GB21802 - Programming Challenges

Week 2 - Problem Solving Paradigms (Search)

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2015-05-6,9

Last updated May 6, 2016

Last Week Results

Week 1

Problems Solved

- Division Of Nlogonia: 28/28
- Cancer Or Scorpio: 24/27
- 3n+1: 25/25
- Request for Proposal: 19/22

Manaba Submission: 29 Students

Week 2

Problems Solved

- Jolly Jumpers: 26/26
- Army Buddies: 15/21
- Rotated Square: 17/17
- File Fragmentation: 4/5
- Contest Scoreboard: 10/11
- Multitasking: 3/5
- Jolibee Tournament: 7/7

Manaba Submission: 25 students

Don't Give Up!

Special Notes

If the UVA Judge is offline

Sometimes the UVA Judge gets overloaded. If this happens near the deadline, send your programs to MANABA before the UVA deadline. Re-send your programs to UVA as soon as it becomes available again.

If uDebug result is different from UVA result

uDebug is a volunteer service. Although it is very useful, its correctness is not guaranteed. If you see a result that indicates that uDebug is wrong, contact the maintainers of that site, they will be grateful for the report.

Always try to submit programs early!

Quick Review of Last Week

- Linear Arrays (unidimensional and bidimensional) are KISS data structures;
  - A lot of problems can be solved by iterating on arrays
  - Always allocate a bit more than you need.
- Bitmasks are quick and efficient ways to store sets of True/False values;
- Tree data structures can search very efficiently
  - Instead of implementing your own, learn to use the structures available from the library.

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

Request For Proposal (Week 1)

You are given a set of proposals with the number of satisfied requirements, and the total cost. Find the proposal with the highest number of requirements, and the lowest total cost.

Search Space: The set of proposals in the input.

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 $< n$

Now you are thinking with search spaces

When we define a problem as a search problem, we can use strategies based on organizing the search space, and systematically examining each solution contained in it.

Questions about the algorithm:

- How is the search space represented?
- How are the solutions ordered?
- In what order are the solutions tested?
- How many solutions are tested?

Thinking with search spaces: Request for Proposal

- How is the search space represented?
  For each proposal, the number of requirements and the cost are stored.
- How are the solutions ordered?
  The array of proposals is ordered by highest number of requirements and lowest cost.
- In what order are the solutions tested?
  Highest item in the array of proposals are checked first.
- How many solutions are tested?
  Only the first item is returned.

Of course, for many problems there is more than one way to fill this pattern.

Search Paradigms

These are some common approaches for search problems:

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

As we said before, 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:

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

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

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 $x^2 + y^2 + x^2 = C$. Since $C \leq 10000$, the maximum range for x, y, z 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 open ended problems – Bisection

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?

Bisection approach:

- Estimate a good min,max interval for D.
- Select the middle value and simulate the result.
- If the result is bigger/smaller than the desired, modify the interval accordingly.
- 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 $O(log_2((b-a)/e))$

UVA - 11936 - Through the desert - uses this logic

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 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:
$V = 42, Coins = 25, 10, 5, 1$
(a One coin means we can always make any value)

However, if V = 6, and coins = 4,3,1, the greedy algorithm does not reach an optimal 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".

$$\text{Imbalance} = \sum_{i=1}^{C} |CM_i - AM|$$

Can you figure out a greedy search solution?

Greedy Example 2 – Load Balancing UVA 410

Insights:

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

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.

Greedy Example 3 - Dragon of LooWater (11292)

List of knights (with heights) and dragons (with diameter). Find the minimal height of knights that cut the heads of all dragons (knights can only cut the heads of dragons when D <= H).

Problem Discussion

- Division
- Social Constraints
- Simple Equations - EXTREME!!
- Bars
- Rat Attack
- Little Bishops
- Water Gate Management
- Through the Desert
- Dragon of Loowater
- Shoemaker's Problem

Search Algorithms are an important area of research

Heuristics