Competitive Programming
234900
Introduction
Agenda

• Workshop Goals
• Introduction to ACM-ICPC
• Overview
  • Class Work
  • Homework
  • Final Competition
• A couple of warm-up questions
Workshop Goals

- Hands-on approach to Algorithms
- Fast programming and team work
- Fun!

Introduction to competitive programming

Assemble a team for the ACM-ICPC regionals
Administration

• 2 Academic Points

• Prerequisites
  • Data Structures 1
  • Algorithms 1
  • (Or equivalent courses in other departments)

• ”Reshima Alef”
  • The course is not considered as a project or seminar
Administration (cont.)

• Mondays 16:30-18:30
• Meetings will usually be held in the computer farm
  • Some meetings will be held in other rooms. Pay close attention to the announcements in the course website.
Course Staff

• Prof. Gill Barequet – Lecturer in charge
• Eden Saig – TA in charge
• Nofar Carmeli
• Yufei Zheng
• Aviram Magen
Grading

• Final grade will be determined by:
  • Class work (Questions solved in class)
  • Homework
  • Final Competition

• During the course, 5 homework tasks will be given
• Attendance is mandatory
  • You can miss one lesson without approval.
  • Additional absences will result in a penalty which can be compensated by solving additional questions at home.
Final Competition

• Final competition will be held at the end of the semester (05/07/2017).
• Format will be similar to the ACM ICPC regionals.

• The best eligible team will be invited to represent the Technion at the ACM ICPC Southwestern Europe Regionals (SWERC)
  • SWERC 2017 will be held in Paris in November.
  • Travel expenses will be covered by the computer science department.
ACM ICPC

• **ACM-ICPC** is an annual multi-tiered competitive competition among the universities of the world.
ACM ICPC

- The competition is running since 1977 and considered as the most prestigious in the world of competitive programming.
ACM ICPC

• The ACM-ICPC Competition has three tiers:

Local Qualification
• Each university selects teams for the regionals.
• Over 300,000 students throughout the world.

Regionals
• 30 different regions. The Technion competes in the "southwestern europe" region (SWERC).

World Finals
• Top 100 teams (2-4 from each region)
ACM ICPC Regionals
Previous Competitions

• In 2017 the Technion is going to participate for the 12th time.
• Previous results:
  • 2006 Romania – Marcello Taub, Michael Gelfand, Kolman Vornovitsky – 30th place
  • 2007 Romania – Gill Cohen, Lior Biran, Yaron Yura – 36th place
  • 2008 Romania – Shachar Papini, Yaniv Sabo, Karmi Grushko – 17th place
  • 2009 Spain – Shachar Papini, Yaniv Sabo, Oshri Adler, Itai Levy – 7th place (Silver)
  • 2010 Spain - Shachar Papini, Yaniv Sabo, Karmi Grushko – 2nd place (Gold)
  • 2011 Spain – Omer Tabach, Tomer Fruman, Noa Korner – 16th place
  • 2012 Spain – Idan Elad, Olivier Hoffman, Mike Harris – 12th place (Bronze)
  • 2013 Spain – Alex Goltman, Oleg Zlotnik, Roei Gelbhart – 10th place (Bronze)
  • 2014 Portugal – Eden Saig, Nofar Carmeli, Ori Brusilovsky – 15th place
  • 2015 Portugal – Omer Daniel, Aviram Magen, Itay Zukier – 18th place
    Nitzan Tur, Ido Hakimi, Dima Kuznetsov – 38th place
  • 2016 Portugal – Omer Daniel, Aviram Magen, Itay Zukier – 15th place
ACM ICPC – Eligibility

Willing and able to participate?  
(Ineligible if answer is no)

Year first of postsecondary studies >= 2013?  
(Eligible if answer is yes)

Born in 1994* or later?  
(Eligible if answer is yes, *1991 if you were in the army)

More than 8 semesters of full-time study?  
(Ineligible if answer is yes)

Coach petitions for extension of eligibility.

http://icpc.baylor.edu/download/regionals/rules/EligibilityDecisionTree-2016.pdf
Competition format

- Teams of three
- One computer for each team
- 8-10 problems
- 5 hours
- Limited reference materials
- Automatic Judge
Competition Format (cont.)

• Programming languages: C, C++, Java
  • We will use C++ exclusively

• Code editor of your choice
  • Eclipse, Codeblocks, vim, emacs

• The focus is on **efficient, correct** solutions.
Judge System

• During the competition, solutions are submitted to the automatic judge system and checked against predefined test cases.
• Run time and memory use are usually very limited.
• The following answers are possible:
  • ACCEPTED
  • COMPILE-ERROR
  • TIME-LIMIT-EXCEEDED
  • RUNTIME-ERROR
  • WRONG-ANSWER
  • PRESENTATION-ERROR
ICPC Scoring and Ranking

• Teams are ranked by the **number of problems** solved.

• **Tie breaker**: Teams that solved the same amount of problems are ranked according to their *total time*.
  • Total time is the sum of problem acceptance times + 20 points of penalty for each unaccepted submission (only for problems that were eventually accepted)

• Example:
  • Total time=  
    +40 (Acceptance time of A)  
    +20 (Bad submission of A)  
    +70 (Acceptance time of B)  
    +20 (Bad submission of B)  
    = 150

<table>
<thead>
<tr>
<th>Time</th>
<th>Problem</th>
<th>Judge Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>20min</td>
<td>A</td>
<td>No</td>
</tr>
<tr>
<td>40min</td>
<td>A</td>
<td>Yes</td>
</tr>
<tr>
<td>65min</td>
<td>B</td>
<td>No</td>
</tr>
<tr>
<td>70min</td>
<td>B</td>
<td>Yes</td>
</tr>
<tr>
<td>95min</td>
<td>C</td>
<td>No</td>
</tr>
</tbody>
</table>
Schedule

Introduction (HW0)
- STL (HW1)
  - Graphs 1
    - Binary Search (HW2)
      - Computational Geometry (HW4)
        - Advanced Data Structures
          - Dynamic Programming (HW3)
            - Dynamic Programming 2
              - Simulations
                - Final Competition
Class Work

• Rooms 018+022
  • We will need your cooperation 😊

• Electronic submission
  • Open until 18:30
  • Pay close attention to the submission instructions

• Each meeting deals with a family of problems
  • 3-4 problems per lesson
Teams

• Starting as pairs
  • First Two Lessons
  • Meet new people!

• Team up after Passover
  • 24/04/2017
  • Teams of three
Class Work (cont.)

• Background information will be published
  • A week before each meeting
  • Prepare for the meetings to solve more questions!

• Late submission is possible
  • Singles
  • Bonus: 0.25 “Class Work” points
  • (Refer to the information sheet)
Online Judge

The International Collegiate Soccer Competition (ICSC) is famous for its well-kept rectangular stadiums. The grass playing fields in ICSC stadiums are always 100 meters long, and 75 meters wide. The grass is mowed every week with special lawn mowers, always using the same strategy: first, they make a series of passes along the length of the field, and then they do the same along the width of the field. All passes are straight lines, parallel to the sides of the field.

- Mowing along the length of the field
- Mowing along the width of the field
Homework

• 4 homework tasks (+HW0)
  • 3-4 questions
  • Each solution should be accepted by the online judge
  • Explain your solution in 400 words (or much less)
    • English only
  • HW1, HW2 – Singles
  • HW3, HW4 – Teams

• HW0
  • Create online judge account
  • Submit a solution for an easy problem
  • Will prepare you for next lesson
Example: Lawn Mower 4954

- The grass field in a the stadium is 100m long, 75m wide.
- The grass is mowed every week with special lawn mowers, always using the same strategy:
  - First, they make a series of passes along the length of the field.
  - Then they do the same along the width of the field.
  - All passes are straight lines, parallel to the sides of the field.
• The new gardner likes to choose random starting positions for each of his passes.
• He is afraid of not doing a good job and being fired, so he has asked you to help him.
Lawn Mower 4954 (cont.)

- Write a program to make sure that the grass in the field is perfectly cut:
  - All parts of the field have to be mowed at least once when the mower goes from end to end, and again when the mower goes from side to side.
  - The grass field in the stadium is 100m long, 75m wide.
Lawn Mower 4954 – Input

• Input:
  • Each test case contains 3 lines. The first line contains two integers: \( n_x \) \((0 < n_x < 1000)\) and \( n_y \) \((0 < n_y < 1000)\), and a real number \( w \) \((0 < w \leq 50)\), which represents the width of that particular lawn mower.
  • Next line contains \( n_x \) real numbers \( x_i \) \((0 \leq x_i \leq 75)\), describing the starting positions of the mower’s center in the top-to-bottom passes.
  • Last line describes the side-to-side passes, with \( n_y \) real numbers \( y_i \) \((0 \leq y_i \leq 100)\).
  • The end of the test cases is signalled with a line that contains the numbers ‘0 0 0.0’.

• Output:
  • ”YES” if all the grass was covered by the gardener (at least once in each direction) and “NO” otherwise.
Lawn Mower 4954 – Sample Input

• 8 11 10.0
  0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0
  0.0 10.0 20.0 30.0 40.0 50.0 60.0 70.0
  80.0 90.0 100.0
  (Output: YES)

• 4 5 20.0
  60.0 10.0 30.0 50.0
  30.0 10.0 90.0 50.0 70.0
  (Output: NO)
Lawn Mower 4954

• Any ideas?
Lawn Mower 4954 – Solution

• Solution
  • Check $x$ and $y$ axis separately.
  • Axis is covered iff the distance between all adjacent starting points is less than or equal to $w$.
  • The distance from the first/last point to the edge should be less or equal to $w/2$.
  • **Sort** the starting points, and go over them once.

• Time complexity: $O\left(n_x \log n_x + n_y \log n_y \right)$
  • $0 < n_x, n_y < 1000 \Rightarrow$ Running time is good enough! 😊
```cpp
#include <iostream>
#include <algorithm>

#define MAX 1001

using namespace std;

int main()
{
    int nx, ny, i;
    double w;
    double x[MAX], y[MAX];
    while (cin >> nx)
    {
        cin >> ny >> w;
        if ((nx == 0) && (y == 0) && (w == 0)) break;
        for (i = 0; i < nx; ++i)
            cin >> x[i];
        for (i = 0; i < ny; ++i)
            cin >> y[i];
        sort(x, x+nx);
        sort(y, y+ny);
        if (((x[0] > w/2) || (x[nx-1] < 75-w/2)) ||
            (y[0] > w/2) || (y[ny-1] < 100-w/2))
            continue;
        for (i = 0; i < nx-1; ++i)
            if (x[i+1] - x[i] > w)
                cout << "NO" << endl;
        if (i != nx-1) continue;
        for (i = 0; i < ny-1; ++i)
            if (y[i+1] - y[i] > w)
                cout << "NO" << endl;
        if (i != ny-1) continue;
        cout << "YES" << endl;
    }
    return 0;
}```
Conclusions from the Code

• Input/Output should be as simple as possible
• No need to check for input errors
• Pay attention to edge cases!
• Feel free to copy and paste
• Simple variable names

• Sometimes you do have to refactor your code and solve bugs. Look for the sweet spot between quick writing and code clarity.
Complexity in the Competition

• In the problem statements we are given input bounds, and from them we can deduce if our expected running time is good enough.
• Usually the code has only 2-3 seconds to run in each submission.
• **Rule of thumb**: \(\sim 10M\) operations is the bound above which a solution is not feasible.
  • For example, time complexity of \(O(N^2)\) is feasible for \(N = 3000\), but not for \(N = 10000\).
  • This rule is good in most cases, but not in all of them. Sometimes \(N^2\) gets accepted and \(10N^2\) doesn’t.
Additional Materials

• In ACM-ICPC, contestants are only allowed to use C++ and the STL data structures (or Java and the standard library)
  • In the next lesson we will study about STL and useful data structures.

• The official documentation of C++ and STL is given in the competition.

• In addition, you are allowed to bring up to 25 pages of printed materials.
  • A good reference sheet is very useful in the competition!
6823 - Counting Substhereengs

• Substrings are strings formed by choosing a subset of contiguous characters from a string (This is well known)

• A Substhereeng is a substring which complies to the following additional requirements:
  • It is non-empty, and composed entirely of base 10 digits.
  • Interpreted in base 10 (allowing extra leading zeros), the resulting integer is a multiple of 3.
Example

• The string ‘130a303’ contains 9 substrings:
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
  • 130a303
Input/Output

- The input contains several test cases.
- A test case consists of a single line that contains a non-empty string $S$ of at most $10^6$ characters.
- Each character of $S$ is either a digit or a lowercase letter.
- For each test case output a line with an integer representing the number of substrings contained in $S$. 
Input/Output

130a303           9
00000000000        55
icpc2014regional   2
6823 - Counting Substhereengs

Any Ideas?
How Can We Solve It?

• We denote: $n = 10^n d_n + 10^{n-1} d_{n-1} + \cdots + d_0$

• For numbers divisible by three:

$$(n \mod 3) = 0 \iff \left( \sum_{i=0}^{n} d_i \mod 3 \right) = 0$$
<table>
<thead>
<tr>
<th>Original Number</th>
<th>2 0 1 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Sum</td>
<td>2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>2 2 0 1 2</td>
</tr>
<tr>
<td></td>
<td>2 0 1 4 4</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Original Number</td>
<td>2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum</td>
<td>2 2 0 1 2</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>2 2 0 1 2</td>
</tr>
</tbody>
</table>
20144

<table>
<thead>
<tr>
<th>Original Number</th>
<th>2 0 1 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Sum</td>
<td>2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>2 2 0 1 2</td>
</tr>
<tr>
<td>Original Number</td>
<td>2 0 1 4 4</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Digit Sum</td>
<td>2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>2 2 0 1 2</td>
</tr>
</tbody>
</table>
20144

<table>
<thead>
<tr>
<th>Original Number</th>
<th>2 0 1 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Sum</td>
<td>2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>2 2 0 1 2</td>
</tr>
</tbody>
</table>

\[
\binom{3}{2} = 3 \\
\underbrace{\text{sum} = 2}_{\text{sum}=2}
\]
20144

<table>
<thead>
<tr>
<th>Original Number</th>
<th>2 0 1 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Sum</td>
<td>0 2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>0 2 2 0 1 2</td>
</tr>
</tbody>
</table>

\[
\binom{3}{2} = 3 \quad \frac{(2)}{2} = 1 \quad \frac{(1)}{2} = 0
\]

\[
\frac{sum=2}{\text{sum}} \quad \frac{sum=0}{\text{sum}} \quad \frac{sum=1}{\text{sum}}
\]

3 + 1 + 0 = 4
<table>
<thead>
<tr>
<th>Original Number</th>
<th>2 0 1 4 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Sum</td>
<td>0 2 2 3 7 11</td>
</tr>
<tr>
<td>Digit Sum mod 3</td>
<td>0 2 2 0 1 2</td>
</tr>
</tbody>
</table>

\[
\binom{3}{2} = 3 \quad \text{sum=2} \quad \binom{2}{2} = 1 \quad \text{sum=0} \quad \binom{1}{2} = 0 \quad \text{sum=1}
\]

\[
3 + 1 + 0 = 4
\]
Solution

• \( total = 0 \)

• For each number in the string:
  • Calculate digit sum mod 3
  • Number of subthreengs: \( N = \binom{\#2}{2} + \binom{\#1}{2} + \binom{\#0 + 1}{2} \)
  • Add \( N \) to the total

• Linear complexity – \( O(10^6) \)
int main() {
    string S;
    while(cin>>S) {
        S += "\$";
        long long total_substhreengs=0;
        vector<long long> hist(3,0);
        hist[0]=1;
        int digit_sum=0;

        for(unsigned int i=0; i<S.size(); ++i) {
            int num = char(S[i])-\'0\';
            if ((num>=0) && (num<=9)) {
                digit_sum += num;
                digit_sum %= 3;
                hist[digit_sum]++;
            } else {
                for(int k=0; k<3; k++) {
                    total_substhreengs += hist[k]*(hist[k]-1)/2;
                }
            }
        }

        cout << total_substhreengs << endl;
    }
    return 0;
}
Players start with a score of N points (typically, \( N = 501 \)) and take turns throwing darts.

- The score of each player decreases by the value of the section hit by the dart, unless the score becomes negative, in which case it remains unchanged.
- The first player to reach a score of 0 wins.
4507 – Darts

• Player A throws the darts at random, and consequently they land with equal probability in each of the sections of the dartboard.

• If Player B aims at a certain section, the dart has the same probability of landing in the correct one as in each of the two adjacent ones (the neighbouring regions to the left and right).
  • Moreover, he is completely aware of his ability and sober enough to aim at the section that maximizes his probability of winning
4507 – Darts

• Given the initial score of both players, can you determine the probability that the first player wins?
• Of course, being the first to throw a dart might be advantageous, so the answer depends on who plays first.
4507 – Darts

• $d_i = 20, 1, 18, 4, \ldots$ Score of each sector
• $p_A(n, m) =$ Probability of A win if it’s A’s turn.
  • $n =$ Score of A
  • $m =$ Score of B
• $p_B(n, m) =$ Probability of B win if it’s B’s turn.

• **Problem**: compute $p_A (N, N)$ and $p_B (N, N)$.
• **Solution**: Dynamic Programming
Recurrence

• $p_A(0, m) = 1$
• $p_B(n, 0) = 1$

• $p_A(n, m) = \frac{1}{20} \sum_{i=1}^{20} (1 - p_B(n - d_i, m))$
• $p_B(n, m) = \max_{1 \leq i \leq 20} \frac{1}{3} \sum_{j=-1}^{1} (1 - p_A(n, m - d_{i+j}))$

• Only valid if $n \geq 20$ and $m \geq 20$!
Solving for $n, m \leq 20$

• When $n, m < d_i$ we get a cyclic dependency:

\[
p_A(n, m) = \frac{1}{20} \sum_{i=1}^{20} (1 - p_B(n - d_i, m)) = \cdots = A + B \cdot p_B(n, m)
\]

\[
p_B(n, m) = \cdots = C + D \cdot p_A(n, m)
\]
Solving for $n, m \leq 20$

- $A, B, C, D$ depend on previous values.
- 2 equations in 2 variables:

\[
\begin{align*}
    p_A(n, m) - B \cdot p_B(n, m) &= A \\
    -D \cdot p_A(n, m) + p_B(n, m) &= C
\end{align*}
\]

\[
\begin{pmatrix}
    p_A^{nm} \\
    p_B^{nm}
\end{pmatrix}
= \begin{pmatrix}
    1 & -B \\
    -D & 1
\end{pmatrix}^{-1}
\begin{pmatrix}
    A \\
    C
\end{pmatrix}
\]

- B maximizes winning probability:
  - Check different numbers and solve.
  - Take best solution.
Notes

• **Time Complexity**: \( O(N^2) \approx 10^6 \)\( \_N \leq 501 \)

• We can prepare one 501\( \times \)501 table and use it for all test cases.

• Don’t forget std::setprecision
Conclusion

• Competitive Programming is fun

• HW0 is out – Start early

• See you next week in the computer farm! 😊