CS3233
Competitive Programming

Dr. Steven Halim

Week 05 – Problem Solving Paradigms
(Dynamic Programming 2)

Outline

- Mini Contest #4 + Break + Discussion + Admins
- A simple DP problem to refresh our memory (Section 3.5.3)
- DP and its relationship with (implicit) DAG (Section 4.7.1)
  - These are CS2020/CS2010 materials
    - Those who have not taken either module must consult Steven separately
- DP on Math Problems (Section 5.4 and 5.6)
- DP on String Problems (Section 6.5)
- More DP techniques (Section 8.3)
- Pointers to other DP techniques in CP2.9

More DP Problems in Chapter 3-4-5-6-8-9 of CP2.9 Book

NON CLASSICAL DP PROBLEMS

Non Classical DP Problems

- **My** definition:
  - Not the pure form (or simple variant) of 1D/2D Max Sum, LIS, 0-1 Knapsack/Subset Sum, Coin Change, TSP where the DP **states** and **transitions** can be "memorized"
  - Requires **original*** **formulation** of DP states and transitions
  - Throughout this lecture, we will talk mostly in *DP terms*
    - **State** (to be precise: "*distinct* state")
    - **Space Complexity** (i.e. the number of distinct states)
    - **Transition** (which entail overlapping sub problems)
    - **Time Complexity** (i.e. num of distinct states * time to fill one state)

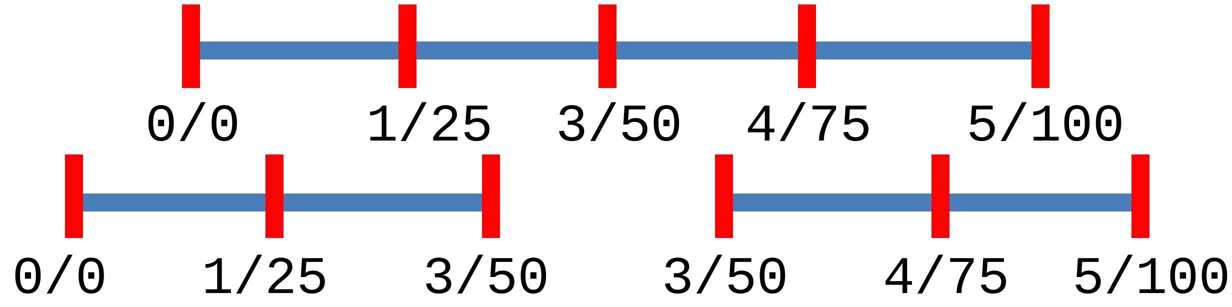
CP2.9, Section 3.5.3

Refresher: Cutting Sticks

- State: index **(l, r)** where **l, r** ∈ **[0..n+1]** and **l < r**
  - Q: Why these two parameters?
- Space Complexity: **O(n²)** distinct states
- Transition: Try all possible cutting points **i** between **l** and **r**,
  - i.e. cut **(l, r)** into **(l, i)** and **(i, r)** with cost **(A[r] - A[l])**
- Time Complexity: There are **O(n)** possible cutting points, thus overall **O(n² * n) = O(n³)**

A = {25, 50, 75}
n = 3, L = 100

A = {0, 25, 50, 75, 100}



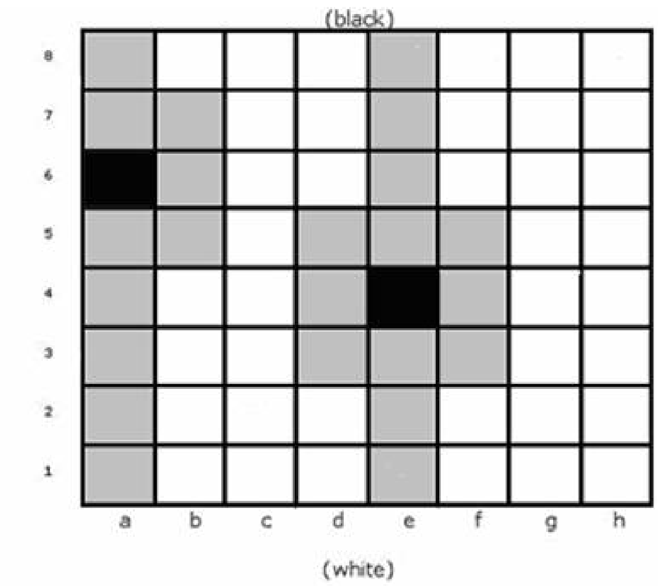
DP on DAG

Overview

- Dynamic Programming (DP) has a close relationship with (usually implicit) Directed Acyclic Graph (DAG)
  - The **states** are the **vertices** of the DAG
  - Space complexity: Number of vertices of the DAG
  - The **transitions** are the **edges** of the DAG
    - Logical, since a recurrence is always **acyclic**
  - Time complexity: Number of edges of the DAG
  - Top-down DP: Process each vertex just once via **memoization**
  - Bottom-up DP: Process the vertices in **topological order**
    - Sometimes, the topological order can be written by just using simple (nested) loops

The Injured Queen Problem

- Like N-queens problem, but the queens are "injured" (can only attack the current column but acts as king otherwise)
- With some of K ($0 \le K \le N$) injured queens positions have been predetermined, count how many possible arrangements of the other (N-K) queens so that no two queens attack each other?



DP on Math Problems

- Some well-known mathematic problems involves DP
  - Some combinatorics problem have recursive formulas which entail overlapping subproblems
    - e.g. those involving Fibonacci number, **f(n) = f(n - 1) + f(n -2)**
  - Some probability problems require us to search the entire search space to get the required answer
    - If some of the sub problems are overlapping, use DP, otherwise, use complete search
  - Mathematics problems involving **static** range sum/min/max!
    - Use dynamic tree DS for dynamic queries

Dice Throwing

- n common cubic dice are thrown ($1 \leq n \leq 24$)
- What is the probability that the sum of all thrown dices is at least x? ($0 \leq x \leq 150$)
- Basic probability = # events / |sample space|
- To compute the |sample space| is easy: It is 6^n
- The # events is harder to compute…

CP2.9, Section 6.5

DP on String Problems

- Some string problems involves DP
  - Usually, we do not work with the string itself
  - But we work with the integer indices to represent suffix/prefix/substring
  - Reason: Too costly to pass (sub)strings around as function parameters

String Partition

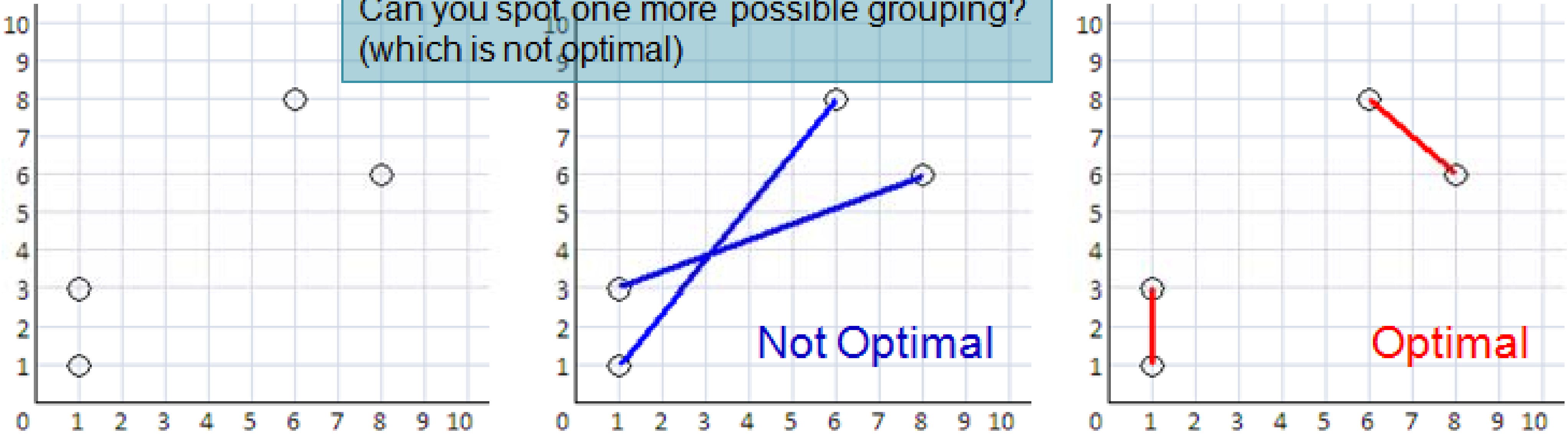
- There are many ways to split a string of digits into a list of non-zero-leading (0 itself is allowed) 32-bit *signed* integers
  - What is the maximum sum of the resultant integers if the string is split appropriately? Examples:
    - 1234554321
      - 1234554321 < 2147483647, so the answer is 1234554321 itself
    - 5432112345
      - 5432112345 > 2147483647, thus 5432112345 must be partitioned
      - There are two ways to partition 5432112345
        - » 5 + 432112345 = 432112350, or
        - » 543211234 + 5 = 543211239 ← the answer
    - 121212121212
      - 121212121212 > 2147483647, thus 121212121212 must be partitioned
      - The answer is: 1 + 2121212121 + 2 = 2121212124

DP with bitmask

- Bitmask technique can be used to represent *lightweight set of Boolean* (up to 2^{64} if using unsigned long long)
- This is important if one of the DP parameter is a "small set"
- We have seen this form earlier in DP-TSP
- One other useful application (there are many others):
  - ***Finding min weighted perfect matching in small general graph***

Forming Quiz Teams

Can you spot one more possible grouping?
(which is not optimal)



N = 2

Cost = 8.60 + 7.61 = **16.21**

Cost = 2.00 + 2.83 = **4.83**

Common DP States (1)

- Position:
  - Original problem: $[x_1, x_2, \ldots, x_n]$
    - Can be sequence (integer/double array), can be string (char array)
  - Sub problems, break the original problem into
    - Sub problem and Prefix: $[x_1, x_2, \ldots, x_{n-1}] + x_n$
    - Suffix and sub problem: $x_1 + [x_2, x_3, \ldots, x_n]$
    - Two sub problems: $[x_1, x_2, \ldots, x_i] + [x_{i+1}, x_{i+2}, \ldots, x_n]$
  - Example: 1D Max Sum, LIS, etc

Common DP States (2)

- Position**<u>s</u>**:
  - This is similar to the previous slide
  - Original problem: $[x_1, x_2, \ldots, x_n]$ and $[y_1, y_2, \ldots, y_n]$
    - Can be two sequences/strings
  - Sub problems, break the original problem into
    - Sub problem and prefix: $[x_1, x_2, \ldots, x_{n-1}] + x_n$ and $[y_1, y_2, \ldots, y_{n-1}] + y_n$
    - Suffix and sub problem: $x_1 + [x_2, x_3, \ldots, x_n]$ and $y_1 + [y_2, y_3, \ldots, y_n]$
    - Two sub problems: $[x_1, x_2, \ldots, x_i] + [x_{i+1}, x_{i+2}, \ldots, x_n]$ and $[y_1, y_2, \ldots, y_i] + [y_{i+1}, y_{i+2}, \ldots, y_n]$
  - Example: String Alignment/Edit Distance, LCS, Matrix Chain Multiplication (MCM), etc
  - PS: Can also be applied on 2D matrix, like 2D Max Sum, etc

Tips: When to Choose DP

- Default Rule:
  - If the given problem is an **optimization** (max/min) or **counting** problem
    - Problem exhibits optimal sub structures
    - Problem has overlapping sub problems
- In ICPC/IOI:
  - If actual solutions are not needed (only final values asked)
    - If we must compute the solutions too, a more complicated DP which stores *predecessor information* and *some backtracking* are necessary
  - The number of distinct sub problems is small enough (< 1M)
    and you are not sure whether greedy algorithm works (why gamble?)
  - Obvious overlapping sub problems detected :O

Dynamic Programming Issues (1)

- Potential issues with DP problems:
  - They may be disguised as (or looks like) non DP
    - It looks like greedy can work but some cases fails…
      - e.g. problem looks like a shortest path with some constraints on graph, but the constraints fail *greedy* SSSP algorithm!
  - They may have subproblems but not overlapping
    - DP does not work if overlapping subproblems not exist
      - Anyway, this is still a good news as perhaps Divide and Conquer technique can be applied

Dynamic Programming Issues (2)

– Optimal substructures may not be obvious
  1. Find correct “states” that describe problem
     – Perhaps extra parameters must be introduced?
  2. Reduce a problem to (smaller) sub problems (with the same states) until we reach base cases
– There can be more than one possible formulation
  - Pick the one that works!

DP Problems in ICPC (1)

- The number of problems in ICPC that must be solved using DP are growing!
  - At least one, likely two, maybe three per contest…
- These new problems are **not** the classical DP!
  - They require deep thinking…
  - Or those that look solvable using other (simpler) algorithms but actually must be solved using DP
  - Do not think that you have “mastered” DP by only memorizing the classical DP solutions!

DP Problems in ICPC (2)

- In 1990ies, mastering DP can make you "king" of programming contests…
  - Today, it is a must-have knowledge…
  - So, get familiar with DP techniques!
- By mastering DP, your ICPC rank is probably:
  - from top ~[25-30] (solving 1-2 problems out of 10)
    - Only easy problems
  - to top ~[15-20] (solving 3-4 problems out of 10)
    - Easy problems + brute force + DP problems

For Week 07 homework ☺

(You can do this over recess week too)

BE A PROBLEM SETTER

Be a Problem Setter

- Problem Solver:
  A. Read the problem
  B. Think of a good algorithm
  C. Write ‘solution’
  D. Create tricky I/O
  E. If WA, go to A/B/C/D
  F. If TLE/MLE, go to A/B/C/D
  G. If AC, stop ☺

- Problem Setter:
  A. Write a <u>good</u> problem
  B. Write <u>good</u> solution<u>s</u>
     - The correct/best one
     - The incorrect/slower ones
  C. Set a <u>good</u> secret I/O
  D. Set problem settings
- A problem setter <u>must</u> think from a different angle!
  – By setting good problems, you will simultaneously be a better problem solver!!

Problem Setter Tasks (1)

- Write a <u>good</u> problem
  - Options:
    - Pick an algorithm, then find problem/story, or
    - Find a problem/story, then identify a good algorithm for it (harder)
  - Problem description must not be <u>ambiguous</u>
    - Specify input constraints
    - Good English!
    - Easy one: longer,
      Hard one: shorter!
- Write <u>good</u> solution<u>**s**</u>
  - Must be able to solve your own problem!
    - To set hard problem, one must increase his own programming skill!
  - Use the <u>best possible</u> algorithm with lowest time complexity
    - Use the inferior ones 'that barely works' to set the WA/TLE/MLE settings...

Problem Setter Tasks (2)

- Set a good secret I/O
  - Tricky test cases to check AC vs WA
    - Usually 'boundary case'
  - Large test cases to check AC vs TLE/MLE
    - Perhaps use input generator to generate large test case, then pass this large input to our correct solution
- Set problem settings
  - Time Limit:
    - Usually 2 or 3 times the timings of your own best solutions
      - Java is slower than C++!
  - Memory Limit:
    - Check OJ setting^
  - Problem Name:
    - Avoid revealing the algorithm in the problem name

FYI: Be A Contest Organizer

- Contest Organizer Tasks:
  - Set problems of *various* topic
    - Better set by >1 problem setter
  - Must balance the difficulty of the problem set
    - Try to make it fun
    - Each team solves some problems
    - Each problem is solved by some teams
    - No team solve all problems
      - Every teams must work until the end of contest

More References

- **Competitive Programming 2.9**
  - Section 3.5, 4.7.1, 5.4, 5.6, 6.5, 8.3, and parts of Ch9
- **Introduction to Algorithms**, p323-369, Ch 15
- **Algorithm Design**, p251-336, Ch 6
- **Programming Challenges**, p245-267, Ch 11
- http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=dynProg
- Best is practice & more practice!