Team Note of jwpassion1

Yoo Jaewon (jwpassion1)

Compiled on January 23, 2026

Contents

1 Data Structure

1.1 PBDS

Time Complexity: $\mathcal{O}(logN)$

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

//중복 비허용 int형 pbds
#define ordered_set tree<int, null_type, less<>, rb_tree_tag,
tree_order_statistics_node_update>
//중복 허용 int형 pbds
#define ordered_set tree<int, null_type, less_equal<>,
rb_tree_tag, tree_order_statistics_node_update>


ordered_set OS; //pbds OS 선언
OS.insert(val); // OS에 val 삽입
OS.erase(val); // OS에서 val 삭제 (중복 비허용일때)
OS.order_of_key(val); // OS에서 val보다 작은 원소의 개수 리턴
OS.find_by_order(K); // OS[K] 리턴 (0-based-index)

void erase(ordered_set &OS, int val){ // OS에서 val 삭제 (중복 허용일
때)
    auto it = OS.find_by_order(OS.order_of_key(val));
    if (*it == val) OS.erase(it);
}
```

2 Graph & Flow

2.1 이분 매칭(Kuh, 느림)

Time Complexity: $\mathcal{O}(N \log N)$

```
vector<int> graph[total_graph_sz];
int from[right_graph_sz], visited[left_graph_sz];
int visitco; //매번 dfs 돌때마다 값 1씩 증가

bool dfs(int node){
    if (visited[node] == visitco) return false;
    visited[node] = visitco;
    for (int i : graph[node]){
```

```
        if (from[i] == -1){
            from[i] = node;
            return true;
        }
        if (dfs(from[i])){
            from[i] = node;
            return true;
        }
    }
    return false;
}
```

3 Fast Fourier Transform

3.1 Fast Fourier Transform

Time Complexity: $\mathcal{O}(N \log N)$

```
typedef complex<long double> cld;

// 곱할 두 벡터를 isfft true로 하고 각각 fft 돌린뒤에 자릿수별로 곱해주고 isfft false로 하고 fft 돌리기
vector<cld> fft(vector<cld>& a, bool isfft){
    int n = 1, isf = isfft ? 2 : -2;
    while (n < a.size()) n <<= 1;
    vector<cld> ret(n);

    for (int i = 0; i < n; i++){
        int tmp = 0, j = 1, rj = n >> 1;
        for (; j < n; j <<= 1, rj >>= 1){
            if (i & j) tmp |= rj;
        }
        if (i < a.size()) ret[tmp] = a[i];
    }

    for (int i = 2; i <= n; i <<= 1){
        cld w = exp(cld(0, isf * M_PI / i));
        int ord = i >> 1;
        for (int j = 0; j < n; j += i){
            cld wn = cld(1, 0);
            for (int k = j; k < (j | ord); k++, wn *= w){
                cld tmp = ret[k | ord];
                ret[k | ord] = ret[k] - wn * tmp;
                ret[k] += wn * tmp;
            }
        }
    }

    if (!isfft) for (cld& i : ret) i /= n;

    return ret;
}
```

4 ETC

4.1 Template

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);


}
```

4.2 Useful Stuff

- Catalan Number
  1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,742900
  $C_n = binomial(n*2, n)/(n+1)$;
  - 길이가 2n인 올바른 괄호 수식의 수
  - n + 1개의 리프를 가진 풀 바이너리 트리의 수
  - n + 2각형을 n개의 삼각형으로 나누는 방법의 수

- Burnside' s Lemma
  경우의 수를 세는데, 특정 transform operation(회전, 반사, ..) 해서 같은 경우

들은 하나로 친다. 전체 경우의 수는? 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, "아무것도 하지 않는다" 라는 operation도 있어야 함!) 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)

- 알고리즘 게임
  - Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.
  - Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.
  - Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 k + 1로 나눈 나머지를 XOR 합하여 판단한다.
  - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.

- Pick' s Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$

- 가장 가까운 두 점 : 분할정복으로 가까운 6개의 점만 확인

- 홀의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S의 크기) <= (S와 연결되어있는 모든 R의 크기)이다.

- 소수 : 10 007 , 10 009 , 10 111 , 31 567 , 70 001 , 1 000 003 , 1 000 033 , 4 000 037 , 99 999 989 , 999 999 937 , 1 000 000 007 , 1 000 000 009 , 9 999 999 967 , 99 999 999 977

- 소수 개수 : (1e5 이하 : 9592), (1e7 이하 : 664 579) , (1e9 이하 : 50 847 534)

- 10^{15} 이하의 정수 범위의 나눗셈 한번은 오차가 없다.

- N의 약수의 개수 = $O(N^{1/3})$, N의 약수의 합 = $O(NloglogN)$

- $\phi(mn) = \phi(m)\phi(n), \phi(pr^n) = pr^n - pr^{n-1}, a^{\phi(n)} \equiv 1 \pmod{n}$ if coprime

- Euler characteristic : v - e + f (면, 외부 포함) = 1 + c (컴포넌트)

- Euler's phi $\phi(n) = n\prod_{p|n}\left(1 - \frac{1}{p}\right)$

- Lucas' Theorem $\binom{m}{n} = \prod \binom{m_i}{n_i} \pmod{p}$ m_i, n_i는 p^i의 계수

- 스케줄링에서 데드라인이 빠른 걸 쓰는게 이득. 늦은 스케줄이 안들어갈 때 가장 시간 소모가 큰 스케줄 1개를 제거하면 이득.

4.3 자주 쓰이는 문제 접근법

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (brute force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 문제를 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 특수 조건을 꼭 활용
- 여사건으로 생각하기
- 게임이론 - 거울 전략 혹은 mex DP 연계
- 겁먹지 말고 경우 나누어 생각
- 해법에서 역순으로 가능한가?
- 딱 맞는 시간복잡도에 집착하지 말자
- 문제에 의미있는 작은 상수 이용
- 스몰투라지, 트라이, 해싱, 루트질 같은 트릭 생각
- 너무 추상화하기보단 풀려야 하는 방식으로 생각하기

- 잘못된 방법으로 파고들지 말고 버리자
- 제발 터널 비전에 빠지지 말자
- 헬프 콜은 적극적으로
- 혼자 멘탈 나가지 않기

4.4 DP 최적화 접근

- C[i, j] = A[i] * B[j]이고 A, B가 단조증가, 단조감소이면 Monge
- l..r의 값들의 sum이나 min은 Monge
- 식 정리해서 일차(CHT) 혹은 비슷한(MQ) 함수를 발견, 구현 힘들면 Li-Chao
- $a <= b <= c <= d$에서 $A[a, c] + A[b, d] <= A[a, d] + A[b, c]$
- Monge 성질을 보이기 어려우면 N^2 나이브 짜서 opt의 단조성을 확인하고 찍맞
- 식이 간단하거나 변수가 독립적이면 DP 테이블을 세그 위에 올려서 해결
- 침착하게 점화식부터 세우고 Monge인지 판별
- Monge에 집착하지 말고 단조성이나 볼록성만 보여도 됨