Topic 7: Binary Search and Meet in the Middle

Competitive Programming I (Fall 2019)

Ninghui Li (Purdue University)

LeetCode: Find the First and Last Position of Element in a Sorted Array (Required)

Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value. Your algorithm's runtime complexity must be in the order of *O*(log *n*). If the target is not found in the array, return [-1, -1].

```
Input: nums = [5,7,7,8,8,10], target = 8    Output: [3,4]

Input: nums = [5,7,7,8,8,10], target = 6    Output: [-1,-1]
```

Find First and Last (C++ Code, no library) 1/2

```
vector<int> searchRange(vector<int>& nums, int target) {
  vector<int> ans(2,-1);
  int b = 0, e = nums.size()-1, m;
  if (e == -1 || nums[b]>target || nums[e]<target) return ans;
  // invariance: nums[b-1]<target && nums[e]>=target
  while (b < e) {
    int m = b + (e-b)/2;  // (b+e)/2 may overflow in rare cases
    if (nums[m] < target)  { b = m+1; }
    else if (nums[m]==target && m==b) {
        b = m;  break;   // found m
    } else  { e = m; }
  }
  if (nums[b] != target)  return ans;
```

Performance optimization

Necessary for invariance

Condition is complicated because we want first appearance.

Find First and Last (C++ Code, no library) 2/2

```
e = nums.size() - 1;
ans[0] = b;  ans[1] = e;
if (nums[e] == target)  return ans;
// invariance: nums[b]==target & nums[e]>target
while (b < e-1) {
  int m = b + (e-b)/2;
  if (nums[m] > target)  { e = m; }
  else { b = m;  }
}
ans[1] = b;
return ans;
```

Suggestion for Writing Binary Search

- Have a clearly-specified invariance.
  - Ensure that it is true at the beginning of the iteration.
  - Ensure that any update within the loop maintains the invariance.
- Ensure that the loop makes progress.
  - Depending on the loop condition and update, it is possible to have an infinite loop.

Find First and Last (C++ Code, using library)

```
vector<int> searchRange(vector<int>& nums, int target) {
  vector<int> ans(2,-1);
  if (nums.size() == 0)     return ans;
  vector<int>::iterator low,up;
  // Returns an iterator pointing to the first element that does not compare less than target
  low = lower_bound (nums.begin(), nums.end(), target);
  if (low == nums.end() || *low != target)     return ans;
  // Returns an iterator pointing to the first element which compares greater than val
  up = upper_bound (low, nums.end(), target);
  ans[0] = low - nums.begin();
  ans[1] = up - nums.begin() - 1;
  return ans;
}
```

Can we solve this in Java using binarySearch?

Find First and Last (What if Java is Used?)

- Java has Arrays.binarySearch(...[] array, …)
  - If array is not sorted, the result is undefined.
  - Returns index of the search key, if it is contained in the array within the specified range; otherwise, (-(*insertion point*) - 1).
  - The *insertion point* is defined as the point at which the key would be inserted into the array: the index of the first element in the range greater than the key, or toIndex if all elements in the range are less than the specified key.
  - If the array contains multiple elements with the specified value, there is no guarantee which one will be found.
    - Cannot solve this problem.

LeetCode: Find Peak Element

A peak element is an element that is greater than its neighbors. Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index. The array may contain multiple peaks; return any one is fine.

```
Input: nums = [1,2,3,1]          Output: 2
Explanation: 3 is a peak element with index 2.

Input: nums = [3,2,1,3,5,6,4]    Output: 0 or 5
```

Find Peak Element Analysis

- A linear scan with complexity O(N) works
- How to speed it up?
- Take a look at the middle element, if it is greater than both neighbors, then it is a peak.
- What do you know about peaks if it is less than its right neighbor?
- There must exist a peak to the right of the middle element!
    - Proof. Either the right neighbor is a peak; or it is less than its right neighbor. If the sequence keeps increasing, the rightmost element is a peak.
- Can be solved by binary search.

Cipele: COCI 2018 October Task 3

Use https://oj.uz/problem/view/COCI18_cipele to submit

Given N left shoes and M right shoes of various sizes, we want to pair as many shoes as possible (a new pair cannot be selected after pairing all the shoes).

Each pair should consist of one left shoe and one right shoe.

Compute the minimal ugliness between all possible shoe pairings, defined to be the maximal absolute difference of the shoe sizes between all pairs of shoes.

(1 ≤ N, M ≤ 100,000) Shoe sizes in the range of 1 .. 10^9

Example Input:

```
3 5
39 42 56
20 30 40 50 90
```

Example Output:

```
9
```

Cipele: Analysis

- Sorting both arrays at the beginning.
- Unclear how to directly compute the minimal ugliness
- Given an ugliness threshold, we can efficiently check whether the threshold is achievable.
  - Assume, wlog, that there are fewer left shoes.
  - For each left shoe, find the smallest unused right show that is within the threshold while leaving sufficient number of right shoes.
  - If such a right show cannot be found, the threshold cannot be achieved.

Lesson on Binary Search

- Understand the binary search API of your language.
- Know how to write your own when you need to do so.
- Binary search is one way to solve optimization problem when one can efficiently check whether an optimization goal is achievable or not.
  - $\log_2 2^{30} == 30$

LeetCode: 3 Sum Closest (Required)

Given an array nums of *n* integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
Input: nums = [-1, 2, 1, -4],   target = 1     Output: 2

Closest 3sum is 2 = -1 + 2 + -1
```

3Sum Closest Analysis

- Bruteforce takes time O(N^3)
- O(N^2) solution exist. First sort the array. Then loop over all possibilities for the first of the three elements, then solve 2Sum closest in linear time.
- How to solve 2Sum closest in linear time when sorted?
- Use two pointers
  - If A[b] + A[e] < target, b++
  - If A[b] + A[e] > target, e--

LeetCode: 4Sum II (Required)

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero. All A, B, C, D have same length of N where $0 \leq N \leq 500$. All integers are in the range of -2^{28} to $2^{28} - 1$.

```
Input: A = [ 1, 2]    B = [-2,-1]    C = [-1, 2]    D = [ 0, 2]
Output: 2
The two tuples are:
1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0
2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0
```

4Sum II Analysis

- Bruteforce takes time O(N^4)
- How to solve this in O(N^3)?
- Sort two of the arrays. Loop over all (N^2) combinations of the other two arrays, use two pointers over the two sorted arrays.
- Can we solve this in O(N^2)?
- Yes, by using O(N^2) extra memory to store all possible sums from two arrays. (This is meet-in-the-middle.)
- What is the complexity for 5 SUM, 6 SUM?

4SUM II (Code in Java)

```
public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {
  HashMap<Integer,Integer> ab = new HashMap<Integer,Integer>();
  for (int x : A) for (int y : B)
    ab.merge(x+y, 1, Integer::sum);
  int ans = 0;
  for (int x : C) for (int y : D)
    ans += ab.getOrDefault(-(x+y), 0);
  return ans;
}
```

Meet in the Middle: Another Example

Given a set of n integers where n <= 40. Each of them is at most 10^{12}, find the maximum subset sum less than or equal S where S <= 10^{18}.

```
Input  : set[] = {38, 34, 18, 12, 6, 5} and S = 42
Output : 41, which is sum of {18, 12, 6, 5}
```

Bruteforce takes time 2^{40}, which is not fast enough. If the n integers are small, one can use dynamic programming, but here numbers are too large.

Maximum Subset Sum: Analysis

- Split the set of integers into 2 subsets say A and B. A having first n/2 integers and B having rest.
- Find all possible subset sums of integers in set A and store in an array X. Sort it.
- Using $2^{n/2}$ memory to speed up searching.
- Enumerate all subsets of B and then find the element in X that achieves maximum sub below S.
- To enumerate all subsets of an n-element set, one way is to enumerate from integers from 0 to 2^{n-1} and interpret each bit as whether an element is included or not.

USACO Cow Routing II (Required)

USACO 2015 January Contest, Bronze 2

Given up to 10,000 cities, and N routes (1 <= N <= 500), each of which goes through up to 500 cities in sequence, and has a specific cost (which one needs to pay if using any portion of the route). Compute the minimal cost of a itinerary going from city A to city B if one can use at most two routes. Output -1 if there is no itinerary using no more than two routes.

```
R1: cost 3, goes through 3 2 1
R2: cost 4, goes through 2 1 4 3
R3: cost 8, goes through 4 1 7 8 2
Optimal cost from 1 to 2 is 7, using R2 to go to 3, then R1
```

USACO Cow Routing II Analysis

- For cases of using only one route, easy to compute.
- For two routes, think about the city in the middle. We can compute and store lowest cost from A to each city, and lowest cost from each city to B.
- Let M be number of cities, using storage O(M), we get complexity O(500N+M)

USACO: It's All About the Base (Required)

One number N, when expressed in two bases X and Y, $10 \le X,Y \le 15{,}000$, are 3 digit numbers where each digit is between 0 and 9. Given the two 3-digit numbers, find X and Y.

```
Input: 419 792    Output: 47 35

The number 8892, written in base 47, is 419.  Written in base 35,
it is 792.
```

USACO: It’s All About the Base (Analysis)

- Compute and store in S all possible numbers for the first 3-digit number and their bases.
- Then compute all possible numbers for the second 3-digit number and check whether it is in S.

Path of Least Resistance (ECNA 2014. Solved by 6 teams)

Given a N-by-M grid (1 ≤ N,M ≤ 100), where each cell specifies a direction and number of steps. Find how to change one cell so that the path from top-left to bottom-right corner is minimal.

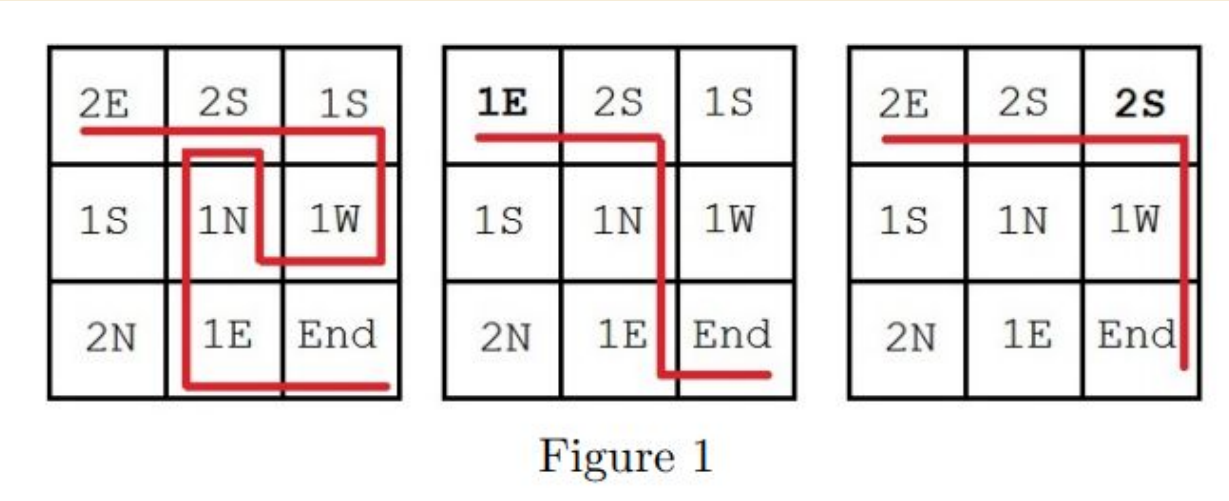

Figure 1

Path of Least Resistance: Analysis

- Uses one BFS to find minimal number of steps to reach each cell from top-left corner.
- Uses another BFS to find minimal number of steps from each cell to reach bottom-right corner.
- For each cell, check whether going to another cell in the same row or column would lead to shorter total distance.