<알고리즘 기초>

* DFS: 재귀, Stack (백트래킹, 단절선찾기, 단절점찾기, 위상정렬, 사이클 찾기 등)

1. 방문
2. 체크인
3. 도착했는가
4. 다음 이동장소 방문
5. 체크아웃

* BFS: queue (최단경로 찾기, 위상정렬 등)

1. 큐에 시작점을 넣음
2. 큐에서 빼옴
3. 목적지인가?
4. 가능한 곳을 큐에 넣으며 체크인

* 완전 탐색 : 재귀함수
* 정렬 (유일성검사/중복제거/빈도구하기/합집합,교집합구하기/이분탐색)

<시간복잡도>

* 투포인터

While(1)

If s>=n || e>=n break

* 이분탐색

While(s<e)

Int mid = (s+e)/2

If~ s=mid+1

If~ e=mid

* 병합정렬

void Merge\_Sort(int s, int e){

int mid = (s+e)/2;

if(s<e){

Merge\_Sort(s, mid);

Merge\_Sort(mid+1,e);

merge(s, mid, e);

}

}

void merge(int s, int m, int e){

int p1 = s;

int p2=m+1;

int k=s;

while(p1<=m&&p2<=e){

if(person[p1].first<=person[p2].first){

tmp[k]=make\_pair(person[p1].first,person[p1].second);

k++;

p1++;

}

else{

tmp[k]=make\_pair(person[p2].first,person[p2].second);

cnt[person[p2].second]+=p1-s;

k++;

p2++;

}

}

while(p1<=m){

tmp[k]=make\_pair(person[p1].first,person[p1].second);

k++;

p1++;

}

while(p2<=e){

tmp[k]=make\_pair(person[p2].first,person[p2].second);

cnt[person[p2].second]+=p1-s;

k++;

p2++;

}

for(int i=s;i<=e;i++){

person[i].first=tmp[i].first;

person[i].second=tmp[i].second;

}

}

* 피보나치수열: dp

<자료구조>

* 선형 자료구조: 배열, 연결리스트, 스택, 큐
* 비선형 자료구조: 트리, 그래프
* 배열: 방문 체크 주로 사용
* 연결리스트
* 스택
* 큐
* 트리
* 이진트리: 전위순회(현-왼-오), 중위순회(왼-현-오), 후위순회(왼-오-현)

Class Node{

Object data

Node left\_child, right\_child

};

* 힙(Min heap, Max heap): 우선순위 큐. 삽입/삭제 연산.

class minHeap{

public:

vector<int> heap;

void heap\_insert(int x){

heap.push\_back(x);

int idx=heap.size()-1;

while(idx>0){

int parent=(idx-1)/2;

if(heap[parent]>heap[idx]){

int tmp=heap[idx];

heap[idx]=heap[parent];

heap[parent]=tmp;

idx=parent;

}

else{

break;

}

}

}

void heap\_delete(){

if(heap.size()==0){

printf("0\n");

}

else{

printf("%d\n",heap[0]);

heap[0]=heap[heap.size()-1];

heap.pop\_back();

int idx=0;

while(idx<heap.size()){

int left\_c=(idx+1)\*2-1, right\_c=(idx+1)\*2;

int minChild;

if(left\_c<heap.size()&&right\_c<heap.size()){

if(heap[left\_c]<=heap[right\_c]){

minChild=left\_c;

}

else{

minChild=right\_c;

}

}

else if(left\_c<heap.size()&&right\_c>=heap.size()){

minChild=left\_c;

}

else if(left\_c>=heap.size()&&right\_c<heap.size()){

minChild=right\_c;

}

else{

break;

}

if(heap[idx]>heap[minChild]){

int tmp=heap[idx];

heap[idx]=heap[minChild];

heap[minChild]=tmp;

idx=minChild;

}

else{

break;

}

}

}

}

};

* 인덱스 트리/세그먼트 트리

class segmentTree{

public:

long long \*nums;

long long \*tree;

segmentTree(int n){

nums=new long long[n+5];

tree=new long long[(int)(log(n)/log(2))\*n+5];

for(int i=1;i<=pow(2,(int)(log(n)/log(2))+1)+5;i++)

tree[i]=0;

};

long long makeTree(int node, int left, int right){

if(left==right){

return tree[node]=nums[left];

}

int mid=(left+right)/2;

tree[node]+=makeTree(node\*2,left, mid);

tree[node]+=makeTree(node\*2+1,mid+1,right);

return tree[node];

}

long long query(int node, int left, int right, int targetLeft, int targetRight){

if(targetRight<left || targetLeft>right){

return 0;

}

else if(targetLeft<=left&&right<=targetRight){

return tree[node];

}

else{

int mid=(left+right)/2;

long long result= query(node\*2, left, mid, targetLeft, targetRight)+query(node\*2+1, mid+1,right,targetLeft,targetRight);

return result;

}

}

void update(int node, int left, int right, int targetIndex, long long diff){

if(targetIndex<left || targetIndex>right){

return;

}

else{

tree[node]+=diff;

if(left==right){

return;

}

else{

int mid=(left+right)/2;

update(node\*2, left, mid, targetIndex, diff);

update(node\*2+1,mid+1,right,targetIndex, diff);

}

}

}

};

* 트라이

Class Node{

Object data

Node child[]

};

class Node {

public:

Node \*\*child;

bool isEnd;

bool isHit;

Node() {

child = new Node\*[26];

for (int i = 0; i < 26; i++) {

child[i] = NULL;

isEnd = false;

isHit = false;

}

}

};

class Trie {

public:

Node root;

Trie() {

}

void insert(char\* word) {

Node\* cur = &root;

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

if (!cur->child[word[i] - 'A']) {

cur->child[word[i] - 'A'] = new Node();

cur = cur->child[word[i] - 'A'];

}

else {

cur = cur->child[word[i] - 'A'];

}

}

cur->isEnd = true;

}

};

* 해싱
* 셋
* 맵

<정수론>

* 유클리드 호제법: A=q\*B+r gcd(A,B)=gcd(B,r) -> r이 0이 될 때까지

int gcd(int a, int b){

while(b!=0){

int tmp=a%b;

a=b;

b=tmp;

}

return abs(a);

}

* 베주 항등식: ax+by=c일 때 c가 gcd(x,y)로 나눠질 수 있어야 해를 찾을 수 있다
* 확장 유클리드 호제법: r=r’’-qr’, s=s’’-qs’, t=t’’-qt’

void ExtendedEuclidean(int Old\_s, int Old\_t, int Old\_r, int s,int t, int r){

while(r!=0){

int q = Old\_r/r;

int tmp\_r=Old\_r-q\*r;

Old\_r=r;

r=tmp\_r;

int tmp\_s=Old\_s-q\*s;

Old\_s=s;

s=tmp\_s;

int tmp\_t=Old\_t-q\*t;

Old\_t=t;

t=tmp\_t;

}

while(!(Old\_t>0&&Old\_s<0)){

Old\_s-=c;

Old\_t+=k;

}

if(Old\_t>1000000000){

printf("IMPOSSIBLE\n");

}

else{

printf("%d\n",Old\_t);

}

}

* 소수: 에라토스테네스의 체

for(int i=2;i<=n;i++){

if(arr[i]==false){

arr[i]=true;

count++;

if(count==k){

printf("%d",i);

return 0;

}

int num=i\*2;

while(num<=n){

if(arr[num]==false){

arr[num]=true;

count++;

if(count==k){

printf("%d",num);

return 0;

}

}

num+=i;

}

}

}

<조합론>

* 순열: nPk
* 중복순열: N^k
* 같은 것이 있는 순열: N!/(p!q!r!)
* 원순열: nPk/k
* 조합: nCk=nCn-k
* 중복조합: k+n-1Ck
* 파스칼의 삼각형: nCk=n-1Ck-1+n-1Ck

int program(int n, int k){

if(n==k || k==0){

return 1;

}

else

return program(n-1,k-1)+program(n-1,k);

}

<그래프>

* Disjoint Set/Union-Find: Union연산, Find연산

class DisjointSet {

public:

DisjointSet(int n) {

for (int i = 0; i <= n; i++) {

parent[i] = i;

}

}

void Union(int a, int b) {

int aRoot = Find(a);

int bRoot = Find(b);

parent[aRoot] = bRoot;

}

int Find(int a) {

if (parent[a] == a)

return a;

else

return parent[a] = Find(parent[a]);

}

};

* 위상정렬

queue<int> q;

for (int i = 1; i <= n; i++) {

if (arr[i] == 0) {

q.push(i);

printf("%d ", i);

}

}

while (!q.empty()) {

int v = q.front();

q.pop();

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

arr[vec[v][i]] --;

if (arr[vec[v][i]] == 0) {

q.push(vec[v][i]);

printf("%d ", vec[v][i]);

}

}

* 최소신장트리(MST): 크루스칼 알고리즘 / 프림 알고리즘

class DisjointSet {

public:

int par[1005];

DisjointSet(int n) {

for (int i = 1; i <= n; i++) {

par[i] = i;

}

}

void Union(int a, int b) {

int aRoot = Find(a);

int bRoot = Find(b);

par[aRoot] = bRoot;

}

int Find(int a) {

if (par[a] == a) {

return a;

}

else {

return par[a] = Find(par[a]);

}

}

};

* LCA 최소 공통 조상

queue<pair<int, int>> q;

q.push(make\_pair(1, 0));

visit[1] = true;

while (!q.empty()) {

int v = q.front().first;

int d = q.front().second;

q.pop();

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

if (visit[vec[v][i]] == false) {

visit[vec[v][i]] = true;

q.push(make\_pair(vec[v][i], d + 1));

par[0][vec[v][i]] = v;

depth[vec[v][i]] = d + 1;

}

}

}

for (int i = 1; i <= 17; i++) {

for (int j = 1; j <= n; j++) {

par[i][j] = par[i - 1][par[i - 1][j]];

}

}

int LCA(int a, int b) {

if (depth[a] < depth[b]) {

int tmp = a;

a = b;

b = tmp;

}

int diff = depth[a] - depth[b];

int k = 0;

while (diff >= 1) {

if(diff%2==1)

a = par[k][a];

diff /= 2;

k++;

}

if (a == b) {

return a;

}

for (k = 16; k >= 0&& a!=b; k--) {

if (par[k][a] != par[k][b]) {

a = par[k][a];

b = par[k][b];

}

}

return par[0][a];

}

* 단절점/단절선

for (int i = 1; i <= v; i++) {

if (!discover[i]) {

DFS(i,true);

}

}

int DFS(int r, bool isRoot) {

discover[r] = ++order;

int low = discover[r];

int child = 0;

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

if (discover[vec[r][i]]) {

low = min(low, discover[vec[r][i]]);

}

else {

child++;

int pre\_low = DFS(vec[r][i], false);

if (!isRoot&&pre\_low >= discover[r]) {

isCut[r] = true;

}

low = min(low, pre\_low);

}

}

if (isRoot) {

if (child >= 2)

isCut[r] = true;

else

isCut[r] = false;

}

return low;

}

* 최단 경로: 다익스트라 알고리즘(음수X), 벨만포드알고리즘(음수O), 플로이드워셜 알고리즘
* 다익스트라

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push(make\_pair(0, k));

cost[k] = 0;

//visit[k] = true;

while (!pq.empty()) {

int c, u;

do {

c = pq.top().first;

u = pq.top().second;

pq.pop();

} while (!pq.empty() && visit[u] == true);

visit[u] = true;

//printf("\n%d %d\n", c, u);

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

int v = vec[u][i].first;

int w = vec[u][i].second;

if (visit[v] == false) {

if (cost[v] > c + w) {

cost[v] = c + w;

pq.push(make\_pair(c + w, v));

}

}

}

}

* 벨만포드

struct edge {

int from, to, cost;

};

bool isCycle = false;

for (int i = 0; i <= W\*H - G - 1; i++) {

for (int j = 0; j < idx; j++) {

if (dist[e[j].from] != INF && (dist[e[j].to] > dist[e[j].from] + e[j].cost)) {

dist[e[j].to] = dist[e[j].from] + e[j].cost;

if (i == W \* H - G - 1) {

isCycle = true;

break;

}

}

}

}

* 플로이드워셜

<동적계획법>

* 최대증가수열
* 행렬곱 접근법(사선 DP)
* 순회

int tsp(int cur, int state) {

if (state == (1 << n) - 1) {

if (!w[cur][0])

return 1e9;

else

return w[cur][0];

}//다시 원래의 도시로 돌아올수있는지 확인

if (dp[cur][state] != 1e9)

return dp[cur][state];

for (int i = 0; i < n; i++) {//다음 방문할 도시 전부 시도

if ((state&(1<<i))==0&&w[cur][i]!=0) {//해당 도시를 방문하지 않았고못가는경우가 아니면

dp[cur][state] = min(dp[cur][state], tsp(i, (state|1 << i)) + w[cur][i]);

}

}

return dp[cur][state];

}