Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 2. pp. 367–375.

Search algorithms at ACM contests

Márk Kósa, János Pánovics

University of Debrecen, Faculty of Informatics
e-mail: mkosa@inf.unideb.hu, panovics@inf.unideb.hu

Abstract

Students of the University of Debrecen majoring in informatics have been participating in regional ACM international collegiate programming contests since 1995. In this paper, we introduce the history and rules of ACM contests. The problems of this programming contest cover several topics from the area of mathematics and informatics. We deal with topics in which search algorithms may be used during problem solving. We show search algorithms which can be applied to solving problems presented in ACM contests of the last few years. Such algorithms are different versions of backtracking algorithm as well as breadth-first search and uniform-cost search. It is very important to find the best algorithm for problem solving in order to have the most efficient solution. We will see that even if using the same algorithm, the performance of these methods are highly dependent on the state space representation of the problem.

Keywords: search algorithms, backtracking, breadth-first search, uniform-cost search, state space representation, ACM contest

MSC: 05C85

1. About ACM contests

The 1st ACM contest was held at the Baylor University, Texas in 1970. Since then this university has been incessantly the main organizer of ACM contests. From 6099 teams selected from 1756 universities in 82 countries competing at 205 sites and hundreds more competing at preliminary contests worldwide, 88 teams of students competed for bragging rights and prizes at The 31st Annual ACM International Collegiate Programming Contest World Finals sponsored by IBM on March 15, 2007, and hosted by ACM Japan Chapter and IBM Tokyo Research Lab. (These data are based on [1].)

From Hungary four universities take part in the contest from year to year, namely Budapest Univerity of Technology and Economics, Eötvös Loránd Univeristy, University of Szeged and University of Debrecen. Unfortunately, there has

never been a Hungarian team in the World Finals. From our region ususally Polish teams (and sometimes Czech teams) advance to the finals.

1.1. Rules of the contest

The most important rules of an ACM contest are the following:

- The language of the contest is English.
- Each team consists of three contestants.
- Each team will be allowed to use one IBM PC-compatible computer.
- Contestants may bring resource materials such as books, manuals, notes and program listings. Contestants may not bring any machine-readable versions of software or data. Contestants may neither bring their own computers, computer terminals and calculators nor any kind of communication devices such as radio sets, cellular phones, pagers and PDAs.
- Usually 7–12 problems are posed.
- The contest is scheduled typically for 5 hours.
- There are size, runtime and memory limitations for the solutions.
- Teams are ranked according to the most problems solved. Teams who solve the same number of problems are ranked by least total time. The total time is the sum of the time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the accepted run plus 20 penalty minutes for every rejected run for that problem regardless of submittal time. There is no time consumed for a problem that is not solved.

1.2. Problem categories

The problems of an ACM contest can be classified into the following categories:

- combinatorics,
- number theory (e.g. prime numbers),
- arithmetic and algebra (e.g. modular arithmetic, big integers),
- computational geometry,
- backtracking,
- graph theory (traversal, single source shortest path, all-pairs shortest path, minimum spanning tree, articulation point, flood fill, network flow, maximum bipartite matching, etc.),

- sorting,
- string processing (e.g. pattern matching),
- greedy algorithms,
- dynamic programming,
- divide and conquer algorithms,
- advanced data structures (e.g. Fibonacci heaps, sets, dictionaries).

2. Search algorithms

We deal with backtracking and graph search (breadth-first search and uniform-cost search) algorithms in this paper because these algorithm families are very commonly used during ACM contests.

The base version of backtracking may be used when the problem's representation graph is a tree graph, the solutions are at the same level in this tree, and we need all of the solutions. The advantage of this algorithm is its small memory need. For example, the well-known Eight Queens' Problem is such a problem. Another example will be introduced in Section 2.1.

The base version of backtracking is inexpedient to solve poblems with an infinit representation graph, or a graph which contains cycles. Though we may use path length limitation or cycle checking in backtracking, but in these cases we can usually find a more appropriate algorithm to solve the problem.

We may use backtracking also to find an optimal solution, namely with dynamically decreasing path length limit. However, we should only use this technique if we have an upper estimation of the cost of the optimal solution.

Two of the graph search algorithms are capable of finding the optimal solution: breadth-first search and uniform-cost search (Dijkstra) algorithms. We use BFS in case of equal cost operators, and uniform-cost search in case of different cost operators.

The depth-first search algorithm and heuristic search algorithms are rarely used at ACM contests. These algorithms are inappropriate to find an optimal solution, moreover during the contest we cannot know whether our heuristics is better than that of the judges, and we have a good chance that it is not.

In the following sections we show five problems with the most suitable algorithm. In case of some problems we show different state space representations, and compare the efficiency of the algorithms mentioned above using these representations.

2.1. Half plus half equals one

There are given ten boxes indicating the digits of two fractions as shown in Figure 1. Fill in the boxes with different digits from 0 to 9 so that the value

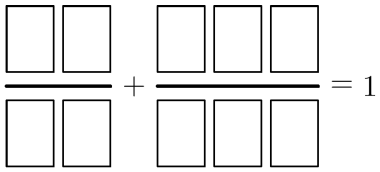

Figure 1: Half plus half equals one

of both fractions is $\frac{1}{2}$ and no number begins with the 0 digit. Give all possible solutions.

This is an example of problems which can be solved most conveniently using the base version of backtracking. The representation graph is a tree, the goal states are in the same level in the tree, and we are searching for all of the solutions.

In Table 1 you can see all the 16 solutions of the problem.

$13/26 = 485/970$	$38/76 = 145/290$
$15/30 = 486/972$	$38/76 = 451/902$
$16/32 = 485/970$	$45/90 = 138/276$
$27/54 = 309/618$	$45/90 = 186/372$
$29/58 = 307/618$	$45/90 = 381/762$
$31/62 = 485/970$	$46/92 = 185/370$
$35/70 = 148/296$	$48/96 = 135/270$
$35/70 = 481/962$	$48/96 = 351/702$

Table 1: Solutions

2.2. The warehouse

We have a square-shaped warehouse with a grid layout as shown in Figure 2. The circle denotes Secret Agent Θ–7, letter E denotes the exit, which is a hole in the ceiling, and the numbered fields denote boxes of certain heights. The floor is filled with deadly acid. Θ–7 can do only two things: he can move to the top of an adjacent box and he can topple the box he is standing on. In Figure 2 you can see an example of the latter move.

The goal is to give the minimum number of steps Θ–7 requires to reach the exit without stepping onto the floor.

We can use the uniform-cost search algorithm to solve this problem because now we are searching for the lowest cost solution.

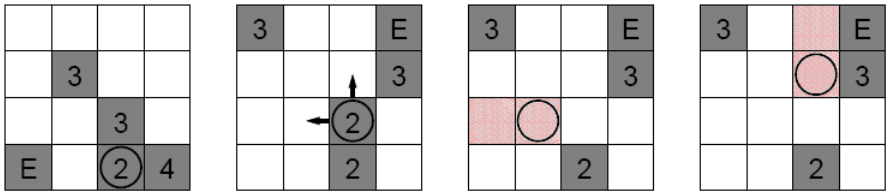

Figure 2: The warehouse

2.3. Ah Ce Emm

This problem is based on a Chinese gamble. Note that the title of the problem refers to the name of the contest: ACM.

Given a random amount of pebbles, you have to lose all these pebbles applying the following options in any order:

- The fire: if you have at least 11 pebbles, then you can throw away exactly 11 pebbles by paying x_1.
- The dragon: if the number of your pebbles is divisible by 3, then you can throw away exactly one third of your pebbles by paying 1 for each pebble thrown away.
- The eagle: you can ask for exactly 7 new pebbles by paying x_2.
- The courage: you can double the number of your pebbles and get one additional pebble by paying 1 for each new pebble you get.

You are not allowed to have more pebbles than initially. What is the minimum cost of losing all the pebbles?

This is also a typical problem which can be solved using the Dijkstra algorithm. We cannot give the answer at once because x_1 and x_2 are variables, not constants.

2.4. The four knights

Place two white and two black knights on a chessboard of 3×3 as you can see in Figure 3. Exchange the white knights with the black knights using legal knight moves, and give one sequence of moves corresponding to the knight swapping.

We present three different state space representations of this problem, and compare the backtracking and the BFS algorithms using these representations.

In the first representation we distinguish all of the four knights as shown in Figure 4. There is one initial state and there are four goal states. The parameters of Move(*knight*, *row*, *column*) operator represent the knight that moves and the destination square. This way we have 32 operators, but at most 8 of them can be applied in a particular state.

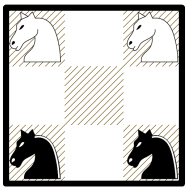

Figure 3: The four knights

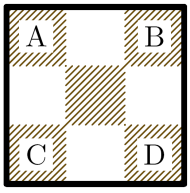

Figure 4: First representation

We hope that our algorithm will be faster if we do not distinguish the knights with the same color. This way we have only one goal state and 16 Move(x, y, u, v) operators, the parameters of which represent the source and destination squares.

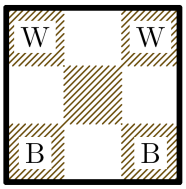

Figure 5: Second representation

Note that if the knights move in a certain (clockwise or counter-clockwise) direction, we can map the board to a vector as shown in Figure 6. As the middle square is never reached, only eight squares have to be mapped. In this vector the adjacent squares are exactly a knight move's distance from each other on the original board. The only parameter of the Move(*position*) operator represents the index in the vector from where a knight will move. This knight will move one square right in the vector, which means that the knight will move a knight move's distance on the original table in the given (clockwise or counter-clockwise) direction. This representation is much better because the precondition and the effect of the operator are much easier to implement.

In Table 2 we compare the three representations and two algorithms. The length of a solution is the length of the solution found first. The search time is the time required to find all solutions.

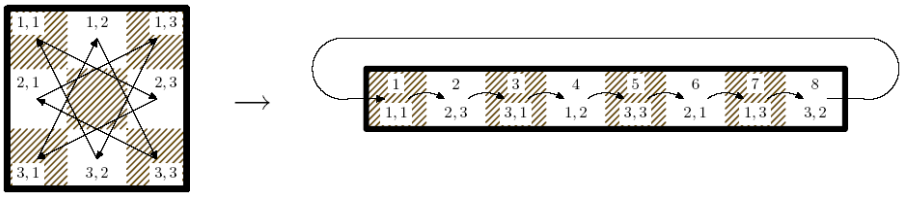

Figure 6: Third representation

	Number of states	Number of operators	Backtracking		Breadth-first search	
			Length of solution	Search time	Length of solution	Search time
1.	280	32	108	21 ms	16	51 ms
2.	280	16	n/a	n/a	16	45 ms
3.	117	8	16	1 ms	16	14 ms

Table 2: Searching of solutions and results

As we did not give the order in which these algorithms should choose the operators, they chose them in a random order. Therefore, backtracking did not stop using the second representation because the knights moved away from their target squares.

The third representation has fewer states because in that case if a knight reaches its target square, we do not let it move any more. Every knight has to make only four moves, so more than half of the original states can be eliminated.

We got the number of states by running breadth-first search for all solutions to traverse the whole graph.

2.5. Number ring

We have nine empty circles which we have to fill in with different numbers from 1 to 9 so that the sum of any two neighboring numbers must not be divisible by any of 3, 5 or 7. In the right hand side of Figure 7 you can see one of the two solutions. As we are looking for all solutions, backtracking is the preferred algorithm.

In the first state representation we apply a little trick: we use 10 positions to store the 9 numbers, where the value of the last position is the same as that of the first position. The set of goal states is given by a condition as opposed to the previous problem. The first idea is that we write down all the numbers, and then swap any two of them. This way we get 36 operators (which is the sum of the first eight natural numbers). This is not really a good representation, we just review it to later see how many states we have altogether.

We can improve the previous representation if we don not touch the numbers

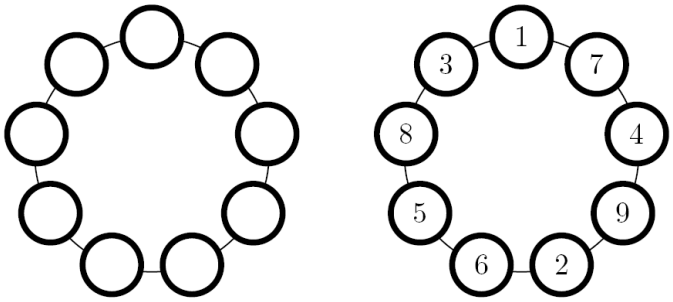

Figure 7: Number ring

at the beginning of the ring, so we can only swap the numbers behind the last "good" position (where the numbers satisfy the goal). This way we get our second representation.

We can further improve this representation if we allow swapping only at the first "bad" position and do not allow swapping ahead of it and behind it. This is the third representation.

Another improvement maybe applied to the previous representation if we place such a number into the first "bad" position that satisfies the goal. This is a quite good representation, which is similar to human's thinking during solving a jigsaw puzzle.

Finally, in the fifth representation we use a totally different technique where we use 9 positions for the 9 numbers, and do not fill in all the circles initially. The value of position #1 is fixed (1) the other values are all different from this (initially 0). We store the first empty position, i.e. the position with a value of 0. We place only such a number into the first empty circle which satisfies the goal condition. This gives us a fairly strong precondition, but we have only 8 operators, not 36. This is like solving a crossword puzzle.

	Number of states	Number of operators	Backtracking		Breadth-first search	
			Length of solution	Search time	Length of solution	Search time
1.	362880	36	n/a	n/a	4	n/a
2.	40320	36	28908	39292 ms	4	6032162 ms
3.	9408	36	n/a	n/a	4	71914 ms
4.	52	36	4	10 ms	4	26 ms
5.	117	8	8	6 ms	8	11 ms

Table 3: Searching of solutions and results

In Table 3 we compare backtracking and BFS algorithms using these representations. Now you can see that the first three representations are not really good for our need. There is a big difference between these representations. The search time of the backtracking algorithm using the second representation is the time of finding the first solution, not all.

3. Conclusion

As you can see from the examples, the following are all very important when solving contest problems:

- thorough knowledge of the problem;
- the knowledge of the applicable problem solving techniques;
- the knowledge of a programming language and/or tool.

All of these knowledges fit in the Computer Science curriculum in such courses as Data Structures and Algorithms, Programming Languages or Artificial Intelligence. We at the University of Debrecen endeavor to form the structure of courses so that we can inspire students to precisely analyze the problems.

References

[1] The ACM-ICPC International Collegiate Programming Contest Web Site, `http://icpc.baylor.edu`

[2] ACM: Association for Computing Machinery, `http://www.acm.org`

[3] Central European Programming Contest, `http://icpc.cs.bme.hu`

Márk Kósa, János Pánovics
University of Debrecen
Faculty of Informatics
H–4032 Debrecen
Egyetem tér 1.
Hungary