GB21802 - Programming Challenges

Week 9 - Programming Challenges Remix!

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2015-06-23,26

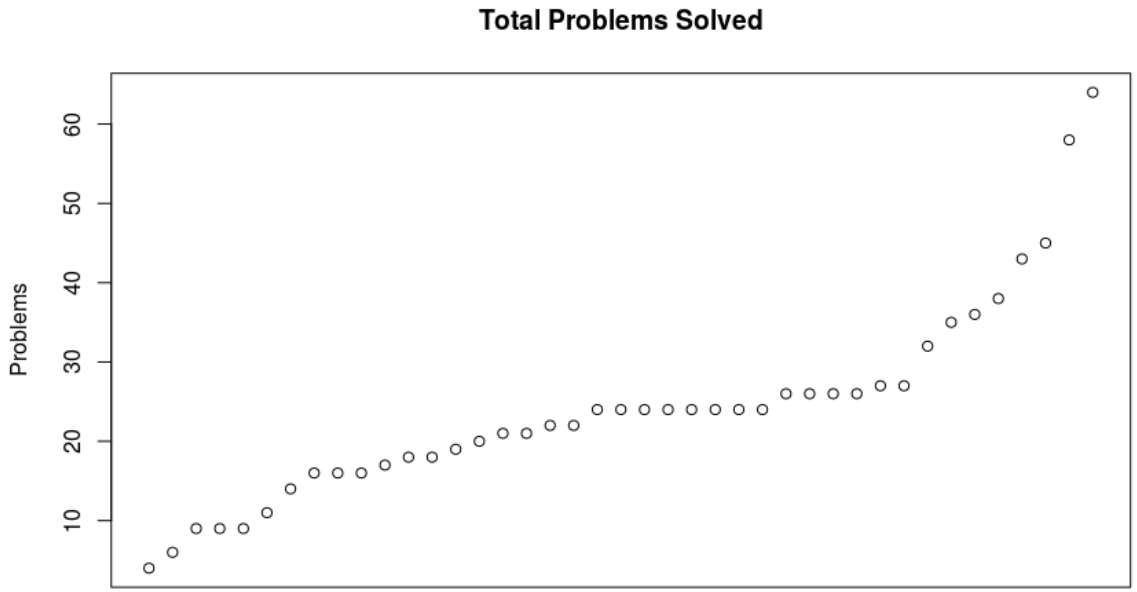
Last updated June 20, 2017

Results for the Previous Week

Results Last week:

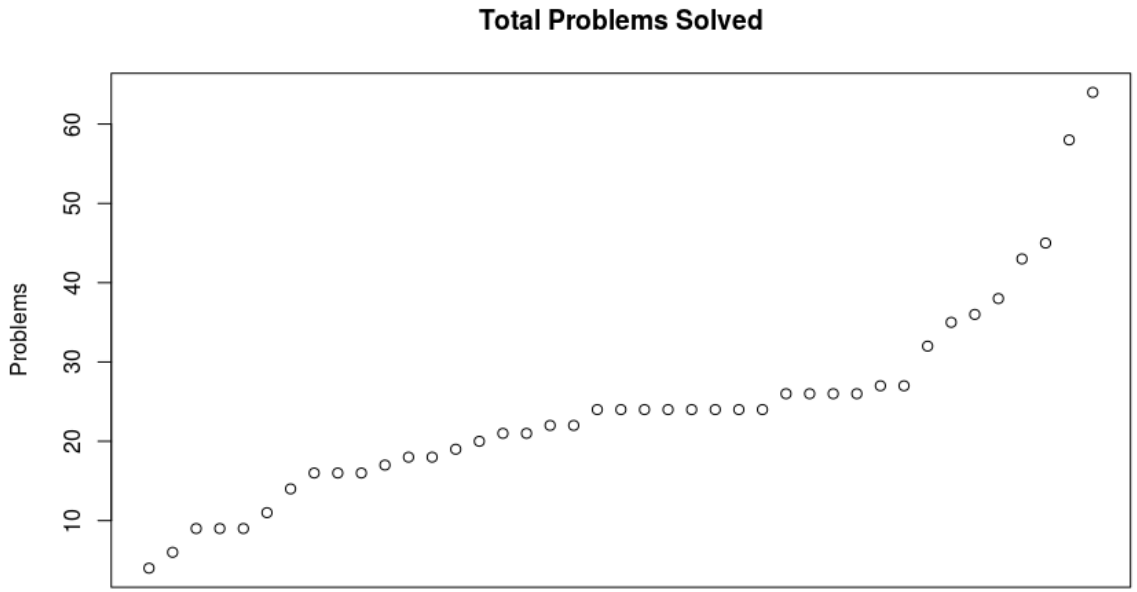
Grade Situation (1)

Total Grades up to Week 7:



- Base A grades: 17 people
- Base B grades: 5 people
- Base C grades: 12 people
- Base D Grades: 10 people (7 abandoned, 3 still close)

Grade Situation (2)



Current Grades A: 17, B: 5, C: 12, D: 10 – Please check your status!

- Final Deadline for problem submissions: 07/05, 23:59
- Grades Announcement: 07/10 (Mon – Manaba)
- Grades Adjustment Deadline: 07/17 (Mon), 17:00

Course Summary – Solving a problem

In this course, we studied and practice many ways to solve problems using computer algorithms. Many problems can be imagined as *searches*.

General Problem Solving:

- Identify the full search approach
- Think about edge and special cases
- See if a better algorithm is needed

Course Summary – Topics Approached

In this course we also saw many examples of specific algorithms for problems.

- Graphs (Minimum spanning tree, Bellman-Ford APSP, . . .)
- Mathematics (Eristhenes Sieve, Prime Factoring)
- Computational Geometry (Convex Hull)
- String (Knuth-Morris-Prat, suffix trie)

Course Summary – Multi-Problems

The most interesting problems are those that mix two or more different algorithms. Or require variations of standard algorithms.

This week, we will try to solve together some of these more interesting problems.

UVA 10937 – Blackbeard the Pirate

Blackbeard has to collect all treasures (up to 10) in the island. He cannot cross water or trees, and he must stay 1 square away from natives.

Black beard speed is 1 square / second. How long does it take to get all treasure and return to the ship?

```
10 10
~~~~~~~~~~          ~ -- Water, can't cross
~~!!!###~~          # -- Trees, can't cross
~##...###~          ! -- Treasure, get these!
~#....*##~          . -- Just sand
~#!..**~~~          @ -- Landing point, return here.
~~....~~~~
~~~....~~~          The solution for this case is: 32
~~..~..@~~          How would YOU solve this problem?
~#!.~~~~~~
~~~~~~~~~~
0 0
```

UVA 10937 – Blackbeard the Pirate

How would you solve this problem?

- Which algorithm do you suggest?
- What is the complexity of that algorithm?
- Where do you need to be careful?

Quiz – What is the complexity of the problem for full recursive search? (Map size $50x50$, number of treasures is 10)

UVA 10937 – Blackbeard the Pirate

Breaking the problem into parts

One way to solve this problem is to break it into two parts:

1. Extract a weighted distance graph from the input map
2. Solve the TSP for the graph

```
10 10
~~~~~~~~~~          ~ -- Water, can't cross
~~!!!###~~          # -- Trees, can't cross
~##...###~          ! -- Treasure, get these!
~#....*##~          . -- Just sand
~#!..**~~~          @ -- Landing point, return here.
~~....~~~~
~~~....~~~
~~..~..@~~
~#!.~~~~~~
~~~~~~~~~~
0 0
```

UVA 10937 – Blackbeard – Extracting the graph

```
10 10
~~~~~~~~~~   ##########   # -- Obstacle (waters and trees)
~~!!!###~~   ##345#####   X -- Obstacles (natives, just for clar
~##...###~   ###..X####   . -- Path
~#....*##~   ##..XXX###   0-9 -- Nodes
~#!..**~~~   ##2.XXX###
~~....~~~~   ##..XX####
~~~....~~~   ###....###
~~..~..@~~   ##..#..0##
~#!.~~~~~~   ##1.######
~~~~~~~~~~   ##########
0 0
```

- We can simply the graph into obstacles, paths and goals
- We are only interested in the treasures and goals, so how to find the pairwise distance between treasures?
- Answer:
- The result is a small graph with **at most** 11 vertices.

UVA 10937 – Blackbeard – Extracting the graph

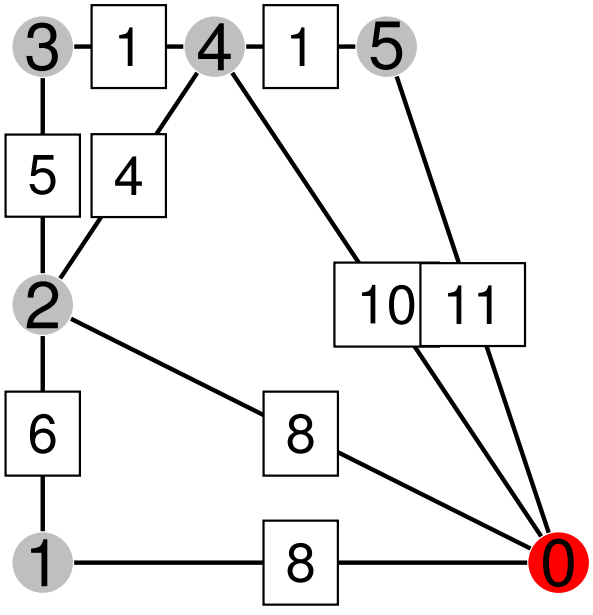
```
10 10
~~~~~~~~~~   ##########   # -- Obstacle (waters and trees)
~~!!!###~~   ##345#####   X -- Obstacles (natives, just for clar
~##...###~   ###..X####   . -- Path
~#....*##~   ##..XXX###   0-9 -- Nodes
~#!..**~~~   ##2.XXX###
~~....~~~~   ##..XX####
~~~....~~~   ###....###
~~..~..@~~   ##..#..0##
~#!.~~~~~~   ##1.######
~~~~~~~~~~   ##########
0 0
```

- We can simply the graph into obstacles, paths and goals
- We are only interested in the treasures and goals, so how to find the pairwise distance between treasures?
- Answer: BFS from each treasure/start point
- The result is a small graph with **at most** 11 vertices.

UVA 10937 – Blackbeard – Extracting the graph

```
##########
##345#####
###..X####
##..XXX###
##2.XXX###
##..XX####
###....###
##..#..0##
##1.######
##########
```

```
BFS from each vertex
-------------------->
Not all paths shown
```



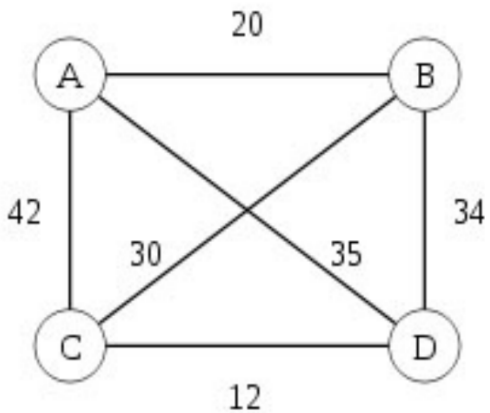
How do we find the minimal cycle starting in **S**, passing by all vertices?

The Traveling Salesman Problem (TSP)

Problem Definition

You have n cities, and their distances. Calculate the cost of the tour that starts and ends at a city s, passing through all other cities.

Exactly what we need! The path for all treasure!



	A	B	C	D
A	0	20	42	35
B	20	0	30	34
C	42	30	0	12
D	35	34	12	0

In the graph above, we have $n = 4$ cities and the minimal tour is A-B-C-D-A, with cost $20 + 30 + 12 + 35 = 97$.

QUIZ: What is the cost of solving TSP with complete search?

Image from Steven Halim – "Competitive Programming"

Characteristics of TSP

- A complete search for TSP costs $O(n! * n)$ – Search each city permutation.
- TSP is a **NP-hard** problem. This means that there is no known polinomial algorithm to solve it.
- However! For small values of n, there are some hacks to make the solution faster.

DP approach to TSP

The complete search for the TSP contains many repeated subsolutions:

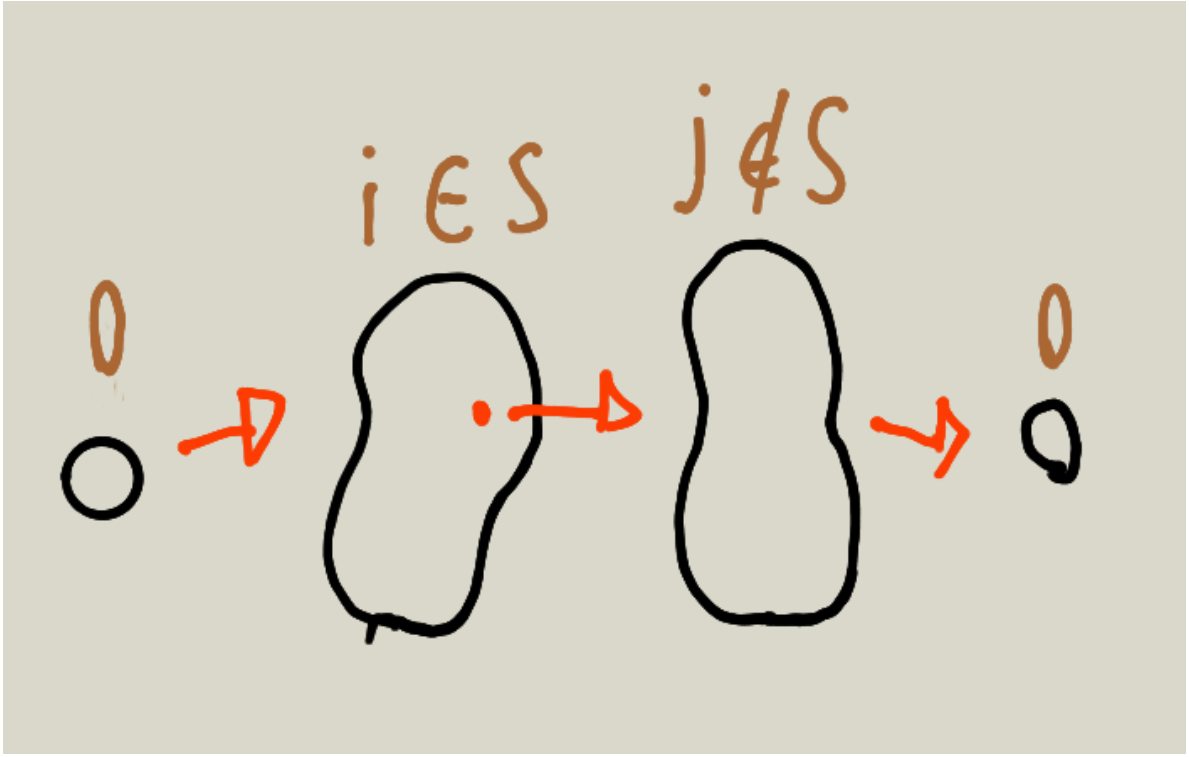
- S–A–B–C–. . .–S
- S–B–A–C–. . .–S

The minimum cost for C–. . .–S is the same. Can we use *memoization* to remember this cost?

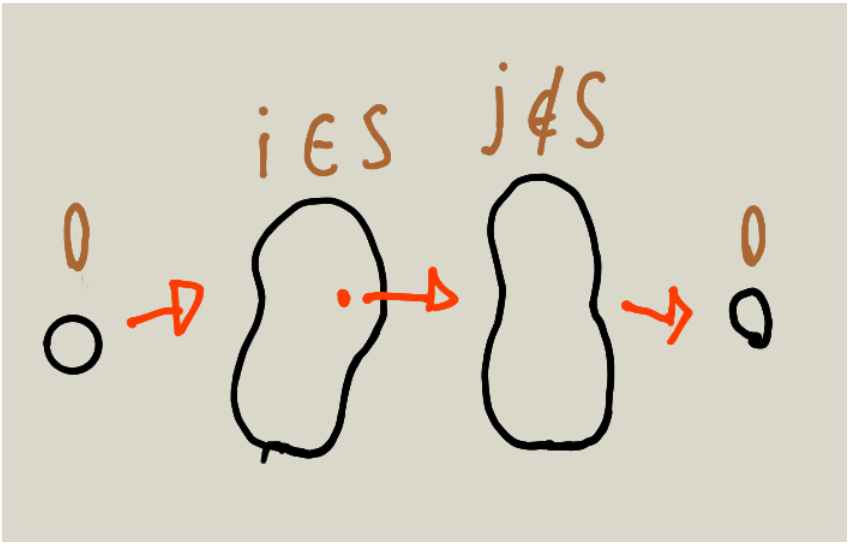
DP approach to TSP (1) – Idea

- We have already visited the cities $S = \{s_1, s_2, \ldots, s_n\}, s_i \neq 0$
- We are **now** in city $s_k \in S$
- What is the shortest path from s_k to 0, that passes in all cities $s_j \notin S$?

DP induction: shortest_path(S,s_k)



DP approach to TSP (2) – Recurrence



DP Recurrence

- We have visited all cities, and must return to the origin: shortest_path(S_{all}, s_k) = $D(s_k, 0)$
- We have visited some cites (S), and must find the next one: shortest_path(S, s_k) = **minimum**($D(s_k, s_i)$ + shortest_path($S \cup s_i, s_i$))
- Initial call: shortest_path(∅,0)

DP approach to TSP (3) – Implementation

- Our DP table is (*all sets*,*all cities*) – $2^n * n$
- We can represent a set of cities using a bitmask
- At each call, we loop through all cities, so the complexity is ($O(2^n * n^2)$)

- TSP using full search: $O(n! * n)$
- TSP using DP: $O(2^n * n^2)$ – Still low, but much better!

DP approach to TSP (4) – Sample Code

```
int dp[n][1<<n] = -1
start = 0

visit(p,v):
   if (v == (1<<n) - 1):
      return cost[p][start]
   if dp[p][v] != -1
      return dp[p][v]

   tmp = MAXINT
   for i in n:
       if not(v && (1 << i):
           tmp = min(tmp,
                     cost[p][i] + visit(i, v | (1<<i)))

   dp[p][v] = tmp
   return tmp
```

UVA 10943 – How do you add?

Problem Description

Given an integer n, how many ways can you add K integers ($0 \leq i \leq n$) so that their sum is equal to n?

Example: $n = 20, K = 2$
$0 + 20, 1 + 19, 2 + 18, \ldots, 20 + 0$

What is the recurrence?

- When K = 1, there is only one way to add to n.
- When K = i > 1, we can test all numbers X between 0 and n, and our result will be the sum of all $(n - X, K - 1)$ sub problems.

How do you add? – Recurrence Example

Recurrence Example: $n = 10, K = 3$

ways(10,3) =	(0,ways(10,2)) +
	(1,ways(9,2)) +
	(2,ways(8,2)) +
	...
	(10,ways(0,2))
ways(8,2) =	(0,ways(8,1)) +
	(1,ways(7,1)) +
	...
	(8,ways(0,1))
=	9
ways (10,3) =	$11 + 10 + 9 + 8 + \ldots + 1 = 66$

How do you add? – Bottom-Up DP

```
dp[maxK][maxN]
tsum[2][maxN]

for (i in maxN):
   tsum[1][i] = i+1
   dp[1][i] = 1

for (i in K):
   tsum[0] = tsum[1]
   for (j in maxN):
      dp[i][j] = tsum[0][j]
      tsum[1][j] = tsum[1][j-1] + tsum[0][j]
```

Time complexity is $O(nK)$

How do you Add? – Mathematical Approach

This problem can also be seen as solving the recurrence/closed form of a binomial combination C. We will come back to recurrences in a later class.

Thinking About DP – 1

DP can come in many forms other than its classical problems. You can solve these problems following the procedure that we have seen so far:

1. Elaborate a recursive, full search solution.
2. Define the distinct states and the transitions
3. Write a program for memoizing the state table (top-down) or constructing the table (bottom up).

DP has an intrinsic relationship with Directed Acyclic Graph (DAG). States are mapped to vertexes, transitions to edges. We will explore this relationship more in the future.

Thinking About DP – 2

Common ways to imagine DP states:

Position in the problem state

- The original problem is an array of values: $\{x_1, x_2, \ldots, x_n\}$
- Subproblems based on position in the array:
  Suffix: $\{x_1, x_2, \ldots, x_{n-1}\} + x_n$
  Prefix: $x_1 + \{x_2, x_3, \ldots, x_n\}$
  Two sub-problems: $\{x_1, x_2, \ldots, x_i\} + \{x_{i+1}, x_{i+2}, \ldots, x_n\}$
- Can be generalized for 2D (x,y): Consider the size of the table!

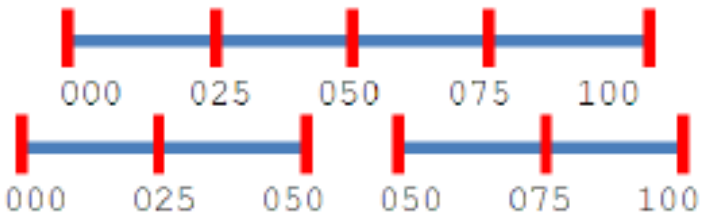
UVA 10003 – Cutting Sticks

Problem Description

Given a stick of length $1 \leq l \leq 1000$ and $1 \leq n \leq 50$ cuts (positions) to be made, the cost of a cut is given by the length of the stick being cut.

Find a cutting sequence that minimizes the cost of cutting the stick.

Example: $l = 100, n = 3, cuts = \{25, 50, 75\}$



- Sequence 1: 25, 50, 75. Cost: 100 + 75 + 50 = 225
- Sequence 2: 50, 25, 75. Cost: 100 + 50 + 50 = 200

What is the recurrence to find the minimum cost?

Cutting Sticks – Recurrence

Basic Idea: Price(start,end) = end-start + Price(start,cut) + Price(cut,end)

Problem: Using start/end, we have 1000*1000 states.

Solution: We use cut indexes instead! (50*50 states)

Recurrence using cut index:

- Price(i,i+1) = 0
- Price(i,j) = min(Size[j] - Size[i] + Price(i,k) + Price(k,j))
  (for all $i < k < j$)

Note that the recurrence costs $O(n)$, and the table size is $O(n^2)$, so the total cost is $O(n^3)$

DP on Math Problems

Many math problems can be implemented as DP:

- Combinatoric problems often have recursive formulas, and overlapping subproblems;
  - Fibonacci Number: $f(n) = f(n-1) + f(n-2)$
- Probability problems often require you to search the entire probability space (tree). These trees usually have overlapping branches.
- Maths problems on static data (sum, min, max)

DP on Strings

- edit distance, substring manipulation, etc.
- Usually, we don't send the string in the recurrence, but indexes on the string

DP Issues

- Many DP problems look like “non DP” problems.

Example

Select positions for a set of flags that cover a certain radius, in order to maximize area coverage.

The area calculation requires geometry, but the flag selection is usually DP.

- Some problem have sub-problems, but they are not overlapping. (In this case, DP will not work, but maybe Divide and Conquer?)

That's all for DP!

This is what we've seen for weeks 3 and 4:

- DP = Complete Search + State table;
- Good when many overlapping subproblems exist;
- Top-Down (recursive) and Bottom-Up (nested loops);
- Classical DP problems;
- (some!) non-classical DP problems;

For the next two weeks, the theme will be graph algorithms!

Problems for Week 4

- Collecting Beepers
- Shopping Trip
- Bar Codes
- Cutting Sticks
- String Popping
- Divisibility
- Marks Distribution
- Squares

World Finals Problem - 2016 Problem C

Problem Description – Ceiling Function

Given n arrays with K values each, each array is organized in a binary search tree in the order of input.

In other words, the first element is the root, the second is the left child of the root if smaller, right child if bigger, so on.

Count the number of different trees generated by the input data.