GB21802 - Programming Challenges

Week 2 - Problem Solving Paradigms (Search)

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2019-04-26

Last updated April 23, 2019

Results for the Previous Week

Week 1: Data Structures

Deadline: UCT 2017-04-28T15:00:01 (2 days, 13:43 hours from now)

Problems Solved -- 0P:21, 1P:5, 2P:5, 3P:10, 4P:6, 5P:2, 7P:2, 8P:3,

#	Name	Sol/Sub/Total	My Status
1	Jolly Jumpers	33/34/54	
2	Newspaper	25/30/54	
3	Army Buddies	12/18/54	
4	Grid Successors	11/12/54	
5	Football (aka Soccer)	6/6/54	
6	File Fragmentation	5/5/54	
7	CD	21/22/54	
8	War	4/5/54	

Week 1: Data Structures

Deadline: 2018/5/10 23:59:59 (1 day, 01:39 hour from now)

Problems Solved -- 0P:14, 1P:4, 2P:2, 3P:6, 4P:5, 5P:1, 6P:1, 7P:3, 8P:9,

#	Name	Sol/Sub/Total	My Status
1	Jolly Jumpers	27/30/45	
2	Newspaper	25/29/45	
3	Army Buddies	24/26/45	
4	Grid Successors	17/17/45	
5	Football (aka Soccer)	14/15/45	
6	File Fragmentation	13/13/45	
7	CD	18/21/45	
8	War	12/13/45	

Java Speed Hints

- Don't add strings in a loop:

```
String a,b;
for (int i = 0; i < N; i++) { a = a + b; } // SLOW!
```

- Use *StringBuilder* instead:

```
StringBuilder sb; String a,b;
for (int i = 0; i < N; i++) { sb.append(b); }
a = sb.toString();
```

- Java *Arraylist*'s **contain()** is O(n)[1]

```
ArrayList<Int> a;
for (int i = 0; i < N; i++) { a.contain(b); }
```

- Use *HashMap* instead (O(1)):

```
HashMap<Int> a;
for (int i = 0; i < N; i++) { a.contain(b); }
```

[1] http://stackoverflow.com/questions/10196343/hash-set-and-array-list-performances

Python Speed Hints

Python Speed

Some students are also having problems with PYTHON getting TLE. (Problem CD).

- You can still get accepted on CD with python, if you remember that the data is Ordered!
- In general, solutions in python will require better algorithms for the bigger problems in this course.

Quick Review of Last Week

- Linear Structures (Arrays, Vectors)
  - Use them a lot!
  - Learn the Library: Sorting, Binary Search, Bitmasks
- Tree Structures (Map, Set)
  - Are fast for querying data
  - Use the standard libraries!
- Union-Find Disjoint Set (UFDS)
  - Good for querying sets, not good for iteration.
  - Simple to implement!
- Segment Trees
  - Good for Max/Min Range Query and dynamic data
  - Hard to implement!

Topic for this week: Search

What is Search

In day to day life, we say we are searching for something when we are trying to find where this something is located.

- Keys of your bycicle;
- Your wallet;
- Your cellphone;

When searching, we think of our Goal (the thing we are searching), and the search space (the number of places where the thing could be hidden)

The thing you search is always in the last place you look

(*By definition!*)

What is a "Search Problem"?

We call a problem a search problem, if we can describe the problem as checking multiple *answers* in order to find one or more *solutions*.

- Answers to a search problem could be correct or incorrect;
- Answers can also possibly have higher or lower scores;
- We can sort the answers by score, or by some other criteria;

Many problems in programming challenges can be described as search problems! (even if sometimes there are better ways to describe them)

Sample Search Problems

Traffic Lights (Week 0)

You are given a set of traffic lights with different *period lengths*
Find The moment in time when all trafic lights are in the green period.

Search Space: The point in time from 1 to MCM(traffic lights).

File Fragmentation (Week 2)

You are given a set of binary fragments. Find a binary value that match all existing fragments.

Search Space: The set of all binary values with size $< n$

Thinking with search spaces

Define a problem as a search space $\rightarrow$ Organize the search space $\rightarrow$ Check every solution.

Questions about the algorithm:

- How is the search space represented (What Data Structure)?
- How are the solutions evaluated (Score)?
- In what order are the solutions tested (What Loop/Recur)?
- How many solutions are tested (Time)?

Thinking with search spaces: File Fragmentation

Input

A set of binary fragments:

0011 100 1100 00 00 001

Output

One binary string that fit all gragments:

001100

Thinking with search spaces: File Fragmentation

- How is the search space represented?
  We store every possible binary B of size n
- How are the solutions evaluated?
  The score of B is the number of fragments that match it.
- In what order are the solutions tested?
  Test in Number order: 0001, 0010, 0011, 0100...;
- How many solutions are tested?
  We need to test 2^n solutions.

There are other possible solutions!

Search Paradigms

These are some common approaches for search problems:

- Complete Search/Brute Force;
- Divide and Conquer;
- Greedy Approach;
- Dynamic Programming (Next week!)

Some problems can use multiple approaches (but not all of them are equally good!)

Theoretical Example (1)

Search Space

You have an array A of n integers ($n < 10K$), where the value of each integer a_i is ($0 \leq a_i \leq 100K$).

Imagine the following problems:

1. Find the Largest and the smallest element of A;
2. Find the k^{th} smallest element of A;
3. Find the largest gap G such that $x, y \in A$ and $G = |x - y|$;
4. Find the longest increasing subsequence of A;

Question: What is the complexity of each problem?

Theoretical Example (2)

How costly would be to search the solutions for each of the four problems described?

- Find the Largest and smallest element of A: O(n) - single pass, and we cannot really go faster than this.
- Find the k^{th} smallest elements of A:
  - Repeat the search k times: $O(n^2)$ in the worst case;
  - Order the number and search: $O(n\log n)$
- Fing the largest gap:
  - Try all possible pairs: $O(n^2)$
  - Greedy: Find the smallest and largest numbers $O(n)$ (you have to prove this works)
- Longest increasing subsequence:
  - Test all possible subsequences (brute force): $O(2^n)$
  - Dynamic programming: $O(n^2)$
  - Greedy search: $O(n\log k)$ – can you prove this?

Complete Search/Brute Force (1)

Complete Search algorithms are expected to test all (or almost all) solutions.

Complete Search are usually called "Brute Force". But because they are often the best way to solve a problem, we use a nicer name here.

Complete Search/Brute Force (2)

Structure of a Complete Search:

- Test all existing solutions
  Usually achieved through either for loops or recursive calls;
- Prune, Prune, Prune
  Remove bad solutions (or bad sets of solutions) as you go, by "breaking" early form loops, or setting good ending conditions to the recursive calls.

Complete Search Example: UVA 725 – Division

Problem Summary

Given an integer N, find all pairs of numbers *abcde* and *fghij* so that $fghij / abcde = N$ and all 10 digits are different.

Example: $N = 62$

79546 / 01283 = 62
94736 / 01528 = 62

Consider this problem for a bit before I show how to solve it using search.

Complete Search Example: UVA 725 – Division

Full Search Solution:

A naive way to solve the problem is to test all $0 \leq x \leq 99999$, calculate $y = x * n$, and test whether x and y have all different digits.

```
for (int x = 0; x < 99999; x++)
{
  y = x*n;
  digits = test(x,y);
  if (digits == 1<<10 - 1) printf("%0.5d/%0.5d=%d\n",y,x,N);
}

int digits(int x, int y)
{
  int used = (x < 10000);
  int tmp;
  tmp = x; while (tmp) {used |= 1 << (tmp%10); tmp /= 10; }
  tmp = y; while (tmp) {used |= 1 << (tmp%10); tmp /= 10; }
  return used;
}
```

Complete Search Example: UVA 725 – Division

Pruning the complete loop:

- What is the absolute minimum and maximum for x? 01234:98765
- Maximum for Y is also 98765, so the actual maximum for x is $x < 98766/n$
- Can we cut the digits test earlier?

Considerations about complete search

- A bug-free complete search should ALWAYS be correct.
  - A complete search tests all solutions, so it should always find the correct one;
  - Of course, in many cases, checking all solutions takes too long;
- Complete Search should always be solution considered (KISS principle)
  - If the problem is so small that a better solution is overkill;
  - If you are running out of ideas, or take too many WAs;
  - Prune, prune, prune!
- Sometimes, you can use a simple complete search on a hard problem to get an idea of what sort of result is expected.
  - Use it to generate solutions for test cases in problems that generate TLEs.

Complete Search Example 2: Simple Equations

Problem Summary – UVA 11565

Find x, y, z so that:

- $x + y + z = A$,
- $x * y * z = B$,
- $x^2 + y^2 + z^2 = C$,
- $1 \leq A, B, C \leq 10000$.

We need to test sets of x,y,z, but how do we set the limits for these values?

Example 2: Simple Equations – initial pruning

Consider $x^2 + y^2 + z^2 = C$.
Since $C \leq 10000$, and $x^2, y^2, z^2 \geq 0$, if $y = z = 0$ then the range for x must be $-100, 100$.

Therefore, here is the Complete Search Loop

```
bool sol = false; int x,y,z;
for (x = -100; x <= 100 && !sol; x++)
  for (y = -100; y <= 100 && !sol; y++)
    for (z = -100; z <= 100 && !sol; z++)
      if (y != x && z != x && z != y &&
          x + y + z == A && x * y * z == B && x*x + y*y + z*z =
             if (!sol) printf("%d %d %d\n", x,y,z);
             sol = true;
          }
```

Can you think of other ways to prune the loop?

Example 2: Simple Equations – more pruning

There are many other ways that we can prune the loop:

- We can change the range using the actual input values of A, B, C
- We only need one solution. We can break the loop once we find it.
- We can consider the other two equations, specially equation 2.

This week's problem: "Simple Equations – Extreme!" has a much higher range for A, B, C. You need a lot of pruning to avoid a TLE!

Complete Search: TIPS 1

The biggest issue with "Complete Search" solutions is: Will it pass the time limit?

If you think that your program is borderline passable, it might be worth it finding and optimizing the critical part of the code.

Tip 1 – Filtering Vs Generating

Filter Programs examine all solutions and remove incorrect ones. Generally iteractive. Generally easier to code. Example: Request for proposal.

Generating Programs gradually build solutions and prune invalid partial solutions. Generally recursive. Generally faster. Example: 8 queens.

Complete Search: TIPS 2

Tip 2 – Prune Early

In the N queen problem, if we imagine a recursive solution that places 1 queen per column, we can prune rows, columns and DIAGONALS.

Also remember to mark impossible places when you enter the recursion, and unmark when you leave, using bitmasks.

Tip 3 – Pre-computation

Sometimes it is possible to generate tables of partial solutions.

Load this data in your code to accelerate computation (at the expense of memory). The programming cost is high, since you have to output the tables in a way to facilitate putting it in the code.

Complete Search: TIPS 3

Tip 4 – Solve the problem backwards

Sometimes a less obvious angle of attack may be easier.

Example: UVA 10360, Rat Attack. A 1024 x 1024 city has $n \leq 20000$ rats in some of its blocks. You have a bomb with radius $d \leq 50$. Where do you place the bomb to kill most rats?

Obvious Approach: Check each of the 1024^2 cells. Cost:
$1024^2 * 50^2 = 2621M$ TLE

Backwards Approach: Make a 1024x1024 matrix of "killed rats". For each rat group, add its value to each cell in the bomb radius:
$n * d^2 = 20000 * 2500 = 50M + 1024 * 1024$.

Complete Search: TIPS 4

Tip 5 – Optimizing the source code

- Loops are usually faster than recursion
- Using built-in data types is usually faster than arrays/vectors
- Printf is usually faster than CIN/COUT
- Many other tips

- Don't forget the Time Optimization vs. Programmer Optimization tradoff!

Divide and Conquer

Divide and Conquer (D&C) is a problem-solving paradigm in which a problem is made simpler by 'dividing' it into smaller parts.

- Divide the original problem into sub-problems;
- Find (sub)-solutions for each sub-problems;
- Combine sub-solutions to get a complete solution;

Examples

Quick Sort, Binary Search, etc...

Canonical Divide and Conquer

1. Sort an static array;
2. You want to find item *n*.
3. Test the middle of the array.
4. If *n* is smaller/bigger than the middle, throw away the second/first half.
5. Repeat

Search time: O(log n) plus sorting time if necessary.

Binary Search on Simulation Problems

Simulation problems usually require us to find a value that solves a complex simulation.

Problem Example: Paying the debt

You have to pay V dollars. You pay D dollars per month, in M months. Each month, before paying, your debt increases by i.

If we fix M,I and V, what is the minimal D?

$V = 1000$, $M = 2$, $i = 1.1$, what is the minimum D?

- $D = 500$:
  - $m_1 : V_0 * 1.1 - D = 600, m_2 : v_1 * 1.1 - D = 160$
- $D = 600$:
  - $m_1 : V_0 * 1.1 - D = 500, m_2 : v_1 * 1.1 - D = -50$

Solving the Simulation Problem

Reverse Engineering Approach:

- Find the derivative of the simulation and solve it to zero.
- Start from the end state of the simulation and calculate the correct value.
- Some sort of Dynamic programming.
- Can be hard for complex simulations!

Binary Search Approach:

- Estimate a minimum and maximum possible answer (a, b)
- Suggest the mean value as the solution, and simulate the result;
- If the result is too big, or too small, correct (a, b)
- Repeat.

Bisection Method – Example

$m = 2, v = 1000, i = 0.1, d = 576.19$

After one Month, debt = 1000 x 1.1 - 576.19 = 523.81
After two Months, debt = 523.81 x 1.1 - 576.19 = 0

Bisection method: Choose the range [a..b], (ex: 0.01 1100.00)

Do a binary search for d in this range

a	b	d	simulation: f(d,m,v,i)	action:
0.01	1100.00	550.005	undershoot by 54.9895	increase d
550.005	1100.00	825.0025	overshoot by 522.50525	decrease d
550.005	825.0025	687.50375	overshoot by 233.757875	decrease d
550.005	687.50375	681.754375	overshoot by 89.384187	decrease d
550.005	618.754375	584.379688	overshoot by 17.197344	decrease d
550.005	584.379688	567.192344	undershoot by 18.896078	increase d
567.192344	584.379688	575.786016	undershoot by 0.849366	increase d
...	...	...	a few iterations later ...	...
...	...	576.190476	stop; error is now less than e	answer = 576.19

Total number of iterations is $O(log_2((b-a)/e))$

Bisection Method Principle

- **Principle**: Search on the solution space and test the answer.
- Another example: UVA 11936 - Through the desert
- This technique requires the solution space to be Unimodal

Greedy

Definition

An algorithm is said to be greedy if it makes the locally optimal choice at each step, with the hope of eventually reaching the global optimal.

For greedy to work, a problem must show two properties:

1. It has optimal sub-structures (Optimal solution of the problem contains optimal solutions for the sub-problems)
2. It has the greedy property: Making locally optimal choices will lead "eventually" to the optimal solution (difficult to prove!)

Greedy Example 0 – Minimal Coverage

Consider an interval [A,B], and a set of n intervals $S = [(a_1, b_1), (a_2, b_2), \ldots (a_n, b_n)]$.

Find the minimal subset of S which completely covers [A,B].

- A = 10, B = 50;
- S = [(5,15), (8,12), (40,60), (30,40), (20,40), (13,25), (33,55), (18,30)]

Greed Example 0 – Minimal Coverage

- Consider the subset S_a, of all intervals that cover A.
- If we choose the item from this subset with the **maximum** end value, we have a new point A_{new}. We can discard all other items.
- Repeat the process using A_{new}

Greedy Methods are often found in problems of "find the subset"

Greedy Example 1 – Coin Change

Given a target value V and a list of coin sizes S, what is the minimum number of coins that we must use to represent V?

Example:

$V = 42, \text{Coins} = 25, 10, 5, 1$

Greed Example 2 – Coin Change

- We can solve this case by always taking the biggest coing that fits the remaining cost: 25x1, 10x1, 5x1, 1x2;
- However, if V = 6, Coins = 4, 3, 1, the greedy algorithm will not find an optimal solution.

Be careful that a greedy algorithm can be wrong!

Greedy Example 2 – Load Balancing UVA 410

Problem Description

- There are C chambers, and $S < 2C$ items.
- Each item has a positive weight M_i.
- You need to assign each item to a chamber in order to minimize "imbalance"

$$A = \sum_{i=1}^{S} M_i / S$$

$$\text{Imbalance} = \sum_{i=1}^{C} |X_i - A|$$

Can you figure out a greedy search solution?

Greedy Example 2 – Load Balancing UVA 410

Problem Description

You have C chambers, and S < 2C specimens with different positive weights. You need to decide where each specimen should go to minimize "imbalance".

Insights:

- A chamber with 1 individual is always better than a chamber with 0 individuals.
- Order of chambers does not matter.

Greedy Example 2 – Load Balancing UVA 410

Problem Description

You have C chambers, and S < 2C specimens with different positive weights. You need to decide where each specimen should go to minimize "imbalance".

Greedy algorithm: Order the individuals by weight, and put one in each chambers until the chambers are full, then add one in each chamber backwards.

A similar approach can be used to solve this week's problem "Dragon of LooWater".

Dragon of Loowater

Input

- list of Dragon heads and their sizes
- list of Knights and their sizes
- a knight can only defeat a head if he is bigger

Output

- What is the minimum cost to defeat the dragon? (knight size)
- Or is it impossible?

Stern-Brocot Number

Input

- Two numbers that make a fraction (5 / 7)

Output

- Search a "fraction tree" and print the path.
- Implicit Data Structure!
- Hint: Try to do it by hand a few times.

Bars

- Knapsack problem

- Tip: You need to prune to make it in time!
- Tip: The bar size could be zero!

Rat Attack

- A 1024 x 1024 matrix has rats in some cells;
- A bomb with power d kills rats in a 2d+1 x 2d+1 matrix;
- Where is the best place to put the bomb?

- We need to check all squares in the city, including the borders.
- But, if we calculate the rats all the time, this becomes very slow. (1024 * 1024 * 100 * 100)
- It is possible to **Pre-process** the number of rats killed in each bomb position. (1024 * 1024 + 100 * 100 * 20000)

Simple Equations

- You are given a set of equations:
  $x + y + z = A, xyz = B, x^2 + y^2 + z^2 = C$
- Given A, B, C, what are the values of x, y, z?

- You need to define limits for x, y and z, how do you do that?
- The equation $x^2 + y^2 + z^2 = C$ might be a good place to start.

Through the Desert

- Simulate a car going through the desert;
- Find the least amount of starting fuel needed to win.

- You could try to calculate the amount of fuel needed in each section of the trip (between gas stations);
- Or you could just simulate the trip, and make a binary search based on the amount of fuel left;

Zones

Input

- A total number of towers and a desired number of towers.
- Each tower has a number of people
- Some people belong to two or three towers!

Output

- Select the towers that maximize the number of people!

Little Bishops

Like 8 queens, but with bishops!

Search Algorithms in CS Research

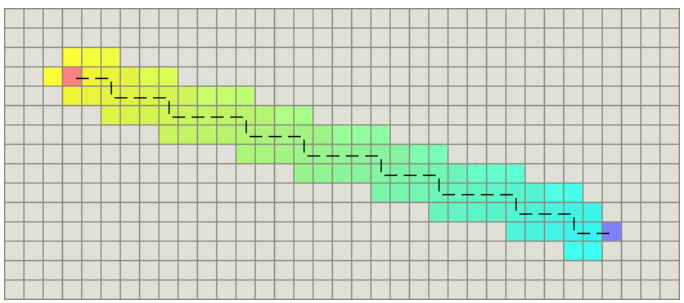
Complete Search and Greedy algorithms feel like something that you only use in your first year of Computer Science, and then never touch again..

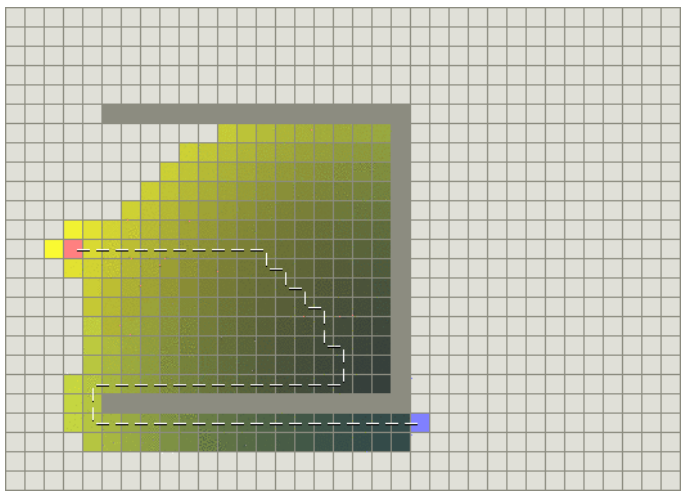
... it turns out however, that search algorithms have a really important role: Heuristics and NP-hard problems.

Heuristic Example: A* Search

A* is a search algorithm for finding a shortest path between two (x/y) coordinates.

1. List all vertices that you can check next;
2. Sort the vertices by sum(distance from start + distance from goal)
3. Explore vertice highest in the sort;





Images from Amit Patel

Heuristic and NP-hard problems

Consider NP-hard problems:

- There are no polinomial algorithms that solve the problem;
- However, a solution to the problem can be tested in polinomial time;

One approach to NP-hard problems is to treat them as search problems, systematically testing solutions (in polynomial time), until an acceptable solution is found.

Heuristic Algorithms

- Hill Climbing
- Evolutionary Algorithms
- Swarm Algorithms
- Etc...

Next Week

Dynamic Programming!