Opracowanie: GRI
Grid

HISTORIA:
• v. 1.2: 2008.04.07, Jakub Lacki, minor cleanups, more remarks
• v. 1.1: 2008.04.06, Jakub Lacki, some corrections after discovering solutions with iteration limit
• v. 1.0: 2008.03.30, Jakub Lacki, everything
• v. 0.1: 2008.03.15, Jakub Lacki, empty template

dokument systemu SINOL 1.6

1 Introduction

At first sight, it seems that a dynamic algorithm can be used to solve this problem. No simple dynamic programming algorithm works here, though, as this problem lacks optimal substructure: before we place a line in one direction, we would like to know the placing of *every* line going in the other direction.

Nevertheless, dynamic programming can be used to solve a similar problem in one dimension. We fill the array calc, where calc[i][j] = the best computing time of i leftmost squares divided with j lines.

In one dimension, we can also use binary search. Once we know the desired computing time, we can easily calculate the number of lines we need to use using greedy approach.

Although model solutions run in exponential time, they utilize both these ideas to improve their time complexity.

2 Model solutions

The simplest solution is to consider all $\binom{n}{r}\binom{m}{s}$ placings of lines. However, if we fix placing of lines going in one direction, then the optimal placing of lines going in the other direction can be computed much faster. There are two possible approaches:

- We can use binary search to find an optimal computing time. Given a candidate for an optimal computing time and placing of all horizontal lines, we can place the vertical lines from the leftmost to the rightmost one using greedy approach: each vertical line should be put as far to the right as possible.

  This solution runs in $O(\binom{n}{r}nm\,log(S))$ time, where S is the sum of all computation times for individual squares. It is implemented in gri.

- Another approach considers all $\binom{n}{r}$ placings of horizontal lines. For each placing, it calculates the minimal processing time using dynamic programming. Let $min[i][j]$ be the minimal processing time of i first columns divided with j meridians. The array min can be easily filled in $O(m^2n)$ time. The overall running time is $O(\binom{n}{r}m^2n)$. It is implemented in gri1.

We can preprocess the data in $O(nm)$ time, so that we can calculate the computation time for any rectangle in constant time. This reduces the nm factor to rs. However, in the worst-case scenario, $r = \frac{n}{2}$ and $s = \frac{m}{2}$, and this optimization makes little difference.

3 Other solutions

Brute-force solution It considers all $\binom{n}{r}\binom{m}{s}$ placings of lines and runs in $\binom{n}{r}\binom{m}{s}nm$ time. This solution scores 40 points. It is implemented in gris1.

Genetic algorithm This algorithm maintains a pool of candidate solutions. In each step it:

- picks a solution, copies it and then mutates the copy, by introducing random changes,
- creates a new solution by crossing over some other solutions (for example, by taking meridians from one solutions and parallels from the other).
- discards the worst solutions, so that the pool doesn't grow too large.

It outputs the value of the best solution found. It scores around 40–60 points, as it can give incorrect results. This solution is implemented in grib1.cpp and grib2.

Brute-force solution with pruning It is similar to the brute-force approach. It places the meridians and parallels starting from the upper left corner and remembers the best computing time obtained so far.

For each partial placing, if for any rectangle,

$$\left\lceil \frac{sum\ of\ all\ computing\ times}{number\ of\ parts\ this\ rectangle\ is\ going\ to\ be\ divided\ into} \right\rceil \geq best\ result\ obtained\ so\ far$$

we know that completing this partial placing would not give us a better result.

This solution is in many cases faster than optimal solutions. However, in some specific situations it can be even worse than the brute-force algorithm.

This solution, scoring around 50 points is implemented in file gris10.

Brute-force solution with pruning and binary search It is the same as the above solution, but instead of improving the best result in each step, it looks for the optimal solution using binary search. It scores 50 points. Its running time can be completely changed by reflecting or rotating the input data. Therefore there are 8 versions of this solution, implemented in files gris2 – gris9.

Brute-force solution with pruning and iteration limit Both of the two previous solutions can be heavily improved by a simple extension: if after a big number of iterations we did not find a solution for a given computing time limit, we assume that a solution does not exist. Both of the two previous solutions can be extended with this trick. All of them are implemented in files gris5 – gris20. They score 70-80 points. The solutions which do not rotate or reflect the input data score 70 points.

Naive algorithms This algorithm initially places all $n-1$ meridians and $m-1$ parallels. In each step it decides (randomly) whether to remove a meridian or a parallel and selects the line to remove in a greedy manner. This process is repeated 1000 times and the best solution found is outputted.

Another naive solution is essentially a genetic algorithm, with the pool size equal to 1. Both solutions score at most 10 points. They are implemented in files grib1 and grib2.

4 Limits

All solutions, except for the brute-force algorithm use little amount of memory. The memory limit of 32MB is sufficient for all solutions mentioned here, including the simplest brute-force approach.

5 Tests

There are ten groups of tests. Each of them consists of 3 to 4 tests.

5.1 Types of tests

This section is only a simplified description of the most interesting types of tests. Check the input generator code for more details.

Distance test The value of each field is a sum of distances to points from a given set. The brute-force solution with pruning runs very slowly on these tests.

Square test The value of field (i,j) is a random integer from the interval $[0, (33 * min(|i-x|, |j-y|)))^2)$, for some x and y. The brute-force solution with pruning runs very slowly on these tests. The genetic algorithm can give wrong results.

Stairs test The value of field (i,j) is a random integer from the interval $C * max(i \bmod p, j \bmod p)$ for some p and C. The genetic algorithm often gives a wrong result.

Chessboard test Each chessboard square consists of several grid squares. The value of each "black" field is a random integer ≤ 1000000, whereas a "white" field value is at most 10 times smaller. The brute-force solution with pruning usually runs slowly on these tests, depending on the chessboard square size.

Steep test The value of each field is an exponential function, taking field position as an argument.

Exponential stairs test The value of field (i,j) is a random integer, lower than $exp(max(|i-x|, |j-x|)$ for a given x and y. Solutions with iteration limits can fail on these tests.

Hurricane (verification) The value of each field in a square of size $\frac{n}{2} \times \frac{m}{2}$ is much bigger than the rests. This is a good test against correcting algorithms, but is not very good gainst others.

5.2 Test data

Filename	n	m	r	s	Test type
gri0.in	7	8	2	1	Sample test
gri1ocen.in	6	6	2	3	All computing times = 1
gri2ocen.in	2	2	1	1	
gri3ocen.in	5	5	4	4	
gri4ocen.in	18	18	1	1	
gri1a.in	2	2	1	1	Random
gri1b.in	10	2	3	1	Random
gri1c.in	2	10	1	3	Random
gri2a.in	6	6	5	5	Random
gri2b.in	8	7	6	4	Random
gri2c.in	8	8	4	4	All computing times = 0
gri3a.in	4	10	2	5	Distance
gri3b.in	10	4	5	2	Distance
gri3c.in	8	8	4	4	Stairs
gri4a.in	10	10	6	4	Random
gri4b.in	10	5	7	2	Square
gri4c.in	5	10	2	7	Square
gri5a.in	15	7	8	2	Exponential
gri5b.in	14	16	7	8	Stairs
gri5c.in	16	14	8	7	Stairs
gri6a.in	10	18	6	8	Exponential stairs
gri6b.in	18	18	5	5	Square
gri6c.in	17	16	11	10	Exponential stairs
gri6d.in	17	17	8	8	Hurricane
gri7a.in	16	16	12	12	Square
gri7b.in	16	16	12	12	Square
gri7c.in	18	18	8	8	Stairs
gri7d.in	18	17	11	11	Exponential stairs
gri7e.in	18	17	9	8	Hurricane
gri8a.in	17	18	13	14	Square
gri8b.in	17	18	13	14	Square
gri8c.in	17	17	7	7	Stairs
gri8d.in	18	18	12	12	Exponential stairs
gri8e.in	17	18	8	9	Hurricane
gri9a.in	18	18	11	12	Exponential stairs
gri9b.in	18	18	12	11	Steep
gri9c.in	18	18	7	7	Stairs
gri9d.in	18	18	7	7	Stairs
gri9e.in	18	18	9	9	Hurricane
gri10a.in	18	18	11	11	Square
gri10b.in	18	18	9	9	Square
gri10c.in	18	18	10	10	Exponential stairs
gri10d.in	18	18	7	7	Distance
gri10e.in	18	18	9	9	Hurricane

5.3 Remarks

The model solution run in less than two seconds on a 2GHz Core 2 Duo mobile CPU. The gri1 solution has a slightly different running time, but it should also score 100 points.

Time limits should be set according to the model and alternative solution running times. As the contestants will know the time limit in advance, it should be strict, as a bigger time limit makes it easier for incorrect solutions (i.e. genetic algorithm) to score more points. I suggest time limit is around two times bigger than the maximal running time of the model and alternative solutions.

5.4 Expected number of points

As there are around 30 solution and over 30 tests, a complete table would not fit.

File names	Expected number of points	Solution type
gri, gri1, gri2, gri3	100	Model and alternative solutions
gris1,gris11	40	Brute-force solution
gris2-gris9, gris12	50–60	Brute-force solution with pruning and binary search
gris10	60	Brute-force solution with pruning
grib1–grib2	40–60	Genetic solutions
grib3–grib4	0–10	Naive solutions
grib5,grib13	70	Brute-force solution with pruning and iteration limit
grib6–grib12, grib14–grib20 iteration limit	70–80	Brute-force solution with pruning, and input data rotation

6 Problem statement changes

1. Minor denotation change: k and l were changed to r and s, as l was too similar to 1.
2. Output format change: instead of outputting a complete solution, it is sufficient to output an optimal computation time. I am deeply convinced that, it does not make the task any easier.

7 Remarks

Tests The number of tests can not be easily reduced. Every tests plays an important role, as there are many, completely different incorrect solutions. Before removing a test, double-check whether an incorrect solutions does not score too much points. If you want to change the input data generator, be aware that the random seeds were carefully chosen. The functions which require random-seed tuning are marked in the input data generator.

Package size I do realize that there are way too many solutions and tests. Unfortunately I was unable to find a symmetric test, which could prove all brute-force solutions wrong.

Missing solutions Solutions gris3-gris10 do not have their Pascal versions. All these solutions either run very quickly or terribly slowly. Therefore, including Pascal versions seemed unnecessary.