Analysis: GAM
Game

HISTORY:
- v. 1.01: 2008.04.08, WTYC - small corrections
- v. 1.00: 2008.03.31, SzW - first version of this document

dokument systemu SINOL 1.7.2

1 Introduction

The problem is a typical game theory problem. Since both players have the same distance to the goal (call that distance D - this is the distance between A's and B's starting point), it's obvious that both players should move on a shortest path. Otherwise at least $D+1$ moves would be required, in which case the other player always wins. Since player A moves first, he will always win unless player B during the move $D/2$ manages to reach the same square as player A (in which case B will win). Note that if D is odd, A always wins.

Because players should move on the shortest path it is easy to find all fields where each player can be after exactly p moves. To do this we can use BFS algorithm twice to find distance from A's and B's starting point to all other fields. After p moves, player A can be on squares where the distance to A's starting point is p and the distance to B's starting point is $D-p$ (and vice versa for player B). This finding is the basic notice that has to be made to correctly solve this problem because only in this way it is possible to find the order in which all states that describe A's and B's position should be processed.

2 Correct solution - time $O(n^3)$, memory $O(n^2)$

This solution uses a simple dynamic programming. Let LA_k be the list of squares where player A can be after k moves and let LB_k be the similar list but for player B. Let $T_{k_{i,j}}$ be **true** if after $D/2-k$ moves player A has a winning strategy if his piece is on the square that is i-th on the $LA_{D/2-k}$ list and player B's piece is on the square that is j-th on the $LB_{D/2-k}$ list. If B has a winning strategy then $T_{k_{i,j}}$ should be **false**. If D is even then list $LA_{D/2}$ should be equal to $LB_{D/2}$. We can easily notice that $T_{0_{i,j}}$ is **true** if and only if $i \neq j$.

To calculate values in matrices T for $k = 1,2,..,D/2$ we have to notice that A has a winning strategy if he can make such a move that after this move B can make only such moves after which A has still winning strategy. More formally, let $NextA_{k,i}$ be the list of squares belonging to $LA_{D/2-k+1}$ where the player A can move from the i-th square on the $LA_{D/2-k}$ list. Let $NextB_{k,j}$ be the similar list for the player B. If for some $i\prime \in NextA_{k,i}$ for all $j\prime \in NextB_{k,j}$ the value of $T_{k-1_{i\prime,j\prime}}$ is **true** then $T_{k_{i,j}}$ is **true**, otherwise it is **false**.

Using the above rule we can calculate T for $k = 1,2,..,D/2$ and the player A has winning strategy in whole game if and only if $T_{D/2_{1,1}}$ is **true** (if the lists are 1-based). The only problem is to quickly find $NextA$ and $NextB$ lists. For some square (x,y) where some player can be after k moves, using results from BFS, we can easily find all squares $(x\prime,y\prime)$ where this player can be after $k+1$ moves. The problem is to find the position of $(x\prime,y\prime)$ on the L_{k+1} list. But because each square can be on only one list when we add some square to some list we can also store in some array position of this square on the appropriate list.

In the worst case $D = O(n)$. Then each L list can have $O(n)$ elements and calculating each T_k takes $O(n^2)$ time. So for $D/2$ matrices it gives $O(n^3)$ time complexity. In the best case $D = O(n^2)$ and then each L list has constant number of elements so each T_k can be calculated in constant time and the time complexity is $O(n^2)$. According to this the overall time complexity is $O(n^3)$.

L lists use $O(n^2)$ memory. All T_k arrays in the worst case use $O(n^3)$ memory but there is no need to store them all. We need only matrices for current and previous k so it takes $O(n^2)$ memory. According to this the overall memory complexity is $O(n^2)$.

This solution was implemented in C++ (`gam.cpp`), C (`gam0.c`) and Pascal (`gam1.pas`).

3 Wrong solution - time $O(n^3 logn)$, memory $O(n^3)$

This solution is similar to the correct solution but instead of *LA*, *LB* lists and T array it stores values of states in some dictionary where $T(a_x, a_y, b_x, b_y)$ is **true** if and only if A has winning strategy when piece A is on the (a_x, a_y) square and piece B is on the (b_x, b_y) square. The implementation of the dictionary adds a factor of *logn* to the time complexity. Because it has to store all states it uses $O(n^3)$ memory.

This solution can score 60% of points (it solves first 9 tests of 15). It was implemented in `gams0.cpp` file.

4 Wrong solution - time $O(n^3)$, memory $O(n^4)$

This solution is similar to the previous one but instead of dictionary it uses 4-dimensional array. It improves speed to $O(n^3)$ but the memory used is $O(n^4)$.

This solution can score 40% of points (it solves first 6 tests of 9) and was implemented in `gams1.cpp` file.

5 Heuristic 1

The heuristic is very simple. Output `A` if D is odd, `B` otherwise. It scores 0 points. It is implemented in `gamb0.cpp` file.

6 Heuristic 2

This solution divides the rectangle which has A's and B's srarting points in opposite corners into four quarters. Then it calculates the number of black squares in the quarters next to A's and next to B's starting points. If there are more black squares next to B then it outputs `A` because the player B has less possibilities to move his piece during first half of the game and player A can use it to avoid meeting B.

This solution scores 0 points. It is implemented in `gamb1.cpp` file.

7 Heuristic 3

It combines Heuristic 1 and 2. If D is odd it outputs `A` otherwise it uses Heuristic B.

This solution scores 0 points. It is implemented in `gamb2.cpp` file.

8 Time and memory limits

The limit for n was set to 300 because for this value:

- the most important - it is easy to divide solutions that use $O(n^2)$ and more memory,
- the time of execution for larger tests is long enough to find too slow solutions.

The memory limit was set to **8 MB** to divide solutions that use $O(n^2)$ and more memory.

9 Tests

Because the answer in this problem is binary, tests were grouped together. Each test with number greater than 1 consists of 3 test files with suffix:

- **a** - one test without any black squares and two tests with small number of black squares (about 3%).
- **b** - one test with the longest possible route - player has to go to the right to the end of the board then two squares down, to the left to the end of the board, two squares down and again to the right, etc. After this test there are two tests similar to the first one but with some random, black squares removed. On average one square from each horizontal wall and from one of three horizontal walls where removed.
- **c** - one test with large number of black squares (about 20%).

In each test case player *B* starts in right, bottom corner. But for each test case two variants where created. In the first variant player *A* starts in the top, left corner, in the other it starts one square to the right. In the result one of these variants will have even (non-trivial) and one have odd length of the shortest path.

In total it gives 3 files with 7 tests, each in two variants, that is 14 tests in each group. So for each group there are $2^{14} = 16K$ possible answers and that is enough to eliminate programs that return random answer.

- `gam0.IN` (1 sek.) Example test from the problem description
- `gamocen1-4.IN` (1 sek.) Simple tests for contestants
- `gam1.IN` (1 sek.) 10 small test cases that checks boundary conditions and correctness
- `gam2-6.IN` (1 sek.) Small tests that should be solved by each algorithm
- `gam7-9.IN` (1 sek.) Larger tests that can be solved only by algorithms that uses $O(n^3)$ memory or less
- `gam10-15.IN` (6 sek.) Large test that should be solved only by correct solution

Time limits where measured on Core 2 Duo 2.2 GHz CPU. In general when setting time limits the correct solution should pass all tests, `gams0` solutions should pass first 9 tests (but here key a resource is memory because it is not possible to separate $O(n^3)$ and $O(n^3 logn)$). The `gams1` solution should pass only first 6 tests.

10 Changes in problem description

There were following changes in the problem description:

- The possibility to put more test cases in one file was added because the answer is binary so a lot of tests is required in each group.
- Information how many scores can be earned for different instances size was added.
- The example test was changed.

11 Remarks

The author wrote in his problem description that there exists a solution that works in $O(n^4)$ time that checks all states. But I think that such solution does not exists because it can not know in which order states should be searched.

12 List of work

The following work was done:

- This document was written.
- Example test, 4 tests for contestants and 15 groups of tests were prepared.
- Document with description of solutions for contestants was written.
- Correct solution in three languages, two slow solutions and three bad solutions were implemented.
- Small changes in problem description were added.
- Input generator and verifier were implemented.