Crossroads
The ACM Student Magazine

Observations on Teamwork Strategies in the ACM International Collegiate Programming Contest

by ***Saman Amirpour Amraii***

Introduction

Thirty years ago, the first International Collegiate Programming Contest (ICPC) **[5]** was held by the Association for Computing Machinery (ACM) **[1]**. Since then, the number of universities participating in this challenge has increased each year. It is not just a programming contest; it challenges contestants' algorithmic knowledge, coding skills, ability to cope with stress, and ability to cooperate efficiently within a team.

Achieving these abilities takes countless hours of practice and experience. During past years, ACM/ICPC teams from Amirkabir University **[2]** have gathered a lot of experience in this field. Each year, they use different methods and strategies, based on observations of previous year's strengths and weaknesses, to develop better teams for the next year's competition.

These techniques are completely experimental and are not based on conventional methods. Although these strategies have been tested practically and altered in order to become more effective, they must be appropriately customized for individual teams. Each team member is unique in their abilities in coding, problem solving, and teamwork. Consequently, their team's overall strengths are different. Hence, each team must tailor these strategies to their own team's requirements.

The main subject of this article is teamwork. The strategies hereunder try to reveal the extreme hidden power of working as a team.

How to Form a Team

Your team is your most important resource. The team serves as a vehicle for cooperation, knowledge sharing, and contribution in order to achieve better results. Never forget that there are three people in each team. Each team member has unique knowledge and strengths. Therefore, each member must be utilized properly.

There are two main methods for forming a team:

1. You know one another before the formation of the team.
2. The university's coaches choose the members of each team.

In the first method, the team is already formed. They know one another, and they are usually friends and want to work with each other to be a part of a team. Naturally, collaboration in this team is high, and cooperation is high. On the other hand, the team members may have similar weaknesses. For example, a team could consist of good programmers but not good problem solvers. Such teams must concentrate on their weaknesses and try to overcome them.

In the second method, the university's coaches are responsible for making new teams. A good way for forming strong teams is to administer several preparation exams and to hold preparation courses. These exams and courses can help the coaches to determine the strengths and weaknesses of each participant. They can then group new teams and observe the cooperative abilities of each person. Based on this information, a coach can form strong teams.

Two main characteristics of a good team are:

1. **High cooperation:** 3 * 1 = 4! [4] If you share your strengths with others in the right way, your outcome is greater than when you work alone. Teamwork is a very important factor. Gathering the best programmers results in nothing if they hate each other!
2. **Balancing skills:** No one is perfect; we all have our own strengths and weaknesses. A good coach must form a team in a way that covers the weaknesses of one member with the strengths of another. Your team is strong

when it is united.

For each required field, it is a good idea to have at least two people who are adept in that field. Imagine a situation in which there is only one skilled programmer on the team, and the other two people are adept in problem solving. This team can easily face a difficult situation in the competition, as their programmer may tire early. In this case, the other team members cannot help the programmer. Thus, overlapping skills are desirable.

Remember, these methods and guides are for forming an ideal programming team. In reality, it is very hard to accomplish. In fact, you must constantly direct your team in order to get closer to this level of perfection. That is what practice is for!

How to Practice

It does not matter how much you know: knowledge without practice is worthless. It is a very good idea to form the team and start practicing one year before the contest. You can practice for one or two hours per day at the beginning and increase the practice time as the contest approaches. Some teams practice for more than eight hours per day.

First, you must find the team's weaknesses in these areas:

1. Individual academic background
2. Teamwork ability

Each field is important. Practicing individually and practicing in a team have different methods and result in different outcomes. You have to do both of them. In this article, we only cover teamwork practicing methods. However, if you do not know where to start for individual practicing, you can read "Introduction to Algorithms" **[3]** as a comprehensive reference to the fundamentals of algorithms. Also, the Valladolid problem set **[6]** is a rich online source of problems. This site has an online system that can judge your solutions.

In the teamwork practicing method, sharing experiences is the key. If you have solved a problem, show it to the others and discuss the important points.

Another important skill is the ability to read other members' code. In other words, the team must have the same coding style. Thus, if someone faces difficulties implementing a problem, the others can help the programmer.

Another important technique is taking specialized exams. In this type of exam, a particular task is done individually. The weakest programmer in the team can be tested in programming, for example. The pressure of this exam can improve one's weaknesses. It is desirable for members to accept different responsibilities during each exam. This way, each will be more prepared in each task.

One of the most dreadful problems in a contest is debugging. It consumes a lot of energy and time. It places a lot of stress on you. Moreover, it brings you some penalties! So, what is the solution to this problem? Write bug-free code! In fact, the only solution is to increase your accuracy during implementation. A good practice technique is to debug code without using the computer. Use a piece of paper and pencil for tracing the code and finding the bug. You will learn common mistakes, and your accuracy will improve. To make this practice harder, debug one another's code. You will learn your team members' common mistakes too.

Practice understanding questions correctly. Get used to underlining important parts of each problem. Read each question several times. Make sure that nothing is omitted, and then begin coding. It is a nightmare to find yourself implementing an incorrect solution.

In some competitions, you can bring some code into the contest. This prewritten code can help you tremendously, so test it as much as you can during your practices.

Teamwork Strategies

This section tries to answer these questions:

- What are the tasks and responsibilities of each of the team members?
- How should team members communicate with each other?
- What are the pros and cons of each teamwork strategy?

In general, there are two main strategies: individual-based methods and group-based methods.

Individual-Based Methods

If you are a "super-coder," this is the best method! Each person picks up a problem, solves it, writes the code, and receives the acceptance. No time is wasted for communication! In reality, most people are not super-coders. If a team member encounters a difficulty, other team members cannot help. They cannot read the super-coder's mind, so the others have no choice but to wait until that person solves the problem. Thus, this method is not appropriate for most teams.

Group-Based Methods

3 * 1 = 4! In these methods, you can see the power of teamwork. When you work in a group, you can share your ideas with others, and you can use their suggestions in order to improve your solution. If you have a bug in the code, there is someone else who can help you find it. In cooperation, your quality of outcome is increased.

These types of methods can be broken down into three submethods: the specialization method, the duty-based method, and the manager-based method.

The Specialization Method

It is common that one team member is more adept at solving backtrack problems while another can solve dynamic programming problems more easily. So, why not use each person in one area of specialization? This model is called the specialization method. In fact, this is a popular strategy.

The advantage of this method is that each problem is solved by the strongest person in that area. However, there is not a high amount of teamwork. People pass on problems that are in another's area of expertise. Team members cannot help each other in debugging because each member has a distinct area of expertise.

The Duty-Based Method

In this method, instead of assigning whole problems, problems are broken into tasks, which are assigned to each team member. It could mean that:

- The first team member reads the problems and solves them (the solver).
- The second team member implements the code (the coder).
- The third team member helps both (the debugger).

This is a highly communicative method. All of the team members are engaged in a problem. There are always two people to solve a problem (the solver and the debugger), and there are two people to implement it (the coder and the debugger). The role of the debugger is important. The debugger helps the solver find the correct solution. The debugger also reads each line of the code while the coder is typing it. Consequently, the number of bugs in the code is greatly reduced.

The main weakness of this method is that there is only one primary coder in the team. Coding takes a lot of energy and concentration. Frequently, a coder is exhausted in the last two hours of the contest.

The other problem is that there is no manager in the team. It becomes hard to decide what to do in these common situations:

- Which question should we solve first?
- If we have solved more than one problem, which one must be implemented?
- Should we try to debug erroneous code, or should we move on?

This method is most appropriate for a team with both practical and theoretical skill sets.

The Manager-Based Method

There is a famous proverb in software engineering that says, "The sooner you start coding, the later you finish!" The gist of this proverb is to put careful thought to the solution before coding, in order to prevent unnecessary bugs.

Do not create bugs! This is an important rule of thumb in the ACM/ICPC. The manager-based method concentrates on producing error-free code.

This method is derived from the duty-based method. There is a solver and a coder, however, the third person has a new task: managing. The manager's role is assigned to only one person. The manager decides on further action in difficult situations, such as the aforementioned.

Having a record, or "status table," of which questions have been read, solved, and/or implemented can be very handy. The manager can see the status of the team by glancing at this table. It is the manager's job to maintain this table.

Another important role of the manager is debugging. The manager must ask the coder about each part of the code in order to understand the code completely. Meanwhile, the manager can make some test cases for the problem. The more you test your solution, the less you hear, "No. Wrong answer." from the judge.

The manager helps the solver as well. Two minds are better than one in solving a problem. Thus, each part of the job is done by two people: two people read the question, solve it, and implement it. This redundancy decreases the probability of bugs in the code.

Unlike the manager, the coder and the solver can exchange their tasks to relieve the coder from stress. The manager must readily make clear decisions in critical situations. Also, the manager must be aware of the team member's morale conditions. Sometimes, the pressure of the contest can have a large impact on a person, and it can decrease that team member's efficiency. A good manager can be very helpful in these situations by thoughtfully encouraging that teammate.

We used this method in the ACM/ICPC World Finals in 2005. We used the duty-based method in the prior year, 2004. It helped us to improve our previous standing from 27th to 17th place. The key point is that we had fewer bugs in our code, and we therefore had more time for working on new problems.

The main disadvantage of this method is that it requires a lot of practice. You must work several hours with one another in order to learn how to communicate and collaborate with each other correctly. So, if you are in a time shortage, this is not a suitable method for you. However, if you have plenty of time, then it is worth trying!

Important Points

A motivated team can do miracles, while a stressed team can ruin months of practicing! Do not forget that the contest is 300 minutes long, and anything can happen at the last moment. So, be hopeful, and do your best.

In the real contest, most of the teams have the same level of intelligence. What separates them is morale. The difficulty of the problems is not the main factor. All of the teams can solve all of the problems given enough time. However, when you are under the enormous pressure of the competition, it is not easy to solve them.

As a result, always be aware of the spirit of the team. If you fall behind, do not get disappointed. If you are ahead, do not become proud. The ranking list changes frequently during the last hour of the contest, so keep trying until the last second!

Do not blame one another for wrong answers. If you received a wrong answer from the judge, all members of the team are equally responsible. If your programmer is implementing the wrong code, this is your fault as well. You did not understand or communicate the problem properly. Keep in mind: you are a team before the contest, during the contest, and after the contest!

Acknowledgments

I would like to thank my team members during past years, particularly Arash Rahimi, Haamed Gheibi, Mohammad Tavakkoli, and Babak Behsaz. I would like to give my special thanks to Martin Margo for his careful reading of this article and his helpful comments. Also, I would like to thank "seKallePuk," the team that I will never forget!

References

1
ACM, http://www.acm.org/.

2
Amirkabir University of Technology, Tehran, Iran. http://www.aut.ac.ir/.

3
T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, The MIT Press, Second Edition, 2001.

4
F. Ernst, J. Moelands, S. Pieterse, *Teamwork in Programming Contests: 3 * 1 = 4*, ACM Crossroads, Winter 1996. http://www.acm.org/crossroads/xrds3-2/progcon.html.

5
ICPC, http://icpc.baylor.edu/icpc.

6
Valladolid Online Judge Site, http://acm.uva.es/.

Biography

Saman Amirpour Amraii (samirpour@acm.org) is an MS student at the University of Tehran, Iran, in the field of artificial intelligence and robotics. He competed in the ICPC finals in 2004 (ranked 27th) and 2005 (ranked 17th). His homepage is available at **http://khorshid.ut.ac.ir/~s.amirpour**.