**목차**

1. **시간복잡도 & 자료구조**
2. **그래프**
3. **정수론**
4. **조합론과 동적계획법**
5. **기하**

**1. 시간복잡도 & 자료구조**

투포인터, 슬라이딩 윈도우

스택 LIFO Top = 스택에 데이터가 삽입될 위치

큐 FIFO Front/Head 삭제될 위치 , 삽입된 위치를 Rear/Tail

Deque = 양 방향에서 삽입 삭제가 이루어짐

깊이 , 레벨 , 높이

preOrder

preOrder

preOrder

Indexed tree -> 이진 트리의 응용

해싱

셋

맵

**2. 그래프**

spanning tree

disjoint set,union-find

MST(minimum spanning tree)

즉, 간선 e개를 퀵 정렬과 같은 효율적인 알고리즘으로 정렬한다면

Kruskal 알고리즘의 시간 복잡도는 O(elog₂e) 이 된다.

Prim의 알고리즘의 시간 복잡도는 O(n^2) 이므로

그래프 내에 적은 숫자의 간선만을 가지는 ‘희소 그래프(Sparse Graph)’의 경우 Kruskal 알고리즘이 적합하고

그래프에 간선이 많이 존재하는 ‘밀집 그래프(Dense Graph)’ 의 경우는 Prim 알고리즘이 적합하다.

LCA(lowest common ancestor)

단절점 :

해법 : 두 묶음을 연결해주는 점

1. 깊이 우선 탐색 후 신장 트리 구축

2. 방문한 순서를 기록

3. 두 묶음->(나보다 방문 순서 낮음, 나보다 방문 순서 높음) 이렇게 나뉨

4. dfs 호출하면서 (실선) 만날 수 있는 젤 낮은 방문 순서를 기록

정점이 자신의 방문 순서보다 자신과 연결된 정점의 low가 크거나 같은 경우가 하나라도 존재하면 해당 정점은 자신을 거치지 않고서는 자신 이전의 정점을 갈 수 없으므로 해당 정점은 단절점

5. 예외 상황 = 루트 루트는 방문 순서 보다 낮은 게 없으니 무조건 단절점이 됨 그래서 루트는 제외 -> 만약 자식이 2개 이상이면 단절점 간선의 개수가 아님

BFS

가중치가 없거나 모든 가중치가 동일한 그래프에서 최단경로를 구하는 경우 가장 빠름

다익스트라 알고리즘 - 출발 정점(S)에서부터 다른 모든 정점까지 최단 경로 구하는 알고리즘

음이 아닌 가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착 최단 경로 문제

O(V^2) or O(ElogE) or O(ElogV) 유동량이나 변화량에 취약 그래서 A\*알고리즘

IDEA

벨만 포드 알고리즘 - 가중 그래프에서의 단일 쌍, 단일 출발, 단일 도착 최단 경로 문제 O(VE)

IDEA

플로이드 워셜 알고리즘 - 전체 쌍 최단 경로 문제 O(V^3)

IDEA

- 어떤 정점 두 정점 사이의 최단 경로는 어떤 경유지(K)를 거치거나 거치지 않는 경로 중 하나

즉 A-B의 최단경로는 A-B or A-K-B

- 만약 경유지(K)를 거친다면 그것을 이루는 부분 경로 역시 최단 경로

A-K-B가 A-B의 최단경로면 A-K와 K-B역시 최단 경로

Algorithm

1. D[i][j] = w

2. 경유지 k에 대해 정점 I에서 정점 j까지의 최단거리

D[i][j] = min( d[i][j], d[i][k] + d[k][j])

다이스트라 n번 더 돌리면 O(n\*ElogE) 더 작지만 플로이드 워셜의 경우 N이 작고 E가 매우 큼

**3. 정수론**

연역법 - 자명한 사실을 이어붙여 결론에 도달

가정을 해놓고 이런 답이 나올것이다

Ex) 사람은 죽는다 -> 그러니깐 세종대왕은 죽는다

귀납법 – 경험적인 증법

어떤 가정을 하고 답이 내놓는다 반례가 없으면 참

Ex) 까마귀는 까맣다 -> 하얀 까마귀는 못 봤으니

귀류법 – 명제의 부정을 참이라고 생각하고 모순을 증명하여 원래의 명제가 참임을 표현

Ex) 알리바이를 댈 때 그 사람을 죽였다고 쳐봐요 근데 난 그때 은행에 있었다

유클리드 호제법

a%b = r이면 a와 b의 최대공약수는 b와 r의 최대공약수와 같다

gcd(a,b) = gcd(b,a%b)

확장 유클리드 호제법

Am+Bn = gcd(m,n)

에라토스테네스의 체 Root(n)보다 작은 소수들로만 나누기

O(NloglogN) O(root(n)/ln(root(n))

**4. 조합론 & 동적계획법**

순열과 조합

순열 : nPk = n!/(n-k)!

조합 : nCk = n!/(n-k)!(k)!

조합의 성질 (파스칼의 삼각형) : nCk = n-1Ck + n-1Ck-1

Memoization : 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술

동적계획법

1. F(i) = i번째인 피보나치 수열
2. F(i) = F(i-1) + F(i-2)
3. 모든 i에 대해 적용

Bottom up, Top down

Lower bound , Upper bound 해당 위치의 index를 return

ex)7

3 4 5 7 7 7 8 10 11

5뒤가 lower bound 8 앞이 upper bound

당연히 7이 하나면 lower bound(7) = upper bound(7)

LCS

Knapsack

행렬의 곱

TSP(NP-Hard)

Time Complexity = Θ(n^2 \* 2^n)

Space Complexity = Θ(n2 n)

NP란 **N**on-deterministic **P**olynomial

모든 NP 문제가 다항 시간에 풀 수 있는 알고리즘이 존재함을 증명할 경우 **P**=**NP**

Decision Problem은 어떠한 인스턴스에 대해서 "yes" or "no"로 답할 수 있는 문제를 말한다.   
쉬운 예를 들자면, 스무고개를 하면 질문을 할때 예 또는 아니오로만 답할 수 있는 질문을 해야한다.

**P : 다항식 시간(polynomial-time)의 알고리즘으로 풀리는 decision problem의 집합**

**NP : 비결정적 다항식시간(polynomial-time nondeterministic) 알고리즘으로 풀리는 decision problem의 집합 ( 2^N, N!..)**

어떠한 problem B에 대해서,  
1. B는 NP이고,  
2. NP의 모든 문제 A에 대해서  A∝B이다.  
그러면 B는 NP-complete이다.

어떤 NP-complete 문제 A에 대해서, A∝tB 이면 문제 B는 NP-hard이다.

**5.기하**

**내적 (Inner product, Scalar product, Dot product)**

**a=(xa,ya) , b=(xb,yb) -> ab = xaxb + yayb = |a||b|cos(theta)**

**외적 (Outer product, Vector product, Cross product)**

**a=(xa,ya) , b=(xb,yb) -> xayb – xbya = |a||b|sin(theta)**

**CCW**

**벡터의 외적을 이용하여 평면상에 세 점이 있을 때 점들의 위치관계를 판단**

**회전방향이 반시계일 경우 (+) 시계인 경우 (-) 일직선에 있는 경우 0**

**외적이 0 이면 평행 CCW가 0 이면 동일 선상**

**기타**

merge sort, heap sorts , index tree ,segment tree, memcpy memset 속도 , 투포인터 , 비트마스크

Struct 선언 차이점

struct Info

{

int node;

int cost;

Info(int n, int c) {

node = n;

cost = c;

}

};

//dynamic

struct Info

{

int node;

int cost;

}Info;

//static

둘이 같지만 위의 경우 무조건 call by value 밑에 경우 자칫 주소값 접근 가능