Discussion and explanation of the Problem Set

1 Foreword

The ACM International Collegiate Programming Contest (ICPC) provides college students with opportunities to interact with students from other universities and to sharpen and demonstrate their problem solving, programming and teamwork skills. The contest provides a platform for ACM, industry and academia to encourage and focus public attention on the next generation of the computing professionals as they pursue excelence.

Another regional contest in Tehran passed. Sweet and bitter experiences and memories are still with us. Two teams (maybe more) qualified for the World Finals to be held in San Antonio, Texas. Congratulations to them, the winners of this year's regional contest. Their performance was great. A first-timer that could come up with third place at last, brought joy to Shahid Beheshti university. On the other hand, Ilúvatar team, from Sharif University of Technology, solved more problems than any other, and proved their excelence in many aspects of a programming contest. I wish them both good luck, in the upcoming World Finals, in Spring 2006.

Looking from a higher standpoint of a competing university student, one might see the deeper effects of having this contest in Iran. The seventh year of this event let more than 70 teams compete, and demonstrate their skills, strive for success, and experience happy and sad moments. A great movement, that can be felt, is looming over computing disciplines in Iran: improve programming skills. We want to contribute to this force, so that the challenge helps our otherwise-doomed state of programming in the universities. No one doubts that the advent of Visual programming languages, among other things, has paved the way for people to develop applications without much thought. Yet, put in a new unpredicted situation, a poor programmer cannot advance a step, and must wait for someone helping him/her in the way. This way, we cannot explore new areas of science, because we are accustomed to using semi-ready recipes for doing everyday stuff. In this situation, holding such contests, as the mission of ACM states, lets people not forget some old skills, and allows for industry centers to find skilled and seasoned resources.

Having seen people who participate in similar contests, and get nothing of the event but mere participation (and possibly some cool food!), I had a plan to build a booklet to discuss the techniques for solving the problems, so that a review of the contest material and how the team behaved during the contest, would help our community flourish better. Besides, this booklet could be a good guide for new-comers to get familiar with the nuts-and-bolts of a programming contest. Unfortunately, work load did not allow me to accomplish my plan before the contest. The enthusiasm of the people after the contest, though, made me prepare this as soon as possible, so that we (the whole Iranian ACM community) can dream of a world-class computing force, and think of better accomplishments in this field, e.g. in international contests. However some other hinderances did not allow for its earlier preparation.

At the end, I would like to thank all those who helped me prepare this booklet, now in your hand, without whose fruitful comments and discussions, strong motivation and support, no such thing would have ever been possible. Among others, Mr Babak Behsaz and Mr Arash Rahimi from AmirKabir UT, Mr Yasser Zhian Tabasy from IAU Mashad, Mohammad Hadee Foroughmand Araabi and Siavosh Benabbas from Sharif UT, really contributed to this work. Anyway, I hope that our teams plan for their preparation in time, and practice well. We have about one year to show up in another regional contest. With practice, I think, we will definitely have much better teams (or teams in better state!) next year. Thus, I wish you all good luck in your way to the next regional contest.

Mohammad Hossein Bateni
2nd February 2006

2 General Issues

Being familiar with the contest environment is an important fact in programming contests. Why does the organizing committee bother to put the description of the contest environment on the web site? To fill some space? Absolutely not! Though C++ is C++, it is very important for a contestant to know which version of what specific brand is the very compiler of the contest. The same goes with Java, be it JDK 1.4 or the new totally-different JDK 5.0 with lots of new features one can make use of. Pascal (better say, Delphi) on the other hand, is also changing, but the new features are not usually related to such contests. However, as Pascal is not to be present in the 2007 World Finals any more, teams are encouraged to use other languages. Therefore, look frequently at the web site, the blog, and the mailing list to find out more about the computing environment, especially the exact compilers present on the contest day. More specifically, learn to work with the IDEs and editors of the contest, and learn the advantages and disadvantages of different languages. Teams are not forced to use one language at the contest and although not recommended, they can choose the implementation language according to the problem needs and the one who is coding at the moment.

The ACM-ICPC contests need various skills: among them are the fast and careful reading of the problem statements, sketching the solutions, be it algorithmic or not, fast and bug-free programming, teamwork, and good management in severe conditions. To be able to make good problem selection decisions, a team needs good reliable raw data about the problems, and thus fast yet careful reading of the problems is a must. Some teams skim through the problems and make an estimate of their difficulty before reading them all completely. Wrong submissions for the *problem B*, even from the top-ranking teams of the contest, showed that we cannot yet boast of this ability: not paying enough attention to the details of the problem statement was a big barrior for some teams in solving the second problem. Another witness is the now-infamous *problem E* that turned into the major point of debate after the contest. This ability, fortunately, is relatively easy to improve: by reading and solving many problems, highlighting important points in the statements, and talking to others, people can learn to read, carefully and fast, the problem statements.

Next in the order of time, comes the problem selection phase: you should choose the problem to tackle. No one doubts that different teams might have different skills, be it programming, thinking or algorithmic. A successful team should always choose the problem that best suits them; if a team have never got a geometric problem accepted, they'd better avoid such problems in the contest, if possible. Large (relative to last year) problem count is another point, highly blamed for the failure of teams, in this year. Primarily, numerous problems allow problem setters to devise problems of different categories with different difficulty levels. Besides, they enable teamwork to be better demonstrated. Were teams given three problems, teamwork would not have been felt at all. Moreover, it helped teams remain busy till the last moments of the contest, while in some contests the last hour is just waste of time, and load of stress.

First six problems are accused of being so simple and the last three of being too difficult. As Mr Zhian Tabasy have interestingly noted the scientific committee was scared of last year's experience; only about 20 teams could solve any problems! The problems were not very harder than this year's but the fact that they were not ordered by their difficulty lead to shameful poor results from participating teams. One of the main concerns of the organizers—the lively efforts of Iranian universities in this field—approximately disappeared, as we did not see some universities even trying to send teams to the contest, this year. Disagreeing with someone claiming the World Finals' problems are usually algorithmic and mental, I belive in the contrary: the World Finals' problem set usually focuses on simulation, and practical aspects of general life issues. The correctness and good understanding of the problem set is much more important than solving complicated algorithmic problems. Mr Babak Behsaz, a former participant of several ICPC contests, noted that the 2004 World Finals had absolutely no problem of the least algorithmic nature, while half of the ten problems were related to the scary field of computational geometry. It is also

worth mentioning that American regional contests, usually do not contain many algorithmic problems. Additionally, consider *problem H*: teams that think their algorithmic wits have been wasted during the contest, should more blame their problem selection (and also in part their problem solving) skills. Mr Zhian Tabasy mentioned that this problem is fairly easy; he *is* right. Furthermore, having solved the problem, a team could write the program in less than 15 minutes. Compare with some problems, that are easy to solve, but long and error-prone to implement.

Time management, and keeping the good work is another very important issue. Some teams, getting one or two wrong answers, loose their strength and cannot solve problems, any more. On the contrary, good teams should be able to tolerate this situation and continue their way during the contest. As an example, *Evenmore!!!* from Sharif UT, solved the first problem in 9 minutes and the second one in the 114th minute. They had a problem with the statement of *Problem B*. While many teams had solved two, and even three, they had only solved one problem. However, they could recover from the bad condition and return to their normal state, and at last, they could get the second place. We should learn to face bad situations. If a couple of wrong answers destroy our spirit, it is quite possible that we get out of the contest soon.

Anyway, one cannot help admitting that our regional contest teams are not as strong as the top world finalists. It would be better to say that our teams did not improve their skills (just like americans) while others (mostly east asian and northeastern european) dramatically improved. As an example, I ask you to take a look at the *Northeastern European Regional Contest (NEERC)*. The contest is held in St. Petersburg, Russia. They are among the top teams of each year's world finals. Looking at their 2002 regionals (three years ago), one finds out that best Iranian teams are comparable to them. The contest data is available on their web site. I strongly recommend you all to take the contest, and even encourage your university teams to take the contest together. They were 9 problems posed; the first team solved them all in less than four hours! I'm almost certain that no Iranian team in the current state, can solve the hardest problem even given 5 hours! My experience shows that good Iranian teams can solve about six, maybe seven. But, don't worry! I think about 5 teams solved more than six problems in their regional contst, of course three years ago.

Now, take a look at the 2003 contest in the same region. Eleven problems were put for the contestants. The best team solved 10. The best Iranian can solve 7 or 8 at best; it's not bad. But, I think that no more than 10 teams of our regional contest, can solve six problems! And about 20 teams solved seven or more in their contest. Yet, the next year is more interesting. Twelve problems were posed; they seem really to be improving, because problems are not easier, only more! Two teams solved all of them—one of them in 4:30. Five teams solved ten or more. Again, take the contest, and see how many you would solve. In case you are making a mistake, I am to correct you: I am in no way trying to humiliate Iranian teams; I am Iranian myself, and just reciting a sad story to you. I remember in one of these contests, a certain team had solved five problems in about an hour with penalty around 190 minutes, while the poor team was not qualified to the world finals at the end, because another team from the same university ranked better at last.

I have made an elaborate search to find out what they do. A few days ago, I found out the IFMO teams (the hosting university and always one of the best) have had about *forty* group programming contests to prepare for this year's contest! So, they are not magicians as I first thought. They are just like Mr Behsaz noted—practicing too much. I believe the best method for practice is to have joint (even with other universities) contests. You can use online judge web site for the problems, and take a contest. It's not a major issue if it turns out that you have already seen a few problems. But, this is not all. Review the contest, and the problems. Discuss the mistakes you made and the problems you could not solve. Review your programs, and each others'. Try to find others' solutions for them, and learn from them. Even discuss the problem selection mistakes you made. What problems should you have chosen first? what next? ... Anyway, have you ever tried to write a program you have once written (or tried to write) before? You make much better design decisions, and make very fewer mistakes. Human beings learn!

One point about the *Time Limit*. Some clarificiations in the contest asked for the time limit. They were all, to the best of my knowledge, answered "No Response", or "A reasonable amount of time". I'd like to note that this is exactly the same answer you would get if you asked it in World Finals. Of course, the (literally) frightening Chief Judge there, would also hit (or beat) you (if not by hand, by his bitter tongue). The main issue in ACM-ICPC is not time limit. Usually a fair polynomial time algorithm would do. And, I am sure that some of the time limits there are about 10 seconds while some are more than three minutes. You should be reasonable and think well. Think what a problem consists. If a problem is dealing with inputs, and formatting output, usually a severe time limit will not prevent your acceptance.

Take a look at the contest problems now, and decide what time each problem actually needed for reading, understanding, sketching and implementing a solution. Try to estimate the best plan in the contest, and determine how many problems a good team could solve.

3 Problems

A. Ancient Keyboard

The problem description is clear. One should only note that the interval of effect of each pair of key presses is semi-closed. Besides, it is worth mentioning that the output can contain empty lines; because the input n might be zero. There were several clarification requests that asked whether the intervals can intersect, or whether there are two intervals corresponding to a single letter, or similar things; as all of these are clearly answered in the problem statement, all received "No Response, read the problem statement."

The problem is rated "Trivial" as it assumes no programming knowledge of any kind. One should only read the problem statement and code it into the computer. As expected, this was the first-solved problem of the contest, where "Dovienya" team solved it in 5 minutes. The acceptance ratio is 68%. The sample test data for this problems, like some other problems of this contest, were carefully designed to help teams that do not read the statement carefully. That is, if a team did not pay attention to some of the details, they would find it out testing their solution with the sample data. Note that this is not the normal strategy in ACM contest, where they normally provide teams with easy sample data, while the real test data is totally tough.

Possible Extensions As the ranges for the input parameters are low, a simple array-based solution and traversing all time units would do in the alotted time. But consider the case that the press and release times were so strictly bounded, e.g. suppose they are at most 10^9, and of course change the output format so that the same consecutive letters can somehow be compressed. How would one solve the problem then? You should think of a better data structure, and note that the intervals that can form in the output, are at most $2n - 1$ distinct intervals, bounded by the a_i's and b_i's of the input.

B. Best SMS to Type

Again, this is a straightforward problem: given a message you should count the number of consecutive characters of the message that are mapped on the same key (excluding whitespaces), and add the fixation times for them to the normal pressing time of all the characters. The main issue in this problem was that consecutive whitespaces do not need fixation time; this is explicitly stated in the problem statement. Nevertheless, many teams (including the top ranking teams) had this problem in the contest. It seems that we have to work more on the skill of reading the problem statement *carefully*.

Note that $1 \leq p, w \leq 1000$ and the message length is at most 1000. Thus, the result would fit in a normal

signed 32-bit integer. The low range for the parameters makes coding time more important than the solution efficiency. A simple way to do it, is to declare two arrays: one containing the first character of each key, and the other one containing the number of characters on each key; then a simple loop on this array would give you the key a specific letter is mapped on. An alternate implementation is two have an array of length 26 initialized at coding time. Further note that, the input is given in milliseconds and the output also asks for milliseconds. But in some ACM-type problems, different units for input and output make the programmer's task difficult.

Once more, the problem is rated "Trivial" as it assumes no programming knowledge of any kind. One should only read the problem statement and code it into the computer. The first correct submission for this problem was in th 17th minute; "Dovienya" solved two problems is 17 minutes. The acceptance ratio is 49%.

C. Changing Phone Numbers

The problem has a long description. One should carefully read the statement to avoid having extra (wrong) assumption about the input data, and understand the problem exactly. There are some points here that lead to *wrong answers*. First, the input gives a number in time y_1 and asks for the same number in time y_2. As the area codes change, the area code of this number is not necessarily the same in times 0 and y_1. One should first find the correct area code and then in the order of increasing time, apply the rules of the corresponding area.

Another point accounting for some wrong answers is that the input description states the *area-code* is at most 5 charachters; it does not state that the area codes would not get larger in the third rule! As a hint for teams, we have added such a rule in the sample input, though the rule was not used in the sample output. A careful study of the sample input would have definitely lead to this understanding.

This problem has a simulation nature, and involves searching, mapping and etc. What is the code of a city? What is the city code in a specific year? What rules apply in a specific interval? Which for this specific city? These searches and mapping can be done with different approaches, giving different complexities; binary search trees, tries or lists can be used. As for this problem, which is among the easy problems of the contest (remember that problem are nearly arranged in the easy-to-difficult order), the time limit allows one to consider all the rules for each query.

This problem is rated "Easy". Although the first correct submission was in minute 156 (again by "Dovienya"), and the acceptance ratio is as low as 19%, teams were just scared of the long problem statement and went for easier problems at first. Actually a few teams had the time to try this problem, and about half of them could get it accepted.

D. Dramatic Multiplications

The problem has the shortest description among all the problems. This is the first problem that needs some thought: we can find the next digit by a simple calculation; thus, we can iterate on the digits until the first digit repeats. It is also easy to show that the length of such a number is no more than 100 digits. So, we can stop there if we could not find a good number. The major mistake, and most of clarification requests here were regarding the size of the resulting number. Some asked about the meaning of *integer* and wanted to know how many bits it would have! Integer is a mathematical concept, not a programming language-depent issue. Note that solving this problem does not need implementing big numbers in any way; just a couple of simple one-digit multiplications and additions!

Starting with the given rightmost digit, one can find the next digit by a couple of simple mathematical operations like multiplication, addition and taking modulus. We repeat this operation until we find a dramatic number. First note that a dramatic number — as well as any other number — cannot have

zeros at the left side! Furthermore, if we have repeated the above operation and we have come up with the same digit twice, and both times with the same carry, we are in a loop and will never find a dramatic number. As there are a total of 100 different such digit-carry combinations, simulating the process for 100 steps is enough.

Another point accounting for some wrong answers is that the implementation for some teams did not give one-digit answers. They assumed there is no one-digit dramatic number. I do not remember any clarification request regarding this issue.

This problem is rated "Easy". The first correct submission was in minute 37 (by "Ilúvatar"), and the acceptance ratio is 22%. Therefore, this team solved three problems in 37 minutes.

E. Entertainment

This problem has by far been the most frequent point of debate among teams, after the contest. Beside the lengthy tasks one should have done in the implementation, careful reading of the problem statement was a serious necessity. First, I should announce that it *was* necessary to print the possible trailing white spaces of the rows of the table. Submissions that only neglected these trailing spaces got "Output Format Error". As some teams objected to the necessity of these white spaces according to the problem statement, I want to attract your attention to some parts of the problem statement, as to clarify the presence of the necessity in the problem statement.

- The 3rd line of the statement: "..., a table entry which is *non-blank* ...", which implicitly states that some table entries might be blank.
- The figures in the statement, which clearly reserve a space (empty cells) for the blank characters.
- The last line of description: "... report the *final* resulting table ...".
- The last line of Input Description: "... Items are always inside the *current* table ...". This and the sentence referred to in the previous item, state that the size of the table might change during the operations, and you have to print the final table.
- The 2nd line of Input Description: "... Characters in the *initial* map are all lowercase ...". That is, the next tables might have characters other than the lowercase letters of the alphabet.
- Finally, I think there was a clarification sent to all teams: "the moves given in the input, all point to non-blank characters". Why should we have said this, if the spaces were not part of the table and the input description ensures you the items are inside the current table?!

As spaces are considered part of the table, they should be printed in the output.

Deleting the required cells in each step can be done with a DFS or BFS. It is important to shift the empty rows and columns after each step, and not once after the final step! Note that this deletion changes the dimensions.

The input format of the problem is also different from the other problems. The dimensions of the initial table is not explicitly given. It's not a lot more difficult to read and parse the input. But if one does not pays attention to this difference, he might get into problem, changing this later. Some teams thought there were empty lines in the input, which was not correct; working with `getline()` in C++ requires some care anyway.

This problem is rated "Medium". The first correct submission was in minute 86 (by "Ilúvatar"), and the acceptance ratio is 5.5%.

F. Fortune at El Dorado

The problem is to find a rectangle containing (border counts) the most number of trees, while the area of the rectangle does not exceed a given parameter. Most of "wrong answers" did not count the trees on the border although explicitly stated in the statement. A few thought the area must be smaller than

A. I think this thought orignates from the second sentence of the second paragraph of the statement. Reading it more carefully, one comes to the conclusion that although the king has told Kamran that the area should be A, choosing a rectangle with area less than A is also acceptable. Furthermore the problem statement explicitly forces the rectangle to have sides parallel to the axes, otherwise in a real ACM contest, if this is not stated, and if the clarification request is answered with "No Response", the sides of the best rectangle can possibly have sides not parrallel to the axes, which makes the problem a lot more difficult.

To solve the problem, you have to decrease the number of rectangles you should consider. The typical solution to this problem is $O(F^3)$, while some cleverly-written $O(F^4)$ solution would also finish in the allowed time limit. The acceptance ratio is 11% while about half of the teams who attempted succeeded. This problem is rated "Medium", as a little algorithmic approach was required to overcome the time limit. "VJ13" from Sharif UT, was the first to solve this problem, in 154 minutes.

In a typical solution, one can use a DP approach to fill an array c_{ij} to hold the number of trees in a rectangle with one corner at $(1, 1)$ and the other at (i, j). This can be in $\Theta(mn)$ time and space. Thinking a little, one comes to the conclusion that we can move the best rectangle so that three sides of the rectangle have a tree on them. Having the above array at hand, one can calculate the number of trees in any rectangle in constant time. Loop over possible x and y values for the bottom-left corner and then on the y coordinate of the other corner; the maximum value for the x coordinate of this corner can be found by a simple division according to the given area A. Paying attention to the non-zero area condition, it's not difficult to find the best answer.

G. Griddy Hobby

The drawing always starts on the border of the rectangle, which is also a corner of one of the border cells. The statement is clear, though some teams asked for the meaning of "non-overlapping rectangles"; it was a mistake, "non-overlapping" should have been replaced by "minimal".

There are a number of solutions to this problem, ranging from straightforward long (about 150 lines of C++ code) to witted short (about 30 lines of Delphi code). A straightforward solution is to draw lines and use some graph algorithm (DFS or BFS) to compute the connected components and then count the ones that are rectangles. Some dirty code it would get, and it is worth mentioning that DFS users (specially in C++) should be aware of "stack oveflow" problems!

Another more witted solution is to draw the lines step by step, marking the points visited, and increment a counter each time we pass a point marked before. So easy to implement! Some teams think that the problems needed lengthy programs, but it is usually possible that thinking more leads to shorter, less bug-prone programs.

This problem is rated "Medium", and the acceptance ratio is 30%. This problem was solved in 122 minutes, by "`Untitled.cpp+`" from Shahid Beheshti University.

Possible Extensions The above solutions work, because the dimensions of the board is not large. For practice, you can consider the board is much larger, say $R, C \leq 10^6$, and then try to solve the problem. Obviously, the simulation approach cannot be taken.

H. Hotel

The statement is clear: we can put some men, some women, or a couple inside a room. Of course, the room cannot be filled more than its capacity. This is the first true algorithmic problem. A straightforward dynamic programming approach is as follows: solve the more general problem of optimizing the situation,

while we have the first a rooms, b men, c women and d couples. Because computing the optimal solution of any of such sub-problems needs the solution for sub-problems with smaller a's. Thus filling a dynamic programing array in increasing order of a is possible. But, this *easy* solution would get "Time Limit Exceeded" to make algorithmic teams even more happy! A nice point is that it is never needed to have more than one room assigned to couples. Otherwise, we can change the arrangement to obtain an assignment at most as costly, with at most one room for couples. Taking advantage of this fact, the problem can be solved in $O(m \cdot f \cdot r)$ time with memory $O(m \cdot f)$.

Mr Babak Behsaz notes that when one faces such problems with lots of constraints, it's a good idea to consider the special cases of the problem, and similar variation. In this way, he believes, one can get a feel of the difficulty of the problem as well as its structure. He first tries to solve the problem where there are only men; and then the case where there are no couples. Having solved this problems, he can easily get to the basic solution given above. Proving a lemma about the number of couple-assigned rooms, he reaches the better solution with respect to time. Decreasing the space complexity of the problem, is a well-known technique in DP solutions. He also notes that the solution fits in a simple 32-bit integer, and this point is important in ACM contests.

To conclude, this problem is rated "Hard", because of the algorithmic nature. No correct submission for this problem, during the contest.

I. Intercepting Missiles

Lengthy problem statement, in addition to the large number of problems did not let any team work on this problem seriously. The problem is: in a two-dimentional world, we have some missiles ready to launch on specific locations all on the x-axis. A number of targets are moving to the right in the sky; they do not change their y-coordinates. Targets are of two types: good and bad. We want to hit the most bad targets while avoiding any good target from being accidentally hit. With some simplifying assumptions in the problem statement, the problem is "maximum matching" and the underlying graph is bipartite. Thus, we just need to calculate the graph and perform a matching algorithm.

The major problem is to find which missile can hit which bombers. For each missile and each target, we can find the time interval in which it can be launched so that it would hit the target. Then we would subtract the intervals relating to lower targets from the interval of a specific target. If the final interval is non-empty, the missile can hit this target.

The problem is rated "Hard", and no team submitted a correct solution.

J. Joy of Mobile Routing

The last problem: a (relatively!) full-fledged geometric problem. Note that each antenna can give service to all the points that are either in its row or column. On the other hand, if another point neighbors a building of height zero (that is no building), it might see some part of an antenna. Also, it is obvious that checking whether each point can see the top of some antenna is enough. This problem contains several geometric special cases that make it more difficult. After finding all accessible points, a fairly easy BFS would give us the shortest possible mobile route.

It is important that the final point need not be accessible through the network. Besides we should take care of the case where the start and end points are the same. Like many other geometric problems, it is vital to code the problem in a structured and clear way; otherwise bugs arise and debugging would be a pain in the neck.

The problem is rated "Hard", and no team submitted any solutions.

4 Best Strategy

In this section, I will give the details of a sample successful strategy. In this strategy, I suppose that all three contestants have good abilities, in reading the problem statement, solving and implementing them.

Figure 1 shows a sample strategy for such a team. Each row describes the status of a contestant, and the letters in the box indicates the problem he/she is working on. The dashed regions correspond to the times the contestant is using the computer. I'm going to explain it here:

Right in the beginning of the contest, the three contestants read the first three problems, having heard that the problems are arranged in an easy-to-hard order. The first contestant finds out `A` is really easy and starts to code it. In the tenth minute, they have one accepted problem; compare to the fact the the first accept for this problem was in 7th minute in our regional contest.

The same situation goes with the problem `B`, where the second contestant finds it easy, and thinks exactly how he is going to code it. The implementation is started in the tenth minute and they get it accepted in the 20th minute.

With the third contestant, the situation is a little bit different. After reading problem `C`, the contestant tries the next problem, because `C` is a little complicated. She finds that `D` is easier to implement. So she starts coding it after they got `B` accepted. She also recommends the first contestant to work on `C`. The problem `D` is finished after twenty minutes of coding.

The first contestant, being recommended by the third, starts to work on `C` and starts coding in the 40th minute, while she helps him to do so. Thirty minutes of terminal time seems enough for coding and probable debugging, if enough time has been spent for developing the implementation details and decisions. In 70 minutes, they have four accepted problems.

Regarding the problem `E`, the second contestant devotes thirty-five minutes to thinking and taking implementation decisions, building the framework in which he would like to code the solution and etc. Then he has twnety-five minutes for coding and debugging on the terminal. A similar story goes with the third contestant and the problem `F`.

The first contestant devotes forty-five minutes to thinking and preparing the implementation details of `G` which has a fairly straight-forward solution. After working for thiry minutes, he finds out there is another more witted yet simpler solution. For the remaining fifteen minutes, he works on this solution. The coding takes twenty five minutes (or less).

After two hours and a half, this imaginary team has got seven accepted problems. The second contestant can definitely solve `H` in forty minutes, and the implementation obviosly takes less than twenty five minutes.

A similar story goes with `I` and `J`, but here the second contestant, after some refreshment, can help the other two in their tasks.

In most of the problems, I allowed more time for coding and debugging, in case something bad happens. Yet the ten problems were finished in 260 minutes. Note that it's a good idea to distribute the load between the contestants if possible. In the extreme case, it's very bad to put the most of the work onto one. He/she will be exhausted if there are more than a few problems! A lone-coder strategy, as Mr Zhian Tabasy noted, is probably doomed in such a contest.

You most probably think this is a fake strategy: It is not possible. Both yes and no! No, it's not possible in the current situation. We witnessed no team could keep up with its standards after the first hour. And yes, it is actually possible because if we improved our skills to near (or even pass) the Russians', we can achieve this. It's interesting to mention that once I told this to a friend, a participant of the same contest, he gave me another timeline, where the hypothetical team would have solved the ten problems in 170 minutes!!

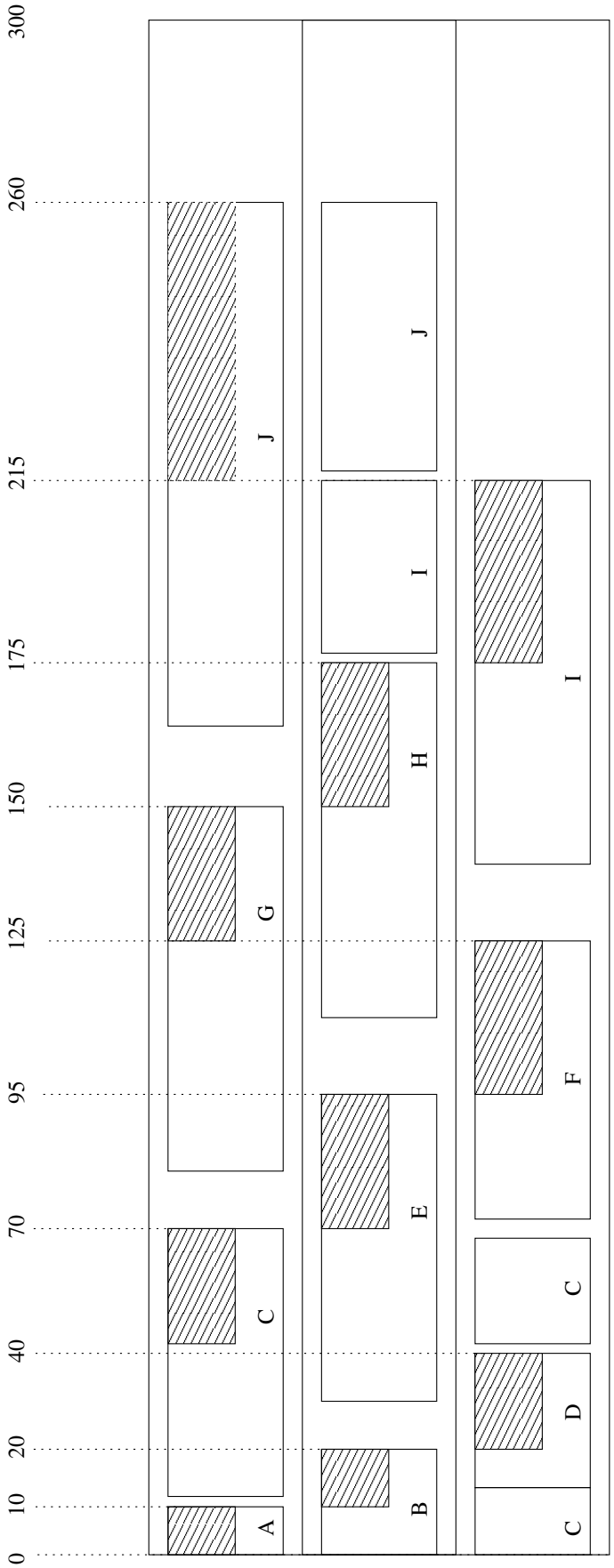

Figure 1: A Good Strategy.