ACM International Collegiate Programming Contest — Training Session I

C. Maria Keet

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

August 2, 2014

Outline

Outline

Approach (1/2)

- Read description
- What is the task?
- What is given?
  - data
  - variables
  - constraint
  - examples
- (cont'd on next page...)

Approach (1/2)

- Read description
- What is the task?
- What is given?
  - data
  - variables
  - constraint
  - examples
- (cont'd on next page...)

Approach (1/2)

- Read description
- What is the task?
- What is given?
  - data
  - variables
  - constraint
  - examples
- (cont'd on next page...)

Approach (2/2)

- 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

Approach (2/2)

- 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

Approach (2/2)

- 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

Approach (2/2)

- 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

Approach (2/2)

- 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

- Can be divided along several dimensions
- Regarding the core problem
  A. maths-y (e.g., probabilities, geometry)
  B. algorithmically/general (still an elegant solution)
  C. seeing patterns, and brute force
- The (im-)balance in the 'what, how, do':
  a. conceptually hard, but (relatively) easier to implement
  b. conceptually (relatively) easy, but laborious to implement
  c. both relatively hard (happens at the finals)
  d. both relatively easy (at least one puzzle in the regionals)

Types of puzzles

- Can be divided along several dimensions
- Regarding the core problem
  A. maths-y (e.g., probabilities, geometry)
  B. algorithmically/general (still an elegant solution)
  C. seeing patterns, and brute force
- The (im-)balance in the 'what, how, do':
  a. conceptually hard, but (relatively) easier to implement
  b. conceptually (relatively) easy, but laborious to implement
  c. both relatively hard (happens at the finals)
  d. both relatively easy (at least one puzzle in the regionals)

Types of puzzles

- Can be divided along several dimensions
- Regarding the core problem
  A. maths-y (e.g., probabilities, geometry)
  B. algorithmically/general (still an elegant solution)
  C. seeing patterns, and brute force
- The (im-)balance in the 'what, how, do':
  a. conceptually hard, but (relatively) easier to implement
  b. conceptually (relatively) easy, but laborious to implement
  c. both relatively hard (happens at the finals)
  d. both relatively easy (at least one puzzle in the regionals)

Team composition (1/2)

- **Your team needs strengths in all three areas (what, how, do)** to have a chance to win
- Your team might be able to get away with maths-y only solving, but more probably with algorithmically, and even better both
- Each member probably won't excel in all three components, but *together* you will
- What are *you* good at?
- What are *your team mates* good at?
- What is your team *together* as a whole good at?

Team composition (1/2)

- **Your team needs strengths in all three areas (what, how, do)** to have a chance to win
- Your team might be able to get away with maths-y only solving, but more probably with algorithmically, and even better both
- Each member probably won't excel in all three components, but *together* you will
- What are *you* good at?
- What are *your team mates* good at?
- What is your team *together* as a whole good at?

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - Bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution
  - the third person looks at the problem 'from the side'

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution; the third person looks at the problem 'from the side'

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution
  - the third person looks at the problem 'untarnished'

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution
  - the third person looks at the problem 'untarnished'

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution
  - the third person looks at the problem 'untarnished'

Team composition (2/2)

- Who will do what?
- Dividing tasks aided by a *team manager*, who does (at least):
  - Keeps track of who is doing what
  - Makes sure problems are not forgotten
  - Keeps track of time
- pair solving option:
  - bouncing off ideas toward solution
  - possibly putting different solved pieces together
- pair programming; One person codes, the other ('co-pilot') watches:
  - Understands the solution
  - Points out bugs
  - Looks things up for pilot
- 'fresh eyes'
  - Sometimes you get stuck in a dead-end looking for a solution
  - the third person looks at the problem 'untarnished'

and finally

- Solve 'the easiest' problem first, i.e.,
  - The one easiest to your team
  - Looking at what others submit first may, or may not, be a good heuristic to choose what's easiest
- Don't submit without doing some testing (recall: time penalty for incorrect submissions!)
- Don't get hung up on one problem for too long, but don't keep jumping between problems without solving any of them either
- There is no harm in last-minute attempts (recall: the number of problems solved is more important than time taken)

and finally

- Solve 'the easiest' problem first, i.e.,
  - The one easiest to your team
  - Looking at what others submit first may, or may not, be a good heuristic to choose what's easiest
- Don't submit without doing some testing (recall: time penalty for incorrect submissions!)
- Don't get hung up on one problem for too long, but don't keep jumping between problems without solving any of them either
- There is no harm in last-minute attempts (recall: the number of problems solved is more important than time taken)

and finally

- Solve 'the easiest' problem first, i.e.,
  - The one easiest to your team
  - Looking at what others submit first may, or may not, be a good heuristic to choose what's easiest
- Don't submit without doing some testing (recall: time penalty for incorrect submissions!)
- Don't get hung up on one problem for too long, but don't keep jumping between problems without solving any of them either
- There is no harm in last-minute attempts (recall: the number of problems solved is more important than time taken)

and finally

- Solve 'the easiest' problem first, i.e.,
  - The one easiest to your team
  - Looking at what others submit first may, or may not, be a good heuristic to choose what's easiest
- Don't submit without doing some testing (recall: time penalty for incorrect submissions!)
- Don't get hung up on one problem for too long, but don't keep jumping between problems without solving any of them either
- There is no harm in last-minute attempts (recall: the number of problems solved is more important than time taken)

Outline

Fitness problem

- 2013 regionals problem (see printout)
- type: basic maths, easy (in the grand scheme of things)
- As exercise: use aforementioned methodological steps
- Solve it—at least the 'what'-part—in 15 minutes

$\Rightarrow$ know your basic maths

Fitness problem

- 2013 regionals problem (see printout)
- type: basic maths, easy (in the grand scheme of things)
- As exercise: use aforementioned methodological steps
- Solve it—at least the 'what'-part—in 15 minutes

$\Rightarrow$ know your basic maths

Similarity problem

- also a 2013 regionals problem (see printout), hardly anyone solved it
- type: algorithmically, difficulty depends on prior knowledge
- First exercise: understand the problem
  - map the problem space (what is asked for, examples, input space, output, ...)
  - what is needed?

Solution (direction)

- Distance `word_A` to `word_B`
- Changes: insert (cost 2), delete (cost 2), replace (just case cost 1, else 2)
- Compare 'words' (strings), letter by letter (but just that would have misalignment issues)
- My association: 1) DNA and RNA alignments, 2) natural languages and computational linguistics
- This was solved a while ago, so there must be an existing algorithm

Solution (direction)

- Distance `word_A` to `word_B`
- Changes: insert (cost 2), delete (cost 2), replace (just case cost 1, else 2)
- Compare 'words' (strings), letter by letter (but just that would have misalignment issues)
- My association: 1) DNA and RNA alignments, 2) natural languages and computational linguistics
- This was solved a while ago, so there must be an existing algorithm

Solution (direction)

- Indeed: Levenshtein distance!
  - How exactly does it do that?
  - Modify the algorithm for the costs that deviate from the standard algorithm
  - Implement
  - Test
- Solved by recognising the problem to be basically the same one that was already solved

$\Rightarrow$ know and understand your algorithms

Solution (direction)

- Indeed: Levenshtein distance!
  - How exactly does it do that?
  - Modify the algorithm for the costs that deviate from the standard algorithm
  - Implement
  - Test
- Solved by recognising the problem to be basically the same one that was already solved

$\Rightarrow$ know and understand your algorithms

Levenshtein distance[1]

- Measure of the similarity between two strings, being the source string (s) and the target string (t)
- Distance is the number of deletions, insertions, or substitutions required to transform s into t
- Named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965, used in, a.o.:
  - Spell checking
  - DNA analysis
  - Plagiarism detection

[1] based on info from http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm

Main steps (1/2)

1. Step 1
   - Set n to be the length of s
   - Set m to be the length of t
   - If $n = 0$, return m and exit
   - If $m = 0$, return n and exit
   - Construct a matrix containing $0..m$ rows and $0..n$ columns
2. Step 2
   - Initialize the first row to $0..n$
   - Initialize the first column to $0..m$
3. Step 3
   - Examine each character of s (i from 1 to n)
4. Step 4
   - Examine each character of t (j from 1 to m)

Main steps (1/2)

1. Step 1
   - Set n to be the length of s
   - Set m to be the length of t
   - If $n = 0$, return m and exit
   - If $m = 0$, return n and exit
   - Construct a matrix containing $0..m$ rows and $0..n$ columns
2. Step 2
   - Initialize the first row to $0..n$
   - Initialize the first column to $0..m$
3. Step 3
   - Examine each character of s (i from 1 to n)
4. Step 4
   - Examine each character of t (j from 1 to m)

Main steps (1/2)

1. Step 1
   - Set n to be the length of s
   - Set m to be the length of t
   - If $n = 0$, return m and exit
   - If $m = 0$, return n and exit
   - Construct a matrix containing $0..m$ rows and $0..n$ columns
2. Step 2
   - Initialize the first row to $0..n$
   - Initialize the first column to $0..m$
3. Step 3
   - Examine each character of s (i from 1 to n)
4. Step 4
   - Examine each character of t (j from 1 to m)

Main steps (1/2)

1. Step 1
   - Set n to be the length of s
   - Set m to be the length of t
   - If $n = 0$, return m and exit
   - If $m = 0$, return n and exit
   - Construct a matrix containing $0..m$ rows and $0..n$ columns
2. Step 2
   - Initialize the first row to $0..n$
   - Initialize the first column to $0..m$
3. Step 3
   - Examine each character of s (i from 1 to n)
4. Step 4
   - Examine each character of t (j from 1 to m)

Main steps (2/2)

5. Step 5
   - If $s[i]$ equals $t[j]$, the cost is 0
   - If $s[i]$ doesn't equal $t[j]$, the cost is 1
6. Step 6
   - Set cell $d[i, j]$ of the matrix equal to the minimum of:
7. Step 7
   - After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell $d[n, m]$

Main steps (2/2)

5. Step 5
   - If $s[i]$ equals $t[j]$, the cost is 0
   - If $s[i]$ doesn't equal $t[j]$, the cost is 1
6. Step 6
   - Set cell $d[i, j]$ of the matrix equal to the minimum of:
     - The cell immediately above plus 1: $d[i-1, j] + 1$
     - The cell immediately to the left plus 1: $d[i, j-1] + 1$
     - The cell diagonally above and to the left plus the cost: $d[i-1, j-1] + \text{cost}$
7. Step 7
   - After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell $d[n, m]$

Main steps (2/2)

5. Step 5
   - If $s[i]$ equals $t[j]$, the cost is 0
   - If $s[i]$ doesn't equal $t[j]$, the cost is 1
6. Step 6
   - Set cell $d[i, j]$ of the matrix equal to the minimum of:
     - The cell immediately above plus 1: $d[i-1, j] + 1$
     - The cell immediately to the left plus 1: $d[i, j-1] + 1$
     - The cell diagonally above and to the left plus the cost: $d[i-1, j-1] + \text{cost}$
7. Step 7
   - After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell $d[n, m]$

Example—Step 1 and 2

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1					
t	2					
a	3					
f	4					
u	5					
l	6					
a	7					

Example—Steps 3-6 when $i = 1$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1				
a	3	2				
f	4	3				
u	5	4				
l	6	5				
a	7	6				

Example—Steps 3-6 when $i = 2$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2			
a	3	2	1			
f	4	3	2			
u	5	4	3			
l	6	5	4			
a	7	6	5			

Example—Steps 3-6 when $i = 3$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2	3		
a	3	2	1	2		
f	4	3	2	1		
u	5	4	3	2		
l	6	5	4	3		
a	7	6	5	4		

Example—Steps 3-6 when $i = 4$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2	3		
a	3	2	1	2	3	
f	4	3	2	1	2	3
u	5	4	3	2	2	
l	6	5	4	3	3	
a	7	6	5	4	4	

Example—Steps 3-6 when $i = 5$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2	3		
a	3	2	1	2	3	
f	4	3	2	1	2	3
u	5	4	3	2	2	3
l	6	5	4	3	3	2
a	7	6	5	4	4	3

Example—Steps 3-6 when $i = 6$

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2	3		
a	3	2	1	2	3	
f	4	3	2	1	2	3
u	5	4	3	2	2	3
l	6	5	4	3	3	2
a	7	6	5	4	4	3

Example—Steps 3-6 when $i = 7$ and final

		t	**a**	**f**	**e**	**l**
	0	1	2	3	4	5
i	1	1				
t	2	1	2	3		
a	3	2	1	2	3	
f	4	3	2	1	2	3
u	5	4	3	2	2	3
l	6	5	4	3	3	2
a	7	6	5	4	4	3

Finalise the similarity problem

- implement the algorithm
- modify algorithm for different costs, according to specs
- check any loose ends; e.g.:
  - what if either m or n is empty?
  - what if $n > m$ (i.e., first string is longer than the second), or vv?
  - does it work with numbers? (e.g.: `gr8` and `great`)

Wires

- 2014 finals problem (see printout), iirc, no-one solved it during the contest
- Type: not predominantly maths-y, not predominantly algorithmically, but seems to be a mix of both
- There isn't a readily available similar solution, though
- Use aforementioned methodological steps and solve first the 'what'-part, then the 'how', and go back to the what, if needed
- Conceptually non-trivial but not extremely hard; laborious to design and to implement

Wires: toward a solution

- Task: find the minimum amount of lines crossed
- What is the input like?
  - 2D plane with arbitrary sized and shaped closed and open line-delinated regions
- Gives two options to pursue:
  - Counting the lines crossed
  - Counting the regions traversed

Wires: toward a solution

- Task: find the minimum amount of lines crossed
- What is the input like?
  - 2D plane with arbitrary sized and shaped closed and open line-delinated regions
- Gives two options to pursue:
  - Counting the lines crossed
  - Counting the regions traversed

Wires: toward a solution

- Task: find the minimum amount of lines crossed
- What is the input like?
  - 2D plane with arbitrary sized and shaped closed and open line-delinated regions
- Gives two options to pursue:
  - Counting the lines crossed
  - Counting the regions traversed

Wires: toward a solution

- Task: find the minimum amount of lines crossed
- What is the input like?
  - 2D plane with arbitrary sized and shaped closed and open line-delinated regions
- Gives two options to pursue:
  - Counting the lines crossed
  - Counting the regions traversed

Wires: solution

- Traversing regions apparently more doable to implement
- Find a way to count the outer, unbounded, regions, e.g.:
  - Add an 'imaginary' line to close the region, and count crossing that line as 0
  - Make one new region comprising the outside option
- Creating and indexing the plane and crossings: a detailed 'how' at Bruce Merry's blogpost: `http://blog.brucemerry.org.za/2014/06/icpc-problem-m-wires.html`
- Compute the shortest path

Wires: solution

- Traversing regions apparently more doable to implement
- Find a way to count the outer, unbounded, regions, e.g.:
  - Add an 'imaginary' line to close the region, and count crossing that line as 0
  - Make one new region comprising the outside option
- Creating and indexing the plane and crossings: a detailed 'how' at Bruce Merry's blogpost: `http://blog.brucemerry.org.za/2014/06/icpc-problem-m-wires.html`
- Compute the shortest path

Wires: solution

- Traversing regions apparently more doable to implement
- Find a way to count the outer, unbounded, regions, e.g.:
  - Add an 'imaginary' line to close the region, and count crossing that line as 0
  - Make one new region comprising the outside option
- Creating and indexing the plane and crossings: a detailed 'how' at Bruce Merry's blogpost:
  `http://blog.brucemerry.org.za/2014/06/icpc-problem-m-wires.html`
- Compute the shortest path

Wires: solution

- Traversing regions apparently more doable to implement
- Find a way to count the outer, unbounded, regions, e.g.:
  - Add an 'imaginary' line to close the region, and count crossing that line as 0
  - Make one new region comprising the outside option
- Creating and indexing the plane and crossings: a detailed 'how' at Bruce Merry's blogpost:
  `http://blog.brucemerry.org.za/2014/06/icpc-problem-m-wires.html`
- Compute the shortest path

Panini World Cup Sticker book

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

$\Rightarrow$ Probabilities (the maths) or the Coupon Collectors Problem

Panini World Cup Sticker book

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

$\Rightarrow$ Probabilities (the maths) or the Coupon Collectors Problem

Panini World Cup Sticker book

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

$\Rightarrow$ Probabilities (the maths) or the Coupon Collectors Problem

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 20 or 27: 9:30-15:30
- Date of the regionals: TBD ("some Saturday between mid Sept and end Oct")