목차

완전 탐색

모든 순열 만들기: https://www.acmicpc.net/problem/10974

```
int Arr[MAX];
bool Select[MAX];
vector<int> V;

void Print()
{
	for (int i = 0; i < V.size(); i++)
		cout << V[i] << " ";
	cout << endl;
}

void permu(int Cnt)
{
	if (Cnt == 3) {
		Print();
```

```
            return;
        }
        for (int i = 0; i < MAX; i++) {
            if (Select[i] == true) continue;
            Select[i] = true;
            V.push_back(Arr[i]);
            permu(Cnt + 1);
            V.pop_back();
            Select[i] = false;
        }
}
```

모든 조합 만들기: https://www.acmicpc.net/problem/2407

```
int Arr[MAX];
bool Select[MAX];
void Print()
{
    for (int i = 0; i < MAX; i++)
        if (Select[i] == true)
            cout << Arr[i] << " ";
    cout << endl;
}
void combi(int Idx, int Cnt)
{
    if (Cnt == 3) {
        Print();
        return;
    }
    for (int i = Idx; i < MAX; i++) {
        if (Select[i] == true) continue;
        Select[i] = true;
        combi(i, Cnt + 1);
        Select[i] = false;
    }
}
```

2^n가지 경우의 수 만들기: https://www.acmicpc.net/problem/1182

```
void solution()
{
	int res = 0;
	int arr[20]; // 0부터 19까지 최대 20개
	int N = 0; // 숫자의 수
	int S = 0; // 원소의 합 목표
	cin >> N >> S;
	for (int i = 0; i < N; i++)
		cin >> arr[i];

	// 1 << N은 1뒤에 0이 N개 붙은 2진수가 됨. 1에서 N번 쉬프트했으니까.
	for (int i = 1; i < (1 << N); i++)
	{
		int sum = 0;
		// cout << i << "[]₩n";
		for (int j = 0; j < N; j++) // 1부터 N - 1까지 자리수를 고르고
		{
			int coef = (i & (1 << j)) > 0;
			sum += coef * arr[j]; // 각 자리의 수 arr[j]에 계수를 처리해서 더해줌
			// cout << "계수: " << coef << " arr[j]: " << arr[j] << '₩n';
		}
		if (sum == S)
			res++;
	}
	cout << res;
}
```

동적 계획법

최장 공통 부분 문자열: https://www.acmicpc.net/problem/9251

```
/*

큰 문제를 작은 문제로 쪼갠다.

어떤 두 문자열 X와 Y가 존재한다.
두 문자열의 LCS인 L이 존재한다.
```

L의 마지막 문자는 X와 Y에 반드시 존재한다.
이때 L의 마지막 문자를 X, Y, L 모두에서 지우고 X', Y', L'라고 하자.
그럼 자명하게 L'는 X'와 Y'의 LCS라고 할 수 있다.

X와 Y의 길이를 각각 i, j라고 하고 LCS의 길이를 L(i, j)라 하자.
만약 X와 Y의 마지막 문자가 같다면 이는 반드시 LCS의 마지막에 포함된다.
L(i, j) = L(i - 1, j - 1) + 1 이 된다는 것이다.

만약 X와 Y의 마지막 문자가 다르다면 두 마지막 문자 중 하나는 제거해도 LCS의 길이에서 달라지는 것이 없다.
제거하는 경우의 수는 두 가지이므로 두 경우의 수 중 큰 쪽을 택한다.
L(i, j) = max(L(i - 1, j), L(i, j - 1))

만약 i와 j 중 하나라도 0이라면 자명하게 L(i, j) = 0 이다.

우리가 구할 것은 L(nX, nY) 가 된다.

*/

```
#include <iostream>

using namespace std;

string str1, str2;
int L[1001][1001];

int main()
{
	ios::sync_with_stdio(false); cin.tie(nullptr); cout.tie(nullptr);
	cin >> str1 >> str2;
	for (int i = 1; i <= str1.length(); i++)
	{
		for (int j = 1; j <= str2.length(); j++)
		{
			if (i == 0 || j == 0)
			{
				L[i][j] = 0;
			}
			else if (str1[i - 1] == str2[j - 1])
```

```
                {
                        L[i][j] = L[i - 1][j - 1] + 1;
                }
                else
                {
                        L[i][j] = L[i - 1][j] > L[i][j - 1] ? L[i - 1][j] : L[i][j - 1];
                }
        }
    }
    cout << L[str1.length()][str2.length()];
    return 0;
}
```

가장 긴 증가하는 부분 수열: https://www.acmicpc.net/problem/14002

```
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
using namespace std;

int a[1001]; // 배열 A
int p[1001]; // 배열 A 원소의 LIS에서의 위치
int lis[1001]; // LIS 구하기 위한 임시 배열

int max_position = 0;
int max_i = 0;

void print_answer(int max_i, int max_position)
{
    //cout << "max_i = " << max_i << ", max_position = " << max_position << endl;
    int j = max_i - 1;
    int pos = max_position - 1;
    if (pos == -1) return;
    while (1) {
        if (p[j] == pos) {
            break;
        }
        j--;
```

```
    }
    print_answer(j, pos);
    cout << a[max_i] << " ";
}

int main(void)
{
    int N;    // 수열 N의 크기
    cin >> N;

    for (int i = 1; i <= N; i++)
        cin >> a[i];

    lis[0] = -99999;
    for (int i = 1; i <= N; i++)
        lis[i] = 99999;

    int k = 1;
    for (int i = 1; i <= N; i++) {
        int positions = std::lower_bound(lis, lis + k, a[i]) - lis + 1;
        p[i] = positions - 1;
        if (max_position < p[i]) {
            max_position = p[i];
            max_i = i;
        }
        lis[p[i]] = a[i];
        //cout << "lis[p[i]] = " << lis[p[i]] << endl;
        if (k == p[i]) k++;
    }
    cout << max_position << "₩n";
    print_answer(max_i, max_position);

    return 0;
}
```

1,2,3 더하기: https://www.acmicpc.net/problem/9095

정수 삼각형: https://www.acmicpc.net/problem/1932

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<vector<int>> triIn, triMax;
int nMax;
void solution()
{
	int N;
	cin >> N; // 높이 수
	for (int i = 0; i < N; i++)
	{
		triIn.emplace_back();
		triMax.emplace_back();
		for (int j = 0; j <= i; j++)
		{
			int tmp;
			cin >> tmp;
			triIn[i].emplace_back(tmp);
			triMax[i].emplace_back(tmp);
			if (i == 0); // 최상층 아무것도안함
			else if (j == 0) // 왼쪽이 없음
				triMax[i][j] += triMax[i - 1][j];
			else if (j == i) // 오른쪽이 없음
				triMax[i][j] += triMax[i - 1][j - 1];
			else
				triMax[i][j] += max(triMax[i - 1][j], triMax[i - 1][j - 1]); // 
triMax[i][j]를 선택가능한 경로중 값이 더 큰 경로
		}
	}
	cout << *max_element(triMax[N - 1].begin(), triMax[N - 1].end());
}
```

동전 바꿔주기: https://www.acmicpc.net/problem/2624

```
int T; // 지폐 금액
int K; // 동전 가지 수 (T 이하)
int coinPrice[101]; // i번째 동전의 가격
int coinCount[101]; // i번째 동전의 개수
```

```
int dp[10001]; // 나올 수 있는 가격

void solve()
{
	cin >> T;
	cin >> K;
	for (int i = 1; i <= K; i++)
		cin >> coinPrice[i] >> coinCount[i];

	dp[0] = 1;
	// 코인 선택
	for (int i = 1; i <= K; i++)
	{
		// 모든 금액에 대해서 dp를 진행.
		for (int j = T; j >= 0; j--)
		{
			for (int k = 1; k <= coinCount[i]; k++)
			{
				if (j - k * coinPrice[i] >= 0)
					dp[j] += dp[j - k * coinPrice[i]];
			}
		}
	}
	cout << dp[T] << "₩n";
}
```

파일 합치기: https://www.acmicpc.net/problem/11066

```
typedef int type;

const int INF = ~(1 << 31);

int T, K; // 테케 수, 챕터 수
type sum[501]; // sum[i] = dp[0] + ... + dp[i - 1]. // sum[i] - sum[j] = dp[j] + .. + dp[i - 1]
type dp[500][500]; // 인덱스는 0부터 시작.


type recurse(int l, int r)
{
```

```
		type& now = dp[l][r];
		type ret = INF;

		if (now)
			return now;
		else if (l == r)
			return 0;
		else
		{
			for (int i = l; i < r; i++)
				ret = min(ret, recurse(l, i) + recurse(i + 1, r));

			ret += sum[r + 1] - sum[l];
			return now = ret;
		}

}

void solve()
{
		cin >> T;
		while (T--)
		{
			memset(dp, 0, sizeof(dp));
			cin >> K;
			sum[0] = 0;
			for (int i = 1; i < K + 1; i++)
			{
				cin >> sum[i];
				sum[i] += sum[i - 1]; // 누적 합
			}
			cout << recurse(0, K - 1) << '₩n';
		}
}
```

동전1: https://www.acmicpc.net/problem/2293

```
/*
가치 -> 동전 루프를 돌아서 가치를 계산할 때 동전이 중복된다.
동전 -> 가치 루프를 돌면?
*/
```

```
#include <iostream>

using namespace std;

int N, K; // 동전 종류, 만들 가치
int coin[101];
int dp[10001]; // 가치 K를 만드는 경우의 수

int main()
{
	coin[0] = -1; // 사용안 함
	dp[0] = 1; // 0원을 만드는 동전의 경우의 수 = 1가지
	cin >> N >> K;
	for (int i = 1; i <= N; i++)
	{
		cin >> coin[i];
	}
	for (int i = 1; i <= N; i++) // 동전의 수를 늘려가면서 DP
	{
		for (int j = 1; j <= K; j++)
		{
			/*
				dp[j]
					coin[i-1]까지로 가치 j를 만드는 경우의 수
				dp[j - coin[i]]
					가치 j - coin[i]를 만드는 경우의 수. (여기에 현재 선택한 coin[i]를 더하면 되니까)
				dp[j] + dp[j-coin[i]]
					coin[i]까지로 가치 j를 만드는 경우의 수
			*/
			if ( j - coin[i] >= 0)
				dp[j] += dp[j - coin[i]];
		}
	}
	cout << dp[K];
	return 0;
}
```

배낭 문제

평범한 배낭: https://www.acmicpc.net/problem/12865

```
int N, K;
int dp[100001];
int main(void)
{
	ios::sync_with_stdio(false); cin.tie(nullptr); cout.tie(nullptr);
	cin >> N >> K;
	for (int i = 1; i <= N; i++)
	{
		int nW, nV;
		cin >> nW >> nV;
		//cout << "이제 i번째까지 넣을 수 있다.₩ni번째를 넣은 경우의 최대 = dp[j - nW] + nV // 안 넣은 경우의 최대 = dp[j] ₩n";
		for (int j = K; j >= nW; j--)
		{
			//cout << "무게 nW: " << nW << "// 남은 무게 j: " << j << "₩n";
			//cout << "넣은 경우: " << dp[j - nW] + nV << "// 안 넣은 경우: " << dp[j] << "₩n";

			dp[j] = dp[j - nW] + nV > dp[j] ? dp[j - nW] + nV : dp[j];
		}
	}
	cout << dp[K];
	return 0;
}
```

사탕: https://www.acmicpc.net/problem/1415

```
bool isPrime[MAX_PRICE]; // isPrime[i]는 i의 소수 여부
ll dp[MAX_PRICE]; // dp[i]는 사탕 가격을 i로 만드는 경우의 수 (순서 상관 x)

int N; // 사탕의 개수
int candyPrice[50]; // 사탕의 가격 (중복 X)
int nCandy[50]; // 각 사탕별 개수
int nCandyKind; // 사탕 종류의 수
int nZero;

ll ans;
```

```
void solve()
{
	memset(isPrime, true, sizeof(isPrime));
	isPrime[0] = isPrime[1] = false;
	// 소수를 구한다.
	for (int i = 2; i * i < MAX_PRICE; i++)
	{
		if (isPrime[i] == false)
			continue;
		else
		{
			for (int j = 2; i * j < MAX_PRICE; j++)
			{
				isPrime[i * j] = false;
			}
		}
	}

	/*
	수도코드
	사탕을 입력받는다.
	사탕의 개수를 세면서 중복으로 저장하지 않는다.
	가격이 0원이 사탕의 개수를 센다.

	dp[0] = nZero + 1; 이다. 왜냐하면 0원 짜리 구매는 모든 경우의 수에 공통으로 더해지기 때문이다.

	사탕 i를 선택 한다.
	for ( i < nCandy
	각 사탕에 대해서 dp[i]를 i가 MAX_PRICE - 1 부터 채운다.
		사탕의 개수만큼 이전 것부터의 영향을 받는다.
			dp[i] += dp[i - price[i] * {c | c <= cnt[i] } ]
	단, 채울 때 사탕이 무료인 경우는 제외한다.

	*/
	cin >> N;
	for (int i = 0; i < N; i++)
	{
```

```
		int sameIdx = -1;
		int tmp = 0;
		cin >> candyPrice[nCandyKind];
		if (candyPrice[nCandyKind] == 0)
			nZero++;
		// 중복 확인
		for (int j = 0; j < nCandyKind; j++)
		{
			if (candyPrice[j] == candyPrice[nCandyKind])
			{
				nCandy[j]++;
				sameIdx = j;
				break;
			}
		}
		if (sameIdx == -1) // 중복이 없는 경우
		{
			nCandy[nCandyKind]++;
			nCandyKind++;
		}
}

dp[0] = (ll)nZero + 1;
// 사탕을 선택.
for (int i = 0; i < nCandyKind; i++)
{
		if (candyPrice[i] == 0)
			continue;
		// 가격을 선택
		for (int j = MAX_PRICE - 1; j >= 0; j--)
		{
			// 사탕의 개수별로 적용
			for (int k = 1; k <= nCandy[i]; k++)
			{
				if (j - candyPrice[i] * k < 0)
					break;
				dp[j] += dp[j - candyPrice[i] * k];
			}
		}
```

```
	}

	for (int i = 2; i < MAX_PRICE; i++)
	{
		if (isPrime[i])
			ans += dp[i];
	}
	cout << ans << "₩n";
}
```

행렬

본대 산책2: https://www.acmicpc.net/problem/12850

```
#include <iostream>
#include <vector>

using namespace std;

typedef long long ll;
typedef vector<vector<ll>> vvll;

typedef struct _matrix {
	int size;
	vvll mat;
} Matrix;

vvll map = {
	{ 0, 1, 0, 1, 0, 0, 0, 0 },
	{ 1, 0, 1, 1, 0, 0, 0, 0 },
	{ 0, 1, 0, 1, 1, 1, 0, 0 },
	{ 1, 1, 1, 0, 0, 1, 0, 0 },
	{ 0, 0, 1, 0, 0, 1, 0, 1 },
	{ 0, 0, 1, 1, 1, 0, 1, 0 },
	{ 0, 0, 0, 0, 0, 1, 0, 1 },
	{ 0, 0, 0, 0, 1, 0, 1, 0 },
};

void mul_mat(Matrix& A, Matrix& B, Matrix& O)
{
```

```
	if (A.size != B.size || A.size != O.size)
	{
		cout << "행렬의 크기가 다릅니다.";
		return;
	}

	int arr_size = A.size;

	for (int i = 0; i < arr_size; i++) {
		for (int j = 0; j < arr_size; j++) {
			ll tmp = 0;
			for (int k = 0; k < arr_size; k++) {
				tmp += A.mat[i][k] * B.mat[k][j];
				tmp %= 1000000007;
			}
			tmp %= 1000000007;
			O.mat[i][j] = tmp;
		}
	}
}

void print_mat(Matrix& arr)
{
	for (int i = 0; i < arr.size; i++)
	{
		for (int j = 0; j < arr.size; j++)
		{
			cout << arr.mat[i][j] << " ";
		}
		cout << "₩n";
	}
}

void init_mat(Matrix& arr, int size)
{
	arr.size = size;
	for (int i = 0; i < size; i++)
	{
		arr.mat.emplace_back(vector<ll>());
```

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

void iden_mat(Matrix& arr)
{
        for (int i = 0; i < arr.size; i++)
        {
                for (int j = 0; j < arr.size; j++)
                {
                        if (i == j)
                                arr.mat[i][j] = 1;
                }
        }
}

ostream& operator<<(ostream& out, Matrix right)
{
        print_mat(right);
        return out;
}

Matrix memo_mat[30]; // A^(2^n) 을 저장함.

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

        init_mat(memo_mat[0], 8);
        memo_mat[0].mat = map;
        memo_mat[0].size = 8;

        for (int i = 1; i < 30; i++)
        {
                init_mat(memo_mat[i], 8);
                mul_mat(memo_mat[i - 1], memo_mat[i - 1], memo_mat[i]);
```

```
	}

	int D;
	cin >> D;

	Matrix ans;
	init_mat(ans, 8);
	iden_mat(ans);

	for (int i = 1, j = 0; i <= D; i = i << 1, j++)
	{
		if (D & i)
		{
			Matrix tmp;
			init_mat(tmp, 8);
			mul_mat(ans, memo_mat[j], tmp);
			ans = tmp;
		}
	}

	cout << ans.mat[0][0];

	return 0;

}
```

그리디

Byte Coin: https://www.acmicpc.net/problem/17521

```
int N; // 요일 수 ( 15 )
long long W; // 초기 자본 ( 10만 이하 자연수 )
int price[15];
int point[15]; // 최고점 1 최저점 -1
long long n_coin;

void solution()
{
	cin >> N >> W;
	int max_cost = 0;
```

```
	if (N == 1) // 하루 밖에 없으면 그냥 가진 자산.
	{
		cout << W;
		return;
	}
	if (N < 1)
		return;
	for (int i = 0; i < N; i++)
	{
		cin >> price[i];
	}
	bool is_up = price[1] > price[0];

	for (int i = 1; i < N - 1; i++)
	{
		bool now = price[i+1] > price[i];
		if (now != is_up)
		{
			is_up = now;
			point[i] = (now ? -1 : 1); // now가 true, is_up false면 최소점을 찾았다는
뜻.
		}
	}

	if (price[1] > price[0]) // 시작 지점 구매 여부
	{
		n_coin = W / price[0];
		W = W % price[0];
	}

	for (int i = 0; i < N; i++)
	{
		if (point[i] == -1) // 최소점이면 산다.
		{
			n_coin += W / price[i];
			W = W % price[i];
		}
		else if (point[i] == 1) // 최대점이면 판다.
		{
```

```
                        W += (long long)price[i] * n_coin;
                        n_coin = 0;
                }
        }
        W += (long long)price[N-1] * n_coin; // 마지막 지점에는 무조건 다 판다.

        cout << W << '₩n';
}
```

카드 합체 놀이: https://www.acmicpc.net/problem/15903

```
#include <iostream>
#include <queue>
using namespace std;
typedef long long ll;
priority_queue< ll, vector<ll>, greater<ll> > pq;

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

        int n, m;
        cin >> n >> m;

        int inp = 0;
        for (int i = 0; i < n; i++)
        {
                cin >> inp;
                pq.push(inp);
        }

        ll x, y;
        for (int i = 0; i < m; i++)
        {
                x = pq.top(); pq.pop();
                y = pq.top(); pq.pop();
                x += y;
                y = x;
                pq.push(x);
                pq.push(y);
        }
```

```
	ll res = 0;
	while (!pq.empty())
	{
		res += pq.top();
		pq.pop();
	}
	cout << res;
	return 0;
}
```

큰 수 만들기: https://www.acmicpc.net/problem/16496

```
int N, sum;
string arr[1000]; // 숫자들

bool compare(string a, string b)
{
	return a + b > b + a; // 이게 핵심. 두 문자열을 더하니 길이가 같고 사전 비교됨
}

void solution()
{
	cin >> N;
	for (int i = 0; i < N; i++)
    {
	cin >> arr[i];
	sum += stoi(arr[i]);
    }
	if (sum == 0)
	{
		cout << 0;
		return;
	}
	sort(arr, arr + N, compare);
	for (int i = 0; i < N; i++)
		cout << arr[i];
}
```

회의실배정: https://www.acmicpc.net/problem/1931

```
/*
```

1. 끝나는 시간이 빠른 순서대로 정렬
2. 끝나는 시간이 같으면, don't care..?
-> No! 시작 시간이 빠른 순서대로 정렬

< 반례 >
2
4 4
1 4
정답 : 2, 오답 : 1

3. 끝나는 시간이 가장 빠른 걸 고른다.
4. 끝나는 시간을 기준으로 고를 수 있는 것들 중에서,
   가장 빠른 것을 고른다.
5. 4번을 반복
*/

```
#include <iostream>
#include <algorithm>

using namespace std;

int N, result = 1; // 1이상 10만 이하
pair<int, int> meeting[100000]; // 회의 번호, 회의 시점, 종점

bool myCompare(pair<int, int> p1, pair<int, int> p2)
{
	if (p1.second == p2.second)
		return p1.first < p2.first;
	return p1.second < p2.second;
}

int main()
{
	cin >> N;
	for (int i = 0; i < N; i++)
	{
		cin >> meeting[i].first >> meeting[i].second;
	}
	sort(meeting, meeting + N, myCompare);
```

```
	int base = meeting[0].second;
	for (int i = 1; i < N; i++)
	{
		if (meeting[i].first >= base)
		{
			base = meeting[i].second;
			++result;
		}
	}
	cout << result;
	return 0;
}
```

백트래킹

N-Queen: https://www.acmicpc.net/problem/9663

```
int arr[15][15];
int N_size; // N * N 체스판의 N개의 퀸. N은 1이상 14이하
int Count; // 퀸을 배치하는 경우의 수
vector<pair<int, int> > v_queen;

void print_queen()
{
	for (auto x : v_queen)
	{
		cout << x.first << " " << x.second << "₩n";
	}
}

int can_queen(int i, int j) // 이 지점에 queen을 둘 수가 있나?
{
	int can = 1; // 초기값은 된다고 가정
	for (auto x : v_queen) // 모든 퀸에 대해서
	{
		if (i == x.first || j == x.second) // 같은 행 또는 같은 행은 공격할 수 있다
		{
			can = 0;
			break;
		}
		int ii = abs(x.first - i);
```

```
                int jj = abs(x.second - j);
                if (ii == jj) // 행과 열로 같은 정도로 떨어져있다 == 대각선상에 있다 == 공격
할 수 있다.
                {
                        can = 0;
                        break;
                }
        }
        return can;
}

void recursion(int ii) // ii 번째 행에 퀸을 배치한다.
{
        /*
        1. 첫번 째 행에 퀸을 배치한다
                2. 다음 퀸을 배치하기 위해 재귀한다.
        2. 배치 가능한 자리가 없으면
                1. 배치가 끝났는지 확인한다
                        1. 끝났으면 + 1 한다.
                2. 배치가 끝나지 않았으면
                        1. 이전 단계로 되돌아가서 다른 자리를 찾는다.
        */

        if (v_queen.size() == N_size) // 모든 퀸을 배치했으면
        {
                Count++; // 경우의 수를 1개 올리고
                return; // 이전 단계로 되돌아가서 다른 경우의 수를 찾는다
        }

        int flag = 0; // 체스판에 퀸을 놨는지
        for (int j = 1; j <= N_size; j++) // 어느 열에 배치가 가능한지 탐색
        {
                if (can_queen(ii, j)) // 체스판에 배치가 가능하면
                {
                        v_queen.emplace_back(ii, j);
                        recursion(ii + 1); // 다음 행에 퀸을 배치한다.
                        v_queen.pop_back();
                        flag++;
                }
```

```
        }

        if (flag == 0) // 체스판에 퀸을 둘 자리가 없으면
                return; // 이전 단계로 돌아가서 다른 경우를 찾아본다.

}

void solution()
{
        cin >> N_size;
        recursion(1);
        cout << Count;
}
```

경로 탐색

다익스트라

네트워크 복구: https://www.acmicpc.net/problem/2211

```
using namespace std;
typedef pair<int, int> pii;
const int MAX_V = 1000;
const int INF = 2147483647;

int N, M; // N(1000이하)개의 컴퓨터, M개의 에지
int A, B, C; // 시작 노드, 끝노드, 가중치 (10 이하)
int K; // 복구할 회선의 개수

// 그래프의 인접 리스트. 연결된 정점 번호, 간선 가중치 쌍을 담는다. (idx 자체가 시작하는 노드)
vector<pii> adj[MAX_V];

vector<int> dijkstra(int src) {
        priority_queue<pii> pq;
        vector<int> dist(N, INF);
        vector<int> edge(N, -1);
        dist[src] = 0;

        pq.push(make_pair(0, src));
        while (!pq.empty()) {
```

```
                int cost = -pq.top().first;
                int here = pq.top().second;
                pq.pop();
                // 만약 지금 꺼낸 것보다 더 짧은 경로를 알고 있다면 지금 꺼낸것을 무시한다.
                if (dist[here] < cost) continue;
                // 인접한 정점을 모두 검사한다.
                for (int i = 0; i < adj[here].size(); i++)
                {
                        int there = adj[here][i].first; // 인접한 좌표
                        int nextDist = cost + adj[here][i].second; // here을 거쳐서 there에 가는
비용
                        // 더짧은 경로를 발견하면 dist[]를 갱신하고 우선순위 큐에 넣는다.
                        if (dist[there] > nextDist) { // there까지 가는 비용 갱신중.
                                //printf("경로 갱신 %d을 거치는 %d까지 최단 비용 %d₩n",
here, there, nextDist);
                                dist[there] = nextDist;
                                edge[there] = here; // there 까지 최단 간선은 here에서 시작함.
                                pq.push(make_pair(-nextDist, there));
                        }
                }
        }
        return edge;
}

// 컴퓨터 번호는 0부터 N - 1까지  (출력시엔 1부터 N). 무방향성
void solution()
{
        cin >> N >> M;
        for (int i = 0; i < M; i++) // 에지 입력 받기
        {
                cin >> A >> B >> C;
                adj[A - 1].emplace_back(make_pair(B - 1, C));
                adj[B - 1].emplace_back(make_pair(A - 1, C));
        }
        vector<int> edge = dijkstra(0);

        cout << edge.size() - 1 << "₩n";
        for (int i = 1; i < edge.size(); i++)
        {
```

```
                cout << i + 1 << " " << edge[i] + 1 << "₩n";
        }
}
```

벨만 포드

웜홀: https://www.acmicpc.net/problem/1865

```
typedef struct {
    int u, v, w;            // start point, end point, weight
} Edge;


const int INF = ~(1 << 31)/2;
const int SIZE = 6000;        // maximum size is M*2 + W + 1 = 5201

int nV, nE, nW;    // 노드수 에지수 웜홀 수
Edge edges[SIZE];            // input graph edges
int dist[SIZE];              // dist[i]: source에서 i까지 최소 거리

// 음의 가중치가 있을 경우 -1 리턴, 정상종료시 0 리턴
int BellmanFord()
{
    // Step 1. Initialize graph (INF로 초기화)
    for (int i = 0; i < nV; i++)
        dist[i] = INF;

    // Step 2. Relax edges repeatedly: O(|V||E|)
    for (int i = 0; i < nV - 1; i++)
        for (int j = 0; j < nE + nW; j++)
            if (dist[edges[j].v] > dist[edges[j].u] + edges[j].w)
                dist[edges[j].v] = dist[edges[j].u] + edges[j].w;

    // Step 3. Check for negative-weight cycles
    for (int j = 0; j < nE + nW; j++)
    {
        if (dist[edges[j].v] > dist[edges[j].u] + edges[j].w)
            return -1;    // negative-weight cycles exist.
    }
```

```
    return 0;               // negative-weight cycles don't exist.
}

int main(void)
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int TC;
    cin >> TC;

    // 테케 반복
    for (int i = 0; i < TC; i++)
    {
        int S, E, T;

        cin >> nV >> nE >> nW;
        nE *= 2; // 양방향이니까 간선을 각각 세줘야함.

        for (int j = 0; j < nE; j += 2)   //도로
        {
            cin >> S >> E >> T;
            edges[j] = { S, E, T };
            edges[j + 1] = { E, S, T };
        }

        for (int j = nE; j < nE + nW; j++)   // 웜홀
        {
            cin >> S >> E >> T;
            edges[j] = { S, E, -T };
        }

        cout << (BellmanFord() ? "YES" : "NO") << "₩n";
    }
    return 0;
}
```

문자열

UCPC는 무엇의 약자일까?: https://www.acmicpc.net/problem/15904

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

	string str;
	int nFound = 0;

	getline(cin, str);

	for (int i = 0; i < str.size(); i++)
	{
		if (nFound == 0 && str[i] == 'U')
			nFound++;
		else if (nFound == 1 && str[i] == 'C')
			nFound++;
		else if (nFound == 2 && str[i] == 'P')
			nFound++;
		else if (nFound == 3 && str[i] == 'C')
		{
			nFound++;
			break;
		}
	}

	cout << (nFound == 4 ? "I love UCPC" : "I hate UCPC");

	return 0;
}
```

닉네임에 갓 붙이기: https://www.acmicpc.net/problem/13163

```
	// 닉네임에 갓 붙이기
	int N; // 사람 수
	string S;

	void solve() {
		cin >> N;
```

```
		cin.ignore(1, '₩n');
		while (N--) {
			string ans = "";
			int idx = 0;
			getline(cin, S); // 한 줄을 다 받는다.


			for (idx = 0; idx < S.length() && S[idx] != ' '; idx++); // 처음 공백의 위치를 찾음.
			ans += "god";
			for (; idx < S.length(); idx++)
				if (S[idx] != ' ')
					ans += S[idx];
			cout << ans << "₩n";
		}
	}
```

그대로 출력하기 2: https://www.acmicpc.net/problem/11719

```
void solve() {
	while ((c = getchar()) != EOF)
		putchar(c);
}
```

구현

모노톤길: https://www.acmicpc.net/problem/11067

```
typedef struct _pos
{
    int y;
    int x;
} Pos;

typedef vector<int> vi;
typedef vector<Pos> vp;

int T; // 테케 수
int n; // 카페 수, 10만 이하
int m; // 출력할 카페 수
int ans;
```

```
vp vPos; // x좌표는 0 ~ 10만, y좌표는 -10만 ~ 10만
vp vAns; // vAns[i] : i + 1번 카페의 좌표

bool compareX(Pos a, Pos b)
{
    return a.x < b.x;
}

bool compareY(Pos a, Pos b)
{
    return a.y < b.y;
}

void printVI(vi v)
{
    for (auto x : v)
        cout << x << " ";
    cout << '₩n';
}

void printVP(vp v)
{
    for (auto x : v)
        cout << x.x << " " << x.y << "₩n";
    cout << '₩n';
}

// vAns를 채운다.
bool setNum(Pos pre, int idx) // 지난 행의 마지막 카페의 좌표, 탐색을 이을 idx
{
    int i = idx;
    vp cols; // 현재 열의 모든 카페 좌표
    if (vAns.size() == vPos.size()) // 다 찾았음.
    {
        return true;
    }

    // 현재 열의 모든 카페를 뒤져보는데,
    // 이 카페들의 y좌표가 pre의 y보다 모두 작거나 같거나 모두 크거나 같게 됨
```

```
    for (i = idx; vPos[i].x == vPos[i + 1].x; i++) // 다음 카페도 현재 열에 있으면
    {
        cols.push_back(vPos[i]);
        if (vPos.size() - 2 == i)
        {
            i++;
            break;
        }
    }
    cols.push_back(vPos[i]); // 현재 행의 마지막 카페

    sort(cols.begin(), cols.end(), compareY); // y에 대해 오름 차순으로 정렬한다.
    if (cols.front().y >= pre.y) // pre보다 항상 y가 크거나 같다.
    {
        for (int j = 0; j < cols.size(); j++)
        {
            vAns.push_back(cols[j]); // 정방향으로 넣음
        }
        setNum(cols.back(), idx + cols.size());
    }
    else if (cols.back().y <= pre.y) // pre보다 항상 y가 작다.
    {
        for (int j = cols.size() - 1; j >= 0; j--)
        {
            vAns.push_back(cols[j]); // 역방향으로 넣음
        }
        setNum(cols.front(), idx + cols.size());
    }
    else
    {
        cout << "error"; // 있어선 안됨
        return false;
    }
}

void solve()
{
    vPos.reserve(100001);
```

```
    vAns.reserve(100001);
    cin >> T;
    for (int i = 0; i < T; i++)
    {
        int tmp1(0), tmp2(0);
        vPos.clear();
        vAns.clear();
        cin >> n;
        for (int j = 0; j < n; j++)
        {
            cin >> tmp1 >> tmp2;
            vPos.push_back({ tmp2, tmp1 }); // y, x 순으로 입력
        }
        sort(vPos.begin(), vPos.end(), compareX);

        // printVP(vPos);
        setNum({ 0, 0 }, 0);
        cin >> m;
        for (int j = 0; j < m; j++)
        {
            cin >> tmp1;
            cout << vAns[tmp1 - 1].x << " " << vAns[tmp1 - 1].y << "₩n";
        }
    }
}
```

펠린드롬 / n진법 변환

회문인 수: https://www.acmicpc.net/problem/11068

```
int T; // 테케 수
int N; // 목표 수
int ans;

typedef vector<int> vi;

vi S; // 숫자

void printVI(vi v)
{
    for (auto x : v)
```

```
        cout << x << " ";
    cout << '₩n';
}

void getChangedDigit(int n, int d, vi& ret) // 숫자 n을 d진법으로 바꿔서 반환.
{
    ret.clear(); // 초기화
    int q; // 몫
    int r; // 나머지
    // n == d여도 루프를 돌아야한다.
    while (n >= d)
    {
        q = n / d;
        r = n % d;
        // 나머지는 붙인다.
        ret.push_back(r);
        n = q;
    }
    ret.push_back(n);
}

bool isPalindromic(vi S) // 회문이면 true
{
    int l(0), r(S.size() - 1);
    while (l <= r)
    {
        if (l == r)
            return true;
        else if ((l + 1 == r) && S[l] == S[r])
            return true;
        else if (S[l] == S[r])
        {
            l++; r--;
        }
        else
            return false;
    }
    return true;
}
```

```
void solve()
{
    S.reserve(100000);

    cin >> T;
    for (int i = 0; i < T; i++)
    {
        ans = 0; // 초기화
        cin >> N;
        for (int j = 2; j <= 64; j++) // 진법 선택
        {
            getChangedDigit(N, j, S);
            // printVI(S);
            if (isPalindromic(S))
            {
                ans = 1;
                break;
            }
        }
        cout << ans << '₩n';
    }
}
```

스위핑

선 긋기: https://www.acmicpc.net/problem/2170

```
#include <iostream>
#include <queue>
using namespace std;

int main(void)
{
        ios::sync_with_stdio(false); cin.tie(nullptr); cout.tie(nullptr);
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > q;
        int N;
        cin >> N;
        for (int i = 0; i < N; i++) // 구간 입력 받기 (완전히 포함되는 구간은 무시해서 받는다)
        {
                int A, B;
```

```
            cin >> A >> B; // 시점 종점
            q.emplace( A, B );
        }

        if (q.size() == 1) // 유효 구간이 하나뿐이라면
        {
            auto pr = q.top();
            cout << pr.second - pr.first;
            return 0;
        }

        int sum = 0;
        auto range = q.top(); q.pop();
        auto px = q.top(); q.pop();

        for (;;)
        {
            if (range.second < px.first) // 이전 구간의 종점이 현재 구간의 시점보다 왼쪽이
면
            {
                sum += range.second - range.first; // 독립된 구간이 정해졌으므로 길이
를 재서 넣는다.
                range = px; // 현재 구간을 새로운 시작 구간으로 설정
            }
            else if (range.second < px.second) // 이전 구간의 종점이 현재 구간의 종점보다
왼쪽이면
            {
                range.second = px.second; // 구간의 종점을 갱신
            }

            if (q.size() == 0) // 더이상 가져올 것이 없으면
            {
                sum += range.second - range.first; // 구간 등록하고 종료.
                break;
            }
            px = q.top(); // 다음 구간을 가져온다.
            q.pop();
        }
        cout << sum;
```

```
	return 0;
}
```

세그먼트 트리

구간 합 구하기: https://www.acmicpc.net/problem/2042

```
using namespace std;

typedef long long ll;

class MySegmentTree
{
public:
	// 배열 arr에 대한 세그먼트 트리를 만든다.
	void init(vector<ll>* arr)
	{
		mp_arr = arr;
		m_height = ceill(log2l((*mp_arr).size())); // n을 2^k꼴로 올림해서 최소 요구 높이 구함
		m_size = powl(2, m_height + 1) - 1; // 필요한 데이터 사이즈

		m_data.resize(m_size, 0);
		init_recurse(1, 1, (*mp_arr).size()); // 노드 번호가 1부터 시작함. (계산 편의상)

	}

	void set_node(int i, ll n) // 외부에서 받아오는 건 0부터
	{
		update(1, 1, (*mp_arr).size(), i + 1, n - (*mp_arr)[i]); // 내부적으로는 1부터
		(*mp_arr)[i] = n;
	}

	ll get_sum(int l, int r)
	{
		return m_sum(1, 1, (*mp_arr).size(), l, r);
	}

	ll size()
	{
		return m_size;
```

```
	}

	ll height()
	{
		return m_height;
	}


private:
	vector<ll>* mp_arr;
	vector<ll> m_data;
	ll m_height = 0;
	ll m_size = 0;

	ll init_recurse(int node, int start, int end)
	{
		if (start == end)
			return m_data[node - 1] = (*mp_arr)[start - 1];  // 노드 번호가 1부터 시작함. (계산 편의상)
		else
		{
			int mid = (start + end) / 2;
			return m_data[node - 1] = init_recurse(node * 2, start, mid) + init_recurse(node * 2 + 1, mid + 1, end);
		}
	}

	void update(int node, int start, int end, int idx, ll diff)
	{
		if (!(start <= idx && idx <= end)) // 갈필요 없는 경로
			return;

		m_data[node - 1] += diff; // 목적지 까지 가는 경로에 있는 모든 sum에 차를 더해줌.

		if (start != end)
		{
			int mid = (start + end) / 2;
			update(node * 2, start, mid, idx, diff);
```

```
                update(node * 2 + 1, mid + 1, end, idx, diff);
        }
    }

    ll m_sum(int node, int start, int end, int l, int r)
    {
        /*
        네 가지 경우로 나뉜다.
        1. 찾는 구간이 현재 구간과 전혀 겹치지 않음
        2. 찾는 구간이 현재 구간을 포함함
        3. 찾는 구간이 현재 구간에 포함됨 -> 더 쪼개야함
        4. 나머지 -> 더 쪼개야함
        */

        if (l > end || r < start) // 전혀 겹치지 않음
            return 0;

        if (l <= start && end <= r) // 찾는 구간이 현재 구간을 포함함
            return m_data[node - 1];

        int mid = (start + end) / 2;

        return m_sum(node * 2, start, mid, l, r) + m_sum(node * 2 + 1, mid + 1, end, l, r);
    }
};



MySegmentTree segt;
vector<ll> arr;

int N; // 수의 개수 100만 이하 자연수
int M, K; // 변경, 구간합 // 1만 이하 자연수

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

    cin >> N >> M >> K;
```

```
	arr.resize(N);
	for (int i = 0; i < N; i++)
	{
		cin >> arr[i];
	}
	segt.init(&arr);

	for (int i = 0; i < M + K; i++)
	{
		int a, b, c;
		cin >> a >> b >> c;
		if (a == 1)
			segt.set_node(b - 1, c);
		else if (a == 2)
			cout << segt.get_sum(b, c) << '₩n';
	}

	return 0;
}
```

북서풍: https://www.acmicpc.net/problem/5419

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>
#include <cmath>

#define DEBUG if (false)

using namespace std;

typedef vector<int> vi;
typedef pair<int, int> pii;
typedef vector<pii> vpii;
typedef long long ll;

typedef vi seg_type;

// idx는 무조건 1부터
class SegTree {
```

```
public:
	SegTree();
	~SegTree();

	// arr을 받아서 이 배열의 원소로 트리를 구성한다 (생성자)
	SegTree(seg_type& par_arr);

	// arr을 받아서 이 배열의 원소로 트리를 구성한다.
	void set(seg_type& par_arr);

	// node: 현재 노드 번호, [start, end): 현재 node의 담당 영역, [left, right): 내가 원하는 영역
	int sum(int left = -1, int right = -1, int node = -1, int start = -1, int end = -1);

	// node: 바꿀 노드 번호, diff: 더할 값, [start, end): 현재 node의 담당 영역
	void update(int change_node = -1, int diff = -1, int node = -1, int start = -1, int end = -1);

private:
	// arr[0] 부터 쓰자.
	seg_type arr;
	// tree[0] 부터 쓰자.
	seg_type tree;

	int a_size;
	int t_height;
	int t_size;

	// 현재 arr과 tree를 기반으로 재귀를 통하여 트리를 구성한다. (반드시 init에서 호출해야 함)
	// 구간은 [start, end) 이다!!!!!
	int set_rcrs(int node = -1, int start = -1, int end = -1);

	// 시작 노드가 0번인 기준.
	// node의 왼쪽 자식.
	int get_l(int node);
	// node의 오른쪽 자식.
	int get_r(int node);
	// node의 부모 자식.
```

```
        int get_p(int node);
        // node의 노드 영역 분할선.
        int get_mid(int node1, int node2);
};

SegTree::SegTree(seg_type& par_arr) {
        set(par_arr);
}

void SegTree::set(seg_type& par_arr) {
        arr = par_arr;// 밖에서 쓰던 애들은 arr[0]부터 채운다.
        a_size = arr.size();
        t_height = (int)ceil(log2(a_size)); // 트리의 높이: cell(log2(대상 배열의 크기)).
        t_size = (1 << (t_height + 1)) - 1; // 트리에 필요한 공간: = 2^(h + 1) - 1
        tree.resize(t_size);

        set_rcrs(0, 0, a_size);
}

int SegTree::set_rcrs(int node, int start, int end) {
        node = (node < 0 ? 0 : node);
        start = (start < 0 ? 0 : start);
        end = (end < 0 ? a_size : end);

        int s(start), e(end - 1); // 범위 조심 [start, end)로 들어왔다.
        DEBUG printf("node %d, start %d, end %d₩n", node, start, end);
        if (s == e) {
                return tree[node] = arr[s];
        }
        else {
                // 반으로 나누기.
                int mid = get_mid(start, end); // 1을 더하는 이유는 범위가 [s, e) 이기 때문.
                return tree[node] = set_rcrs(get_l(node), start, mid) + set_rcrs(get_r(node), mid,
end);
        }
}

int SegTree::sum(int left, int right, int node, int start, int end) {
        node = (node < 0 ? 0 : node);
```

```
	start = (start < 0 ? 0 : start);
	end = (end < 0 ? a_size : end);
	left = (left < 0 ? 0 : left);
	right = (right < 0 ? a_size : right);

	int s(start), e(end - 1); // 범위 조심 [start, end)로 들어왔다.
	int l(left), r(right - 1); // 범위 조심 [start, end)
	DEBUG printf("node %d, start %d, end %d₩n", node, start, end);
	if (e < l || r < s)  // 내가 원하는 구간이 아예 배제되는 경우
		return 0;
	else if (l <= s && e <= r) // 내가 원하는 구간에 속하는 경우 -> return하여 합 해줘야 함
		return tree[node];
	else {
		int mid = get_mid(start, end);
		return sum(left, right, get_l(node), start, mid) + sum(left, right, get_r(node), mid,
end);
	}
}


void SegTree::update(int change_node, int diff, int node, int start, int end) {

	node = (node < 0 ? 0 : node);
	start = (start < 0 ? 0 : start);
	end = (end < 0 ? a_size : end);
	change_node = (change_node < 0 ? 0 : change_node);
	diff = (diff < 0 ? 0 : diff);

	int s(start), e(end - 1); // 범위 조심 [start, end)로 들어왔다.
	if (change_node < s || change_node > e)
		return;

	tree[node] += diff;

	if (s != e) {
		int mid = get_mid(start, end);
		update(change_node, diff, get_l(node), start, mid);
		update(change_node, diff, get_r(node), mid, end);
	}
```

```
        else {
                arr[s] += diff; // 얘도 갱신 해줘야지...
        }
}

int SegTree::get_l(int node) {
        int ret = 2 * node + 1;

        if (0 <= ret || ret <= t_size)
                return ret;

        cout << "left 자식이 없습니다.₩n";
        return -123;
}
int SegTree::get_r(int node) {
        int ret = 2 * node + 2;

        if (0 <= ret || ret <= t_size)
                return ret;

        cout << "right 자식이 없습니다.₩n";
        return -321;
}
int SegTree::get_p(int node) {
        int ret = (int)(node - 1) / 2;

        if (node == 0)
                cout << "부모의 부모는 없습니다.₩n";

        if (0 <= ret || ret <= t_size)
                return ret;

        cout << "부모가 존재하지 않습니다.₩n";
        return -999;
}

int SegTree::get_mid(int node1, int node2) {
        return (node1 + node2 - 1) / 2 + 1;
}
```

```
const int MAX = 75000;
int T; // 테케
int N; // 섬의 수
ll ans;

vpii P; // p[i]: i번째 섬의 좌표
vi newY; // newY[i] = j. i번째로 작은 y좌표를 j로 맵핑한다.
vi S; // S[i] i번째 섬까지 오면서 y좌표에 존재하는 섬의수
SegTree sgt; // newY의 부분함 세그트리



auto cmp_y = [](pii& a, pii& b) {
	if (a.second < b.second)
		return true;
	return false;
};


auto cmp_xy = [](pii& a, pii& b) {
	if (a.first < b.first)
		return true;
	else if (a.first == b.first) {
		if (a.second > b.second)
			return true;
		else
			return false;
	}
	else {
		return false;
	}
};

void solve() {

	cin >> T;
	while (T--) {
```

```
cin >> N;

ans = 0;
S.resize(N);
P.clear();
P.reserve(N);
newY.resize(N);

int x, y;
for (int i = 0; i < N; i++)
{
	cin >> x >> y;
	P.emplace_back(x, y);
}

//for (int i = 0; i < MAX; i++)
//{
//	P.emplace_back(i, MAX - i - 1);
//}

sort(P.begin(), P.end(), cmp_y); // y 오름차순으로 정렬한다.
int cnt = 0;
newY[0] = 0;
for (int i = 1; i < P.size(); i++) { // 오름차순으로 새로운 좌표를 부여하고
	if (P[i].second != P[i - 1].second)
		cnt++;
	newY[i] = cnt;
}

for (int i = 0; i < P.size(); i++) // 적용
	P[i].second = newY[i];

sort(P.begin(), P.end(), cmp_xy); // x가 같다면 y좌표 순으로 정렬.

sgt.set(S);

for (int i = 0; i < P.size(); i++)  // 가장 좌측 상단부터 확인함.
{
	ans += sgt.sum(P[i].second);
```

```
                        sgt.update(P[i].second, 1);
                }
                cout << ans << "₩n";
        }
}
```

최솟값: https://www.acmicpc.net/problem/10868

```
class MySegmentTree
{
public:
        // 배열 arr에 대한 세그먼트 트리를 만든다.
        void sum_init(vector<ll>* arr)
        {
                mp_arr = arr;
                m_height = ceill(log2l((*mp_arr).size())); // n을 2^k꼴로 올림해서 최소 요구 높이 구함
                m_size = powl(2, m_height + 1) - 1; // 필요한 데이터 사이즈

                m_data.resize(m_size, 0);
               sum_init_recurse(1, 1, (*mp_arr).size()); // 노드 번호가 1부터 시작함. (계산 편의상)

        }

        void small_init(vector<ll>* arr)
        {
                mp_arr = arr;
                m_height = ceill(log2l((*mp_arr).size())); // n을 2^k꼴로 올림해서 최소 요구 높이 구함
                m_size = powl(2, m_height + 1) - 1; // 필요한 데이터 사이즈

                m_data.resize(m_size, 0);
                small_init_recurse(1, 1, (*mp_arr).size()); // 노드 번호가 1부터 시작함. (계산 편의상)

        }

        void sum_set_node(int i, ll n) // 외부에서 받아오는 건 0부터
        {
                sum_update(1, 1, (*mp_arr).size(), i + 1, n - (*mp_arr)[i]); // 내부적으로는 1부터
                (*mp_arr)[i] = n;
```

```
	}

	void small_set_node(int i, ll n)
	{

	}

	ll get_sum(int l, int r)  // 외부에서 받아오는 건 0부터
	{
		return m_sum(1, 1, (*mp_arr).size(), l + 1, r + 1);
	}

	ll get_small(int l, int r)  // 외부에서 받아오는 건 0부터
	{
		return m_small(1, 1, (*mp_arr).size(), l + 1, r + 1);
	}

	ll size()
	{
		return m_size;
	}

	ll height()
	{
		return m_height;
	}


private:
	vector<ll>* mp_arr;
	vector<ll> m_data;
	ll m_height = 0;
	ll m_size = 0;

	ll sum_init_recurse(int node, int start, int end)
	{
		if (start == end)
			return m_data[node - 1] = (*mp_arr)[start - 1];  // 노드 번호가 1부터 시
작함. (계산 편의상)
```

```
		else
		{
			int mid = (start + end) / 2;
			return m_data[node - 1] = sum_init_recurse(node * 2, start, mid) + sum_init_recurse(node * 2 + 1, mid + 1, end);
		}
	}

	ll small_init_recurse(int node, int start, int end)
	{
		if (start == end)
			return m_data[node - 1] = (*mp_arr)[start - 1]; // 리프 노드는 자기 자신의 값.
		else
		{
			int mid = (start + end) / 2;
			return m_data[node - 1] = min(small_init_recurse(node * 2, start, mid), small_init_recurse(node * 2 + 1, mid + 1, end)); // 최소값 리턴.
		}
	}

	void sum_update(int node, int start, int end, int idx, ll diff)
	{
		if (!(start <= idx && idx <= end)) // 갈필요 없는 경로
			return;

		m_data[node - 1] += diff; // 목적지 까지 가는 경로에 있는 모든 sum에 차를 더해줌.

		if (start != end)
		{
			int mid = (start + end) / 2;
			sum_update(node * 2, start, mid, idx, diff);
			sum_update(node * 2 + 1, mid + 1, end, idx, diff);
		}
	}

	ll m_sum(int node, int start, int end, int l, int r)
	{
```

```
		/*
		네 가지 경우로 나뉜다.
		1. 찾는 구간이 현재 구간과 전혀 겹치지 않음
		2. 찾는 구간이 현재 구간을 포함함
		3. 찾는 구간이 현재 구간에 포함됨 -> 더 쪼개야함
		4. 나머지 -> 더 쪼개야함
		*/

		if (l > end || r < start) // 전혀 겹치지 않음
			return 0;

		if (l <= start && end <= r) // 찾는 구간이 현재 구간을 포함함
			return m_data[node - 1];

		int mid = (start + end) / 2;

		return m_sum(node * 2, start, mid, l, r) + m_sum(node * 2 + 1, mid + 1, end, l, r);
	}

	ll m_small(int node, int start, int end, int l, int r)
	{
		if (l > end || r < start) // 전혀 겹치지 않음
			return 1000000001; // 최대 값

		if (l <= start && end <= r) // 찾는 구간이 현재 구간을 포함함
			return m_data[node - 1];

		int mid = (start + end) / 2;

		return min(m_small(node * 2, start, mid, l, r), m_small(node * 2 + 1, mid + 1, end,
l, r)); // 자기 아래 노드중 작은 걸 리턴
	}
};



MySegmentTree segt;
vector<ll> arr;
```

```
int N; // 수의 개수 10만 이하 자연수
int M; // 찾을 구간 10만 이하 자연수

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

	cin >> N >> M;
	arr.resize(N);
	for (int i = 0; i < N; i++)
	{
		cin >> arr[i];
	}
	segt.small_init(&arr);

	for (int i = 0; i < M; i++)
	{
		int a, b;
		cin >> a >> b;
		cout << segt.get_small(a - 1, b - 1) << '₩n';
	}

	return 0;
}
```

좌표 압축

좌표 압축: https://www.acmicpc.net/problem/18870

```
typedef vector<int> vi;

const int SIZE = 1000001;

int N;

vi p; // 원본
vi a;

void solve() {

	int tmp = 0;
```

```
	int idx = 0;

	p.reserve(SIZE);
	a.reserve(SIZE);

	cin >> N;
	for (int i = 0; i < N; i++)
	{
		cin >> tmp;
		p.push_back(tmp);
		a.push_back(tmp);
	}

	sort(a.begin(), a.end(), less<int>()); // 작은 것부터 = 오름차순 정렬
	auto rm_i = unique(a.begin(), a.end()); // 중복되는 원소를 뒤로 밀고 인덱스(iterator) 반환.
	a.erase(rm_i, a.end()); // 중복 제거 후


	for (int i = 0; i < N; i++)
	{
		idx = lower_bound(a.begin(), a.end(), p[i]) - a.begin(); // p[i]가 a의 어느 index에 있는지.
		cout << idx << " "; // 몇 번째 - 1 = index이므로 그대로 출력
	}
}
```

이분 매칭

열혈강호: https://www.acmicpc.net/problem/11375

```
vector<int> staff[MAX]; // staff[i]: i번째 A의 원소가 원하는 B의 원소들
int work[MAX]; // work[i] = x는 i를 하는 사람이 x라는 뜻.
bool visit[MAX]; // 이미 매칭해서 볼필요 없는 직원

int N; // 직원 수 [1, 1000]
int M; // 일의 수 [1, 1000]
int ans;

//x번째 노드가 매칭에 성공하면 True, 실패하면 False. dfs로 구현.
bool bimatch(int x)
{
```

```
	// x 직원이 할 수 있는 모든 일 t에 대하여
	for (int i = 0; i < staff[x].size(); i++)
	{
		int t = staff[x][i];
		// 일 t와 직원 x가 매칭된적이 있으면 또 볼필요가 없다.
		if (visit[t]) continue;
		visit[t] = true;
		// t를 하는 사람이 없거나, 원래 t를 하던 사람이 다른 일을 할 수 있으면
		if (work[t] == 0 || bimatch(work[t]))
		{
			work[t] = x; // t는 x가 한다.
			return true;
		}
	}

	return false; // 다른 직원들과 중복되지 않으면서 할 수 있는 일이 없는 직원이다.

}

void solve()
{
	int n_works;
	int tmp_work;
	cin >> N >> M;

	for (int i = 1; i <= N; i++)
	{
		cin >> n_works;
		for (int j = 0; j < n_works; j++)
		{
			cin >> tmp_work;
			staff[i].push_back(tmp_work);
		}
	}
	for (int i = 1; i <= N; i++)
	{
		fill(visit + 1, visit + M + 1, false); // DFS 도는데 확인한 곳을 초기화.
		if (bimatch(i)) {
			ans++;
```

```
                }
        }
        cout << ans << "₩n";
}
```

이분 탐색

K번째 수: https://www.acmicpc.net/problem/1300

```
typedef long long ll;
ll N, K; // N 배열 크기, K 자연수
ll leqNum(ll n)
{
        ll ret = 0;

        for (ll i = 1; i <= N; i++) // 행 선택
        {
                if (i > n) // n행 초과면 n보다 작은 수가 없기 때문.
                        break;
                ret += min(n / i, N); // f(i, n)에 해당
        }

        return ret;
}

void solution()
{
        cin >> N >> K;

        /*A.resize(N, vi (N));
        B.reserve(N * N);

        test();*/

        ll left = 1;
        ll right = N * N;
        ll mid, num, ans;
        while (left <= right)
        {
```

```
                mid = (left + right) / 2;
                num = leqNum(mid);
                if (num < K)
                        left = mid + 1;
                else
                {
                        ans = mid;
                        right = mid - 1;
                }
        }
        cout << ans << '₩n';
}
```

소수

소수 찾기: https://www.acmicpc.net/problem/1978

```
int prime_arr[1001] = { 1 }; // 1 소수 0 소수 아님
int in_arr[1001];

int is_prime(int n)
{
        if (prime_arr[n] == 1)
                return 1;
}

void solution()
{
        int N;
        cin >> N;
        for (int i = 0; i < N; i++)
                cin >> in_arr[i];
        prime_arr[1] = 0;
        for (int i = 2; i <= 1000; i++) // 소수인지 판정할 수
        {
                int is_prime = 1;
                for (int j = 2; j <= 1000; j++) // 나눠볼 수
                {
                        if (prime_arr[j] == 0) // 소수가 아니라고 판정된 수들로 나눠볼 필요가
없음.
```

```
                continue;
            if (j * j > i) // 이보다 큰 j에 대해서 계산 불필요
                break;
            is_prime = i % j;
            if (is_prime == 0) // 소수라고 확인되면 종료
                break;
        }
        DEBUG cout << i << " " << is_prime << endl;
        prime_arr[i] = is_prime;
    }
    int cnt = 0;
    for (int i = 0; i < N; i++)
        if (prime_arr[in_arr[i]] != 0)
            cnt++;
    cout << cnt;
}
```

기하학

정사각형: https://www.acmicpc.net/problem/9015

```
int T;
int N;
typedef pair<int, int> pii;

vector<pii> ve;
set<pii> st;
int ans;

void solve() {

    ve.resize(3000);

    cin >> T;

    while (T--) {

        ans = 0;
        st.clear();
```

```
		cin >> N;
		for (int i = 0; i < N; i++)
		{
			cin >> ve[i].first >> ve[i].second;
			ve[i].first *= 2;
			ve[i].second *= 2;
			st.insert(ve[i]);
		}

		for (int i = 0; i < N; i++)
		{
			for (int j = i + 1; j < N; j++)
			{
				int sx(ve[i].first), sy(ve[i].second), tx(ve[j].first), ty(ve[j].second);
				int a(sx), b(sy), c((sx + tx)/2), d((sy + ty)/2);
				// c + b - d, d - a + c가 평면 범위 벗어나는 경우 쳐내 꼭
				bool cond1 = st.find({ c + b - d, d - a + c }) != st.end();
				if (!cond1)
					continue;

				bool cond2 = st.find({ c - b + d, d + a - c }) != st.end();
				if (!cond2)
					continue;

				int tmp = (c + b - d - a) * (c + b - d - a) + (d - a + c - b) * (d -
a + c - b);
				ans = max(ans, tmp);
			}
		}
		cout << ans/4 << "₩n";
	}
}
```

정렬

단어 정렬: https://www.acmicpc.net/problem/1181

```
void solution()
{
	int N;
```

```
	string str;
	auto str_cmp = [](string s1, string s2) {
		if (s1.length() < s2.length()) // 정렬되어있다 = true
			return true;
		else if (s1.length() == s2.length())
			return s1 < s2;
		else
			return false;
	};
	set < string, decltype(str_cmp) > st(str_cmp);
	cin >> N;
	while (N-- > 0)
	{
		cin >> str;
		st.insert(str);
	}
	for (auto x : st)
	{
		cout << x << "₩n";
	}
}
```

매우 큰 배열 다루기

매우 큰 수 덧셈, 뺄셈

```
#include <iostream>
#include <math.h>
#include <complex>
#include <vector>
#include <algorithm>

using namespace std;

template <typename T>
T power_of_2_ge_than(T n) {
	T ret = 1;
```

```
	while (n > ret) ret <<= 1;
	return ret;
}

typedef complex<double> base;

void fft(vector<base>& a, bool inv) {
	int n = (int)a.size();
	for (int i = 1, j = 0; i < n; i++) {
		int bit = n >> 1;
		while (!((j ^= bit) & bit)) bit >>= 1;
		if (i < j) swap(a[i], a[j]);
	}
	for (int i = 1; i < n; i <<= 1) {
		double x = (inv ? 1 : -1) * M_PI / i;
		base w = { cos(x), sin(x) };
		for (int j = 0; j < n; j += i << 1) {
			base th(1);
			for (int k = 0; k < i; k++) {
				base tmp = a[i + j + k] * th;
				a[i + j + k] = a[j + k] - tmp;
				a[j + k] += tmp;
				th *= w;
			}
		}
	}
	if (inv) {
		for (int i = 0; i < n; i++) a[i] /= n;
	}
}

vector<int> multiply(vector<int>& A, vector<int>& B) {
	vector<base> a(A.begin(), A.end());
	vector<base> b(B.begin(), B.end());
	int n = power_of_2_ge_than(max(a.size(), b.size())) * 2;

	a.resize(n);		b.resize(n);
	fft(a, false);		fft(b, false);
```

```
	for (int i = 0; i < n; i++)
		a[i] *= b[i];
	fft(a, true);

	vector<int> ret(n);
	for (int i = 0; i < n; i++)
		ret[i] = (int)round(a[i].real());
	return ret;
}


// a = a + b * (10 ^ k);
void addTo(vector<int>& a, const vector<int>& b, int k)
{
	// a = 123, b = 4321, k = 1라고 하면
	// 123 + 4321 * 10 ^ 1
	// {3, 2, 1} + {0, 1, 2, 3, 4} 이렇게 되어있는 셈.

	// 먼저 길이를 맞춰준다 {3, 2, 1, 0} {1, 2, 3, 4}
	if (a.size() < b.size() + k)
		a.resize(b.size() + k);
	// k부터 시작하는 이유는 그 이전은 0이기 때문. {3, 2, 1, 0} {0, 1, 2, 3, 4} 이렇게 계산하겠
다는 말.
	int carry = 0;
	for (int i = k; i < b.size() + k; i++)
	{
		a[i] += b[i - k] + carry;
		carry = a[i] / 10; // 올림수
		a[i] %= 10; // 올리고 남은 것
	}
	if (carry > 0) // 마지막 자리에서 올림수가 나오면
		a.push_back(carry); // 추가해줌
}

void normalize(vector<int>& num)
{
	num.push_back(0);
	for (int i = 0; i < num.size() - 1; i++)
	{
```

```
                if (num[i] < 0)
                {
                        int borrow = (abs(num[i]) + 9) / 10;
                        num[i + 1] -= borrow;
                        num[i] += borrow * 10;
                }
                else
                {
                        num[i + 1] += num[i] / 10;
                        num[i] %= 10;
                }
        }
        while (num.back() == 0 && num.size() > 1)
                num.pop_back();
}


string tmp;
vector<int> A[3];

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

        A[0].reserve(300001);
        A[1].reserve(300001);

        for (int i = 0; i < 2; i++)
        {
                cin >> tmp;
                for (int j = tmp.size() - 1; j >= 0; --j)
                {
                        A[i].emplace_back(tmp[j] - '0');
                }
        }

        A[2] = multiply(A[0], A[1]);

        int r = 0;
```

```
	vector<int>& ans = A[0];
	ans = vector<int>(1, 0);

	for (int i = 0; i < A[2].size(); i++, r++)
	{
		vector<int> tmp_vec;
		while (A[2][i] > 0)
		{
			tmp_vec.push_back(A[2][i] % 10);
			A[2][i] /= 10;
		}
		addTo(ans, tmp_vec, r);
	}

	normalize(ans);

	for (int i = ans.size() - 1; i >= 0; i--)
	{
		cout << ans[i];
	}

	/*long long sum = 0;
	long long r = 1;

	for (int i = 0; i < A[2].size(); ++i)
	{
		sum += A[2][i] * r;
		r *= 10;
	}
	cout << sum;*/
	return 0;
}
```

무한 정밀도 실수 -> 파이썬

```
from decimal import *
from math import *
import sys
```

```
# from sys import stdin
# for n in map(int, stdin.read().split()): #이렇게 바로 map에 넣어도 됨.

ins = sys.stdin.readline().rstrip().split(' ') // 입력 가져와서 파싱
getcontext().prec = 100 // 부동소수점 정확도 설정 (저 자리수만큼 보장은 아닌듯. 넉넉필수)

a = Decimal(ins[0])
b = Decimal(ins[1])
c = Decimal(ins[2])

# 일반적인 연산. Math 함수도 적용됨

print("%.13f" % S)
```

입출력 다루기

비교 함수 만들기

자주 사용하는 algorithm 헤더 함수