GB21802 - Programming Challenges

Week 9 - Programming Challenges Remix!

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2019-06-21,24

Last updated June 20, 2019

Grade Dates

- Final Submission Date: 7/10
- Grade Announcement: 7/14
- Grade Registration: 7/20

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
~~~~~~~~~~
~~!!!###~~
~##...###~
~#....*##~
~#!..**~~~
~~....~~~~
~~~....~~~
~~..~..@~~
~#!.~~~~~~
~~~~~~~~~~
0 0
```

```
~ -- 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?
```

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 50*x*50, 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

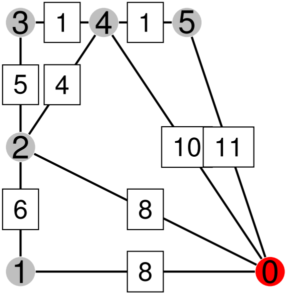
```
10 10
~~~~~~~~~~  ##########  # -- Obstacle (waters and trees)
~~!!!###~~  ##345#####  X -- Obstacles (natives, just for clar
~##...###~  ###..X####  . -- Path
~#....*##~  ##..XXX###  0-9 -- Nodes
~#!..**~~~  ##2.XXX###
~~....~~~~  ##..XX####
~~~....~~~  ###....###
~~..~..@~~  ##..#..0##
~#!.~~~~~~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



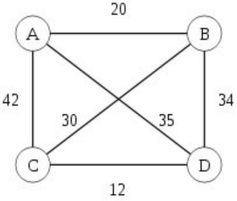
How do we find the minimal cycle starting in **S**, passing by all vertices?

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:

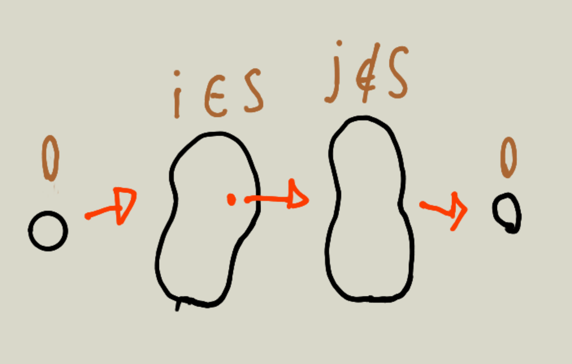
- S–A–B–C–. . .–S
- S–B–A–C–. . .–S

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

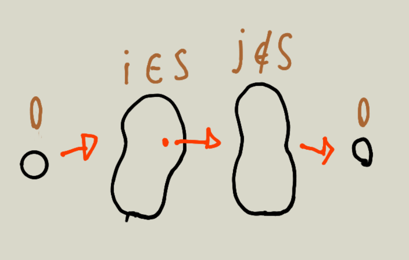
DP approach to TSP (1) – Idea

- 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) = $\min_{s_i \notin S}(D(s_k, s_i) +$ shortest_path$(S \cup s_i, s_i))$
- Initial call: shortest_path($S = \emptyset$,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

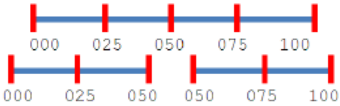
```
int dp[n][1<<n] = -1
start = 0

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 10003 – Cutting Sticks

Problem Description

- In a stick of length l ($1 \leq l \leq 1000$)
- Make N cuts at positions $\text{cuts} = \{c_1, c_2, \ldots, c_N\}$ ($1 \leq N \leq 50$)
- The cost of a cut is the size of the sub-stick that you cut.
- What order of cuts minimize the cost?

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



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

UVA 10003 – Cutting Sticks – Questions

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

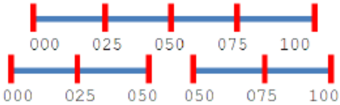
Quiz 1

- What is the algorithm for a full search?
- What is the complexity of this algorithm? And the maximum time?

Quiz 2

- This problems smells of **DP**...
- But what are the **states**, and what is the **transition**?

UVA 10003 – Cutting Sticks – Recurrence



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

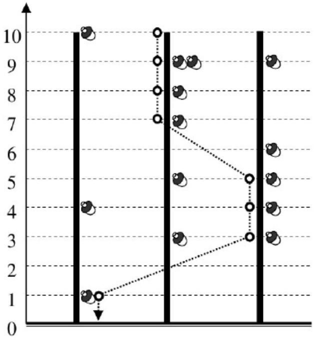
Recurrence

Let's think of a Top-down DP based on a recursive function:

- $A = \{0, c_1, c_2, \ldots c_N, N+2\}$ is the set of all cutting points, plus the start and end point.
- $\text{cost}(a_i, a_j) = \text{dist}(a_i, a_j) + \min_{i \le k \le j}(\text{cost}(a_i, a_k) + \text{cost}(a_k, a_j))$
- $\text{cost}(a_i, a_i) = 0$

This requires at most a (N, N) DP table for memoization, and O(N) for each iteration.

UVA 1231 – ACORN



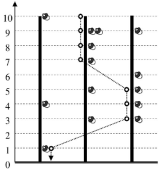
- Begin at the top of a tree, and get the maximum number of acorns.
- You can go down **1 height** on the tree.
- OR **change tree** for the cost of **f height** (In this figure, $f = 2$)

- Number of trees: $1 \leq T \leq 2000$
- Height of trees: $1 \leq H \leq 2000$
- Length of fall : $1 \leq f \leq 500$

Let's skip the full search. This smells of DP, but how to solve it?

QUIZ: Describe the **transition** and the **state table**

UVA 1231 – ACORN – Simple Recurrence



Simple Recurrence

- acorn[t_i][h] – number of acorns in tree t_i at height h
- $\text{cost}(t_i, 0) = \text{acorn}[t_i][0]$
- $\text{cost}(t_i, j) = \text{acorn}[t_i][j] + \max_{k \neq t_i}(\text{cost}(t_i, j-1), \text{cost}(t_k, j-f))$
  (Don't forget to check $j - f < 0$)
- Final cost: $\max_{1 \leq i \leq T}(\text{cost}[t_i][H])$

QUIZ: What is the problem with this recurrence?

UVA 1231 – ACORN – Better DP table

The DP table of last slide is A[H][T], with size $2000 * 2000 = 4M$. Each function call is $O(H * T * T)$, so total complexity is $4M * 2000 = 8B$

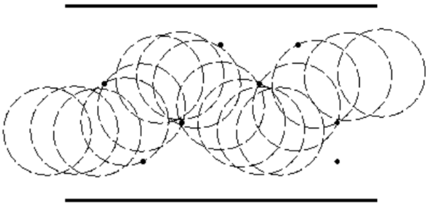
- Cost of changing tree is constant for any two trees.
- It is not necessary to keep all trees, only the best.

Better Recurrence – $O(H * T)$

We use the table dp[H] which contains the best solution at height H.

- $\text{dp}[0] = \max_{1 \leq j \leq T} \text{acorn}[j][0]$
- $\text{acorn}[j][i] += \max(\text{acorn}[j][i-1], \text{maxac}[i-f])$
- $\text{dp}[i] = \max_{1 \leq j \leq T}(\text{acorn}[j][i])$

UVA 295 – Fatman!



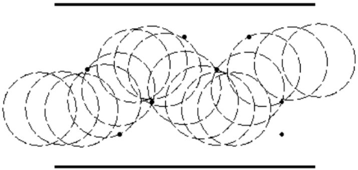
Problem Description

Find the **maximum diameter D of the circle** that can pass the corridor.

- The corridor has length L and width W;
- The corridor has $0 \leq N \leq 100$ obstacles, represented by (x_i, y_i);
- Obstacles are **points** with $0 \leq x_i \leq L, 0 \leq y_i \leq W$;

QUIZ: How do you solve this problem? (to 4 decimal places)

UVA 295 – Fatman – Breaking up the problem



One way to solve some problems is to break them down into smaller components.

We can break this problem into two sub-problems:

1. Is it possible for a circle of size $0 \leq R \leq W$ to pass?
2. What is the maximum R that can pass?

QUIZ: Assume that (2) is "fast enough", how do we solve (1)?

UVA 295 – Fatman – Binary Search

- Is it possible for a circle of size $0 \leq R \leq W$ to pass?
- What is the maximum R that can pass?

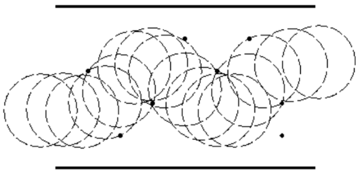
Assuming that we have a “fast” algorithm to test if R can pass ($T(R)$), we could use Binary Search to find the maximum R:

1. Start with $R_l = 0, R_h = W$, Test $T(R_l + R_h/2)$;
2. If fails, $R_h = R_l + R_h/2$, else $R_l = (R_l + R_h)/2$; repeat $T(R_l + R_h/2)$.
3. Repeat until $R_h - R_l < 0.0001$.

This requires $\log_2(100 * 10000) = 20$ operations.

QUIZ: How can we test $T(R)$ “fast enough”?

UVA 295 – Fatman – Squeezing through



- R can pass between two objects i and j if $euclid(i,j) \geq R$
- R can pass between an object i and a wall if $y_i \geq R || y_i \leq W - R$

Algorithm for T(R)

- Create a Graph G where the obstacles and walls are vertices;
- If R can **not** pass between i and j, add an Edge E_{ij};
- If there is a **path** between both walls, R cannot pass;

UVA 295 – Fatman – Squeezing through

T(R) sample code – part 1, construct graph

```
def test(R):
  nb = []                       # list of neighbor list
  for i in range(len(N)+2): nb[i] = list()

  for i in range(len(N)):  # N is list (x,y) of obstacles
    if (N[i][1] < R): nb[0].append(i+1)
    if (W - N[i][1] < R): nb[len(N)+1].append(i+1)
    if (i+1) in nb[0] and (i+1) in nb[len(N)+1]: return 0
       # quick check 1
  if not (len(nb[0]) and len(nb[len(N)+1]): return 1
       # quick check 2

  for i in range(len(N)):
    for j in range(len(N)):
      if dist(N[i],N[j]) < R: nb[i+1].append(j+1)

  ... next we test the graph ...
```

UVA 295 – Fatman – Squeezing through

QUIZ: What is the total cost of this approach?

T(R) sample code – part 2, testing the graph

```
def test(R):
  nb = []                      # list of neighbor list
  for i in range(len(N)+2): nb[i] = list()
  for i in range(len(N)):    ... border test ...
  for i in range(len(N)):
    for j in range(len(N)): ... build graph ...

  curnode = 0; visited = list(); tovisit = list()
  while 1: # DFS
    if (curnode == len(N)+1) return 0    # reached wall
    visited.add(curnode)
    for i in nb[curnode]: tovisit.append(i)
    while(curnode in visited):
       if not (len(tovisit)): return 1   # not reached wall
       curnode = tovisit.pop()
```

UVA 714 – Copying books

Problem Description

- There are M books and K scribes ($1 \leq K \leq M \leq 500$).
- The each book has p_i pages ($1 \leq p_i \leq 1000000$)
- Assign books to each scribe, and **minimize** maximum job.
- Books must be assigned in blocks.

```
9 3
Input 1: 100 200 300 400 500 600 700 800 900
Output 1: 100 200 300 400 500 / 600 700 / 800 900 (max 1700)

5 4
Input 2: 100 100 100 100 100
Output 2: 100 / 100 / 100 / 100 100 (max 200)
```

- QUIZ: Describe the full search (and complexity)
- QUIZ: Describe a better algorithm?

UVA 714 – Copying books – Decomposition approach

```
9 3
Input 1: 100 200 300 400 500 600 700 800 900
Output 1: 100 200 300 400 500 / 600 700 / 800 900 (max 1700)

5 4
Input 2: 100 100 100 100 100
Output 2: 100 / 100 / 100 / 100 100 (max 200)
```

- Someone has probably suggested DP. It is certainly possible.
- We could also use "Binary Search + Test" from the last problem:
  - Binary search the maximum cost (100000*500 = 26 comparisons)
  - Test if the maximum cost is possible (T(max))
  - QUIZ: What is a "fast enough" algorithm for T(max)?

UVA 714 – Copying books – Testing a solution

```
9 3
Input 1: 100 200 300 400 500 600 700 800 900
Output 1: 100 200 300 400 500 / 600 700 / 800 900 (max 1700)
```

One possible Test: Greedy Algorithm to test Maximum M

```
def test(M):
  scribe = 0; book = 0;
  while scribe < K:
    sum = 0
    while sum + page[book] < M:
      sum += page[book]; book += 1
      if book == M: return 1    # assigned all books
    scribe ++
  return 0                      # did not assign all books
```

Caution: This code gives WA – in case of tie, you need lowest jobs first!

Take-home messages

Composite Problems

Many interesting problems use a combination of algorithms:

- Blackbeard: BFS + TSP
- Fatman: Geometry + Graph + Binary Search
- Books: Greedy + Binary Search

Do not forget simple approaches

Binary-search-and-test is **very powerful** if:

- You need to find a bounded maximum or minimum;
- The feasibility test is simple to perform; (code-simple)

UVA 1079 – A careful Approach

Problem Description

- Choose the landing time t_i for $2 \leq N \leq 8$ planes;
- The minimum gap $|t_i - t_j|$ must be as large as possible;
- Each plane i has a maximum and minimum allowed landing time: $0 \leq \min_i \leq t_i \leq \max_i \leq 1440$

```
Input:
3 planes
1- 0 to 10
2- 5 to 15
3- 10 to 15
```

```
Solution:
Maximum Minimum Gap: 7.5 minutes
P1 - Arrive at 0
P2 - Arrive at 7.5
P3 - Arrive at 15
```

Final Quiz: Let's solve this problem
(hint: it is composite of 3 problems!)

This Week's Problems

- Blackbeard the Pirate – UVA 10937
- Cutting Sticks – UVA 10003
- Free Parenthesis – UVA 1238
- ACORN – UVA 1231
- Copying Books – UVA 714
- How big is it? – UVA 10012
- Fatman – UVA 295
- A careful approach – UVA 1079

The End!

I hope you enjoyed the course!
Have a nice summer!