GB20602 - Programming Challenges

Week 3 - Search Algorithms

Claus Aranha
caranha@cs.tsukuba.ac.jp

University of Tsukuba, Department of Computer Sciences

(last updated: May 4, 2022)

Version 2022.1

Part I: Introduction

Outline

This week, we study "Search Based" approaches to programming challenges:

- Complete Search (Brute Force)
- Divide and Conquer
- Greedy Search

What is Search?

In daily life, we **search** for things all the time:

- Where are the keys of your bycicle;
- Where is your wallet;
- Where is your cellphone;

Every search has:

- An **Objective**: what are we searching;
- A **Search Space**: where we are searching;

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

What is a “Search Algorithm”?

The search algorithm **list all possibilities** (using a loop or recursion), and **test them one by one** until the correct solution is found.

- The correct solution can be unique or not;
- Sometimes, we can calculate the **distance** to the correct solution;
- Sometimes, we can list the possibilities **in different orders**;

Many problems (in programming challenge and real life) can be described as search problems. Search can be slow, but we always find a solution (someday).

The Generic Search Algorithm

This is the general idea of any **Search Algorithm**:

1. Sort the possibilities, and choose the first one: $i = 1$;
2. Test if solution a_i is correct;
3. If a_i is not correct, $i = i + 1$, and go back to 2.

How can we make the search algorithm better?

- What is the size of the search space? (space of possibilities)
- What is the best order for the seach space?
- Can we reduce the search space?

Search Algorithm Example "Black Friday"

- You have n numbers between 1 and 6 ($1 \leq n \leq 100$)
- What is the highest number that is not repeated?
- Example: $A = \{3, 5, 4, 5, 6, 1, 2, 3, 1, 6\}$, Solution: 4

Solution 1 ($O(N + 6)$)

- Create count array C (from 1 to 6)
- Loop array A, and count the numbers, adding their value to C
- Loop array C, and print highest i so that $C_i == 1$

Solution 2 ($O(N \log N + N)$)

- Sort Array A;
- Loop Array A, if $A_i \neq A_{i+1} \neq A_{i+2}$, that is the solution

Looping, Sorting and Searching an Array for many problems

Consider the following input array:

Array A with n **unsorted** random integers. $n < 10.000$, $0 \leq a_i \leq 100.000$

For each problem, think **how do you loop the array** and **how much time it takes**

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; $G = |a_i - a_j|$;
4. Find the **longest increasing subsequence (LIS)** of A;

Example of LIS: $A = [3, 2, 5, \overline{1}, 4, \overline{2, 3, 5}, 7, \overline{6, 10}]$

Looping, Sorting and Searching an Array for many problems

1. **Find the Largest and smallest element of A:**
   - Loop through A one time, keep track of a_{max}, a_{min}, cost: $O(n)$.
2. **Find the k^{th} smallest elements of A:**
   - Loop through A, k times, remove smallest each time. Cost: $O(nk)$, worst case: $O(n^2)$
   - Sort A, return $A[k]$. Cost: $O(n \log n)$
3. **Find the largest gap:**
   - Try all possible pairs: Loop i, Loop j, keep max $|a_i - a_j|$. Cost: $O(n^2)$
   - Think: Largest gap = biggest - smallest. Find the smallest and largest: $O(n)$.
4. **Longest increasing subsequence:**
   - Test all possible subsequences: $O(2^n)$ – We will make this fast next week.

Part II: Complete Search

Idea of Complete Search

In the Complete Search algorithm, we loop through all possibilities, until we find the correct answer.

Some people call this algorithm **Brute Force**. This name is negative, but sometimes a Complete Search is very useful.

How to think of a complete search algorithm

- Find the loop that checks all possibilities; (or a recursive function)
- Write the program to check if a possibility is correct or not;
- **PRUNING**: Remove from the loop cases that are impossible;
  - Use "break", "continue" or good loop limits.

Complete Search Example: Division

Problem Summary

Input: One integer N.
Output: All pairs of numbers with 5 digits (*abcde* and *fghij*) that satisfy:

1. $fghij / abcde = N$
2. $a, b, c, d, e, f, g, h, i, j$ are all different.

Example: $N = 62$

79546 / 01283 = 62
94736 / 01528 = 62

Think: How can you solve this problem using **Complete Search**?

Complete Search Example: Division

Naive Solution

Let abcde = x, fghij = y. Loop all x from 0 to 99999, calculate $y = x * n$.
Test if all digits of x and y are different.

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

int count_digits(int x, int y) {
  int used = (x < 10000); % bit array: each bit mark one digit
  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: Division

Prunning the Loop

The naive algorithm tests many numbers that **will never be the answer**. (eg: x = 00021).

We can **prune** the loop, to test a smaller number of cases.

- Prune 1: Minimum and Maximum values of x:
  x_{min} : 01234, x_{max} : 98765. Other values have repeated digits!
- Prune 2: Consider $y = x * N$. The maximum of y is also 98765, so maximum of x is:
  x_{max} : $98765/N$
- Can you think of other ways to prune?

Notes about Complete Search algorithms

- A Complete Search should always give the correct solution
  - It checks all cases, so it should always find the correct one;
- For some problems, the Complete Search is the right algorithm.
  - If the problem is small, use Complete search, no need for complex algorithm;
  - Prune, prune, prune to make the problem small;
- For a very hard problem, a Complete Search can give you a starting point;
  - Even if Complete Search is "Time Limited Exceeded", you can use it to check test cases;

Complete Search Example 2: Simple Equations

Problem Summary

Input: A, B, C, $1 \leq A, B, C \leq 10000$.

Find x, y, z so that:

- $x + y + z = A$,
- $x * y * z = B$,
- $x^2 + y^2 + z^2 = C$,

To solve this problem we can loop and test on of x, y, z (3-nested loop).

But what should be the minimum and maximum value of the loops?

Complete Search 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 $-100, 100$.

```
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 == C) {
             printf("%d %d %d\n", x,y,z);
             exit(0);
          }
```

QUESTION: How can we prune this loop even more?

Complete Search 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 use equation 2 to think of other ways to prune.

Complete Search Example 3: Calculating Dart Scores

Problem Summary

- Given a score S, $1 \leq S \leq 180$, find three dart scores that add to S.
- One dart score can be 1 to 20, "double" and "triple"

- For this program, maybe it is easier to use recursion than loop.
- Function: "findscore(S, darts)". Print the score when you return.
- How to prune this search?

Part III: Binary Search

Divide and Conquer Idea

The Bynary Search algorithm is based on the idea of **Divide and Conquer** (D&C).

D&C means to make a problem simple by dividing it in smaller parts. It is easier to solve the smaller parts than the big one.

Examples of algorithms using D&C approach

Binary Search, Quick Sort, Merge Sort, many algorithms on Tree Graphs, etc...

Binary Search Algorithm

You want to find number k in a very large array A of size n:

1. Sort A; Set begin = 0; End = n
2. Test $a_{(end-begin)/2}$ (The middle of begin and end)
3. If $a_{(end-begin)/2} > k$, $end = end - begin/2$
4. If $a_{(end-begin)/2} < k$, $begin = end - begin/2$
5. Go to 2 until you find k

The search time is $O(n \log n)$ (sorting) + $O(\log n)$ for each search.

If the number of searches is (very) small, it is better to just loop the array.

If the number of searches is big, or if the array is sorted in the beginning, it is better to use binary search.

Binary Search Problem: Room Painting

Problem Description

- Input: $n < 100.000$ can sizes and $m < 100.000$ paint sizes, in random order.
- For each paint size m_i, find the smallest n_j so that $n_j > m_i$.

Full Search Solution: Loop m times around n – simple to code, but total cost is $O(m * n) = 10.000.000.000$.

Binary Search Solution: Sort n, then do binary search m times. Total cost is: $O(n \log n) + O(m \log n) \approx 200.000 * 11 \approx 2.000.000$.

Remember you can use the "upper_bound" function in C++. No need to code by hand.

Binary Search for Simulation

You can also use Binary Search to calculate complicated calculations.

- Consider the complicated function $f(x)$;
- You want to find x' so that $f(x') = 0$;
- Estimate x_{min} and x_{max};
- Do a **Binary Search** between x_{min} and x_{max}, using $f(x)$ to test.

Note: This approach only works if the calculation is **monotonic**.

Example 1: Paying a Debt

Problem Description

- Pay a total of V. You pay D per month, for M months.
- Each month V increases before paying, $V = V * i$.
- Given M,i and V, what is the minimum D?

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

- if $D = 500$:
  - $m_1 : V_0 * 1.1 - D = 600, m_2 : V_1 * 1.1 - D = 160$: D is too small
- if $D = 600$:
  - $m_1 : V_0 * 1.1 - D = 500, m_2 : V_1 * 1.1 - D = -50$: D is too big

Example 1: Paying a Debt with Binary Search

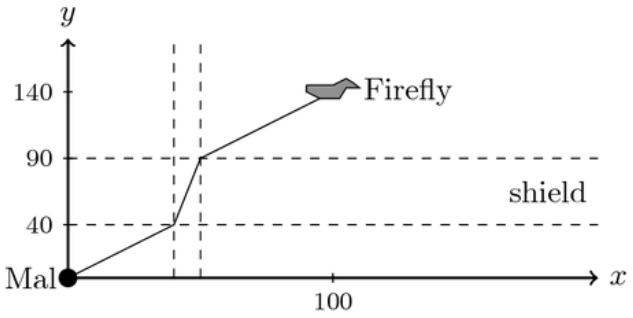
- Input: $m = 2, v = 1000, i = 1.1$
- Choose initial range: (ex: [0.01 to 1100.00]); Initial d: 550.005
- $f(d, m, v, i) = \text{loop}(v * i - d)$, m times
- Do binary search in this range;

a	b	d	simulation: f(d,m,v,i)	action:
0.01	1100.00	550.005	error: -54.98	increase a
550.005	1100.00	825.002	error: 522.50	decrease b
550.005	825.002	687.503	error: 233.75	decrease b
550.005	687.503	681.754	error: 89.38	decrease b
550.005	618.754	584.379	error: 17.19	decrease b
550.005	584.379	567.192	error: -18.89	increase a
567.192	584.379	575.786	error: -0.84	increase a
...	...	...	a few iterations later ...	...
...	...	576.190	error $< \epsilon$	stop: answer = 576.19

Total number of steps: $O(log_2((b-a)/\epsilon))$ **Attention: Choose a stop criteria carefully!**

Binary Search in a Simulation, Example 2: carefulascent

- Given the position of the ship (x, y), and a fixed vertical speed ($v = 1$),
- Find horizontal speed (h)
- There are n shields that change the horizontal speed.
- Acceptable error: $10^6 = 0.000001$; calculate with more digits than this.



Part VI: Greedy Search

The Greedy Search Algorithm

Definition: The Greedy Algorithm makes a **locally optimal choice** at each step.

Common implementation:

- The solution in broken in "steps";
- Sort the input, so the "best step" is first;
- Remove the first step from the input, add to the output;
- Repeat.

Be Careful

- If the "best step" is wrong, you will get **wrong answer**;
- Some problems CANNOT be solved by greedy method;
- Calculate your "best step" carefully on paper; **Check for trick cases**.

Example Greedy: ICPC Team Selection

- You have $3 * n$ students to separate in n teams of 3 students.
- Each student has a skill between 1 and 100. Each team has score = median score.
- Choose teams to maximize total score.

- **Example**: Student scores $\{4, 8, 6, 8, 10, 9\}$
- Team selection example 1: $\{6, 8, 10\}, \{4, 8, 9\}$: Total 16
- Team selection example 2: $\{6, 9, 10\}, \{4, 8, 8\}$: Total 17

Example Greedy: ICPC Team Selection

Greedy Algorithm:

- Separate problem into steps;
- Select the best step from the input;
- Repeat

For this problem, one step is "select a team fron the input".

We can think of many ways to select the "best" team. Which one is correct?

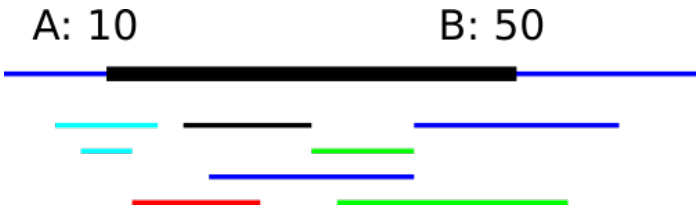
- Select the best 3 students
- Select 2 best, 1 worst student
- Select 1 best, 2 worst student
- Select Best, median, worst student
- Other?

Test it on pen and paper!

Traditional Greedy Search: Minimal Coverage

- **Input**: integers A, B, and a set of n intervals $S = [(a_1, b_1), (a_2, b_2), \ldots (a_n, b_n)]$.
- **Output** Minimal subset of S that 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)]



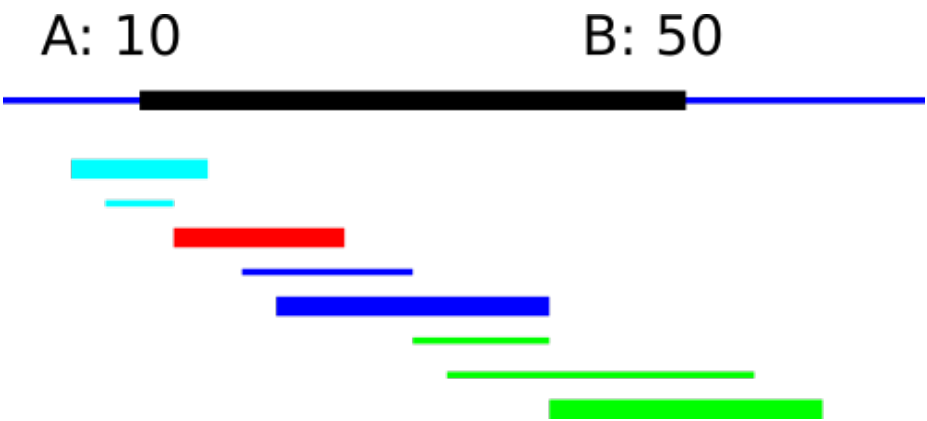
Traditional Greedy Search: Minimal Coverage

- Greedy Step: Choose an interval that covers position C
- Initial Step: $C = A$
- "Best" Step:
  - Sort S by left value; Select subset S_c where $C \in s_i$
  - Choose s_i with largest right value.
- C becomes the right value of the interval;

It is common to use Greedy Algorithms in problems of the type "Find the best Subset";

Traditional Greedy Search: Minimal Coverage

Solution: Select the thick lines



Bad Greedy Example: Coin Change

Given a target value V and a set of coin sizes S, select a group C of coins (with repetition) that adds to V, so that the size of C is minimum.

Example Input: $V = 42, S = \{25, 10, 5, 1\}$

Example Output: $25 + 10 + 5 + 1 + 1$: Size 5

Greedy Algorithm

- Greedy Step: Add one coin to C
- "Best" Step: Add the biggest possible coin

Bad Greedy Example: Coin Change

Wrong Answer!

Sample Data – Greedy works here.

- $V = 42, S = \{25, 10, 5, 1\}, C = \emptyset$, Take 25
- $V = 17, S = \{25, 10, 5, 1\}, C = \{25\}$, Take 10
- $V = 7, S = \{25, 10, 5, 1\}, C = \{25, 10\}$, Take 5
- $V = 2, S = \{25, 10, 5, 1\}, C = \{25, 10, 5\}$, Take 1
- $V = 1, S = \{25, 10, 5, 1\}, C = \{25, 10, 5, 1\}$, Take 1

Trick Case! $\{3, 3\} < \{4, 1, 1\}$ – There is no "best step" rule for this problem

- $V = 6, S = \{4, 3, 1\}, C = \emptyset$, take 4
- $V = 2, S = \{4, 3, 1\}, C = \{4\}$, take 1
- $V = 1, S = \{4, 3, 1\}, C = \{4, 1\}$, take 1

About these Slides

Individual images in some slides might have been made by other authors. Please see the following pages for details.

Image Credits I

[Page 27] Careful Ascent Image CC-BY-SA, from Egor Dranischnikow