ACM International Collegiate Programming Contest — Training Session II

C. Maria Keet

Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za

August 16, 2014

Approach

- Read description
- What is the task?
- What is given?
  - data, variables, constraints, examples
- Solve the problem
  - ‘essentially solve it’
    - input data space
    - recognise underlying core issues
    - similarity with other problems
    - result: a solution on paper, knowing **what** needs to be done
  - solve it algorithmically (**how** to do it—e.g., a sort, OSPF, complexity, ...)
  - code and test it, i.e., **do** it and verify solution

Types of puzzles

- Regarding the core problem
  A. maths-y (e.g., probabilities, geometry)
  B. algorithmically/general (still an elegant solution)
  C. seeing patterns, and brute force
- For the algorithms: which class of algorithm would be needed?
  i. e.g.: simple sorting, searching, graphs, numerical, combinatorial, sets, strings, geometry?
  ii. within the class, which type? e.g., geometry: convex hulls, range search, polygons, shape similarity, ...
- The (im-)balance in the 'what, how, do':
  a. conceptually hard, but (relatively) easier to implement
  b. conceptually (relatively) easy, but laborious to design and/or implement
  c. both relatively hard (happens at the finals)
  d. both relatively easy (at least one puzzle in the regionals)

Types of puzzles

- Regarding the core problem
  A. maths-y (e.g., probabilities, geometry)
  B. algorithmically/general (still an elegant solution)
  C. seeing patterns, and brute force
- For the algorithms: which class of algorithm would be needed?
  i. e.g.: simple sorting, searching, graphs, numerical, combinatorial, sets, strings, geometry?
  ii. within the class, which type? e.g., geometry: convex hulls, range search, polygons, shape similarity, ...
- The (im-)balance in the 'what, how, do':
  a. conceptually hard, but (relatively) easier to implement
  b. conceptually (relatively) easy, but laborious to design and/or implement
  c. both relatively hard (happens at the finals)
  d. both relatively easy (at least one puzzle in the regionals)

Types of puzzles

- Regarding the core problem
  - A. maths-y (e.g., probabilities, geometry)
  - B. algorithmically/general (still an elegant solution)
  - C. seeing patterns, and brute force
- For the algorithms: which class of algorithm would be needed?
  - i. e.g.: simple sorting, searching, graphs, numerical, combinatorial, sets, strings, geometry?
  - ii. within the class, which type? e.g., geometry: convex hulls, range search, polygons, shape similarity, ...
- The (im-)balance in the 'what, how, do':
  - a. conceptually hard, but (relatively) easier to implement
  - b. conceptually (relatively) easy, but laborious to design and/or implement
  - c. both relatively hard (happens at the finals)
  - d. both relatively easy (at least one puzzle in the regionals)

Tips for competitive programming

- General tips:
  - Type Code Faster
  - Quickly Identify Problem Types
  - Do Algorithm Analysis
  - Master Programming Languages
  - Master the Art of Testing Code
  - Practice and More Practice
- Know your problem solving paradigms in CS: complete search, divide and conquer, greedy, dynamic programming

Competitive programming 1, by Steven and Felix Halim: `https://sites.google.com/site/stevenhalim/`

Today

- 3 ICPC problems, one ‘bonus’
- Problems again selected for differences in approaches, emphases what/how/do, level of difficulty
- If you don’t finish a problem, try at home and make sure you’ve implemented it
- Some sources you may want to have a look at:
  - Steven Skiena’s “Algorithm Design Manual”
  - Paul Zeitz’s “The art and craft of problem solving”
  - ACM-ICPC Live Archive with hundreds of problems `https://icpcarchive.ecs.baylor.edu/`

Outline

Student IDs

- 2013 regionals problem (see printout), many teams solved it
- type: algorithmically, some coding, relatively easy (in the grand scheme of things)
- As exercise: use aforementioned methodological steps
- Solve at least the ‘what’ and ‘how’-parts in 30 minutes

⇒ automata are helpful, design important to make sure you don’t miss anything.

Student IDs

- 2013 regionals problem (see printout), many teams solved it
- type: algorithmically, some coding, relatively easy (in the grand scheme of things)
- As exercise: use aforementioned methodological steps
- Solve at least the ‘what’ and ‘how’-parts in 30 minutes

$\Rightarrow$ automata are helpful. design important to make sure you don’t miss anything.

Outline

Designing Railway routes

- 2013 regionals problem (see printout), few solved it
- type: algorithmically, somewhat challenging
- Solve it

⇒ My direction of solution is one of 'modify existing algorithm'

Designing Railway routes

- 2013 regionals problem (see printout), few solved it
- type: algorithmically, somewhat challenging
- Solve it

$\Rightarrow$ My direction of solution is one of ‘modify existing algorithm’

Designing Railway routes: toward solution

- Task: second-best solution
- Has lots of constraints on the input data
- Check first example, and draw and check the second example (in the input/output files, given)
- Do you have to start at node 1? no.
- What algorithms do we have to compute trees and shortest paths? Can we use that here?
- OSPF/Dijkstra's algorithm, spanning tree algorithms
- But can we get the 2nd-best from that?

Designing Railway routes: toward solution

- Task: second-best solution
- Has lots of constraints on the input data
- Check first example, and draw and check the second example (in the input/output files, given)
- Do you have to start at node 1? no.
- What algorithms do we have to compute trees and shortest paths? Can we use that here?
- OSPF/Dijkstra's algorithm, spanning tree algorithms
- But can we get the 2nd-best from that?

Designing Railway routes: toward solution

- Task: second-best solution
- Has lots of constraints on the input data
- Check first example, and draw and check the second example (in the input/output files, given)
- Do you have to start at node 1? no.
- What algorithms do we have to compute trees and shortest paths? Can we use that here?
- OSPF/Dijkstra's algorithm, spanning tree algorithms
- But can we get the 2nd-best from that?

Designing Railway routes: toward solution

- Task: second-best solution
- Has lots of constraints on the input data
- Check first example, and draw and check the second example (in the input/output files, given)
- Do you have to start at node 1? no.
- What algorithms do we have to compute trees and shortest paths? Can we use that here?
- OSPF/Dijkstra's algorithm, spanning tree algorithms
- But can we get the 2nd-best from that?

Designing Railway routes: toward solution

- Task: second-best solution
- Has lots of constraints on the input data
- Check first example, and draw and check the second example (in the input/output files, given)
- Do you have to start at node 1? no.
- What algorithms do we have to compute trees and shortest paths? Can we use that here?
- OSPF/Dijkstra's algorithm, spanning tree algorithms
- But can we get the 2nd-best from that?
    - Compute OSPF for each node, take the 2nd-lowest value, or mutate lowest value?
    - Modify a greedy algorithm for 'second-best': Kruskal? Prim?
    - Which one works best? or neither, and use another solution?

Outline

Baggage problem: sorting luggage

- A 2014 finals problem (see printout), no-one solved it during the finals
- type: algorithmically;
- difficulty: judges thought it would be solved first; imho: it's a tough nut to crack
- First exercise: understand the problem
  - map the problem space (what is asked for, examples [can you find new constraints/regularities?], input space, output)

Baggage: toward a solution

- How to tackle the problem? I've heard:
  - 'mathsy, with some neat proofs'
  - brute force
  - pattern finding
  - It appeared to be a variation on "Taits counter puzzle"
- One can prove that you need *at least n* moves, but that doesn't solve the problem.

Baggage: toward a solution

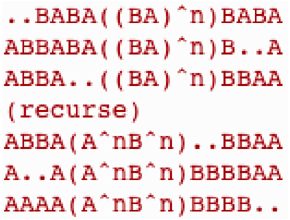
- How to tackle the problem? I've heard:
  - 'mathsy, with some neat proofs'
  - brute force
  - pattern finding
  - It appeared to be a variation on "Taits counter puzzle"
- One can prove that you need *at least n* moves, but that doesn't solve the problem.

Baggage: toward a solution

- How to tackle the problem? I've heard:
  - 'mathsy, with some neat proofs'
  - brute force
  - pattern finding
  - It appeared to be a variation on "Taits counter puzzle"
- One can prove that you need *at least n* moves, but that doesn't solve the problem.

Baggage: brute force

- Brute force approach (thanks to SnapDragon on TopCoder):
    - $3 \leq n \leq 7$ is a bit of a pain (pre-compute by hand)
    - $n + 4$ and larger can be done using recursion as follows:



```
..BABA((BA)^n)BABA
ABBABA((BA)^n)B..A
ABBA..((BA)^n)BBAA
(recurse)
ABBA(A^nB^n)..BBAA
A..A(A^nB^n)BBBBAA
AAAA(A^nB^n)BBBB..
```

Baggage: patterns

- Finding a pattern (thanks to CMK)
- More detail at `https://keet.wordpress.com/2014/06/28/acm-icpc-2014-solution-to-problem-a-baggage/`
- Approach of the algorithm:
  1. move around the As and Bs to pair them as AA and BB. alternate moving AB from the right to the left, starting at the last AB (position 2n-2) and then every other 4 to its left, and BA from left to right, starting from position 3 and every other 4 positions to the right (i.e., 7, 11, etc.)
  2. sort those pairs in the remainder to a total of n moves. The BBs are ferried to the right from left to right, and the AAs from the right to the left, also alternating

Baggage: patterns

- Finding a pattern (thanks to CMK)
- More detail at `https://keet.wordpress.com/2014/06/28/acm-icpc-2014-solution-to-problem-a-baggage/`
- Approach of the algorithm:
  1. move around the As and Bs to pair them as AA and BB.
     alternate moving AB from the right to the left, starting at the last AB (position 2n-2) and then every other 4 to its left, and BA from left to right, starting from position 3 and every other 4 positions to the right (i.e., 7, 11, etc.)
  2. sort those pairs in the remainder to a total of n moves.
     The BBs are ferried to the right from left to right, and the AAs from the right to the left, also alternating

Baggage: patterns

- Finding a pattern (thanks to CMK)
- More detail at `https://keet.wordpress.com/2014/06/28/acm-icpc-2014-solution-to-problem-a-baggage/`
- Approach of the algorithm:
  1. move around the As and Bs to pair them as AA and BB. alternate moving AB from the right to the left, starting at the last AB (position 2n-2) and then every other 4 to its left, and BA from left to right, starting from position 3 and every other 4 positions to the right (i.e., 7, 11, etc.)
  2. sort those pairs in the remainder to a total of n moves. The BBs are ferried to the right from left to right, and the AAs from the right to the left, also alternating

Baggage: patterns

- Finding a pattern (thanks to CMK)
- More detail at `https://keet.wordpress.com/2014/06/28/acm-icpc-2014-solution-to-problem-a-baggage/`
- Approach of the algorithm:
  1. move around the As and Bs to pair them as AA and BB. alternate moving AB from the right to the left, starting at the last AB (position 2n-2) and then every other 4 to its left, and BA from left to right, starting from position 3 and every other 4 positions to the right (i.e., 7, 11, etc.)
  2. sort those pairs in the remainder to a total of n moves. The BBs are ferried to the right from left to right, and the AAs from the right to the left, also alternating

Outline

Young and Successful

- Not an ACM ICPC problem from recent years (or at all)
- Description on printout
- Identify type
- Try to solve it
- Solve it algorithmically

$\Rightarrow$ Basically a version of a *range search*:
input: A set S with n points in d-dimensional space E, and a query asking for the points in region Q

Young and Successful

- Not an ACM ICPC problem from recent years (or at all)
- Description on printout
- Identify type
- Try to solve it
- Solve it algorithmically

$\Rightarrow$ Basically a version of a *range search*:
input: A set S with n points in d-dimensional space E, and a query asking for the points in region Q

Young and Successful

- Not an ACM ICPC problem from recent years (or at all)
- Description on printout
- Identify type
- Try to solve it
- Solve it algorithmically

$\Rightarrow$ Basically a version of a *range search*:
input: A set S with n points in d-dimensional space E, and a query asking for the points in region Q

Scheduled training dates

- Aug 2: 9:30-15:30
- Aug 16: 9:30-15:30
- **Aug 30: 9:30-15:30**
- Sept 13: 9:30-15:30
- Sept 27: 9:30-15:30
- Date of the regionals: 4 October, 2014