.org/wiki/Algorithm_X).

[Knuth's Paper describing the Dancing Links optimization](http://www-cs-faculty.stanford.edu/~uno/papers/dancing-color.ps.gz) - Gzip'd postscript file.