

CSC3100 Tutorial 2

Xingjian Wang 117010266@link.cuhk.edu.cn

Build array from permutation

- Consider an array 'arr' of size n. The array contains a permutation of numbers from 0 to n - 1. We wish to build a new array 'ans' such that 'ans[i] = arr[arr[i]]'
- e.g. arr = [0, 3, 4, 2, 1], ans = [0, 2, 1, 4, 3]
- Given that $n < 2^{15} - 1$ and 'arr' is an integer(32-bits) array. Derive an algorithm that computes 'ans' in $O(n)$ time and $O(1)$ space

Build array from permutation

- Common method:
Using function like 'swap(int []arr, int pos1, int pos2)' to swap the values of different indices until reaching the resulting array.


Build array from permutation

- Common method:
Using function like 'swap(int []arr, int pos1, int pos2)' to swap the values of different indices until reaching the resulting array.
- However, this requires an intermediate array and takes up $O(n)$ space, which is not acceptable.


Build array from permutation

- Common method:
Using function like 'swap(int []arr, int pos1, int pos2)' to swap the values of different indices until reaching the resulting array.
- However, this requires an intermediate array and takes up $O(n)$ space, which is not acceptable.
- Luckily, the extra $O(n)$ space is already allocated for us.

Build array from permutation

 16 *bits*

$$(65535)_{10} = (1111111111111111)_2$$

 32 *bits*

$$(4294967295)_{10} = (11111111111111111111111111111111)_2$$

Build array from permutation

- We can store 2 16-bits number in an 32-bits number
- If we store 8 and 4 in an 32-bits integer 'i', then the hexadecimal representation of 'i' should be looking like 'i = 0x00080004'.
- The first 16-bits are used to represent 8 and the second 16-bits are used to represent 4.
- We can retrieve the first half by doing 'i >> 16' and the second half by 'i & 0x0000FFFF'

Build array from permutation

```
public class helloworld {  
    public static void main(String[] args) {  
  
        int arrLength = 5;  
        int []arr = {0, 3, 4, 2, 1};  
  
        for (int i = 0; i < arrLength; i++) {  
            arr[i] = (arr[arr[i]] << 16) | arr[i];  
        }  
  
        for (int i = 0; i < arrLength; i++) {  
            arr[i] = arr[i] >> 16;  
        }  
  
        for (int i = 0; i < arrLength; i++) {  
            System.out.println(arr[i]);  
        }  
    }  
}
```


2 Sum

- Given an array of integers 'nums' and an integer 'target', return indices of the two numbers such that they add up to 'target'. You may assume that each input would have exactly one solution and you may not use the same number twice.
- e.g. 'nums = [2, 7, 11, 15]', 'target = 9', 'Output = [0, 1]'

2 Sum

- Common solution:
Double for loop. Complexity is $O(n^2)$

```
public class helloworld {  
  
    public static void main(String[] args) {  
        int arrLength = 4;  
  
        int[] a = {2, 7, 11, 15};  
        int target = 9;  
        int[] res = new int[2];  
        boolean breakFlag = false;  
  
        for (int i = 0; i < arrLength; i++) {  
            for (int j = 0; j < i; j++) {  
                int num = a[i] + a[j];  
                if (num == target) {  
                    breakFlag = true;  
                    res[0] = j;  
                    res[1] = i;  
                }  
            }  
            if (breakFlag)  
                break;  
        }  
  
        System.out.println(res[0] + " " + res[1]);  
    }  
}
```

2 Sum

- Better solution:
Hash map. Complexity is $O(n \log n)$

```
public static void main(String[] args) {
    int arrLength = 4;

    int[] a = {2, 7, 11, 15};
    Map<Integer, Integer> myMap = new HashMap<Integer, Integer> ();
    for (int i = 0; i < arrLength; i++) {
        myMap.put(a[i], i);
    }

    int target = 9;
    int[] res = new int[2];
    boolean breakFlag = false;

    for (int i = 0; i < arrLength; i++) {
        Integer diff = target - a[i];
        Integer val = myMap.get(diff);
        if (val != null) {
            if (val.equals(diff)) {
                continue;
            } else {
                res[0] = val;
                res[1] = i;
            }
        }
    }

    System.out.println(res[0] + " " + res[1]);
}
```

2 Sum

- Solution pattern:
 1. Construct a data structure from the original array
 2. Iterate through the array 'nums'
 3. For each 'nums[i]', check if 'target - nums[i]' is in the array(or the constructed data structure)
 4. If it is in the array, we have found what the question wants. If no, continue the iteration

2 Sum

- Array operation complexity:
Constructor: $O(n)$
Array.get(): $O(1)$
Array.containsValue(int value): $O(n)$
- Hashmap operation complexity (based on red-black tree):
Constructor: $O(n \log n)$
Hashmap.get(): $O(\log n)$
Hashmap.containsKey(int key): $O(\log n)$

References

- <https://leetcode.com/problems/two-sum/> (2 sum)
- <https://leetcode.com/problems/build-array-from-permutation/> (permutation)