[1. n! 끝자리 0의 개수]

- n!을 할 때 인수 5가 몇 개 들어가는지 세면됨

ex) n=30 : 5, 10, 15 20, 25=(5x5), 30 이므로 6개.

[2. 인접하지 않은 인덱스만 골라서 원소 합 최대 구하기 – House robber]

- DP로 푸는 거임. 배열하나 설정하고, 해당 배열에 max(현재인덱스 + 2번째 전 인덱스, 1번째 전인덱스)를 하면서, 계속 더해나가면 됨.

int rob(vector<int>& nums) {

if(nums.empty() || nums.size()==0) return 0;

if(nums.size()==1) return nums[0];

int prev\_1=nums[1];

int prev\_2=nums[0];

int cur;

for(int i=2;i<nums.size();i++){

if(nums[i]+prev\_2>prev\_1){

cur=prev\_1;

prev\_1=nums[i]+prev\_2;

prev\_2=max(cur,prev\_2);

}

else

prev\_2=prev\_1;

}

return max(prev\_1,prev\_2);

}

[3. 이진수 스트링의 합]

- 스트링을 인트로 변환하려면 atoi(s.c\_str); 을 사용하면 됨.

- 인트를 바이너리 스트링으로 출력하려면 biset 라이브러리를 이용하면 됨.

ex) biset<셋의 크기>(인트 변수).to\_string();

- 또는 answer = sum%2; sum/=2 를 한후 얻은 문자열을 reverse하면 됨.

- 이 문제는 각 스트링의 오른쪽 끝부터 시작하여 carry값을 유지하며 진행함.

즉, 둘다 1이었을 경우 합은 2 인데, 2%2==0 이므로, 해당 위치에는 0을 대입하고 캐리 값은 2 /1 = 1을 가지고 올라감. 그럼으로써, 캐리가 될 수 있도록 함.

string addBinary(string a, string b) {

int i=a.size()-1;

int j=b.size()-1;

int cur=0;

string answer;

while(i>=0 || j>=0|| cur!=0){

if(i>=0){

cur+=a[i]=='0'?0:1;

i--;

}

if(j>=0){

cur+=b[j]=='0'?0:1;

j--;

}

answer+=((cur%2)==0?'0':'1');

cur/=2;

}

reverse(answer.begin(),answer.end()); //이렇게 해주는 것이 O(n)으로 풀수있는 방법임.

return answer;

}

[4. 배열의 각 원소들을 연이어서 수를 만들 때, 그 수의 +1을 리턴하는 문제 – Plus One]

- 즉, [4,3,2,1] 이면 [4,3,2,2]를 리턴하는 문제, 왜냐하면 4321+1=4322

수의 범위가 크지 않다면 아래 방법처럼 간단한 덧셈과 인트 -> 스트링 -> 벡터변환으로 풀 수 있음.

vector<int> plusOne(vector<int>& digits) {

int answer=digits[0];

for(int i=1;i<digits.size();i++){

answer=answer\*10+digits[i];

}

answer++;

string temp=to\_string(answer);

vector<int> ret;

for(int i=0;i<temp.size();i++){

ret.push\_back(temp[i]-48);

}

return ret;

}

그러나 수의 범위가 크다면, 덧셈을 이용할 수 없음. 아래 방법을 이용하면 됨.

vector<int> plusOne(vector<int>& digits) {

vector<int>::reverse\_iterator iter=digits.rbegin();

//rbegin 하려면 reverse\_iterator 라고 명시해줘야한다.

vector<int> answer;

int carry=1;

for(;iter!=digits.rend();iter++){

if(carry+\*iter>=10){

carry=1;

answer.push\_back(0);

}

else{

answer.push\_back(\*iter+carry);

carry=0;

}

}

// 인풋이 9 였을 때, answer에 10 이 들어가야되는데 0 만들어감.

// 이런 경우른 대비해서 끝났을 때 carry가 1이면 1을 한번더 push 해줌

if(carry==1)

answer.push\_back(1);

reverse(answer.begin(),answer.end());

return answer;

}

[4. 해당 수가 power of prime인지 확인하는 문제]

- 사실 power of three 문제였는데, 3도 소수이므로 동일하게 풀 수 있다.

- 3의 제곱들 중 최댓 값을 안다면(1162261467, Math.log(0x7fffffff)) 해당 수가 3으로 나누어지면 power of three이다.

- 마찬가지로 다른 소수의 제곱들 중 최댓 값을 알면 풀 수 있다.

[5. 옆 사람과 가장 멀리 떨어져 앉는 문제]

- 모든 인덱스에 대해서 나의 오른쪽 사람과의 거리를 right 배열에 저장하고, 왼쪽 사람과의 거리를 left에 저장한다. 그 후 두 배열의 같은 인덱스에 대해서 min값을 구하고, 최종적으로 구해진 배열의 max값을 리턴하면 된다.

- 이때 양쪽 끝에는 사람이 없는 것으로 간주한다. 이것 때문에 나의 알고리즘이 속도가 많이 느려졌다. 이것을 처리하려고 count를 사용했는데, 이때 마지막 자리와 끝 자리와의 거리가 구해진다면, 그 사이의 거리는 min처리에서 제대로 처리되므로 상관 없다.

-90.24%의 속도가 나왔다.

int alloc\_right(int\* cur, vector<int>& seats,int index,int count){

if(index>=seats.size()) return count+1;

if(seats[index]==1) return 1;

cur[index]=alloc\_right(cur,seats,index+1,count+1);

return cur[index]+1;

}

int alloc\_left(int\* cur, vector<int>& seats,int index,int count){

if(index<0) return count+1;

if(seats[index]==1) return 1;

cur[index]=alloc\_left(cur,seats,index-1,count+1);

return cur[index]+1;

}

int maxDistToClosest(vector<int>& seats) {

int\* left=new int[seats.size()]{0,};

int\* right=new int[seats.size()]{0,};

for(int i=0, j=0;i<seats.size();i++){

if(seats[i]==1) continue;

j=i;

i=i+alloc\_right(right,seats,j,0)-1;

}

for(int i=seats.size()-1,j=i;i>=0;i--){

if(seats[i]==1) continue;

j=i;

i=i-alloc\_left(left,seats,j,0)+1;

}

vector<int> answer;

for(int i=0;i<seats.size();i++){

answer.push\_back(min(left[i],right[i]));

}

int maximum=INT\_MIN;

for(int i=0;i<answer.size();i++)

maximum=max(maximum,answer[i]);

return maximum;

}

[6. 큐로 스택 구현하기]

- 큐는 항상 front에서 pop하기 때문에, stack처럼 LIFO로 pop 하려면, pop할 때 queue.size()-1만큼 큐 앞에 원소를 지워서 뒤에 붙여준 후, 마지막에 pop을 해야 됨. 이때 C++ 에서는 pop을 하면 원소가 지워지기만 하고 void 가 return 되므로, queue.front() 로 먼저 해당 원소를 저장해 주어야 함.

queue<int> Que;

/\*\* Push element x onto stack. \*/

void push(int x) {

Que.push(x);

}

/\*\* Removes the element on top of the stack and returns that element. \*/

int pop() {

int a;

for(int i=0;i<Que.size()-1;i++){

Que.push(Que.front());

Que.pop();

}

a=Que.front();

Que.pop();

return a;

}

/\*\* Get the top element. \*/

int top() {

return Que.back();

}

/\*\* Returns whether the stack is empty. \*/

bool empty() {

return Que.empty();

}

[7. 균형트리인지 확인하기]

-postOrder로 트리를 탐색하는 데, 이때 재귀로 각 트리의 왼쪽 서브트리와 오른쪽 서브트리의 높이 중 큰 값을 취하여 리턴해주고, 만약 왼쪽, 오른쪽 서브트리의 높이차가 1이상이면 flag=false로 하고, 아니면 falg=true인 상태로 놔둠.

bool flag=true;

int postOrder(TreeNode\* root){

if(!root) return 0;

int left=0,right=0;

left+=postOrder(root->left);

right+=postOrder(root->right);

if(abs(left-right)>1)

flag=false;

return max(left,right)+1;

}

bool isBalanced(TreeNode\* root) {

postOrder(root);

return flag;

}

[8. License kek Formatting – 주어진 K에 대해서 dash 사이에 K개의 문자를 배치하는 문제]

- 문자열 전체를 대문자로 바꾸기 위해서는 std::transform(S.begin(),S.end(),S.begin(), ::toupper)를 하면 된다.

- 문자열 끝에서부터 시작하여 dash를 제외한 alphanumeric의 개수를 count로 세주고, given K만큼 세어졌을 때, dash를 추가한다.

- 또한 문자열 맨 처음의 연속된 dash를 방지하기 위하여, dash가 아닌 제일 첫번째 문자의 인덱스를 문자열 시작점으로 한다.

string licenseKeyFormatting(string S, int K) {

string answer;

int count=1;

int begin=0;

while(S[begin]=='-') begin++; // 맨 앞에 연이은 dash를 방지함.

for(int i=S.size()-1;i>=begin;i--){

if(count%(K+1)==0){

answer+='-';

count=1;

i++; // K개 만큼 센후 dash추가하려는 데, 그 원소가 dash가 아닐 경우를 대비.

continue;

}

if(S[i]!='-'){

answer+=S[i];

count++;

}

}

std::transform(answer.begin(),answer.end(),answer.begin(),::toupper);

reverse(answer.begin(),answer.end());

return answer;

}

[9. Find Pivot Index – 인덱스를 기준으로 왼쪽과 오른쪽 원소들의 합이 같은 인덱스(pivot)을 찾는문제]

int pivotIndex(vector<int>& nums) {

nums.push\_back(0);

int pivot=0;

int left=0;

int right=0;

for(int i=1;i<nums.size();i++)

right+=nums[i];

while(pivot<nums.size()-1){

if(left==right)

break;

left+=nums[pivot];

right-=nums[pivot+1];

pivot++;

}

return pivot==nums.size()-1? -1 : pivot;

}

[10. power of two – 주어진 수가 2의 제곱 꼴인지 확인하는 문제]

- 비트연산을 이용하여 푼다. 2의 제곱들은 1000, 0100, 0010, 0001 등 비트형태에서 1을 하나 가지고 있다. 이때 2의 제곱들 -1은 0111, 0011, 0001, 0000 등 2의 제곱들보다 한자리 낮은 수부터 1로 가득 차있다.

-그러므로 n&(n-1)==0 이면 2의 제곱이다.

bool isPowerOfTwo(int n) {

return n>0 && (n&(n-1))==0 ;

}

[11. Reverse Vowels of a String – 모음 위치를 반대 순서로 바꾸는 문제]

- 한번 훑으면서 모음을 벡터에 저장하고, 그 모음의 인덱스를 다른 벡터에 저장한 후, 인덱스의 역순에 따라서 모음을 넣어주면 됨.

-주의할 점은 vector가 비었을 때, vec.back()을 호출하면 heap overflow 에러가 발생한다는 것임.

if(s.size()<2)

return s;

vector<char> table;

vector<int> index;

for(int i=0; i<s.size();i++){

if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'||

s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O'||s[i]=='U'){

table.push\_back(s[i]);

index.push\_back(i);

}

}

reverse(index.begin(),index.end());

string answer;

int i=0;

while(i<s.size()){

if(!index.empty()&&i==index.back()){

answer+=table.back();

table.pop\_back();

index.pop\_back();

}

else{

answer+=s[i];

}

i++;

}

return answer;

[12. **Second Minimum Node In a Binary Tree – 두개의 자식 중 최소 값이 부모가 되는 이진트리에서 두번째 최소값 찾는 문제]**

- 전부 순회를 하되, 두번째 값을 LONG\_MAX로 초기화 해놓고, 순회할때마다 해당 노드의 value가 최소값(루트값) 보다 크고, 현재의 second값보다 작은지 확인하여, 조건이 만족하면 second=value를 하면 됨.

int first;

long second;

void search(TreeNode\* root){

if(!root) return;

if(first<root->val && root->val<second)

second=root->val;

search(root->left);

search(root->right);

}

int findSecondMinimumValue(TreeNode\* root) {

first=root->val;

second=LONG\_MAX;

search(root);

return second==LONG\_MAX? -1: second;

}

[13. **Convert a Number to Hexadecimal – 십진수를 16진수 스트링으로 바꾸는 문제]**

- 스트링하나에 0부터 f까지 지정해 준 후, 주어진 십진수를 16으로 % 연산을 하여 해당 인덱스의 값을 결과 값에 더해나간다.

- 마지막에 해당 값을 reverse해주면 된다.

- 이때 -를 다루기 위해서는 unsigned int 로 인풋값을 재 지정해준다. 그럴경우 -1이 unsigned int의 최대값이 된다.

string toHex(int num) {

unsigned int n=num;

string table="0123456789abcdef";

string answer;

do{

answer+=table[n%16];

n/=16;

}while(n);

return {answer.rbegin(),answer.rend()};

}

[14 **Quad Tree Intersection - 4개의 자식이 있는 두개의 트리에 대하여 각 자식끼리의 OR연산을 하는 문제]**

**-** 난 아래처럼 최종 모습이 4개의 자식을 가진 노드를 만드는 건줄 알았는 데, 아닌 듯?

bool search(Node\* root){

if(root->isLeaf) return root->val;

return search(root->topLeft)||search(root->topRight)||search(root->bottomLeft)||search(root->bottomRight);

}

Node\* intersect(Node\* quadTree1, Node\* quadTree2) {

bool one=search(quadTree1->topLeft)||search(quadTree2->topLeft);

bool two=search(quadTree1->topRight)||search(quadTree2->topRight);

bool three=search(quadTree1->bottomLeft)||search(quadTree2->bottomLeft);

bool four=search(quadTree1->bottomRight)||search(quadTree2->bottomRight);

Node\* answer=new Node();

answer->val = true;

answer-> isLeaf = false;

answer->topLeft = new Node(one,true,nullptr,nullptr,nullptr,nullptr);

answer->topRight = new Node(two,true,nullptr,nullptr,nullptr,nullptr);

answer->bottomLeft = new Node(three,true,nullptr,nullptr,nullptr,nullptr);

answer->bottomRight = new Node(four,true,nullptr,nullptr,nullptr,nullptr);

return answer;

}

- 마지막에 4개의 자식이 모두 같은 boolean 값을 가지면 하나로 합쳐줘야함.

- 아래 2개의 방법은 다른 사람들이 푼 것으로, 읽으니까 이해할 수 있었음.

Node\* intersect(Node\* quadTree1, Node\* quadTree2) {

if(quadTree1->isLeaf && quadTree1->val) return quadTree1;

if(quadTree2->isLeaf && quadTree2->val) return quadTree2;

if(quadTree1->isLeaf && !quadTree1->val) return quadTree2;

if(quadTree2->isLeaf && !quadTree2->val) return quadTree1;

auto tl = intersect(quadTree1->topLeft, quadTree2->topLeft);

auto tr = intersect(quadTree1->topRight, quadTree2->topRight);

auto bl = intersect(quadTree1->bottomLeft, quadTree2->bottomLeft);

auto br = intersect(quadTree1->bottomRight, quadTree2->bottomRight);

if(tl->val == tr->val && tl->val == bl->val && tl->val == br->val && tl->isLeaf && tr->isLeaf && bl->isLeaf && br->isLeaf)

return new Node(tl->val, true, nullptr, nullptr, nullptr, nullptr);

else

return new Node(false, false, tl, tr, bl, br);

}

Node\* intersect(Node\* quadTree1, Node\* quadTree2)

{

// if a is leaf+true, choose a.

// if b is leaf+false, choose a.

if (quadTree1->isLeaf && quadTree1->val == true ||

quadTree2->isLeaf && quadTree2->val == false)

return quadTree1;

// if b is leaf+true, choose b.

// if a is leaf+false, choose b;

if (quadTree2->isLeaf && quadTree2->val == true ||

quadTree1->isLeaf && quadTree1->val == false)

return quadTree2;

// intersect.

Node\* ret = new Node (false, false,

intersect (quadTree1->topLeft, quadTree2->topLeft),

intersect (quadTree1->topRight, quadTree2->topRight),

intersect (quadTree1->bottomLeft, quadTree2->bottomLeft),

intersect (quadTree1->bottomRight, quadTree2->bottomRight));

// merge if all children are leaves and have the same value.

if (ret->topLeft->val == ret->topRight->val && ret->topLeft->val == ret->bottomLeft->val &&

ret->topLeft->val == ret->bottomRight->val &&

ret->topLeft->isLeaf && ret->topRight->isLeaf && ret->bottomLeft->isLeaf && ret->bottomRight->isLeaf)

{

ret->val = ret->topLeft->val;

ret->isLeaf = true;

ret->topLeft = ret->topRight = ret->bottomLeft = ret->bottomRight = NULL;

}

// done.

return ret;

}

[15. **Most Common Word – 문자열에서 금지된 단어를 제외하고 가장 많이 등장하는 단어를 찾아내는 문제]**

- map을 이용해서 각 단어마다 등장횟수를 센 후, map을 순회하면서 해당 단어가 banned되어 있지 않고, 현재의 단어보다 많이 등장하였다면 answer를 바꾸어 줌.

- 문자열을 소문자로 바꾸기 위해서는 transfrom(s.begin(),s.end(),s.begin(),::tolower)를 하면 됨.

- vector안에서 해당 아이템, 또는 문자열을 찾기 위해서는 find(vec.begin(),vec.end(),item)을 하면 해당 item의 iterator가 반환됨. 그러므로 !=vec.end() 를 조건으로 주면 해당 아이템이 없으면 false, 있으면 true가 됨.

std::transform(paragraph.begin(),paragraph.end(),paragraph.begin(),::tolower);

unordered\_map<string,int> table;

string temp;

for(int i=0;i<paragraph.size();i++){

if('a'<=paragraph[i] && paragraph[i]<='z')

temp+=paragraph[i];

else{

table[temp]++;

temp="";

}

}

table[temp]++; //마지막에 고려되지 못한 단어를 위해서

unordered\_map<string,int>::iterator iter=table.begin();

string answer="";

int count=0;

for(;iter!=table.end();iter++){

if(std::find(banned.begin(),banned.end(),iter->first)==banned.end()&&iter->second>count

&& 'a'<=iter->first[0] && iter->first[0]<='z'){// 해당 문자열이 banned이고, iter가 구두점이나 공백이 아니라면

answer=iter->first;

count=iter->second;

}

}

return answer;

}

[16. **Subtree of Another Tree – 트리 s가 트리 t를 서브트리로 가지고 있는지 확인하는 문제]**

**-** 이렇게 푸는게 맞는지 모르겠는데, 일단 오래걸렸음.

- 먼저 트리 t의 shape을 vector에 담고, 트리 s에서 트리 t의 root를 찾은 순간, 그때부터 트리 t와 같은 subtree구조를 가졌는지 확인하는 방식임.

bool answer=false;

vector<int> t\_table;

int i;

int first;

void t\_search(TreeNode\* root){ //4, 1, min, min, 2, min, min

if(!root){

t\_table.push\_back(INT\_MIN);

return;

}

t\_table.push\_back(root->val);

t\_search(root->left);

t\_search(root->right);

}

void s\_search(TreeNode\* root){

if(!root) return;

if(root->val==first){ //if tree t's root is found, then check whether the subtree is same with t

check(root);

}

if(answer==true)

return;

s\_search(root->left);

s\_search(root->right);

}

void check(TreeNode\* root){

if(!root && t\_table[i]==INT\_MIN){

if(i==t\_table.size()-1){

answer=true;

return;

}

i++;

return;

}

if((!root && t\_table[i]!=INT\_MIN) || ( root &&root->val !=t\_table[i])){

i=0;

return;

}

i++;

check(root->left);

check(root->right);

}

bool isSubtree(TreeNode\* s, TreeNode\* t) {

t\_search(t);

first=t->val;

i=0;

s\_search(s);

return answer;

}

- 그리고 훨씬 좋은 solution을 봐버렸다.

class Solution {

public:

bool isSubtree(TreeNode\* s, TreeNode\* t) {

if(!s) return false;

if (isSame(s,t)) return true;

return isSubtree(s->left,t) || isSubtree(s->right,t);

}

bool isSame(TreeNode \*s, TreeNode \*t)

{

if (!s && !t) return true;

if (!s || !t) return false;

if (s->val != t->val) return false;

return isSame(s->left, t->left) && isSame(s->right, t->right);

}

};

[17. **Remove Duplicates from Sorted List – 링크드리스트에서 중복 원소 제거하는 문제]**

- Discussion 찾아보니까 new로 생성한거 아니면 dlete할 필요 없다는 거 같은데 잘 모르겠음.

- 일단 같은 원소가 아닐때까지 loop돌고 다른원소 발견하면 이전 원소의 맨 앞에거랑 연결해주는 방식으로 했음. 이것이 속도가 좀 빠름.

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* answer= head;

ListNode\* temp;

int value;

while(head){

value=head->val;

temp=head;

while(temp->next && value==temp->next->val) temp=temp->next;

if(!temp->next){

head->next=nullptr;

head=head->next;

break;

}

else{

head->next=temp->next;

head=head->next;

}

}

return answer;

}

[18. **Add to Array-Form of Intege – 벡터로 정수가 주어지고, 또 다른 정수 K가 주어져서 두개를 더한 후 벡터모양을 리턴하는 문제]**

- 벡터를 일단 정수로 바꿔서 더할려고 했는데, 정수 값을 엄청 크게 줘서 안됨.

- 그래서 carry를 사용해서 풀었는 데, 내가봤는데도 너무 느린 효율을 가짐

- 참고로 K%10 은 마지막 숫자를 리턴해줌. K/10은 마지막 숫자를 제외한 숫자를 리턴함.

vector<int> addToArrayForm(vector<int>& A, int K) {

int cur=0;

int carry=0;

vector<int> answer;

for(int i=A.size()-1;i>=0;i--){

cur=A[i]+K%10+carry;

if(cur>=10) {

carry=1;

cur%=10;

}

else carry=0;

K/=10;

answer.push\_back(cur);

}

while(K){ //A의 자릿수가 K보다 작은 경우를 cover함.

cur=K%10+carry;

if(cur>=10){

carry=1;

cur%=10;

}

else carry=0;

answer.push\_back(cur);

K/=10;

}

if(carry==1) answer.push\_back(1); //답이 1082 막 이런데 벡터에 082 이런거 까지만 들어가는 경우 cover함

return {answer.rbegin(),answer.rend()};

}

- 이거는 나랑 같은 방식인데 설명을 더 잘해놓음.

/\* An important observation ---

1) num%10 gives us the last digit of a number

2) num = num/10 cuts off the last digit of the number

3) numVector.back() gives us the last digit of the number in vector form

4) numVector.pop\_back() cuts off the last digit of the number in vector form

5) The extra space required can be reduced by overwriting the first vector.

\*/

class Solution

{

public:

vector<int> addToArrayForm(vector<int>& a, int k);

};

/\* Returns the sum of 2 numbers in vector form \*/

vector<int> Solution :: addToArrayForm(vector<int>& a, int k)

{

// Get the length of the first number

int n = a.size();

// Vector to store the answer

vector<int> answer;

/\* Start adding both the numbers from the end \*/

int carry = 0;

// As long as one of the number exists, keep adding them

while(!a.empty() || k!=0)

{

// Get the last digits of both the numbers. If a vector has finished off, the last digit is zero

int lastDigit\_1 = a.empty() ? 0 : a.back();

int lastDigit\_2 = k%10;

// Sum up the digits and add the carry

int sum = lastDigit\_1 + lastDigit\_2 + carry;

answer.push\_back(sum%10);

carry = sum/10;

// Remove the last digits of both the numbers

if(!a.empty()) a.pop\_back();

k = k/10;

}

// If the carry is remaining, add it

if(carry!=0) answer.push\_back(carry);

// Reverse the answer, since we were summing up from the end

reverse(answer.begin(), answer.end());

// Return the answer in vector format

return answer;

}

[19. **Long Pressed Name – 중복적으로 눌린 키보드 문자로 원래의 이름을 만들 수 있는지 확인하는 문제.]**

- 문제 설명하기가 힘들어서 원본을 붙여 넣음.

Your friend is typing his name into a keyboard.  Sometimes, when typing a character c, the key might get long pressed, and the character will be typed 1 or more times.

You examine the typed characters of the keyboard.  Return True if it is possible that it was your friends name, with some characters (possibly none) being long pressed.

- 두 문자열이 match 되면 pass, 아니면 long pressed인지 확인하여 그만큼 string typed를 진행시키고, 아니면 return false.

- while 문이 종료되었을 때, i가 name.size()보다 작으면, string typed의 character의 숫자가 모자른 것이므로 false.

bool isLongPressedName(string name, string typed) {

int i=0,j=0;

char prev=name[0];

while(j<typed.size()){

// if the two strings is matched.

if(name[i]==typed[j]){

prev=name[i];

i++;j++;

continue;

}

//if typed has been long pressed.

if(prev!=name[i] && prev==typed[j]){

while(prev==typed[j]) j++;

continue;

}

return false;

}

//if the type has not enough character.

return i>=name.size()? true : false;

}

[20. **Path Sum III – 주어진 트리의 노드들을 더하여 주어진 sum을 만들 수 있는 path가 몇 개인지 구하는 문제]**

- 두개의 재귀를 생각해야해서 어려운 문제임. 덕분에 좋은 공부가 되었음.

- 전수조사 밖에 답이없는 것 같음.

int search(TreeNode\* root, int sum, int pre){

if(!root) return 0;

return (root->val+pre == sum) + search(root->left,sum,pre+root->val) + search(root->right,sum,pre+root->val);

}

int pathSum(TreeNode\* root, int sum) {

if(!root) return 0;

return search(root,sum,0) + pathSum(root->left,sum)+pathSum(root->right,sum);

}

[21. **Find Smallest Letter Greater Than Target – 주어진 target보다 큰 가장 작은 character를 반환하는 문제]**

- 그냥 간단하게 target의 ascii code값 보다 큰 값이 나오면 return 해주는 것으로 했음, 끝날때까지 찾지 못하면, 첫번째 원소. 이거는 O(n)임.

char nextGreatestLetter(vector<char>& letters, char target) {

if(letters[letters.size()-1]<=target) return letters[0];

for(int i=0;i<letters.size();i++){

if(target<letters[i])

return letters[i];

}

return letters[0];

}

-이것을 이진탐색으로도 풀 수 있음.

char nextGreatestLetter(vector<char>& letters, char target) {

int left=0;

int right=letters.size()-1;

int mid;

while(left<=right){

mid=(right-left)/2+left; // to handle INT overflow.

if(mid==0){

if(letters[mid]>target) return letters[0];

else{

left=mid+1;

continue;

}

}

if(letters[mid-1]<=target && letters[mid]>target) return letters[mid];

else if(letters[mid]<=target)

left=mid+1;

else right=mid-1;

}

return letters[0];

}

- upper\_bound를 사용한 풀이 방법도 있음. 이거 알아두자.

- upper\_bound는 이진탐색 기반의 탐색 법임. 사용전 list가 정렬되어 있어야함.

- key값을 초과하는 가장 첫번째 원소의 반복자를 리턴함.

- upper\_bound(vec.begin(),vec.end(),value)의 형식으로 사용함.

- lower\_bound는 해당 value를 포함하여 이상인 값을 리턴함. 즉, lower는 이상, upper는 초과!

char nextGreatestLetter(vector<char>& letters, char target) {

vector<char>::iterator iter=upper\_bound(letters.begin(),letters.end(),target);

return iter==letters.end() ? letters[0] : \*iter;

}

p.s) 세상 많이 편해졌다.

[22. **Longest Harmonious Subsequence – max와 min의 차이가 정확히 1인 subsequence 찾는 문제]**

- 여기서 subsequence는 substring이 아니라, 조합될수 있는 모든 가지수임.

ex) apple의 subsequence는 a,ap,al,ae,app,apl,ape … 이런 식임.

- 이 문제는 map을 활용하여 2번의 for문을 쓰는게 관건이었음.

- 생각하기 쉽지 않아서 Discussion의 도움을 받음.

int findLHS(vector<int>& nums) {

unordered\_map<int,int> table;

for(int i : nums){

table[i]++;

}

int answer=0;

unordered\_map<int,int>::iterator iter=table.begin();

for(;iter!=table.end();iter++){

if(table.count(iter->first-1)>0)

answer=max(answer,iter->second+table[iter->first-1]);

}

return answer;

}

[23. **Longest Continuous Increasing Subsequence – 벡터 내 오름차순 부분배열 중 가장 긴 길이를 구하는 문제] 2020-01-26 7:30.**

- empty인 vector에 대해서 vec.empty()로 체크하는 것 보다 vec.size()==0 으로 체크하는 것이 훨씬 빠르다.

int findLengthOfLCIS(vector<int>& nums) {

if(nums.size()==0) return 0;

int count=1;

int answer=1;

int i=0;

while(i<nums.size()-1){

while(i+1<nums.size() && nums[i]<nums[i+1]){

count++;

i++;

}

answer=max(answer,count);

count=1;

i++;

}

return answer;

}

[24. **Symmetric Tree – 트리가 대칭 모양인지 구하는 문제]**

- 나는 left subtree와 right subtree를 preorder search를 통해서 vector에 각 value값을 저장하고, 마지막에 두개의 벡터가 같은 값을 가지는지 확인하였음.

- 이때 null node에 대해서는 INT\_MIN을 대입함으로써, 모양을 기억함.

vector<int> left\_tree;

vector<int> right\_tree;

void leftSearch(TreeNode\* root){

if(!root){

left\_tree.push\_back(INT\_MIN);

return;

}

left\_tree.push\_back(root->val);

leftSearch(root->left);

leftSearch(root->right);

}

void rightSearch(TreeNode\* root){

if(!root){

right\_tree.push\_back(INT\_MIN);

return;

}

right\_tree.push\_back(root->val);

rightSearch(root->right);

rightSearch(root->left);

}

bool isSymmetric(TreeNode\* root) {

if(!root) return true;

leftSearch(root->left);

rightSearch(root->right);

if(left\_tree.size()!=right\_tree.size())

return false;

for(int i=0;i<left\_tree.size();i++)

if(left\_tree[i]!=right\_tree[i])

return false;

return true;

}

- Discussion의 방법인데, 좀 더 빠르고 간단함.

bool isSymmetric(TreeNode \*root) {

if (!root) return true;

return helper(root->left, root->right);

}

bool helper(TreeNode\* p, TreeNode\* q) {

if (!p && !q) {

return true;

} else if (!p || !q) {

return false;

}

if (p->val != q->val) {

return false;

}

return helper(p->left,q->right) && helper(p->right, q->left);

}

[25. **Maximum Subarray – 최대부분배열문제] 2020-01-27 4:30 PM.**

- dp를 이용해서 푸는건데, dp 벡터에 현재 인덱스의 값과 바로 이전까지 구한 최대부분배열을 더하여 현재 인덱스 값보다 크면 그 값을 그 인덱스의 dp값으로 하는 방식임.

- O(n) solution인데, 떠올리기 쉽지 않음.  
int maxSubArray(vector<int>& nums) {

int maximum=nums[0];

vector<int> dp;

dp.push\_back(nums[0]);

for(int i=1;i<nums.size();i++){

dp.push\_back(max(nums[i],nums[i]+dp[i-1]));

maximum=max(maximum,dp[i]);

}

return maximum;

}

[26. **Base 7 – 주어진 수를 7진수 string으로 표현하는 문제]**

- 이게 인풋이 그냥 7이면 곱하기 10을 해도 7%7=0 이므로 자릿수가 올라가지 않아서, 처음부터 string으로 다루는 것이 포인트임.

string convertToBase7(int num) {

if(num==0)

return "0";

string base7;

int temp=num;

num=abs(num);

while(num){

base7+=to\_string(num%7);

num/=7;

}

reverse(base7.begin(),base7.end());

return temp<0? '-'+base7 : base7;

}

[27. **Add Strings – 두개의 string으로 주어진 정수를 더하는 문제] 2020-01-27 5:45 PM**

- string 최대길이가 5100이므로 정수로 바꿔서 더할 수 없고, carry를 활용해야함.

- 더 짧게 짤 수 있을 것 같긴한데, 알고리즘은 대체로 나와 같을 것이라 예상됨.

string addStrings(string num1, string num2) {

int carry=0;

string answer;

int temp;

int i,j;

for(i=num1.size()-1,j=num2.size()-1;i>=0 && j>=0 ; i--,j--){

temp=(num1[i]-48)+(num2[j]-48)+carry;

answer+=to\_string(temp%10);

if(temp>=10) carry=1;

else carry=0;

}

while(i>=0){

temp=(num1[i]-48+carry);

answer+=to\_string(temp%10);

i--;

if(temp>=10) carry=1;

else carry=0;

}

while(j>=0){

temp=(num2[j]-48+carry);

answer+=to\_string(temp%10);

j--;

if(temp>=10) carry=1;

else carry=0;

}

if(carry)

answer+=to\_string(carry);

return {answer.rbegin(),answer.rend()};

}

[28. **Climbing Stairs – 계단을 한번에 1개나 2개씩 오를 수 있을 때, 특정 높이의 계단까지 가는 방법의 수를 구하는 문제] 2020-01-27 6:20**

**-** 간만에 DP를 풀어서 그런지, 피보나치임을 눈으로 보고도 알아채지 못 했음.

int climbStairs(int n) {

if(n<=2) return n;

int temp;

int prev=1;

int cur=2;

for(int i=3; i<=n;i++){

temp=cur;

cur=prev+cur;

prev=temp;

}

return cur;

}

[29. **Binary Watch – 손목시계에 들어온 LED의 개수로 표현할 수 있는 모든 시간을 구하는 문제] 2020-01-27 7:00**

- 이건 내가 못푸는 문제임. Discussion 보고도 한참 걸렸음.

- bitset을 이용하는 문제임. 어렵다.

- bitset<10> a으로 선언할 수 있는데, a에 자리수가 10개인 비트를 저장할 수 있다는 것이다.

이때 bitset<10>(3) a 로 선언하면, 10개의 비트에 대해서 십진수 3을 이진수로 표현하여 a에 저장해준다. 즉 0000 00011 이 된다.

- a.set() 을 하면 모든 bit이 1이되고, a.reset() 하면 0이 된다. a.set(3,true)로 하면 3+1인 4번 비트를 1로 설정한다.

- a.flip()을 하면 모든 비트를 반전하고, a.flip(3)을 하면 3+1인 4번 비트를 반전한다.

- a.to\_string() 하면 전체 비트를 string화 시킨다.

- a[4], a.test[4] 처럼 배열처럼 접근할 수 있다.

- a.any()는 비트 중 하나라도 1이면 1을 반환하고, a.none()은 모두 0이어야만 1을 반환한다.

- a.test(n)은 n+1번째 비트가 1인지 0인지 검사한다.

vector<string> readBinaryWatch(int num) {

vector<string> answer;

for(int h=0;h<12;h++){

for(int m=0;m<60;m++){

if(bitset<10>(h).count()+bitset<10>(m).count()==num)

answer.push\_back(to\_string(h)+":"+(m<10? "0":"")+to\_string(m));

}

}

return answer;

}

-이런걸 생각해 내는 사람들도 대단함.

[30. **Student Attendance Record I – 학생이 상을 받을 수 있는 출석률을 가졌는지 아닌지 판단하는 문제]**

- 간단한 문제임. if문을 여러 개 사용하면 됨.

- 결석이 2회이상이면 탈락이고, 3연속 지각을 하면 탈락임.

bool checkRecord(string s) {

bool absent=false;

int late=0;

for(int i=0;i<s.size();i++){

if(absent && s[i]=='A')

return false;

else if(s[i]=='A'){

absent=true;

late=0;

continue;

}

else if(late==2&&s[i]=='L')

return false;

else if(s[i]=='L'){

late++;

continue;

}

late=0;

}

return true;

}

[31. **Number of 1 Bits – uint\_8 로 주어진 bit에 1이 몇 개인지 세는 문제]**

**-** bit연산을 통해서 풀었음. 주어진 n과 1을 &연산을 하여 true면 count++를 함. 그 이후 n=n>>1을 하여 n==0일 때까지 loop를 돌림.

int hammingWeight(uint32\_t n) {

int count=0;

while(n){

n&1? count++:count;

n=n>>1;

}

return count;

}

[32. **Pascal's Triangle II – 파스칼 트라이앵글의 특정 row의 원소를 구하는 문제]**



- 벡터 메모리 해제를 위해 vector<int>().swap(answer)를 이용하였다.

vector<int> getRow(int rowIndex) {

if(rowIndex==0) return {1};

vector<int> prev(2,1);

vector<int> answer;

vector<int> res=prev;

for(int i=0;i<rowIndex-1;i++){

answer.push\_back(1);

for(int j=0;j<prev.size()-1;j++){

answer.push\_back(prev[j]+prev[j+1]);

}

answer.push\_back(1);

prev=answer;

res=answer;

answer.clear();

}

vector<int>().swap(prev); // deallocate prev

vector<int>().swap(answer);

return res;

}

- 다음 문장에 의거하여 더 빠르게 풀 수 있다.

*Based on math, the kth element for nth row is C(n, k) = n! / (k!\*(n-k)!), then res[k] = res[n - k]*

*so the relationship between res[i] and res[i-1] is n! / (k!(n-k)!) / n!****/****((k-1)!(n-k + 1)!) = (n - k + 1) / k;*

*Note that this solution is math derived from number of Combinations.*

*Each line of Pascal's Triangle is a full set of Combination number based on k .*

*comb(k,p) = k! /( p! \*(k-p)!) = comb(k,k-p)*

*if p < k-p*

*comb(k,p) = comb(k,p-1) \* (k-p+1) / p*

*Because :*

*comb(k,p) = [ k \* (k-1) \* (k-2) \*... (k-p+1)] / [1 \* 2 \* 3 \*...(p)]*

vector<int> getRow(int rowIndex) {

vector<int> answer(rowIndex+1,1);

for(int i=1;i<=(rowIndex+1)/2;i++){

answer[i]=answer[rowIndex-i]=(long)answer[i-1]\*(long)(rowIndex-i+1)/i;

}

return answer;

}

[33. **Implement Queue using Stacks – 스택을 이용해 큐를 구현하는 문제] 2020-01-28 5:00**

- 스택을 2개 사용하여, push는 스택1에 push하고, pop이나 peek이 호출된 순간, 스택2가 비어있으면 스택1의 모든 원소를 pop하여 스택 2에 넣고, 스택 2의 top을 리턴한다. 스택2가 비어있지 않을 경우 그냥 스택 2의 탑을 리턴한다.

- 이때 pop 호출시 해당 top을 삭제하여 준다.

stack<int> stk1;

stack<int> stk2;

/\*\* Push element x to the back of queue. \*/

void push(int x) {

stk1.push(x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

int pop() {

if(!stk2.empty()){

int temp=stk2.top();

stk2.pop();

return temp;

}

while(!stk1.empty()){

stk2.push(stk1.top());

stk1.pop();

}

int temp=stk2.top();

stk2.pop();

return temp;

}

/\*\* Get the front element. \*/

int peek() {

if(!stk2.empty())

return stk2.top();

while(!stk1.empty()){

stk2.push(stk1.top());

stk1.pop();

}

return stk2.top();

}

/\*\* Returns whether the queue is empty. \*/

bool empty() {

if(stk1.empty() && stk2.empty())

return true;

return false;

}

[34. **Number of Equivalent Domino Pairs - 2차원 벡터에서 순서에 상관없이 동일한 원소를 가진 행벡터를 구하는 문제] 2020-01-28 5:20**

- 나는 일단 잘 안떠올라서 sort시키고 시작했음. sort 후 map의 키로서 pair<int,int>를 활용함.

- map의 키로서 pair를 사용할 경우 unordered\_map은 사용 불가능함. 그래서 더 느린듯.

int numEquivDominoPairs(vector<vector<int>>& dominoes) {

for(int i=0;i<dominoes.size();i++)

sort(dominoes[i].begin(),dominoes[i].end());

map<pair<int,int>,int> table;

for(int i=0;i<dominoes.size();i++){

table[make\_pair(dominoes[i][0],dominoes[i][1])]++;

}

map<pair<int,int>,int>::iterator iter=table.begin();

int answer=0;

for(;iter!=table.end();iter++)

answer+=(iter->second)\*(iter->second -1)/2;

return answer;

}

- 다른 방법을 찾아봤는데, Discussion 에서 이런 신박한 방법이 있었음.

**Explanation**

You need to distinguish the different dominoes and count the same.

I did it in this way:  
f(domino) = min(d[0], d[1]) \* 10 + max(d[0], d[1])  
For each domino d, calculate min(d[0], d[1]) \* 10 + max(d[0], d[1])  
This will put the smaller number on the left and bigger one on the right (in decimal).  
So same number same domino, different number different domino.

Take the example from the problem:  
dominoes = [[1,2],[2,1],[3,4],[5,6]]  
now we transform it into [12,12,34,56].

int numEquivDominoPairs(vector<vector<int>>& dominoes) {

unordered\_map<int, int> count;

int res = 0;

for (auto& d : dominoes) {

res += count[min(d[0], d[1]) \* 10 + max(d[0], d[1])]++;

}

return res;

}

- 천재 xx들…

[35. **Maximum Product of Three Numbers – 벡터의 원소 중 3개를 골라 곱하였을 때의 최댓값을 구하는 문제] 2020-01-28 5:48**

- 최댓값은 벡터에 음수가 존재할 경우, 가장 작은 음수 2개와 가장 큰 양수 하나를 곱하거나, 가장 큰 양수 3개를 차례로 곱하는 2가지 방법만이 존재한다.

- 그러므로 가장 큰 양수 3개와, 가장 작은 음수 2개를 찾은 후 두개를 비교하면 된다.

int maximumProduct(vector<int>& nums) {

int max1=INT\_MIN+2,max2=INT\_MIN+1, max3=INT\_MIN;

int min1=INT\_MAX-1,min2=INT\_MAX;

for(int i=0;i<nums.size();i++){

if(nums[i]>max1){

max3=max2;

max2=max1;

max1=nums[i];

}

else if(nums[i]>max2){

max3=max2;

max2=nums[i];

}

else if(nums[i]>max3) max3=nums[i];

if(nums[i]<min1){

min2=min1;

min1=nums[i];

}

else if(nums[i]<min2) min2=nums[i];

}

return max(max1\*max2\*max3,min1\*min2\*max1);

}

[36. **Pairs of Songs With Total Durations Divisible by 60 – 주어진 벡터에서 두 원소의 합이 60으로 나누어 떨어지는 원소 쌍의 개수를 구하는 문제] 2020-01-28 6:14**

- 일단 map을 이용하여, 60-원소값 즉, 해당 원소가 60으로 나누어떨어지기 위해서 더해져야 하는 값을 확인하고, 해당 값이 존재하면 answer에 더해준다.

- 이때 i<j인 쌍에서만 확인한다. 즉 중복된 두개의 원소쌍에 대해서는 한번만 연산한다는 것이 중요하다.

- table에는 key값으로 60을 가질 수 없기 때문에, time[i]==60일 경우 %60을 한번 더해줌으로써 table[0]을 검색하게 해준다.

int numPairsDivisibleBy60(vector<int>& time) {

unordered\_map<int,int> table;

int answer=0;

for(int i=0;i<time.size();i++){

answer+=table[(60-(time[i]%60))%60]; // to handle when a time[i]==60, since there will be no value 60 in the table.

table[time[i]%60]++;

}

return answer;

}

[37. **Sum of Nodes with Even-Valued Grandparent – 현재 노드의 grandparent가 짝수인 모든 노드의 합을 구하는 문제] 2020-01-29 6:37**

**-** helper함수 하나와 재귀를 사용하여 풀었다.

int sum=0;

int search(TreeNode\* root){

int temp=0;

if(!root) return 0;

if(root->left)

temp+=root->left->val;

if(root->right)

temp+=root->right->val;

return temp;

}

int sumEvenGrandparent(TreeNode\* root) {

if(!root) return 0;

if(root->val%2==0){

sum+=search(root->left)+search(root->right);

}

sumEvenGrandparent(root->left);

sumEvenGrandparent(root->right);

return sum;

}

[38. **Deepest Leaves Sum – 가장 깊은 노드들의 value의 합을 구하는 문제] 2020-01-29 6:47**

- 첫번째로 max\_depth를 구하기 위해 트리를 한번 돌고, 두번째로 max\_depth인 노드들의 합을 구하기 위해 한번 돌고, 트리를 총 2번 도는 알고리즘임.

int max\_depth=INT\_MIN;

void findMaxDepth(TreeNode\* root,int depth){

if(!root) return;

max\_depth=max(max\_depth,depth);

findMaxDepth(root->left,depth+1);

findMaxDepth(root->right,depth+1);

}

int sumOfDeepest(TreeNode\* root,int depth){

if(!root) return 0;

if(depth==max\_depth)

return root->val;

return sumOfDeepest(root->left,depth+1)+sumOfDeepest(root->right,depth+1);

}

int deepestLeavesSum(TreeNode\* root) {

findMaxDepth(root,0);

return sumOfDeepest(root,0);

}

[39. **Group the People Given the Group Size They Belong To – 주어진 groupsize의 각 인덱스의 위치한 숫자가, 해당 인덱스가 포함될 그룹의 사이즈임]**

**-** 문제를 이해하는 것 자체가 힘듬. 일단 0번째 인덱스의 원소가 3이면, 0번 ID를 가진 사람의 그룹 사이즈가 3이어서, 사이즈가 3인 그룹에 해당 ID를 할당해야하는 것임.

- map에 벡터를 value로 사용하는 방법을 배웠음. 해당 key가 empty일때는 table[key]=vector<int>{value} 로 맨처음에 값을 넣어주고, 만약에 값이 존재하면 즉, table[key].count()>0 이 true이면 그냥 바로 table[key].push\_back(value); 해주면 됨.  
vector<vector<int>> groupThePeople(vector<int>& groupSizes) {

unordered\_map<int,vector<int>> table;

vector<vector<int>> answer;

for(int i=0;i<groupSizes.size();i++){

int key=groupSizes[i];

if(table.count(key)>0)

table[key].push\_back(i);

else table[key]=vector<int>{i};

if(table.find(key)->second.size()>=key){

answer.push\_back(table[key]);

table.erase(key);

}

}

return answer;

}

[40. **Max Increase to Keep City Skyline – 각 row, col에 대하여 해당 원소가 증가할 수 있는 최대치의 합을 구하는 문제]**

- 각 row와 column의 최대 값을 구한 벡터를 2개를 가진 후, 각 원소가 증가할 수 있는 최대치를 구하여 더해줌.

int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {

vector<int> row\_height;

vector<int> col\_height;

int temp\_max=INT\_MIN;

for(int i=0;i<grid.size();i++){

for(int j=0;j<grid[i].size();j++)

temp\_max=max(temp\_max,grid[i][j]);

row\_height.push\_back(temp\_max);

temp\_max=INT\_MIN;

}

for(int i=0;i<grid[0].size();i++){

for(int j=0;j<grid.size();j++)

temp\_max=max(temp\_max,grid[j][i]);

col\_height.push\_back(temp\_max);

temp\_max=INT\_MIN;

}

int sum=0;

for(int i=0;i<grid.size();i++){

for(int j=0;j<grid[i].size();j++){

sum+=min(row\_height[i],col\_height[j])-grid[i][j];

}

}

return sum;

}

[41. **Binary Search Tree to Greater Sum Tree – 이진탐색트리에서 자신보다 큰 노드의 값을 모두 합친 값이 자신의 값이 되도록 수정하는 문제]**

- 전역 변수를 사용하는게 포인트임. 사용안하고 풀어보려다가 시간 개오래 걸림.

int current\_sum=0;

void modify(TreeNode\* root){

if(!root) return;

modify(root->right);

root->val+=current\_sum;

current\_sum=root->val;

modify(root->left);

return;

}

TreeNode\* bstToGst(TreeNode\* root) {

modify(root);

return root;

}

[42. **Encode and Decode TinyURL – 내 마음대로 URL을 encode한 후 decode하는 문제]**

**-** 그냥 간단하게 각 문자에 3을 더한 ascii 코드값을 string에 저장하고, decode할 때 3을 빼줌.

string enc;

string dec;

// Encodes a URL to a shortened URL.

string encode(string longUrl) {

enc.clear();

for(int i=0;i<longUrl.size();i++)

enc+=longUrl[i]+3;

enc+='\0';

return enc;

}

// Decodes a shortened URL to its original URL.

string decode(string shortUrl) {

dec.clear();

for(int i=0;shortUrl[i]!='\0';i++)

dec+=shortUrl[i]-3;

return dec;

}

[43. **Insert into a Binary Search Tree – BST에 삽입하는 문제]**

- 자신 보다 큰 노드를 발견할때까지 오른쪽으로 가고, 자신 보다 큰 노드를 만났을 때, 왼쪽 자식이 비어있으면 거기에 넣고, 아니면 왼쪽으로 한칸가서 다시 자신보다 큰 노드를 발견할 때 까지 반복.

void search(TreeNode\* root, int val){

if(root->val<val){

if(!root->right){

root->right=new TreeNode(val);

return;

}

search(root->right,val);

}

else if(!root->left){

root->left= new TreeNode(val);

return;

}

else search(root->left,val);

}

TreeNode\* insertIntoBST(TreeNode\* root, int val) {

search(root,val);

return root;

}

- 근데 이문제 iterative로 풀면 O(1) space만 쓰면 된다. recursive로 풀면 stack에 함수가 쌓여서 O(n) space가 필요하다.

[44. **Maximum Binary Tree – 주어진 벡터에 대해서 맥스 값을 root로 왼쪽 오른쪽을 나누고, 같은 방식으로 트리를 만드는 문제] 2020-01-30 4:20**

**-** 처음에는 iterator로 풀려고 했는데, 자꾸 bad\_allocation 이라는 runtime error가 발생해서, 그냥 인덱스 가지고 for문 돌리는 것으로 바꿈

- 내생각에 이거는 O(n^2)에 푸는 건데, Discussion을 확인해 봐야겠음.

TreeNode\* constructTree(vector<int>& nums){

int maximum=INT\_MIN;

int max\_index=0;

for(int i=0;i<nums.size();i++)

if(maximum<nums[i]){

maximum=nums[i];

max\_index=i;

}

vector<int> left;

vector<int> right;

for(int i=0;i<max\_index;i++)

left.push\_back(nums[i]);

for(int i=max\_index+1;i<nums.size();i++){

right.push\_back(nums[i]);

}

TreeNode\* root=new TreeNode(maximum);

if(max\_index!=0) root->left=constructTree(left);

if(max\_index!=nums.size()-1) root->right=constructTree(right);

return root;

}

TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

return constructTree(nums);

}

- 다음 방법은 stack을 이용한 건데, 이해는 했는데, 이게 O(n)인지는 좀 확실치 않음, 스택에 다 넣었다가 빼는 과정도 있어서.. 그래도 O(2n)정도인듯.

TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

vector<TreeNode\*> stk;

for (int i = 0; i < nums.size(); ++i)

{

TreeNode\* cur = new TreeNode(nums[i]);

while (!stk.empty() && stk.back()->val < nums[i])

{

cur->left = stk.back();

stk.pop\_back();

}

if (!stk.empty())

stk.back()->right = cur;

stk.push\_back(cur);

}

return stk.front();

}

[45. **Sort the Matrix Diagonally – 주어진 2차원 벡터에 대해서 모든 대각선을 오름차순으로 sort하는 문제] 2020-01-30 4:40**

**-** 쉽게 생각해서 O(n^2)에 풀었는데, 각 원소의 오른쪽 아래원소가 자신보다 작으면 두개를 스왑하는 방식으로, 그것을 주어진 2차원 벡터의 row수만큼 반복함.

vector<vector<int>> diagonalSort(vector<vector<int>>& mat) {

for(int k=0;k<mat.size();k++)

for(int i=0;i<mat.size()-1;i++)

for(int j=0;j<mat[i].size()-1;j++)

if(mat[i][j]>mat[i+1][j+1])

swap(mat[i][j],mat[i+1][j+1]);

return mat;

}

[46. **All Elements in Two Binary Search Trees – 주어진 두개의 BST의 원소들을 오름차순으로 벡터에 집어넣는 문제] 2020-01-30 5:03**

- BST이므로 중위순회를 하면 sort가 됨. 이것을 각각 벡터로 하나씩 가지고 있고, 두개의 벡터의 원소들을 비교해가면서 answer 벡터에 넣어줌.

-마지막에 빈 벡터와 위의 sort 벡터 2개를 swap함으로써 메모리 해제를 해줌.

- 이때 벡터 2개를 사용안하고 벡터 1개에 집어넣고 sort하는 방식도 있긴함. 근데 이러면 O(nlogn)이 걸림. 내 방법은 해봐야 O(4n)임.

void inOrder(TreeNode\* root,vector<int>& bst){

if(!root) return;

inOrder(root->left,bst);

bst.push\_back(root->val);

inOrder(root->right,bst);

}

vector<int> getAllElements(TreeNode\* root1, TreeNode\* root2) {

vector<int> bst1;

vector<int> bst2;

vector<int> answer;

inOrder(root1,bst1);

inOrder(root2,bst2);

vector<int>::iterator left=bst1.begin();

vector<int>::iterator right=bst2.begin();

while(left!=bst1.end()|right!=bst2.end()){

if(left==bst1.end()){

while(right!=bst2.end()){

answer.push\_back(\*right);

right++;

}

break;

}

if(right==bst2.end()){

while(left!=bst1.end()){

answer.push\_back(\*left);

left++;

}

break;

}

if(\*left>\*right){

answer.push\_back(\*right);

right++;

}

else{

answer.push\_back(\*left);

left++;

}

}

vector<int>().swap(bst1);

vector<int>().swap(bst2);

return answer;

}

[47. **Construct Binary Search Tree from Preorder Traversal – preorder로 주어진 벡터로 트리를 만드는 문제] 2020-01-30 5:34**

**-** 아 이거 학교에서 했던건데 알고리즘이 떠오르지 않아서 되게 과격한 방식으로 풀었음. Discussion을 좀 확인해봐야겠음.

- 일단 나의 알고리즘은, 벡터의 첫번째 원소가 항상 root이고 이제 벡터를 왼쪽 서브트리와 오른쪽 서브트리로 나눠야 하는데, 첫번째 원소보다 큰 값을 가진 첫번째 원소가 오른쪽 서브트리의 루트고, 왼쪽 서브트리의 루트는 두번째 원소임.

- 위와 같은 방식으로 벡터 2개를 구해서 각각 재귀해줌. 그런데 이거 iterator로 풀수 있을 것 같은 데 왜 안풀리냐 ㅡㅡ, 그래서 그냥 index로 품. 속도 보장 못함.

TreeNode\* helper(vector<int> preorder){

if(preorder.size()==0) return nullptr;

TreeNode\* root=new TreeNode(preorder[0]);

int next\_index=preorder.size();

for(int i=1;i<preorder.size();i++)

if(preorder[i]>preorder[0]){

next\_index=i;

break;

}

vector<int> left;

vector<int> right;

for(int i=1;i<next\_index;i++)

left.push\_back(preorder[i]);

for(int i=next\_index;i<preorder.size();i++)

right.push\_back(preorder[i]);

if(right.size()!=0)

root->right=helper(right);

if(left.size()!=0)

root->left=helper(left);

return root;

}

TreeNode\* bstFromPreorder(vector<int>& preorder) {

return helper(preorder);

}

- 이런식으로 하면 iterator 쓸수 있을 듯.

TreeNode\* bstFromPreorder(vector<int>& preorder) {

return helper(preorder.begin(), preorder.end());

}

TreeNode \* helper(vector<int>::iterator begin, vector<int>::iterator end) {

if (begin == end) {

return nullptr;

}

auto node = new TreeNode(\*begin);

auto right = upper\_bound(begin + 1, end, \*begin);

node->left = helper(begin + 1, right);

node->right = helper(right, end);

return node;

}

[48. **Letter Tile Possibilities – 주어진 string으로 만들 수 있는 모든 sequence의 개수를 출력하는 문제]**

**-** 이 문제 어렵다 ㄷㄷ.

- Discussion 보고 배껴서 풀긴 했는데, 아직도 이해가 잘 안감.

unordered\_set<string> answer;

unordered\_map<char,int> table;

void helper(string current,int length){

if(current!="")

answer.insert(current);

if(current.size()>=length)

return;

for(unordered\_map<char,int>::iterator iter=table.begin();iter!=table.end();iter++){

if(iter->second>0){

iter->second--;

current+=iter->first;

helper(current,length);

current.pop\_back();

iter->second++;

}

}

}

int numTilePossibilities(string tiles) {

for(int i=0;i<tiles.size();i++)

table[tiles[i]]++;

helper("",tiles.size());

return answer.size();

}

[49. **Delete Leaves With a Given Value – 주어진 target value와 동일한 값을 가지고 있는 leaf node를 삭제하는 문제] 2020-01-31 8:50**

- 리프노드를 삭제했을 때, 그 부모가 리프노드가 되고 또 리프노드가 된 그 노드가 target value를 가지고 있으면 또 지워줘야함.

- 또한 트리 전체의 root 까지 지워야 하는 경우를 대비하여, root를 가리키는 임의의 노드를 생성하여, 그것을 가상의 루트로 함. 여기서는 answer node임.

bool end=false;

void remove(TreeNode\* root, int target){

if(!root) return;

if(root->left){

if(root->left->val==target && !root->left->left && !root->left->right){

root->left=nullptr;

end=false;

}

}

if(root->right){

if(root->right->val==target && !root->right->left && !root->right->right){

root->right=nullptr;

end=false;

}

}

remove(root->left,target);

remove(root->right,target);

}

TreeNode\* removeLeafNodes(TreeNode\* root, int target) {

TreeNode\* answer=new TreeNode(target-1);

answer->left=root;

while(!end){

end=true;

remove(answer,target);

}

return answer->left;

}

[50. **Partition Labels – 주어진 string을 분할하는 문제인데, 각 문자는 하나의 partition에만 속할 수 있음. 이렇게 나눌 수 있는 최대의 파티션을 구하고, 각 파티션의 길이를 리턴하는 문제] 2020-01-31 9:20**

- 나는 brute force로 푼거 같음. 현재 고려하는 문자를 cur라고 하고 그 위치를 left로 함. 그 후 가장 오른쪽에 위치한 cur를 찾고 그위치를 right로 하고, left와 right 사이의 문자 중 right 보다 더 오른쪽에 존재하는 문자를 찾아봄. 있으면 그 문자가 right가 됨. 그후 right-left+1이 해당 partition의 길이임.

-위 과정을 주어진 string이 끝날때까지 하면됨.

- O(n^2) solution 인듯.

int rightEnd(string& S,char cur,int left){

int right=0;

for(int i=left;i<S.size();i++)

if(S[i]==cur) right=i;

return right;

}

vector<int> partitionLabels(string S) {

vector<int> answer;

char cur=S[0];

int left=0,right=0;

int i=0;

while(i<S.size()){

cur=S[i];

left=i;

right=rightEnd(S,cur,left);

for(int j=left+1;j<right;j++){

int temp\_right=rightEnd(S,S[j],j);

if(right<temp\_right){

right=temp\_right;

}

}

answer.push\_back(right-left+1);

i=right+1;

}

return answer;

}

-생각해보니 다르게 풀 수 있을 것 같음. map 써가지고 O(n)시간안에 푸는건데, O(n)이긴 한데 좀 오래걸리는 O(n)임.

vector<int> partitionLabels(string S) {

unordered\_map<char,vector<int>> table;

for(int i=0;i<S.size();i++){

if(table.count(S[i])>0) table[S[i]][1]=i;

else table[S[i]]=vector<int>{i,i};

}

int i=0;

int left,right;

char cur;

vector<int> answer;

while(i<S.size()){

cur=S[i];

left=table.find(S[i])->second[0];

right=table.find(S[i])->second[1];

for(int j=left+1;j<right;j++){

if(table.find(S[j])->second[1]>right){

right=table.find(S[j])->second[1];

}

}

answer.push\_back(right-left+1);

i=right+1;

}

return answer;

}

- void **swap**(a, b) : a와 b를 교환

- T **exchange**(T t, U u) : t에 u를 할당, 원래 t를 반환

- 이거는 O(n) solution 임. Discussion에서 봄. 확실히 간단함.

vector<int> partitionLabels(string S) {

vector<int> res, pos(26, 0);

for (auto i = 0; i < S.size(); ++i) pos[S[i] - 'a'] = i;

for (auto i = 0, idx = INT\_MIN, last\_i = 0; i < S.size(); ++i) {

idx = max(idx, pos[S[i] - 'a']);

if (idx == i) res.push\_back(i - exchange(last\_i, i + 1) + 1);

}

return res;

}

[51. **Reveal Cards In Increasing Order – 주어진 단계에 따랐을 때, 오름차순으로 카드가 정렬될 수 있도록 하는 문제. 설명을 봐야됨] 2020-01-31 10:09**

In a deck of cards, every card has a unique integer.  You can order the deck in any order you want.

Initially, all the cards start face down (unrevealed) in one deck.

Now, you do the following steps repeatedly, until all cards are revealed:

1. Take the top card of the deck, reveal it, and take it out of the deck.
2. If there are still cards in the deck, put the next top card of the deck at the bottom of the deck.
3. If there are still unrevealed cards, go back to step 1.  Otherwise, stop.

Return an ordering of the deck that would reveal the cards in **increasing order.**

The first entry in the answer is considered to be the top of the deck.

- 이 문제 처음에는 가늠도 안갔음. 어떻게 풀어야 하는지.

- Discussion을 참고했더니, 하는 단계를 역으로 하는 해법이 있었음. 세상의 천재는 많음. ㄹㅇ…

vector<int> deckRevealedIncreasing(vector<int>& deck) {

sort(deck.begin(),deck.end());

vector<int> answer;

while(deck.size()){

// second step

if(answer.size()){

answer.insert(answer.begin(),answer.back());

answer.pop\_back();

}

//first step

answer.insert(answer.begin(),deck.back());

deck.pop\_back();

}

return answer;

}

[52. **All Paths From Source to Target – 그래프의 모든 paths를 반환하는 문제] 2020-02-04 11:43**

**-** “from node 0 to node N-1”이 말이 키포인트 임. 문제를 잘 읽어야 함.

- dfs로 푸는건데 생각해내기 쉽지 않음.

- 와 코딩 열심히 해야겠따 ㄹㅇ...

void dfs(vector<vector<int>>& graph,vector<vector<int>>& answer,vector<int> table,int cur){

table.push\_back(cur);

if(cur==graph.size()-1){

answer.push\_back(table);

return;

}

for(int i=0;i<graph[cur].size();i++)

dfs(graph,answer,table,graph[cur][i]);

}

vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {

vector<vector<int>> answer;

vector<int> table;

dfs(graph,answer,table,0);

return answer;

}

[53. **Binary Tree Pruning – 1을 포함하지 않는 모든 subtree를 제거하는 문제] 2020-02-04 11:51**

**-** 이 문제는 리프노드이면서 value가 0인 노드를 리프에서부터 제거해서 루트로 올라오면 된다.

- 조건에 부합하여 제거된 노드의 부모가 다시 조건에 부합하게 되면, 그역시 삭제해야 한다.

void helper(TreeNode\* root){

if(!root) return;

helper(root->left);

helper(root->right);

if(root->left && !root->left->left && !root->left->right && root->left->val==0)

root->left=nullptr;

if(root->right&& !root->right->left && !root->right->right && root->right->val==0)

root->right=nullptr;

}

TreeNode\* pruneTree(TreeNode\* root) {

helper(root);

return root;

}

[54. **All Possible Full Binary Trees – 어려워서 못풀었음.]**

**-**  DFS 랑 BFS로 풀라그랬는데 실패함. discussion봤는데 divide and conquer 길래 읽어봐도 뭔소린지 모르겠음. 그래서 youtube를 참고하려고 함.

- youtube에도 중국인 설명밖에 없음. 모르겠다 이문제 ㄷㄷ;

- 이해했다. 1개로 만들수 있는 트리, 3개로 만들 수 있는 트리를 계속저장해가면서 불러오는거 같다.

- 이게 정답인데, 이해할려면 시간 좀 걸릴 듯.

unordered\_map<int, vector<TreeNode\*>> cache;

vector<TreeNode\*> allPossibleFBT(int N) {

vector<TreeNode\*> res;

if(cache[N].size() != 0) return cache[N];

if(N == 1) {

res.push\_back(new TreeNode(0));

} else {

for (int i = 1; i < N; i += 2) {

int l = i, r = N - i - 1;

for (TreeNode\* left : allPossibleFBT(l)) {

for (TreeNode\* right : allPossibleFBT(r)) {

TreeNode \* root = new TreeNode(0);

root->left = left;

root->right = right;

res.push\_back(root);

}

}

}

}

cache[N] = res;

return res;

}

[55. **Find and Replace Pattern – 주어진 pattern과 같은 순열을 가지는 word들을 반환하는 문제]**

- 나는 unordered\_map을 이용해서, 주어진 string의 pattern을 숫자로 나타내었음. 그리고 word들을 숫자로 나타낸 것과 비교함.

vector<string> findAndReplacePattern(vector<string>& words, string pattern) {

vector<string> answer;

unordered\_map<char,int> p\_table;

int count=1;

for(int i=0;i<pattern.size();i++){

if(!p\_table[pattern[i]]) p\_table[pattern[i]]=count++;

}

string answer\_pattern;

for(int i=0;i<pattern.size();i++){

answer\_pattern+=p\_table[pattern[i]]+48;

}

unordered\_map<char,int> match;

count=1;

string temp;

for(string word:words){

for(int i=0;i<word.size();i++){

if(!match[word[i]]) match[word[i]]=count++;

}

for(int i=0;i<word.size();i++){

temp+=match[word[i]]+48;

}

if(temp.compare(answer\_pattern)==0)

answer.push\_back(word);

count=1;

temp="";

match.clear();

}

return answer;

}

[56. **Find Elements in a Contaminated Binary Tree – 오염된 트리를 재구성하고 target value를 찾는 문제]**

1. root.val == 0
2. If treeNode.val == x and treeNode.left != null, then treeNode.left.val == 2 \* x + 1
3. If treeNode.val == x and treeNode.right != null, then treeNode.right.val == 2 \* x + 2

- 위 공식대로 트리를 재구성한 후 target을 찾으면 된다.

- set을 이용하여 모든 노드 값을 저장하는 것이 target을 더 빨리 찾는 방법이다. 왜냐하면 순회를 하지 않아도 되기 때문이다.

TreeNode\* answer;

unordered\_set<int> value\_set;

void recover(TreeNode\* root){

if(!root) return;

if(root->left)

root->left->val=2\*root->val+1;

if(root->right)

root->right->val=2\*root->val+2;

value\_set.insert(root->val);

recover(root->left);

recover(root->right);

}

FindElements(TreeNode\* root) {

root->val=0;

answer=root;

recover(root);

}

bool helper(TreeNode\* root,bool flag,int target){

if(!root) return flag;

if(root->val==target) return true;

flag=helper(root->left,flag,target);

flag=helper(root->right,flag,target);

return flag;

}

bool find(int target) {

//return helper(answer,false,target);

return value\_set.count(target);

}

[57. **Matrix Block Sum – 주어진 조건에 맞는 모든 원소를 더해 벡터로 만드는 문제]**

**-** 일단 문제 이해하는데 좀 걸림.

- for문을 4개나 썼는데, 더 빨리 풀 수 있는방법이 있을 것 같음. 전에 합해놓은 값을 그대로 이용한다던가 하는…

vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int k) {

vector<vector<int>> res;

vector<int> temp\_answer;

int temp\_sum=0;

for(int i=0;i<mat.size();i++){

for(int j=0;j<mat[i].size();j++){

for(int r=i-k;r<=i+k;r++){

if(r<0 || r>=mat.size())

continue;

for(int c=j-k;c<=j+k;c++){

if(c<0||c>=mat[i].size())

continue;

temp\_sum+=mat[r][c];

}

}

temp\_answer.push\_back(temp\_sum);

temp\_sum=0;

}

res.push\_back(temp\_answer);

temp\_answer.clear();

}

return res;

}

[58. **Minimum Add to Make Parentheses Valid – 주어진 string에서 괄호가 valid하게 되기위해 필요한 괄호의 개수를 반환하는 문제]**

**-** 그냥 각 케이스의 경우를 생각해서 풀었음.

int minAddToMakeValid(string S) {

int res=0;

int left=0,right=0;

bool left\_first=false;

for(int i=0;i<S.size();i++){

if(S[i]=='('){

left++;

res++;

continue;

}

if(S[i]==')'){

if(left){

left--;

res--;

continue;

}

res++;

continue;

}

}

return res;

}

[59. **Score After Flipping Matrix – 주어진 matrix를 자유롭게 toggle 즉, 0과 1을 뒤바꾸어 이진수로 취급할 때의 최대 합을 구하는 문제]**

**-** 일단 맨 첫 비트가 1이되도록 각 row를 toggle해주고, 이후 모든 columns을 탐색하면서 0의 개수가 1의 개수보다 많으면 toggle한다.

int sumUp(vector<vector<int>> temp){

int sum=0;

int power=0;

for(int i=0;i<temp.size();i++){

for(int j=temp[i].size()-1;j>=0;j--,power++){

if(temp[i][j])

sum+=pow(2,power);

}

power=0;

}

return sum;

}

int matrixScore(vector<vector<int>>& A) {

//row toggle

for(int i=0;i<A.size();i++){

if(A[i][0]==0){// if leading bit is 1, then toggle it.

for(int j=0;j<A[i].size();j++)

A[i][j]^=1;

}

}

//col toggle when 0s are more than 1s in the current column.

int zeros=0;

int ones=0;

for(int i=0;i<A[0].size();i++){

for(int j=0;j<A.size();j++){

if(A[j][i]==0) zeros++;

else ones++;

}

if(zeros>ones){ // toggle the column.

for(int j=0;j<A.size();j++)

A[j][i]^=1;

}

ones=0;

zeros=0;

}

return sumUp(A);

}

[60. **Maximum Level Sum of a Binary Tree – 노드들의 총합이 가장 큰 minimum level을 찾는 문제]**

- 나는 pair와 map을 이용하여 bfs로 풀었다. pair 접근할 때, -> 로 접근하는 것이 아니라 ‘.’ 마침표로 접근한다. root.first 이렇게

- 생각해보니 dfs로 풀면 훨씬 빠를 것 같다. que도 pair도 필요없이 재귀로만 풀 수 있을 듯.

void bfs(queue<pair<TreeNode\*,int>>& que,unordered\_map<int,int>& level\_sum){

if(que.empty()) return;

pair<TreeNode\*,int> root=que.front();

que.pop();

level\_sum[root.second]+=root.first->val;

if(root.first->left) que.push(make\_pair(root.first->left,root.second+1));

if(root.first->right) que.push(make\_pair(root.first->right,root.second+1));

bfs(que,level\_sum);

}

int maxLevelSum(TreeNode\* root) {

queue<pair<TreeNode\*,int>> que;

int sum=root->val;

que.push(make\_pair(root,1));

unordered\_map<int,int> level\_sum;

bfs(que,level\_sum);

int max\_level=1;

for(unordered\_map<int,int>::iterator iter=level\_sum.begin();iter!=level\_sum.end();iter++){

if(iter->second>sum){

sum=iter->second;

max\_level=iter->first;

}

}

return max\_level;

}

[61. **Path In Zigzag Labelled Binary Tree – 레벨마다 내림차순, 오름차순이 바뀌는 포화 이진트리에서 주어진 label까지의 경로를 출력하는 문제]**

- 주어진 트리에 숫자가 순서대로는 들어가 있으므로, 부모노드는 자신의 value/2라는 것을 이용함.

-이때 홀수 레벨은 오름차순이고, 짝수 레벨은 내림차순이므로 계산을 따로 해줌.

vector<int> pathInZigZagTree(int label) {

//leftmost node of level n is 2^(n-1) when root's level is 1.

//rightmost node of level n is 2^n -1.

//find rightmost node greater than label. Then we can find the level of label.

//if n is odd, then the nodes of the level have normal sequence, otherwise reverse sequence.

//In normal case, parents node's value is label / 2, we can use it.

vector<int> answer;

answer.push\_back(label);

int level=1;

while(pow(2,level)-1<label) level++; //find the label's level.

int temp=label;

int position=0;

while(level>1){

if(level%2==0) position=(pow(2,level)-1-temp)/2; //find next postion.

else position=(temp-pow(2,level-1))/2;

level--; // we find the postion of the upper level. so we don't need current level anymore.

if(level%2==0) temp=pow(2,level)-1-position;

else temp=pow(2,level-1)+position;

answer.push\_back(temp);

}

return {answer.rbegin(),answer.rend()};

}

[62. **Maximum Nesting Depth of Two Valid Parentheses Strings – 최소 깊이를 가지도록 괄호들을 두개의 그룹으로 나누는 문제]**

- 문제 설명부터 이해하기 힘듬. 주어진 스트링에서 나열된 순서와 상관 없이, 왼쪽 오른쪽 짝만 맞으면 한 그룹에 들어갈 수 있음. 이렇게 짝을 맞추면서 두개의 그룹으로 나눌 때, 최소의 깊이를 가지게 하면 됨.

- discussion을 보고서 문제를 이해했고, 답도 거기서 힌트를 얻었음.

- 일단 각 괄호의 깊이를 모두 구함. 그리고 홀수는 A에 짝수는 B에 넣음. 즉, 깊이가 하나 깊어질 때마다 다른 그룹에 넣어주면 깊이가 최소가 됨.

- 이런 문제는 누가 생각해내는 거야…

vector<int> maxDepthAfterSplit(string seq) {

vector<int> depth;

int cur\_depth=0;

for(int i=0;i<seq.size();i++)

if(seq[i]=='(') depth.push\_back(++cur\_depth);

else depth.push\_back(cur\_depth--);

vector<int> answer;

for(int i=0;i<depth.size();i++)

depth[i]%2? answer.push\_back(0):answer.push\_back(1);

return answer;

}

- for문을 두개 안쓰고, 하나만 쓰면서 깊이를 구하자마자 바로 answer에 넣을 수도 있음.

vector<int> maxDepthAfterSplit(string seq) {

vector<int> depth;

vector<int> answer;

int cur\_depth=0;

for(int i=0;i<seq.size();i++){

if(seq[i]=='(') depth.push\_back(++cur\_depth);

else depth.push\_back(cur\_depth--);

depth[i]%2? answer.push\_back(0):answer.push\_back(1);

}

return answer;

}

[63. **Distribute Coins in Binary Tree – 모든 노드가 1개의 coin을 갖도록 분배할 때, 코인이 움직이는 개수를 구하는 문제]**

**-** 한번에 한칸 움직일 수 있고, 그때 움직인 coin의 개수를 총합한 것이 답이다.

- bottom up으로 풀려고 했고, 실제로 discussion의 방법들과 동일했는데, 나는 중간에 막혔다.

- discussion을 참고하여 풀었다. abs로 move를 더해주는 것이 key였다.

int getCoin(TreeNode\* root,int& move){

if(!root) return 0;

root->val+=getCoin(root->left,move);

root->val+=getCoin(root->right,move);

move+=abs(root->val-1);

return root->val-1;

}

int distributeCoins(TreeNode\* root) {

int move=0;

getCoin(root,move);

return move;

}

[64. **Queens That Can Attack the King – king과 같은 row,column,diagnal 에 있는 가장 가까운 queen의 좌표를 찾는 문제]**

- 처음에는 BFS를 사용할까 생각했는데, 그냥 8x8 배열 만들어 퀸 좌표를 모두 넣은 다음, 킹에서부터 가장 가까운 것 찾는 for문8번돌리는게 시간상 빠를 것 같아서 그렇게 함.

vector<vector<int>> queensAttacktheKing(vector<vector<int>>& queens, vector<int>& king) {

vector<vector<int>> answer;

vector<vector<bool>> chess;

chess.assign(8,vector<bool>(8)); //2차원 벡터 메모리 초기화

for(int i=0;i<queens.size();i++)

chess[queens[i][0]][queens[i][1]]=true;

int x=king[0];

int y=king[1];

//left

for(int i=x-1;i>=0;i--){

if(chess[i][y]){

answer.push\_back(vector<int>{i,y});

break;

}

}

//right

for(int i=x+1;i<chess.size();i++){

if(chess[i][y]){

answer.push\_back(vector<int>{i,y});

break;

}

}

//up

for(int i=y-1;i>=0;i--){

if(chess[x][i]){

answer.push\_back(vector<int>{x,i});

break;

}

}

//down

for(int i=y+1;i<chess.size();i++){

if(chess[x][i]){

answer.push\_back(vector<int>{x,i});

break;

}

}

//up-left

for(int i=x-1,j=y-1;i>=0 && j>=0;i--,j--){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//up-right

for(int i=x+1,j=y-1;i<chess.size() && j>=0;i++,j--){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//down-left

for(int i=x-1,j=y+1;i>=0&&j<chess.size();i--,j++){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//down-right

for(int i=x+1,j=y+1;i<chess.size()&&j<chess.size();i++,j++){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

return answer;

}

[65. **Battleships in a Board – 인접하지 않은 the number of rows of Xs or cols of Xs 를 찾는 문제]**

- 푸는 건 어렵지 않았음. 그냥 top-left 부터 시작해서 X만날 때 연이은 X를 모두 ‘.’으로 바꾸어주면 됨.

int countBattleships(vector<vector<char>>& board) {

int res=0;

int temp;

for(int i=0;i<board.size();i++){

for(int j=0;j<board[i].size();j++){

if(board[i][j]=='X'){

res++;

board[i][j]='.';

temp=j+1;

while(temp<board[i].size()&&board[i][temp]=='X') board[i][temp++]='.';

temp=i+1;

while(temp<board.size()&&board[temp][j]=='X') board[temp++][j]='.';

}

}

}

return res;

}

- follow up에 O(1) memory without modifying the value of board가 있어서 discussion을 들어가 봄. head of ships 를 찾는 문제로 변형 됨.

int countBattleships(vector<vector<char>>& board) {

if(board.empty() || board.front().empty()) return 0;

int res=0;

for(int i=0;i<board.size();i++)

for(int j=0;j<board[i].size();j++) //just find ship's head.

res+= board[i][j]=='X' && (i==0|| board[i-1][j]!='X') && (j==0||board[i][j-1]!='X');

return res;

}

[66. **Count Square Submatrices with All Ones – 사각형이 몇 개 존재하는 지 구하는 문제]**

- 처음에는 top-left를 기준으로 만들어지는 square의 개수를 만드려고 다음과 같은 코드를 짰음.

- 그런데 time limit exceeded 함.

- 짱구를 더 굴려 보자구.

int isSquare(int x, int y, int size,vector<vector<int>> matrix){

int res=0; //how many square current coordinates can make.

int temp=1; // current size of square +1

bool flag=true; // whether current considering square is valid square or not.

while(temp<size){

cout<<"x : "<<x<<" y : "<<y<<" temp : "<<temp<<endl;

for(int j=y;j<=y+temp;j++){

if(x+temp>=matrix.size()||j>=matrix[0].size()||!matrix[x+temp][j]){

flag=false;

break;

}

}

cout<<"pass1"<<endl;

// thr bottom-right value doesn't need to be considered twice. so i<x+temp

for(int i=x;i<x+temp;i++){

if(y+temp>=matrix[0].size()||i>matrix.size()||!matrix[i][y+temp]){

flag=false;

break;

}

}

if(!flag) break;

cout<<"pass2"<<endl;

res++;

temp++;

}

return res;

}

int countSquares(vector<vector<int>>& matrix) {

int res=0;

int minSize=min(matrix.size(),matrix[0].size());

for(int i=0;i<matrix.size();i++){

for(int j=0;j<matrix[i].size();j++){

if(matrix[i][j]){

res+=isSquare(i,j,minSize,matrix) + 1; // the rightmost 1 is for square of side 1

}

}

}

return res;

}

- 특정되는 사각형 수가 크기가 N X N 일때, 총 사각형의 개수는1^2 + 2^2 + … + N^2 임.

을 참고하면 될 듯.

- 위에 것도 접근이 너무 어려움. DP로 푸는 거였음. 하 나는 DP가 너무 약해…

- 근데 이 DP는 진짜 생각해내기 너무 어려운 것 같은데?

- 내 생각에는 이건 그냥 암기다.

int countSquares(vector<vector<int>>& matrix) {

int res=0;

for(int i=0;i<matrix.size();i++){

for(int j=0;j<matrix[i].size();j++){

if(matrix[i][j] && i && j) matrix[i][j]+=min(matrix[i][j-1],min(matrix[i-1][j-1],matrix[i-1][j]));

res+=matrix[i][j];

}

}

return res;

}

1. if matrix[i][j] == 0, skip
2. use bottom right to count the square, if a 2 \* 2 grid like  
   " A B "  
   " C X "  
   every character means the nums of the square that position can represent. For example A = 2 represents 2 squares in the "A" position, one is itself and the other is a 2 \* 2 square , a 2 \* 2 square means A is in the position like  
   " 1 1 "  
   " 1 A "  
   so X should be Math.min(A+1, B+1, C+1) which ensures X is in the postion like  
   " 1 ... 1 " //k \* k, k = Math.min(A+1, B+1, C+1)  
   " 1 .... . "  
   " 1 ... X "  
   for example  
     
   in the yellow grid, we know (D4=3) (E4=2) (D5=2) and then how to calculate the (E5).  
   D4 = 3 means the largest sqaure it can represent is the black one(3\* 3) === we can find 3 sqaures which its bottom right is D4 (1\* 1, 2\* 2, 3\* 3 squares).  
   Then the same as E4(red square) and D5(green one). When it comse to E5, (E5) = Math.min(D4+1, E4+1, D5+1) means the largest square E5 can represent which is (C3 ~ E5)
3. because we are using the bottom right to count the square, the "A B C" will always be calculated before "X". So just go through the grid then and add all nums in the grid

[67. **Spiral Matrix III – 나선모양으로 걸으면서 거친 좌표를 차례대로 벡터에 넣는 문제]**

- 이 문제 왜캐 어려웠지… Discussion을 안보고 풀라고 애썼는 데, 실패했다.

- 이 문제의 포인트는 boundary 바깥으로 나가는 것을 고려하지 않아도 된다는 것이었다. 나선형만 제대로 짜 놓으면 어차피 다시 안으로 들어올 것이고, 주어진 boundary안에 좌표가 있을 때만 answer에 더해주면 되는 것이었다.

vector<vector<int>> spiralMatrixIII(int R, int C, int r0, int c0){

vector<vector<int>> res={{r0,c0}};

int max\_step=R\*C;

vector<int> cur\_pos={r0,c0};

vector<vector<int>> dir={{0,1},{1,0},{0,-1},{-1,0}};

int head=0;

int cur\_step=1;

int how\_many\_step=0;

while(cur\_step<max\_step){

if(head==0 || head==2) how\_many\_step++; // when east or west, the step is added one.

for(int i=0;i<how\_many\_step;i++){ //how many step we have to go before turning right.

//by using the limit R and C, we don't need to consider a walk around outside.

//because it will be returned into the boundary. we just need to check the walk occurs in the boundary or not.

cur\_pos[0]+=dir[head][0];

cur\_pos[1]+=dir[head][1];

if(0<=cur\_pos[0] && cur\_pos[0]<R && 0<=cur\_pos[1] && cur\_pos[1]<C){

cur\_step++;

res.push\_back(cur\_pos);

}

}

head=(head+1)%4;

}

return res;

}

[68. **Reduce Array Size to The Half – 배열의 길이를 반 이하로 만들기 위하여 삭제해야하는 원소의 최소개수를 구하는 문제. 이때 각 원소는 중복될 수 있음]**

- 내가 생각한 방법의 point는 각 원소의 개수대로 sort를 하는 것이었다. 그래서 pair를 이용했다.

- map 만들때 O(n) sort할 때 O(nlogn)이 걸렸고, 마지막에 length 찾을때 O(n)이 걸렸으므로 O(nlogn)알고리즘 인 것 같다.

- 주의할 점음 pair는 ->이 아니라 . 으로 first, second에 접근한다.

- iterator의 위치를 변경할때는 advance(iter,3)의 방식으로 3번째 다음 인덱스로 iter를 옮길 수 있고, iter간의 거리도 distance(iter1, iter2) 의 형식으로 구할 수 있다.

- 추가로 sort할 때 sort(iter.begin(),iter.end(),greater<int>()); 의 형식으로 하면 내림차순 정렬이 가능하다.

int minSetSize(vector<int>& arr) {

unordered\_map<int,int> table;

vector<int> answer\_array;

for(int ele : arr){

table[ele]++;

}

vector<pair<int,int>> sorted\_pair; //(count,number)

for(unordered\_map<int,int>::iterator iter=table.begin();iter!=table.end();iter++)

sorted\_pair.push\_back(make\_pair(iter->second,iter->first));

sort(sorted\_pair.begin(),sorted\_pair.end());

int half=arr.size()/2; // size is always even.

int current\_length=0;

for(int i=sorted\_pair.size()-1;i>=0;i--){

current\_length+=sorted\_pair[i].first;

answer\_array.push\_back(sorted\_pair[i].second);

if(current\_length>=half) break;

}

return answer\_array.size();

}

[69. **Counting Bits - 0부터 주어진 숫자까지의 수들의 bit 표현에서 1의 개수를 세어 vector에 넣는 문제]**

**-** 나는 그냥 bitset을 이용했다 bitset<32>(i).count()로.

vector<int> countBits(int num) {

vector<int> answer;

for(int i=0;i<=num;i++)

answer.push\_back(bitset<32>(i).count());

return answer;

}

- 다음과 같은 logic으로도 풀 수 있다.

There is one imporant observation we can make about the number of bits in each number.

1. Each Power of 2 has exactly only 1 bit. (2 : 0010 , 4: 0100, 8:1000, 16:10000)
2. Each number after the power of 2 follows a pecular pattern :  
   0 → 0  
   1 → 0  
   2 → 1 + dp[0] Nearest Power of 2  
   3 → 1 + dp[1] 1 greater than nearest  
   4 → 1 + dp[0] Nearest  
   5 → 1+ dp[1] 1 greater than nearest  
   6 → 1+ dp[2] 2 greater than nearest  
   7 → 1+ dp[3] 3 greater than nearest  
   8 → 1+ dp[0] Nearest  
   9 → 1+ dp[1]  
   10 → 1+ dp[2]  
   11 → 1+ dp[3]  
   12 → 1+ dp[4]

You can easily see the pattern here.

[70. **Complex Number Multiplication – 복소수 2개를 주고 곱하여 string으로 출력하는 문제]**

**-** 그냥 각 복소수의 실수와 허수부분을 나누어 곱해주는 방식으로 했는데, 부호 구분하는 데서 오래 걸렸다.

string complexNumberMultiply(string a, string b) {

string answer;

vector<int> a\_sub;

vector<int> b\_sub;

string temp="";

int i=1;

if(a[0]=='-'){

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i=0;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp));

}

temp.clear();

if(b[0]=='-'){

i=1;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i=0;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp));

}

i=a.size()-2;

temp.clear();

while('0'<=a[i] && a[i]<='9') i--;

if(a[i]=='-'){

i++;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i++;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stod(temp));

}

i=b.size()-2;

temp.clear();

while('0'<=b[i] && b[i]<='9') i--;

if(b[i]=='-'){

i++;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i++;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp));

}

answer+=std::to\_string(a\_sub[0]\*b\_sub[0]+a\_sub[1]\*b\_sub[1]\*-1);

answer+='+';

answer+=std::to\_string(a\_sub[1]\*b\_sub[0]+a\_sub[0]\*b\_sub[1]);

answer+='i';

return answer;

}

- find 함수 이용 간단하게 풀 수 있는 discussion이 있었다.

public:

string complexNumberMultiply(string a, string b) {

pair<int, int> av = parse(a);

pair<int, int> bv = parse(b);

int ra = av.first \* bv.first - av.second \* bv.second;

int rb = av.first \* bv.second + av.second \* bv.first;

return to\_string(ra) + "+" + to\_string(rb) + "i";

}

pair<int, int> parse(const string& a) {

int plus = find(a.begin(), a.end(), '+') - a.begin();

int i = find(a.begin(), a.end(), 'i') - a.begin();

int ra = stoi(a.substr(0, plus));

int rb = stoi(a.substr(plus + 1, i - plus));

return {ra, rb};

}

[71. **Fizz Buzz Multithreaded -Thread 사용하는 문제]**

- 덕분에 thread 공부했다. 하지만 나는 풀 수 없었기에, discussion 보고 thread 어떻게 쓰는지 배웠다.

private:

int n;

int count;

mutex m;

condition\_variable cv;

public:

FizzBuzz(int n) {

this->n = n;

this->count = 1;

}

void fizz(function<void()> printFizz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 3 != 0 || count % 5 == 0))

cv.wait(lock);

if (count > n) return;

printFizz();

++count;

cv.notify\_all();

}

}

void buzz(function<void()> printBuzz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 != 0 || count % 3 == 0))

cv.wait(lock);

if (count > n) return;

printBuzz();

++count;

cv.notify\_all();

}

}

void fizzbuzz(function<void()> printFizzBuzz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 != 0 || count % 3 != 0))

cv.wait(lock);

if (count > n) return;

printFizzBuzz();

++count;

cv.notify\_all();

}

}

void number(function<void(int)> printNumber) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 == 0 || count % 3 == 0))

cv.wait(lock);

if (count > n) return;

printNumber(count++);

cv.notify\_all();

}

}

[72. **[951] Flip Equivalent Binary Trees – when we choose any node and swap left and right subtrees, we call it a flip operation. Write a function whether two given trees are flip equivalent.]**

- no matter a tree is filped or not, its children have to be same or just changed the order.

- So we can just compare every possible situation – Greedy using DFS.

bool flag=true;

void helper(TreeNode\* root1,TreeNode\* root2){

if(!root1->left && !root1->right && !root2->left && !root2->right)

return;

if(root1->left && root2->left && root1->left->val == root2->left->val)

helper(root1->left,root2->left);

else if(root1->left && root2->right && root1->left->val==root2->right->val)

helper(root1->left,root2->right);

else if(root1->left){

flag=false;

return;

}

if(!flag) return;

else if(root1->right && root2->left && root1->right->val==root2->left->val)

helper(root1->right,root2->left);

else if(root1->right && root2->right && root1->right->val==root2->right->val)

helper(root1->right,root2->right);

else if(root1->right){

cout<<"flag"<<endl;

flag=false;

}

}

bool flipEquiv(TreeNode\* root1, TreeNode\* root2) {

if(!root1 && !root2) return true;

if((root1 && !root2) ||(!root1 && root2)) return false;

helper(root1,root2);

return flag;

}

[73. [**1123] Lowest Common Ancestor of Deepest Leave]**

**-** find deepest level from root. If the deepest depth is same, then the root is the answer.

- if the depth is different. determine which one has deeper depth between left and right child.

- and doing recursive with the child who has deeper detph.

int helper(TreeNode\* root){

if(!root) return 0;

return 1+max(helper(root->left),helper(root->right));

}

TreeNode\* lcaDeepestLeaves(TreeNode\* root) {

int left=helper(root->left);

int right=helper(root->right);

if(left==right) return root;

else if(left>right) return lcaDeepestLeaves(root->left);

else return lcaDeepestLeaves(root->right);

}

[74. **[986]** **Interval List Intersections – Given two sorted closed interval, return the intersection of these two lists.]**

- At first try, I use map to record interval of two lists into one store with using vector in the map for checking every interval’s end.

- I think I just need O(total interval of A and B + map.size()). But time limit exceeded occurs.

class Solution {

public:

vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {

map<int,vector<int>> inter;

vector<vector<int>> answer;

int i;

for(vector<int> ele : A){

for(i=ele[0];i<ele[1];i++){

inter[i]={1,0,0};

}

inter[i]={1,1,0};

}

for(vector<int> ele : B){

for(i=ele[0];i<=ele[1];i++){

if(inter[i].empty()) inter[i]={1,0,0};

else inter[i][0]++;

}

inter[i-1][2]=1;

}

map<int,vector<int>>::iterator iter=inter.begin();

int left=0;

for(;iter!=inter.end();iter++){

if(iter->second[0]==2){

left=iter->first;

while(iter->second[0]==2){

if(iter->second[1]==1 || iter->second[2]==1)

break;

else iter++;

}

answer.push\_back({left,iter->first});

}

}

return answer;

}

};

- I guess the reason why the time limit exceed occurs is because I search all the element in each intervals. So I should figure out how to solve this problem by just using left and right limit of each intervals only.

- the final answer is that if A[i][1] is less than B[j][0], i++, vice versa j++.

- after that we can find overlapping range by two lists. From there, we just determine which one is left and right limit.

- see the code.

class Solution {

public:

vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {

vector<vector<int>> answer;

for(int i=0, j=0;i<A.size() && j<B.size();){

if(A[i][1]<B[j][0]) i++;

else if(A[i][0]>B[j][1]) j++;

else{

answer.push\_back({max(A[i][0],B[j][0]),min(A[i][1],B[j][1])});

if(A[i][1]<B[j][1]) i++;

else j++;

}

}

return answer;

}

};