This course material is now made available for public usage.
Special acknowledgement to School of Computing, National University of Singapore
for allowing Steven to prepare and distribute these teaching materials.

CS3233
Competitive Programming

Dr. Steven Halim

Week 12 – Harder Stuffs


Jan-Apr 2009


Jan-Apr 2010


Jan-Apr 2011


Jan-Apr 2012

Outline

- Mini Contest #10 (the last one) + Discussion + Break
- CLASS PHOTO!!
- Admins
- Last Lecture (let’s not get too ambitious):
  - Problem Decomposition (Section 8.2)
  - Meet in the Middle/Bidirectional Search

The harder ones

(you have seen some of these before; now, let's demistify some of them)

Soft skills needed:

Ability to spot the individual components and break them apart!

This is based on what I know from ~ 1422 UVa problems

PROBLEM DECOMPOSITION

Problem Decomposition (1)

Binary Search the Answers + X

- We have seen this form earlier (Chapter 3.3)
- But the “X” component of this ‘classical’ combination can be “many thing”, not just simulation problem
- So far, I have seen that X can be:
  - Greedy algorithm: UVa 714, 11516
  - MCBM: UVa 10804, 11262
  - SSSP: UVa 10186
  - Max Flow: UVa 10983
- Tips to spot this type: If you guess the answer, will the problem turn into a True/False problem?

Problem Decomposition (2)

Involving DP 1D Range Sum/Max/Min

- This one can be easily decomposed
- Tips to spot this type:
  The problem ask you for **static range queries**
  - Especially the 1D one
  - Usually range sum, but can also be max/min queries, how?
- **Range Sum Query**: Pre-process the answers in O(n)
  - dp[0] = ans[0]
  - dp[i] = dp[i-1] + ans[i] $\forall$ i $\in$[1..n-1]
- So that each **RSQ** can be answered in O(1)
  - rsq(i, j) =dp[j] if i == 0, or dp[j] - dp[i - 1] if j > 0

Problem Decomposition (3)

SSSP/APSP/SCC contraction + DP/Something else

- Another 'classical' combination is to use shortest path (or SCC contraction) as one sub problem to transform the original problem into a shortest path table (or a DAG) and then pass this table (or DAG) to a DP/other solution
  - BFS/Dijsktra's to build shortest path matrix → DP-TSP (UVa 10937, 10944, 10405, 11813, NOI 2011)
  - Run Dijkstra's algorithm → build DAG from SP information → Counting paths on DAG (UVa 10917)
  - Run Floyd Warshall's algorithm → do something else (UVa 1233, 10793, 11463)
  - Run Tarjan's SCC algorithm to contract SCC → Longest Path in DAG (UVa 11324)
- Tips to spot this type: Shortest path (or SCC) is one of the component, but not the only one…

Problem Decomposition (4)

X + Y

- Here, X is the "main issue"
  - But that problem is written in Y flavour
- Tips to spot this type: Usually,
  - X is either: BFS, Complete Search, Binary Search, (mostly Chapter 3 stuffs), and
  - Y is either: Graph, Mathematics, or Geometry (mostly Chapter 4-5-7 stuffs)
- Example: UVa 11730
  - Actually a BFS (SSSP on unweighted graph) problem
  - But the graph is implicitly derived via Mathematical rules

Problem Decomposition (5)

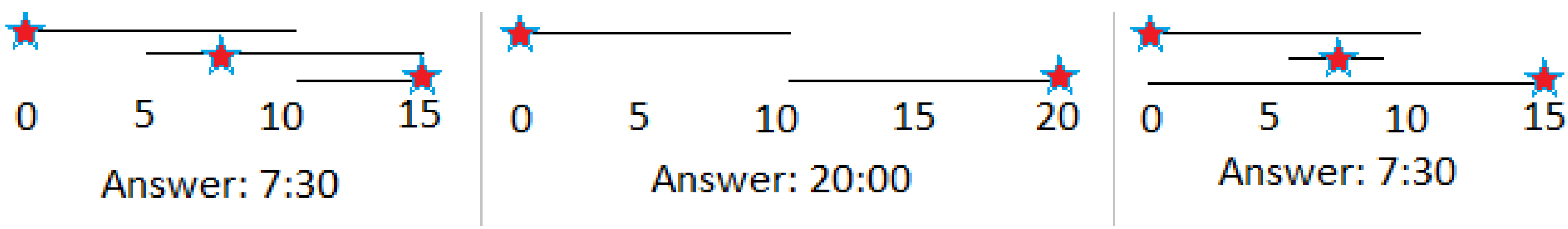
Involving (Advanced) Data Structures/DS

- Tips to spot this type: If you got a problem "AC" but very slow (TLE)
- Consider the possibility that some operations in your algorithm can be optimized by using a better DS
  - This better DS are usually harder to implement though
- These DSes are usually:
  - Balanced BST: map/set,
    or the self-coded one due to the need to *augment data*
  - Binary Indexed (Fenwick) Tree
  - Segment Tree, etc

Problem Decomposition (6)

Three (or More?) Components

- UVa 1079 – A Careful Approach
  - http://uva.onlinejudge.org/external/10/1079.html
  - ACM ICPC World Finals 2009 problem
- Solution:
  - Complete Search + Binary Search the Answer + Greedy :O



Problem Decomposition (7)

- There are many other possible combinations…
- Note: If there are X basic types of contest problems…
  - There can be ${}_{X}C_{2}$ possible pairs of combinations
  - And there can be ${}_{X}C_{3}$ triples…
- You will get more familiar to spot the individual components as you master them
  - All the best

a.k.a. Bidirectional Search

MEET IN THE MIDDLE

UVa 11212 – Editing a Book

Rujia Liu's Problem

- Given *n* equal-length paragraphs numbered from 1 to *n*
- Arrange them in the order of 1, 2, ..., *n*
- With the help of a clipboard,
  you can press Ctrl-X (cut) and Ctrl-V (paste) several times
  - You cannot cut twice before pasting, but you can cut several contiguous paragraphs at the same time - they'll be pasted in order
- The question: What is the minimum number of steps required?
- Example 1: In order to make {2, 4, (1), 5, 3, 6} sorted,
  you can cut 1 and paste it before 2 → {**1,** 2, 4, 5, (3), 6}
  then cut 3 and paste before 4 → {1, 2, **3,** 4, 5, 6} → done √
- Example 2: In order to make {(3, 4, 5), 1, 2} sorted,
  you can cut {3, 4, 5} and paste it after {1, 2} → {1, 2, **3, 4, 5**} √
  or cut {1, 2} and paste it before {3, 4, 5} → {**1, 2**, 3, 4, 5} √

Loose Upper Bound

- Answer: *k*-1
  - Where *k* is the number of paragraph in the wrong position
- Trivial but wrong algorithm:
  - Cut a paragraph that is in the wrong position
  - Paste that paragraph in the correct position
  - After *k*-1 such cut-paste, we will have a sorted paragraph
    - The last wrong position will be in the correct position at this stage
  - But this may not be the shortest way
- Examples:
  - {(3), 2, 1} → {(2), 1, **3**} → {1, **2**, 3} → 2 steps
  - {(5), 4, 3, 2, 1} → {(4), 3, 2, 1, **5**} → {(3), 2, 1, **4**, 5} → {(2), 1, **3**, 4, 5} → {1, **2**, 3, 4, 5} → 4 steps

The Actual Answers

- {3, 2, 1}
  - Answer: 2 steps, e.g.
    - {(3), 2, 1} → {(2), 1, **3**} → {1, **2**, 3}, or
    - {3, 2, (1)} → {**1**, (3), 2,} → {1, 2, **3**}
- {5, 4, 3, 2, 1}
  - Answer: Only **3** steps, e.g.
    - {5, 4, (3, 2), 1} → {**3, (2**, 5), 4, 1} → {3, 4, (1, **2), 5**} → {**1, 2**, 3, 4, 5}
- How about {5, 4, 9, 8, 7, 3, 2, 1, 6}?
  - Answer: 4, but very hard to compute manually
- How about {9, 8, 7, 6, 5, 4, 3, 2, 1}?
  - Answer: 5, but very hard to compute manually

Some Analysis

- There are at most n! permutations of paragraphs
  - With maximum n = 9, this is 9! or 362880
  - The number of vertices is not that big actually
- Given a permutation of length n (a vertex)
  - There are ${}_nC_2$ possible cutting points (index i, j $\in$ [1..n])
  - There are n possible pasting points (index k $\in$ [1..(n-(j-i+1))])
  - Therefore, for each vertex, there are about $O(n^3)$ branches
- The worst case behavior if we run single BFS on this search space graph: $O(V+E) = O(n! + n!*n^3) = O(n!*n^3)$
  - With n = 9, this is $9! * 9^3 = 264539520$ ~ 265 M, TLE (or maybe MLE…)

All other details are hidden for NUS ACM ICPC/Singapore IOI teams only :)

Actually we will still meet again next week for final contest :D

SOME PARTING WORDS

What You Have Been Exposed To
(as of Tonight, Wed 04 Apr 2012)

- Competitive Coding Style
- Extensive usage of libraries
- Bitmask
- BIT/FT
- Iterative BF Techniques: Subset, Permutation
- Recursive backtracking
- Some classical Greedy problems
- Binary Search the Answer
- The thinking process to get DP states and transitions
- Graph DS, Traversal: DFS/BFS, MST (briefly), SSSP: Dijkstra's, Bellman Ford's, APSP: Floyd Warshall's
- Tarjan's SCC algorithm
- More DP techniques
- Network Flow: Edmonds Karps'
- Bipartite Graph: MCBM++
- Mathematics-related problems: Log techniques, Big Integer, Prime Factor techniques, Modulo arithmetic
- Various string processing skills
- Suffix Tree/Array: String Matching, Longest Repeated Substring, Longest Common Substring
- Basic geometry routines
- Algorithms on polygon
- Problem decomposition
- Meet in the middle/bidirectional search

What You Have NOT Been Exposed To
(as of Tonight, Wed 04 Apr 2012)

- Many more cool and exotic algorithms out there :O
- Maybe read CP3 in the future ☺
- Or join NUS ACM ICPC trainings
- Or do self study