ACM International Collegiate Programming Contest — Training Session III

C. Maria Keet

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

August 30, 2014

Outline

Today

- Identifying a CS 'problem-solving paradigm' from the description
- Look at a whole set of problems of a regional contest:
  - categorise the problems
  - find strategy: which problem to tackle first, by whom, why?
- If you don't finish a problem, try at home and make sure you've implemented it, can submit to the ICPC and UVa sites anytime

Outline

Problem-solving paradigms in computing[1]

CS Complete search (brute force)

- solve problem searching the entire search space

D&C Divide & Conquer

- make problem 'simpler' by dividing into sub-problems (usually half the size)

Gr Greedy

- make locally optimal choice at each step

DP Dynamic Programming

- problem that has overlapping subproblems and optimal substructure

[1] content in this section based on "Competitive programming 1", by Steven and Felix Halim

Problem-solving paradigms in computing[1]

CS Complete search (brute force)

- solve problem searching the entire search space

D&C Divide & Conquer

- make problem 'simpler' by dividing into sub-problems (usually half the size)

Gr Greedy

- make locally optimal choice at each step

DP Dynamic Programming

- problem that has overlapping subproblems and optimal substructure

[1] content in this section based on "Competitive programming 1", by Steven and Felix Halim

Problem-solving paradigms in computing[1]

CS Complete search (brute force)

- solve problem searching the entire search space

D&C Divide & Conquer

- make problem 'simpler' by dividing into sub-problems (usually half the size)

Gr Greedy

- make locally optimal choice at each step

DP Dynamic Programming

- problem that has overlapping subproblems and optimal substructure

[1] content in this section based on "Competitive programming 1", by Steven and Felix Halim

Problem-solving paradigms in computing[1]

CS Complete search (brute force)
- solve problem searching the entire search space

D&C Divide & Conquer
- make problem 'simpler' by dividing into sub-problems (usually half the size)

Gr Greedy
- make locally optimal choice at each step

DP Dynamic Programming
- problem that has overlapping subproblems and optimal substructure

[1] content in this section based on "Competitive programming 1", by Steven and Felix Halim

Problem-solving paradigms in computing[1]

CS Complete search (brute force)

- solve problem searching the entire search space

D&C Divide & Conquer

- make problem 'simpler' by dividing into sub-problems (usually half the size)

Gr Greedy

- make locally optimal choice at each step

DP Dynamic Programming

- problem that has overlapping subproblems and optimal substructure

[1] content in this section based on "Competitive programming 1", by Steven and Felix Halim

Notes on CS

- Bug-free code never gives WA (wrong answer)
- Useful for 'small values', but inefficient for larger spaces (resulting in TLE (time limit exceeded)) (recall algorithm complexity)
- Can run CS on small instances of a hard problem to get some patterns from its output
- Can serve as verifier for faster non-trivial algorithms

Some tips on CS

- Filtering the right answer from a set of values vs. generating only the right values (latter more efficient)
- Think about the data space: remove infeasible space upfront
- Utilise symmetries
- Pre-computation: make better data structure, such that time making it outweighs the 'loss' in time searching
- Solve it 'backwards' from the data/results space fitting to the problem

Notes on D&C

- Divide the original problem into sub-problems — usually by half or nearly half
- Find (sub-)solutions for each of these sub-problems – which are now easier
- If needed, combine the sub-solutions to produce a complete solution for the main problem
- Binary search seems easy, but there are some creative options there

Notes on D&C

- Divide the original problem into sub-problems — usually by half or nearly half
- Find (sub-)solutions for each of these sub-problems – which are now easier
- If needed, combine the sub-solutions to produce a complete solution for the main problem
- Binary search seems easy, but there are some creative options there

Notes on Greedy

- Problem must exhibit two things in order for a greedy algorithm to work for it:
  1. It has optimal sub-structures. (Optimal solution to the problem contains optimal solutions to the sub-problems.)
  2. It has a greedy property. (If we make a choice that seems best at the moment and solve the remaining subproblems later, we still reach optimal solution. We never have to reconsider our previous choices)
- Risky: if you get WA and code is correct, the problem may not be greedy after all
- Use when input size is too large for CS or DP

Notes on Greedy

- Problem must exhibit two things in order for a greedy algorithm to work for it:
  1. It has optimal sub-structures. (Optimal solution to the problem contains optimal solutions to the sub-problems.)
  2. It has a greedy property. (If we make a choice that seems best at the moment and solve the remaining subproblems later, we still reach optimal solution. We never have to reconsider our previous choices)
- Risky: if you get WA and code is correct, the problem may not be greedy after all
- Use when input size is too large for CS or DP

Outline

Problem set

- See printout
- Most problems are taken from, or based on, the UVa problem set (includes ICPC problem set)
- Each problem falls in one or more PSP, ad hoc, and/or another
- Read them, classify them
- Go online to `http://survey.cs.uct.ac.za/limesurvey/index.php/784949/lang-en` and select your choices
- Discuss afterward, and solve them

Solution

- Charming canines – Divide & Conquer (binary search)
- Work reduction – Greedy
- Trainsorting – Dynamic Programming
- Wine trading in Gergovia – Greedy
- The jackpot – Dynamic Programming
- Durban prawns – Complete Search
- Mobile Phone Coverage – Complete Search + Geometry
- Blowing fuses – ad hoc (simulation)

Outline

A complete regional

- Divide in groups of 3
- Read the problem set: LA 2010 regionals
- Categorise the problems (mathsy, algorithmically, more theory/more programming; which type of algorithm/problem solving strategy)
- Make a plan of work as a team (who will act as team manager, how much time for a problem, which problem first, who works on which 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 27: 9:30-15:30
- Date of the regionals: 4 October, 2014