Some Tricky Problems

- [x] ~~731~~
  - 错了一次，一定要注意第一个区间（`--one.upper_bound(start)` 很大，以至于覆盖了输入区间的情况） @Zebo L
- [x] ~~399~~
  - 有一次test 没有过，但不好说对错，就是query中含有 `["x", "x"]`，但是`"x"`在之前的条件中没有出现，这样是返回`-1`， 还是`1`，一定要问清楚 @Zebo L
- [x] ~~c++ comparator~~
  - 试了好几遍，用不用 `const` 好像都OK, 看来跟compiler 有关

```
struct Less{
  bool operator ()(const T&x, const T&y) const {
    ......
  }
};
```

- [ ] Hash map 原理:
  - A hash function is a function which when given a key, generates an address in the table.
  - A hash function that returns a unique hash number is called a **universal hash function.**
    - it always returns a number for an object.
    - two equal objects will always have the same number
    - two unequal objects not always have different numbers
  - Collision:
    - Use array of linked list: separate chaining collision resolution
    - We make sure that our lists won't become too long. This is usually implmented by maintaining a ***load factor*** that keeps a track of the average length of lists. If a load factor approaches a set in advanced threshold, we create a bigger array and *rehash* all elements from the old table into the new one.
  - 主要用mod的方法
  - 主要考Java

拓扑排序 count

- [x] ~~365~~
  - 辗转相除法：`method of successive division`

788 变形

- [x] ~~837~~
    - `dp[j]` explanation: the probability that at particular time (anytime), the person can get exactly `j` cards.
    - We have $dp[j] = \frac{1}{W} \sum_{i=1}^{W} dp[j-i]$
- [x] ~~152 以下方法很奇妙，再看看 @Zebo L~~
    - [ ] 再研究一下 @Zebo L

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int res = nums[0];
        for(int i=1, imax=nums[0], imin=nums[0]; i<nums.size
(); ++i){
            if(nums[i] < 0) swap(imax, imin);
            imax = max(nums[i], nums[i]*imax);
            imin = min(nums[i], nums[i]*imin);
            res = max(res, imax);
        }
        return res;
    }
};
```

- [x] ~~773~~
    - 错了三次，写起来不好写，一定要细心
- [x] ~~857~~
    - 这么多题中唯一一个能一次写对的，我擦，太nm 弱了

843

- [x] ~~198, 213, 337~~
    - 198, 213 太简单了，就不看了
    - 337:
        - 典型的dfs：
            - 记两个value：
                - The maximum money the robber can get if he decides to robber the root;

        - The maximum money the robber can get if he decides not to robber the root;
    - Use `pair<int, int>` as return type to get these two information from a particular tree.
    - Then doing a recursion.

```
class Solution {
    typedef pair<int, int> ii;
    ii dfs(TreeNode*root){
        if(!root) return ii(0, 0);
        ii l = dfs(root->left), r = dfs(root->right);
        return ii(max(l.first, l.second)+max(r.first, r.secon
d), l.first+r.first+root->val);
    }
public:
    int rob(TreeNode* root) {
        ii ans = dfs(root);
        return max(ans.first, ans.second);
    }
};
```

- [x] ~~685 这个一定要看，自己的解法不是最优的 @Zebo L~~
    - Union Find Combine with degree calculation
    - 错三次
    - A lot of corner cases:
        - Most simple case: `ONLY LOOP`, Union find can capture it
        - The degree of a particular node is greater than one:
            - We need to decide which edge to erase:
                - If there is no loop, remove the later one
                - If there is a loop, return the one that is in the loop
                - A easy way, could be move the high degree node edges at the tail of the original array, with the order preserved
                - Then do a union find again.
        - We probably need to do Union find twice.
        - Alternatively, we can check degree first.

761

- [x] ~~212:~~
    - This one is relatively Easy
    - 面筋题，用BFS + bitmap要好些
    - 如果搜格点中有没有某个词，dfs跟Trie可能好点

308： 看看自己的解法就行了

- [x] ~~68, String alignment:~~
    - 居然一次写对
- [ ] class继承关系
    - After Google
- [x] ~~坐板凳问题:~~
    - 只能用heap
- [x] ~~数数问题：以下，很难写对，遇到一定要细心细心细心!!!~~

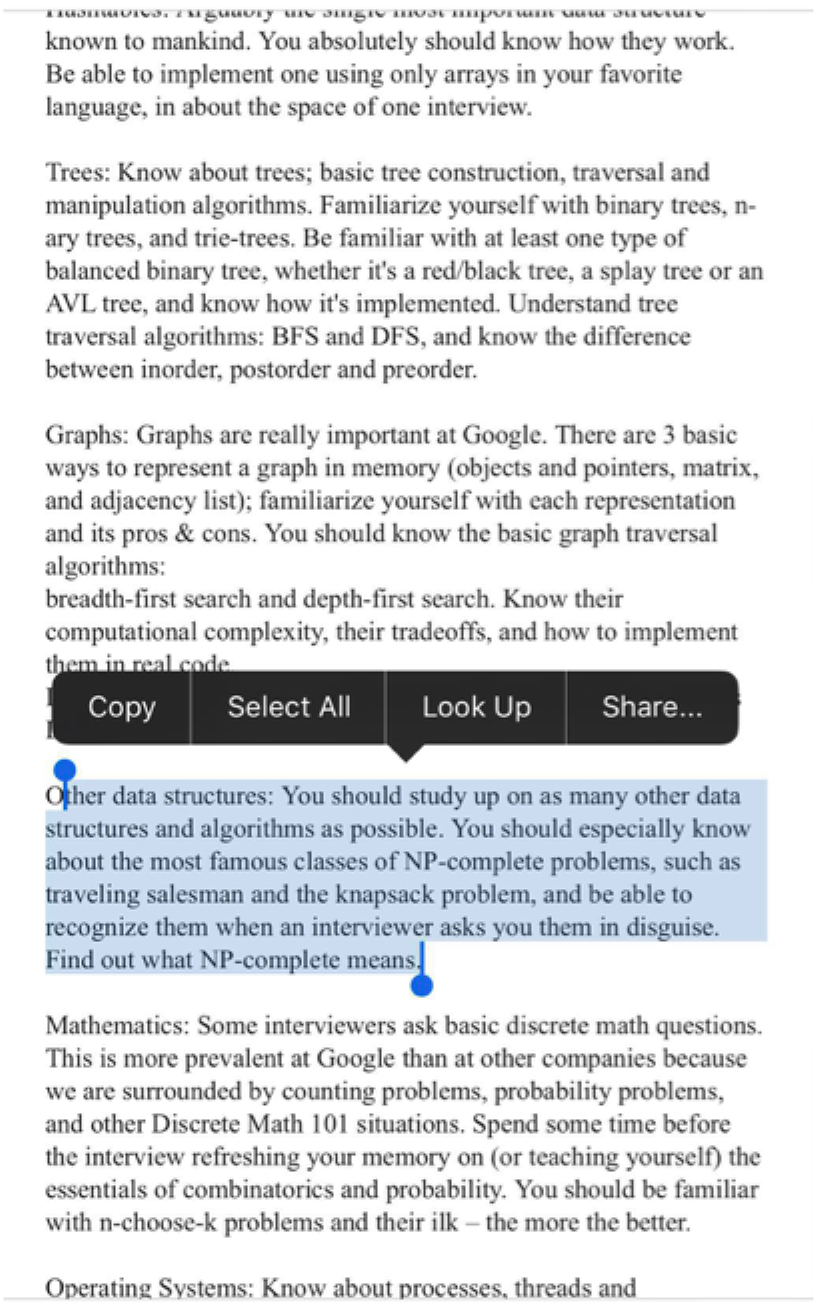
```
int hardHelper(int i, vector<int> &digs){
    int m = digs[i] - int(i==0), n = digs.size();
    if(i==n) return 0;
    int sum = 0;
    for(int k=n-1; k>i; --k){
        sum += m * int(pow(10, k-i-1)) * digs[n-k-1];
    }
    sum += min(m, digs[n-i-1]);
    return sum + hardHelper(i+1, digs);
}



int hardCnt(int n){
    vector<int> digs;
    int N = n;
    while(n){
        digs.push_back(n%10);
        n /= 10;
    }
    int m = 0;
    int ans = int(m <= N);
    int l = digs.size();
    reverse(digs.begin(), digs.end());
    ans += hardHelper(0, digs);
    ans += int(pow(10, l-1)) - 1;
    return ans;
}
```

- [x] ~~Maze:~~
  - [x] ~~https://en.wikipedia.org/wiki/Maze_generation_algorithm~~
  - [x] ~~https://blog.csdn.net/juzihongle1/article/details/73135920~~
    - Recursive backtracker:
      - Initially at the starting point
      - Put the initial cell into visited
      - Push the initial cell into stack
      - If the current cell has unvisited neighbors
        - Randomly select a neighbor to go
        - Make passenger
        - Push the new neighbor into stack and mark it a visited
      - else
        - if stack is empty: FINISHED
        - pop stack and set the popped cell as the current cell

    - Randomized Prim's Algorithm:
      - Started from one cell in the maze cell list
      - Select one of the wall:
        - If the other side of the wall has not been visited:
          - make it passenger and push the new cell in to maze cell list
          - Push the new cell's walls into wall list
        - else
          - reselect a cell
      - Until all the cells and walls are in the maze

    - Recursive Division:
      - 4-Tree:
        - dfs(rectangle):
          - dfs(upper-left);

        - dfs(upper-right);
        - dfs(lower-left);
        - dfs(lower-right);
        - randomly select 3 Big walls to make them connected.
- [x] ~~MAZE I, II, III~~
    - Only Maze III need to review
        - 错2次
        - 注意初始条件，跟截止条件
        - Dijkstra： Using priority_queue or set to track the minimum path arriving at a particular point.
        - 之前对Dijkstra理解完全不够
    - Sparse Matrix:
        - use `vector<set<int>>` to save obstacles
        - use `unordered_map<int, set<int>>` to save relevant rows and columns.



known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

Trees: Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees, and trie-trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Graphs: Graphs are really important at Google. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms:
breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code.

Other data structures: You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise. Find out what NP-complete means.

Mathematics: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because we are surrounded by counting problems, probability problems, and other Discrete Math 101 situations. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.

Operating Systems: Know about processes, threads and

- [ ] 看guild
- [x] ~~Magic Square~~
    - 奇数：旋转大法
    - 偶数：dfs暴力解
- [x] ~~自行车题一定要看下~~
    - [x] ~~Biparticle Matching~~
    - [x] ~~Stable Marriage (Not we want)~~
    - [x] ~~直接bit map dp + dfs 能搞定~~
    - [ ] Hungarian Algorithm
    - [x] ~~python module lsa: $O(n^3)$~~

```
from scipy.optimize import linear_sum_assignment
```

- [ ] 复习下Hacker Cup 第二题，Union Find经典
- [ ] 电梯
- [x] ~~https://www.geeksforgeeks.org/top-10-algorithms-in-interview-questions/~~
  - [x] ~~Partial Finished ，left are really difficult algorithms~~

  +Frequent asked Interview Algorithms
  - [x] ~~Subtree checking~~
- [x] ~~Swap Operating system~~
  - To replace pages or segments of data in memory. Swapping is a useful technique that enables a computer to execute programs and manipulate data files larger than main memory. The operating system copies as much data as possible into main memory, and leaves the rest on the disk. When the operating system needs data from the disk, it exchanges a portion of data (called a *page* or *segment*) in main memory with a portion of data on the disk.
- [x] ~~LRU~~
  - 一定要细心细心细心：
  - 定义`List` 的时候`head` 跟 `tail` 用指针，不要用Node object，否则地址有可能变
- [x] ~~Morris Traversal:~~

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        while(root){
            cout<<root->val;
            if(!root->left){
                res.push_back(root->val);
                root = root->right;
            }
            else{
                auto p = root->left;
                while(p->right) p = p->right;
                p->right = root;
                p = root->left;
                root->left = NULL;
                root = p;
            }
        }
        return res;
    }
};
```

- [ ] Guess the word
- [x] ~~Crack the safe~~
- [x] ~~Remove Boxes~~
- [x] ~~坐座位~~
- [x] ~~时间戳hash map~~