Graduation Gazua!!!

Soongsil University
2018 Graduation Gazua!!! (sys7961,hyo123bin,skdudn321)

Table of Context

Graph Algorithm

Dinic (Maximum Flow)

```
using std::queue;
const int INF = 0x7fffffff / 2;
struct Dinic {
	struct edge {
		int to, cap, rev;
		edge() {}
		edge(int a, int b, int c) :to(a), cap(b), rev(c) {}
	};
	vector<vector<edge>> G;
	vector<int> level;
	vector<int> iter;
	int source;
	int sink;
	Dinic(int n, int source, int sink) :source(source), sink(sink) {
		G.resize(n);
		level.resize(n);
		iter.resize(n);
	}
	void add_edge(int from, int to, int cap)
	{
		edge e(to, cap, G[to].size());
		edge re(from, 0, G[from].size());
		G[from].push_back(e);
		G[to].push_back(re);
	}
	bool bfs(int s) {
		std::fill(level.begin(), level.end(), -1);
		std::fill(iter.begin(), iter.end(), 0);
		queue<int> que;
		level[s] = 0;
		que.push(s);
		while (!que.empty()) {
			int v = que.front();
			que.pop();
			for (int i = 0; i < G[v].size(); i++) {
				edge & e = G[v][i];
				if (e.cap > 0 && level[e.to] < 0) {
					level[e.to] = level[v] + 1;
					que.push(e.to);
				}
			}
		}
		return level[sink] >= 0;
	}
	int dfs(int v, int t, int f) {
		if (v == t) return f;
		for (int &i = iter[v]; i < G[v].size(); i++) {
			edge &e = G[v][i];
			if (e.cap > 0 && level[v] < level[e.to]) {
				int d = dfs(e.to, t, std::min(f, e.cap));
				if (d > 0) {
					e.cap -= d;
					G[e.to][e.rev].cap += d;
					return d;
				}
			}
		}
		return 0;
	}
	int flow() {
		int flow = 0;
		while (bfs(source)) {
			int f;
			while ((f = dfs(source, sink, INF)) > 0) {
				flow += f;
			}
		}
		return flow;
	}
};
```

Bipartite Matching

```
using std::vector;
struct BipartiteMatching {
	vector<int> L;
	vector<int> R;
	vector<vector<int>> G;
	vector<int> dist;
	BipartiteMatching(int n, int m) {
		L.resize(n, -1);
		R.resize(m, -1);
		G.resize(n);
		dist.resize(n);
	}
	void add_edge(int from, int to) {
		G[from].push_back(to);
	}
```

```
bool bfs() {
	std::queue<int> que;
	for (int i = 0; i < L.size(); i++) {
		if (L[i] == -1) {
			dist[i] = 0;
			que.push(i);
		}
		else {
			dist[i] = -1;
		}
	}
	bool flag = false;
	while (!que.empty()) {
		int idx = que.front();
		que.pop();
		for (int to : G[idx]) {
			if (R[to] == -1)flag = true;
			else if (dist[R[to]] == -1) {
				dist[R[to]] = dist[idx] + 1;
				que.push(R[to]);
			}
		}
	}
	return flag;
}
bool dfs(int idx) {
	for (int to : G[idx]) {
		if (R[to] == -1 || (dist[idx]<dist[R[to]] && dfs(R[to]))) {
			R[to] = idx;
			L[idx] = to;
			return true;
		}
	}
	dist[idx] = -1;
	return false;
}
int matching() {
	int ret = 0;
	while (bfs()) {
		for (int i = 0; i < L.size(); i++) {
			if (L[i] == -1 && dfs(i)) {
				ret++;
			}
		}
	}
	return ret;
}
};
```

Minimum Cost Maximum Flow using Primal Dual

```
struct MCMF {
	struct edge {
		int to;
		int cap;
		int cost;
		int rev;
	};
	vector<vector<edge>> G;
	vector<int> dist;
	vector<int> chk;
	vector<pair<int, int>> from;
	vector<int> pi;
	vector<int> level;
	vector<int> iter;
	int source, sink;
	const int INF = 0x7fffffff / 2;
	MCMF(int n, int source, int sink) :source(source), sink(sink) {
		G.resize(n);
		dist.resize(n);
		chk.resize(n);
		from.resize(n, { -1,-1 });
		level.resize(n);
		iter.resize(n);
	}
	void add_edge(int u, int v, int cap, int cost) {
		edge ori{ v,cap,cost,G[v].size() };
		edge rev{ u,0,-cost,G[u].size() };
		G[u].push_back(ori);
		G[v].push_back(rev);
	}
	void getPotential() {
		std::fill(dist.begin(), dist.end(), INF);
		std::fill(chk.begin(), chk.end(), false);
		std::fill(from.begin(), from.end(), std::make_pair(-1, -1));
		std::fill(level.begin(), level.end(), INF);
		std::queue<int> que;
		que.push(source);
		dist[source] = 0;
		level[source] = 0;
		while (!que.empty()) {
			int idx = que.front();
```

```
            que.pop();
            chk[idx] = false;
            for (int i = 0; i < G[idx].size(); i++) {
                auto& e = G[idx][i];
                int to = G[idx][i].to;
                if (e.cap > 0 && dist[to] > dist[idx] + e.cost) {
                    dist[to] = dist[idx] + e.cost;
                    level[to] = level[idx] + 1;
                    from[to] = { idx,i };
                    if (!chk[to]) {
                        chk[to] = true;
                        que.push(to);
                    }
                }
            }
        }
        pi = dist;
    }
    bool dijkstra() {
        std::fill(dist.begin(), dist.end(), INF);
        std::fill(chk.begin(), chk.end(), false);
        std::fill(from.begin(), from.end(), std::make_pair(-1, -1));
        std::fill(level.begin(), level.end(), INF);
        int n = dist.size();
        dist[source] = 0;
        using node = pair<int, int>;
        std::priority_queue<node, vector<node>, std::greater<node>> heap;
        heap.emplace(dist[source], source);
        while (!heap.empty()) {
            int idx = heap.top().second;
            heap.pop();
            if (chk[idx])continue;
            chk[idx] = true;
            for (int i = 0; i < G[idx].size(); i++) {
                edge& e = G[idx][i];
                if (e.cap > 0 && dist[e.to] > dist[idx] + e.cost - pi[e.to] + pi[idx]) {
                    assert(!chk[e.to]);
                    dist[e.to] = dist[idx] + e.cost - pi[e.to] + pi[idx];
                    level[e.to] = level[idx]+1;
                    from[e.to] = { idx,i };
                    heap.emplace(dist[e.to], e.to);
                }
            }
        }
        int idx = sink;
        int cap = INF;
        int cost = dist[sink] + pi[sink] - pi[source];
        if (dist[sink] == INF || cost > 0) {//Minimum Cost Flow
            return false;
        }
        return true;
    }
    int dfs(int v, int t, int f) {
        if (v == t) return f;
        for (int &i = iter[v]; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && dist[v] + e.cost - pi[e.to] + pi[v] == 0 && level[v] <
level[e.to]) {
                int d = dfs(e.to, t, std::min(f, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }


    pair<int, int> flow() {
        int total_cap = 0;
        int total_cost = 0;

        getPotential();
        while (dijkstra()) {
            std::fill(iter.begin(), iter.end(), 0);
            int f;
            while ((f = dfs(source, sink, INF)) > 0) {
                total_cap += f;
                total_cost += f*(dist[sink] + pi[sink] - pi[source]);
            }
            for (int i = 0; i < dist.size(); i++) {
                if (dist[i] < INF)
                    pi[i] += dist[i];
            }
        }
        return{ total_cap,total_cost };
    }
};
```

Maximum Flow Special Case

L-R Maximum Flow
방법 1) a번 정점에서 b번 정점으로 가는 하한 l, 상한 r인 간선이 있을 때, a번 정점에서 b번 정점으로 가는 유량 l, 비용 -1인 간선, 유량 r-l, 비용 0인 간선으로 만든 뒤 mincost-maxflow
방법 2) a번 정점에서 b번 정점으로 가는 하한 l, 상한 r인 간선이 있을 때, 새로운 source에서 b번 정점으로 가는 유량 l인 간선 추가, a번 정점에서 새로운 sink로 가는 유량 l인 간선 추가, a번 정점에서 b번 정점으로 가는 유량 r-l인 간선으로 만들고, 기존 sink에서 기존 source로 가는 유량 무한인 간선 추가 이후 최대 유량이 l의 합이 되는지 확인

부분 순서 집합의 반사슬의 크기
4) 부분 순서 집합의 최대 반사슬의 크기가 최소 Path Cover의 크기와 같은 이유 (Dilworth's theorem)
정의 3. 부분 순서 집합은 사이클이 없는 방향성 그래프로, 임의의 정점 i, j, k에 대해서 i에서 j로 가는 에지와, j에서 k로 가는 에지가 있으면, i에서 k로 가는 에지가 항상 존재하는 성질을 가진다.
정의 4. 부분 순서 집합의 반사슬은, 정점의 부분집합으로, 부분집합의 임의의 정점 i, j에 대해, i -> j로도, j -> i로도 에지가 없는 집합을 뜻한다.
쉽게 설명하자면, DAG에 플로이드 돌리면 부분 순서 집합이다. 대전 리저널에도 나왔었고 (2012 K) 그렇게 낯선 개념은 아니다. 예시 문제를 들면 "DAG가 주어졌을 때, 서로 경로가 없는 최대 정점 집합을 출력하라" 같은 문제가 있겠다. 플로이드를 돌려서 부분 순서 집합으로 만들어 주고 반사슬을 구하면 된다.
부분 순서 집합의 최대 반사슬은 최소 Path Cover라는 내용이 Dilworth's Theorem이다. Path Cover는 대충 이렇게 구할 수 있다. 이제 이것의 크기와 최대 반사슬의 크기가 같음을 보인다.

Theorem. 부분 순서 집합에서, 최소 Path Cover의 크기 == 최대 반사슬의 크기
Minimum Cut 찾기
Flood Fill 후 간선들을 보며 한 쪽은 방문한 점이고, 다른 한 쪽은 방문하지 않은 점이면 Cut인 간선(Flood Fill하면서 하면 안됨)
Minimum Vertex Cover 찾기
Flood Fill 후 왼쪽에서 방문하지 않은 점(매칭 된 점들 중 일부), 오른쪽에서 방문한 점(매칭 된 점들 중 일부) 가 Minimum Vertex Cover에 포함되는 점이다.

Gomory Hu tree (all pair flow)

```
const int N = 3010;
vector<pii> G[N]; // 양방향필요
int n;//vertex
int m;
int flow[N][N];
int p[N];
void gomoriHu() {
	for (int i = 0; i < n; i++)p[i] = 0;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			flow[i][j] = INF;
		}
	}
	for (int i = 1; i < n; i++) {
		Dinic D(n + 1, i, p[i]);
		for (int i = 0; i < n; i++) {
			for (pii e : G[i]) {
				D.add_edge(i, e.first, e.second);
			}
		}
		int f = D.flow();
		for (int j = i + 1; j <= n - 1; j++) {
			if (D.level[j] >= 0 && p[j] == p[i])p[j] = i;
		}
		flow[i][p[i]] = flow[p[i]][i] = f;
		for (int j = 0; j < i; j++) {
			flow[j][i] = flow[i][j] = std::min(flow[j][p[i]], f);
		}
	}
	for (int i = 0; i < n; i++) {
		flow[i][i] = 0;
	}
}
```

Articulation Point & Bridge (simple)

pair<vector<bool>,vector<pii>> cut_VE : pair<단절점,단절선>

```
struct Tarjan {
	vector<vector<int>> G;
	vector<bool> chk;
	vector<int> dis, low;
	int n, dfo = 0;
	Tarjan(int n) :n(n) {
		chk.resize(n + 1);		dis.resize(n + 1);		low.resize(n+ 1);
		G.resize(n + 1, vector<int>(n + 1));
	}

	void add_edge(int from, int to) {
		G[from].push_back(to);
	}
	void init() {
		fill(chk.begin(), chk.end(), 0);
		fill(dis.begin(), dis.end(), 0);
		fill(low.begin(), low.end(), 0);
		dfo = 0;
```

```
}
//tarjan scc
void TJ_dfs(int x, vector<vector<int>> &scc, vector<int> &S) {
	chk[x] = true;
	dis[x] = low[x] = ++dfo;
	S.push_back(x);
	for (auto &to : G[x]) {
		if (chk[to])   // back , cross
			low[x] = std::min(low[x], dis[to]);

		else if (dis[to] == 0) {
			TJ_dfs(to, scc, S);
			low[x] = std::min(low[x], low[to]);
		}
	}

	if (dis[x] == low[x]) {
		int last = scc.size();
		scc.push_back(vector<int>());
		while (!S.empty()) {
			int idx = S.back(); S.pop_back();
			chk[idx] = false;
			scc[last].push_back(idx);
			if (idx == x) break;
		}
		std::sort(scc[last].begin(), scc[last].end());
	}
}
vector<vector<int>> TJ_scc() { // 1 indexed
	init();
	vector<vector<int>> scc;
	vector<int> S;
	for (int i = 1; i <= n; i++) if (dis[i] == 0) TJ_dfs(i, scc, S);
	return scc;
}

//cut_VE
void cut_dfs(int x, vector<bool> &cut, vector<int> &p, vector<pii>& edge) {
	chk[x] = true;
	dis[x] = low[x] = ++dfo;

	int child = 0;
	for (auto &to : G[x]) {
		if (to == p[x]) continue;

		if (chk[to])
			low[x] = std::min(low[x], dis[to]);
		else {
			child++;
			p[to] = x;
			cut_dfs(to, cut, p, edge);
			low[x] = std::min(low[x], low[to]);

			// 단절점
			if (p[x] == 0 && child >= 2) cut[x] = true;
			if (p[x] != 0 && low[to] >= dis[x]) cut[x] = true;

			//단절선
			if (low[to] > dis[x])
				edge.push_back({ to,x });
		}
	}
}
pair<vector<bool>, vector<pii>> cut_VE() {//1 indexed
	init();
	vector<bool> cut;
	vector<int> p;
	cut.resize(n + 1, 0); p.resize(n + 1, 0);
	vector<pii> edge;

	for (int i = 1; i <= n; i++)
		if (dis[i] == 0)
			cut_dfs(i, cut, p, edge);

	return std::make_pair(cut, edge);
}
};
```

Articulation Point & Bridge (advanced)
vertex_query (a, b, c) : c를 자르면 a, b가 분할되지 않는가
edge_query (u, v, a, b) : 간선(u,v)를 자르면 a, b가 분할되지 않는가

```
struct Tarjan_Query { // query

	bool rdy = false;
	struct dot {
		bool ar = false;
		int s = 0, e = 0, low = 0, dep = 0;
	};
	vector<dot> D;
	vector<vector<int>> G;
```

```
vector<vector<int>> p;
int dfo = 0,n;

Tarjan_Query(int n) :n(n){
	D.resize(n + 1);
	G.resize(n + 1, vector<int>());

	p.resize(log2(n) + 1, vector<int>(n + 1));
}

void add_edge(int a, int b) {
	G[a].push_back(b);
}
void dfs(int a, int depth) {

	D[a].s = D[a].low = ++dfo;
	D[a].dep = depth;

	int child = 0;
	for (auto &next : G[a]) {
		if (D[next].s == 0) {//tree edge
			p[0][next] = a;
			child++;
			dfs(next, depth + 1);
			D[a].low = std::min(D[a].low, D[next].low);
			//단절점
			if (p[0][a] == 0 && child >= 2) D[a].ar = true;
			if (p[0][a] != 0 && D[next].low >= D[a].s) D[a].ar = true;
		}
		else if (next != p[0][a])
			D[a].low = std::min(D[a].low, D[next].s);
	}
	D[a].e = ++dfo;
}
int near_child(int idx, int find) {
	int dx = D[find].dep - D[idx].dep - 1;
	for (int i = 0; i < 17; i++)
		if ((dx >> i) & 1) find = p[i][find];
	return find;
}
bool sub(int u, int v) { return D[u].s <= D[v].s && D[v].e <= D[u].e; }


void ready() {
	dfs(1, 1);
	for (int i = 1; (1 << i) <= n; i++)
		for (int j = 1; j <= n; j++)
			p[i][j] = p[i - 1][p[i - 1][j]];
}
bool vertex_query(int a, int b, int c) { // c를 끊으면 a-b를 갈 수 있는가
	if (!rdy) rdy = true, ready();
	if (D[c].ar == false) return true;
	if (!sub(c, a) && !sub(c, b)) return true; //외부 외부

	if (sub(c, a) == sub(c, b)) {	//내부 내부
		int x = near_child(c, a), y = near_child(c, b);
		if (x == y) return true;
		else if (D[x].low < D[c].s && D[y].low < D[c].s) return true;
		return false;
	}
	//내부 외부
	int x;
	if (sub(c, a))	x = near_child(c, a);
	else x = near_child(c, b);

	if (D[x].low < D[c].s) return true;
	return false;
}
bool edge_query(int u, int v, int a, int b) { // u-v를 끊으면 a-b를 갈 수 있는가
	if (!rdy) rdy = true, ready();
	if (D[u].s < D[v].s) std::swap(u, v);
	//단절선아님
	if (D[u].low <= D[v].s) return true;
	//단절선
	if (sub(u, a) == sub(u, b)) return true;
	return false;
}
};}
```

Heavy Light Decomposition

```
struct HeavyLightDecomposition {
	vector<int> parent;
	vector<bool> root;
private:
	vector<vector<int>> G;
	vector<int> weight;
	vector<int> top;
	vector<int> idx;
	vector<int> check;
	template <typename T>
	struct indexed_tree {
		vector<T> unit, sum;
```

```
int k = 1;
indexed_tree(int n) {
	while (k < n)k *= 2;
	unit.resize(k * 3, 0);
	sum.resize(k * 3, 0);
}
void update(int left, int right, T val) {
	int L = left + k, R = right + k;
	int Ls = L, Rs = R;
	T Lsum = 0, Rsum = 0;
	int len = 1;
	while (Ls >= 1) {
		sum[Ls] += Lsum;
		sum[Rs] += Rsum;

		if (L <= R) {
			if (L % 2 == 1) {
				unit[L] += val;
				sum[L] += val*len;
				Lsum += val*len;
			}
			if (R % 2 == 0) {
				unit[R] += val;
				sum[R] += val*len;
				Rsum += val*len;
			}
		}
		L = (L + 1) / 2, R = (R - 1) / 2;
		Ls /= 2; Rs /= 2;
		len *= 2;
	}
}
T query(int left, int right) {
	int L = left + k, R = right + k;
	int Ls = L, Rs = R;
	T ret = 0;
	int Llen = 0, Rlen = 0;
	int len = 1;
	while (Ls >= 1) {
		ret += unit[Ls] * Llen + unit[Rs] * Rlen;

		if (L <= R) {
			if (L % 2 == 1) {
				ret += sum[L];
				Llen += len;
			}
			if (R % 2 == 0) {
				ret += sum[R];
				Rlen += len;
			}
		}
		L = (L + 1) / 2, R = (R - 1) / 2;
		Ls /= 2; Rs /= 2;
		len *= 2;
	}
	return ret;
	}
};
indexed_tree<int> tree;
//LCA
vector<vector<int>> ancestor;
vector<int> height;
int lca_log = 20;
void processing(int node, int Parent) {
	weight[node] = 1;
	ancestor[0][node] = parent[node] = Parent;
	for (int p : G[node]) {
		if (p == Parent) continue;
		height[p] = height[node] + 1;
		processing(p, node);
		weight[node] += weight[p];
	}
}
int getLCA(int a, int b) {
	if (height[a] < height[b]) std::swap(a, b);
	int dx = height[a] - height[b];
	for (int i = 0; i < lca_log; i++)
		if ((dx >> i) & 1)
			a = ancestor[i][a];
	if (a == b) return a;
	for (int i = lca_log; i >= 0; i--)
		if (ancestor[i][a] != ancestor[i][b])
			a = ancestor[i][a], b = ancestor[i][b];
	return ancestor[0][a];
}
void initLCA(int root) {
	ancestor.resize(lca_log + 5, vector<int>(G.size()));
	height.resize(G.size(), 0);
	processing(root, root);
	for (int i = 1; (1 << i) < G.size(); i++)
		for (int j = 1; j < G.size(); j++)
			ancestor[i][j] = ancestor[i - 1][ancestor[i - 1][j]];
```

```
    }

    int cnt = 1;
    void chaining(int node, int Top) {
        idx[node] = cnt;
        top[node] = Top;
        check[node] = true;
        cnt++;
        std::priority_queue<pair<int, int>> heap;
        for (int next : G[node]) {
            if (next != parent[node]) {
                heap.push({ weight[next],next });
            }
        }
        if (heap.empty()) { // is leaf
            return;
        }

        chaining(heap.top().second, Top);
        heap.pop();
        while (!heap.empty()) {
            chaining(heap.top().second, heap.top().second);
            heap.pop();
        }
    }
public:

    bool edge;
    HeavyLightDecomposition(int n, bool edge) :edge(edge), tree(n + 1) {
        G.resize(n + 1);
        weight.resize(n + 1);
        top.resize(n + 1);
        idx.resize(n + 1);
        parent.resize(n + 1, -1);
        check.resize(n + 1, false);
        root.resize(n + 1, false);
    }
    void add_edge(int a, int b) {
        G[a].push_back(b);
        G[b].push_back(a);
    }
    void init() {
        for (int i = 1; i < check.size(); i++) {
            if (!check[i]) {
                root[i] = true;
                initLCA(i);
                chaining(i, i);
            }
        }
    }
    void update(int A, int B, int Cost) {
        int lca = getLCA(A, B);
        while (true) {
            if (top[A] == top[lca]) {
                tree.update(idx[lca], idx[A], Cost);
                break;
            }
            else {
                tree.update(idx[top[A]], idx[A], Cost);
                A = parent[top[A]];
            }
        }
        while (true) {
            if (top[B] == top[lca]) {
                tree.update(idx[lca], idx[B], Cost);
                break;
            }
            else {
                tree.update(idx[top[B]], idx[B], Cost);
                B = parent[top[B]];
            }
        }
        if (edge) {
            tree.update(idx[lca], idx[lca], -Cost * 2);
        }
        else {
            tree.update(idx[lca], idx[lca], -Cost);
        }
    }
    int query(int A, int B) {
        int lca = getLCA(A, B);
        int ret = 0;
        while (true) {
            if (top[A] == top[lca]) {
                ret += tree.query(idx[lca], idx[A]);
                break;
            }
            else {
                ret += tree.query(idx[top[A]], idx[A]);
                A = parent[top[A]];
            }
        }
```

```
        while (true) {
            if (top[B] == top[lca]) {
                ret += tree.query(idx[lca], idx[B]);
                break;
            }
            else {
                ret += tree.query(idx[top[B]], idx[B]);
                B = parent[top[B]];
            }
        }
        if (edge) {
            ret -= tree.query(idx[lca], idx[lca]) * 2;
        }
        else {
            ret -= tree.query(idx[lca], idx[lca]);
        }
        return ret;
    }
};
```

2-SAT

scc의 번호를 매긴다. 위상 순서 뒤쪽에 있는 정점이 True
x를 무조건 true 로 만들고 싶다. => 전처리 (x -> ㄱx) 간선 추가 후 2sat 돌린다.

Centroid Decomposition

```
#include<bits/stdc++.h>
using std::vector;
using std::pair;
using lint = long long int;
using pii = std::pair<int, int>;


struct edge {
    int to;
    int rev;
    bool vaild;
};

vector<edge> G[100100];
bool chk[100100];
int weight[100100];
int parent[100100];
void add_edge(int a, int b) {
    edge e = { b,G[b].size(),true };
    edge re = { a,G[a].size(),true };
    G[a].push_back(e);
    G[b].push_back(re);
}

int dfs(int idx) {
    chk[idx] = true;
    weight[idx] = 1;
    for (edge& e : G[idx]) {
        if (e.vaild && !chk[e.to]) {
            parent[e.to] = idx;
            weight[idx] += dfs(e.to);
        }
    }
    chk[idx] = false;
    return weight[idx];
}
int find_centroid(int idx) {
    parent[idx] = -1;
    dfs(idx);
    int n = weight[idx];
    while (true) {
        bool flag = false;
        for (edge& e : G[idx]) {
            if (e.vaild && parent[idx] != e.to && weight[e.to]>n / 2) {
                idx = e.to;
                flag = true;
                break;
            }
        }
        if (!flag)break;
    }
    return idx;
}
struct node {
    std::unordered_set<int> V;
};
node P[100100];
vector<int> CG[100100];
bool chk2[100100];
int divide(int idx) {
    idx = find_centroid(idx);
    //calculate
    for (edge& e : G[idx]) {
        if (e.vaild) {
            e.vaild = false;
            G[e.to][e.rev].vaild = false;
```

```
                /*
                        find answer
                */
                CG[idx].push_back(divide(e.to));
            }
        }
        P[idx].V.insert(idx);
        for (int to : CG[idx]) {
            for (int i : P[to].V) {
                P[idx].V.insert(i);
            }
        }
        return idx;
}
void update(int idx, int p, bool add) {
        int d = P[idx].dist[p];
        auto& set = P[idx].smallest;
        if (add)set.insert(d);
        else set.erase(set.find(d));
        if (idx == p)return;
        for (int to : CG[idx]) {
            if (P[to].V.count(p)) {
                update(to, p, add);
                return;
            }
        }
}
int getClosest(int idx, int p) {
        int ret = 0x7fffffff / 2;
        if (!P[idx].smallest.empty())ret = std::min(ret, *P[idx].smallest.begin() + P[idx].dist[p]);

        if (idx == p)return ret;
        for (int to : CG[idx]) {
            if (P[to].V.count(p)) {
                ret = std::min(ret, getClosest(to, p));
                break;
            }
        }
        return ret;
}
```

Geometry

Basic

```
using data_Ty = long double;
const data_Ty INF = LLONG_MAX;
const double PI = 2.0*acos(0.0);
const double EPSILON = 1e-9;
struct point {
        data_Ty x, y;
        bool operator==(point p) { return x == p.x && y == p.y; }
        point operator*(double p)const { return{ data_Ty(x*p), data_Ty(y*p) }; }
        point operator+(const point& p)const { return{ x + p.x, y + p.y }; }
        point operator-(const point& p)const { return{ x - p.x, y - p.y }; }
        bool operator<(const point& p)const { return x < p.x || (x == p.x && y < p.y); }
        double norm()const { return hypot(x, y); }
        point normalize()const { return{ data_Ty(x / norm()), data_Ty(y / norm()) }; }
        double polar()const {//radian
                return fmod(atan2(y, x) + 2 * PI, 2 * PI); }
        double dot(const point& p)const { return x*p.x + y*p.y; }
        double cross(const point& p)const { return x*p.y - p.x*y; }
        point project(const point& p)const {
                point r = p.normalize();
                return r*r.dot(*this);
        }
};
data_Ty abss(data_Ty a) {
        if (a < 0)return -a;
        return a;
}
data_Ty ccw(data_Ty ax, data_Ty ay, data_Ty bx, data_Ty by, data_Ty cx, data_Ty cy) {
        return bx*cy - ay*bx - ax*cy - by*cx + ax*by + ay*cx;
}
data_Ty ccw(point a, point b, point c) {
        return ccw(a.x, a.y, b.x, b.y, c.x, c.y);
}
struct line {
        data_Ty a, b, c;//ax+by+c=0
        line() {}
        line(point A, point B) {
                a = B.y - A.y;
                b = -(B.x - A.x);
                c = A.y*(B.x - A.x) - A.x*(B.y - A.y);
                /*    data_Ty g = std::abs(gcd(gcd(a, b), c));
                a /= g; b /= g; c /= g;*/
```

```
	}
	line(data_Ty m_u, data_Ty m_d, data_Ty B) { // y=(m_u/m_d)x+B
		a = m_u;
		b = -m_d;
		c = B;
		/*	data_Ty g = std::abs(gcd(gcd(a, b), c));
		a /= g; b /= g; c /= g;*/
	}
	line(data_Ty m_u, data_Ty m_d, point p) { // (y-p.y)=(m_u/m_d)*(x-p.x)
		a = m_u;
		b = -m_d;
		c = -(a*p.x + b*p.y);
	}
	data_Ty getX(data_Ty y) { return (-b*y - c) / a; }
	data_Ty getY(data_Ty x) { return (-a*x - c) / b; }
	double DistToPoint(point p) { // Distance
		return std::abs(a*p.x + b*p.y + c) / sqrt((double)(a*a + b*b));
	}
	bool getLineIntersectionPoint(line p, point& q) {// return : 교차 여부 ,point p: 교차점
		point a, b, c, d;
		if (this->b == 0) {
			a = { getX(0), 0 };
			b = { getX(1), 1 };
		}
		else {
			a = { 0, getY(0) };
			b = { 1, getY(1) };
		}
		if (p.b == 0) {
			c = { p.getX(0), 0 };
			d = { p.getX(1), 1 };
		}
		else {
			c = { 0, p.getY(0) };
			d = { 1, p.getY(1) };
		}
		double det = (b - a).cross(d - c);
		if (fabs(det) < EPSILON)return false;
		q = a + (b - a)*((c - a).cross(d - c) / det);
		return true;
	}
};
struct segment {
	point a, b;
	data_Ty diffX() {
		return abss(b.x - a.x);
	}
	data_Ty diffY() {
		return abss(b.y - a.y);
	}
	long long sign(long long a) { // inner method
		if (a > 0) return 1;
		if (a < 0)return -1;
		return 0;
	}
	bool intersect(segment p) {
		long long a1 = ccw(a, b, p.a);
		long long a2 = ccw(a, b, p.b);
		long long b1 = ccw(p.a, p.b, a);
		long long b2 = ccw(p.a, p.b, b);
		bool A = sign(a1)*sign(a2) < 0;
		bool B = sign(b1)*sign(b2) < 0;
		return A&&B;
	}
	bool inner(point p) {
		long long A = ccw(a, b, p);
		bool X = false;
		bool Y = false;
		if (std::min(a.x, b.x) <= p.x && p.x <= std::max(a.x, b.x)) {
			X = true;
		}
		if (std::min(a.y, b.y) <= p.y && p.y <= std::max(a.y, b.y)) {
			Y = true;
		}
		return A == 0 && X&&Y;
	}
	bool endpoint(point p) {
		return a == p || b == p;
	}
	bool overlap(segment p) {
		long long a1 = ccw(a, b, p.a);
		long long a2 = ccw(a, b, p.b);
		long long b1 = ccw(p.a, p.b, a);
		long long b2 = ccw(p.a, p.b, b);
		if (a1 == 0 && a2 == 0 && b1 == 0 && b2 == 0) {
			if ((inner(p.a) && inner(p.b)) || (p.inner(a) && p.inner(b))) {
				return true;
			}
			if (a == p.a && (sign(b.x - a.x) != sign(p.b.x - p.a.x) || sign(b.y - a.y) !=
sign(p.b.y - p.a.y))) {
				return false;
			}
```

```
                if (a == p.b && (sign(b.x - a.x) != sign(p.a.x - p.b.x) || sign(b.y - a.y) !=
sign(p.a.y - p.b.y))) {
                        return false;
                }
                if (b == p.a && (sign(a.x - b.x) != sign(p.b.x - p.a.x) || sign(a.y - b.y) !=
sign(p.b.y - p.a.y))) {
                        return false;
                }
                if (b == p.b && (sign(a.x - b.x) != sign(p.a.x - p.b.x) || sign(a.y - b.y) !=
sign(p.a.y - p.b.y))) {
                        return false;
                }
                if (inner(p.a) || inner(p.b) || p.inner(a) || p.inner(b)) {
                        return true;
                }
        }
        return false;
    }
    long double distToPoint(point p, point& q) {
        line A(a, b);
        line B(-A.b, -A.a, p);
        A.getLineIntersectionPoint(B, q);
        if (inner(q)) {
            return (p - q).norm();
        }
        return std::min((p - a).norm(), (p - b).norm());
    }
};
struct polygon : vector<point> {
    data_Ty ccw(int a, int b, int c) {
        return ::ccw(at(a), at(b), at(c));
    }
    data_Ty ccw(point a, point b, point c) {
        return ::ccw(a, b, c);
    }
    bool isInsider(point p) {
        segment q = { p,{ p.x + 1000000007, p.y + 1000000009 } };
        int sz = size();
        int cnt = 0;
        for (int i = 0; i < sz; i++) {
            segment k = { at(i), at((i + 1) % sz) };
            if (q.intersect(k)) {
                cnt++;
            }
            if (k.inner(p)) { // 선분위에 점 있는 경우
                return true;
            }
        }
        return cnt % 2 == 1;
    }
    bool isInsider(polygon& p) {
        int size = p.size();
        for (int i = 0; i < size; i++) {
            if (!isInsider(p[i])) {
                return false;
            }
        }
        return true;
    }
    long double getArea() {
        int sz = size();
        long double ret = 0;
        for (int i = 0; i < sz; i++) {
            ret += ccw(0, i, (i + 1) % sz);
        }
        return fabs(ret / 2);
    }
    polygon getConvexHull() {
        polygon P = *this;
        if (P.size() <= 2) {
            return P;
        }
        std::sort(P.begin(), P.end());
        polygon Convex;
        for (int _ : {0, 0}) {
            polygon half;
            for (point& p : P) {
                while (half.size() >= 2) {
                    point A = *(half.end() - 2);
                    point B = *(half.end() - 1);
                    if (ccw(A, B, p) <= 0) {
                        half.pop_back();
                        continue;
                    }
                    break;
                }
                half.push_back(p);
            }
            half.pop_back();
            Convex.insert(Convex.end(), half.begin(), half.end());
            std::reverse(P.begin(), P.end());
        }
```

```
        return Convex;
    }
    polygon cutPolygontoSegment(const segment& p) {
        int sz = size();
        vector<bool> inside;
        auto next = [&](int i) { return (i + 1) % sz; };
        for (auto& q : *this) inside.push_back(ccw(p.a, p.b, q) >= 0);
        polygon ret;
        for (int i = 0; i < sz; i++) {
            int j = next(i);
            if (inside[i]) ret.push_back(at(i));
            if (inside[i] != inside[j]) {
                line a(at(i), at(j));
                line b(p.a, p.b);
                point q;
                assert(a.getLineIntersectionPoint(b, q));
                ret.push_back(q);
            }
        }
        return ret;
    }
    polygon cutPolygontoConvexhull(polygon& p) {
        int sz = p.size();
        if (isInsider(p)) {
            return p;
        }
        if (p.isInsider(*this)) {
            return *this;
        }
        polygon ret = *this;
        auto next = [&](int i) {return (i + 1) % sz; };
        for (int i = 0; i < sz; i++) {
            ret = ret.cutPolygontoSegment({ p[i], p[next(i)] });
        }
        return ret;
    }
    data_Ty getDistToFarthest() {
        polygon Convex = getConvexHull();
        int idx = 1;
        data_Ty dist = -1;
        int n = Convex.size();
        for (int i = 0; i < n; i++) {
            while (true) {
                point vec1 = Convex[(i + 1) % n] - Convex[i];
                point vec2 = Convex[(idx + 1) % n] - Convex[idx];
                if (ccw({ 0,0 }, vec1, vec2) > 0) {
                    idx = (idx + 1) % n;
                    continue;
                }
                break;
            }
            dist = std::max<data_Ty>(dist, (Convex[i]- Convex[idx]).norm());
        }
        return dist;
    }
    bool isPolygonOverlap(polygon poly) {
        for (point &p : *this) {
            if (poly.isInsider(p)) {
                return true;
            }
        }
        for (point &p : poly) {
            if (isInsider(p)) {
                return true;
            }
        }
        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < poly.size(); j++) {
                segment p = { at(i),at((i + 1) % size()) };
                segment q = { poly[j],poly[(j + 1) % poly.size()] };
                if (p.intersect(q)) {
                    return true;
                }
            }
        }
        return false;
    }
};
```

Sort by angle

```
sort(b, b+n-1, [&](const pnt &p, const pnt &q){
    auto hypot = [&](pnt a){
        return 1ll * a.x * a.x + 1ll * a.y * a.y;
    };
    if(p.x == q.x && p.y == q.y) return false;
    if(p.y == 0 && p.x > 0 && q.y == 0 && q.x > 0) return hypot(p) < hypot(q);
    if(p.y == 0 && p.x > 0) return true;
    if(q.y == 0 && q.x > 0) return false;
    if(p.y == 0 && q.y == 0) return hypot(p) < hypot(q);
```

```
    if(1ll * p.y * q.y <= 0) return p.y > q.y;
    lint tmp = ccw(p, q);
    if(tmp == 0) return hypot(p) < hypot(q);
    return tmp > 0;
});
```

Data Structure

Segment Tree Range Query And Range Update

```
#include<bits/stdc++.h>
using std::vector;
using lint = long long;
struct segment_tree {
	struct T {
		lint pro, sum;
	};
	vector<T> tr;
	int S = 1;
	segment_tree(int sz) {
		while (S < sz)S *= 2;
		tr.resize(S * 2 + 10);
	}
	void propagate(lint pos,lint len) {
		if (pos >= S) {
			tr[pos].pro = 0;
			return;
		}
		lint val = tr[pos].pro;
		tr[pos * 2].sum += val*(len/2);
		tr[pos * 2].pro += val;
		tr[pos * 2 + 1].sum += val*(len/2);
		tr[pos * 2 + 1].pro += val;
		tr[pos].pro = 0;
	}
	void update(lint pos, lint l, lint r, lint ql, lint qr, lint val) {
		lint m = (l + r + 1) / 2;
		if (r < ql || qr < l) return;
		propagate(pos, (r - l + 1));
		if (ql <= l && r <= qr) {
			tr[pos].sum += val * (r - l + 1);
			tr[pos].pro += val;
			return;
		}
		update(pos * 2, l, m - 1, ql, qr, val);
		update(pos * 2 + 1, m, r, ql, qr, val);
		tr[pos].sum = tr[pos * 2].sum + tr[pos * 2 + 1].sum;
	}
	lint range(lint pos, lint l, lint r, lint ql, lint qr) {
		lint m = (l + r + 1) / 2;
		if (r < ql || qr < l) return 0;
		propagate(pos, (r - l + 1));
		if (ql <= l && r <= qr) {
			return tr[pos].sum;
		}
		return range(pos * 2, l, m - 1, ql, qr) + range(pos * 2 + 1, m, r, ql, qr);
	}
	void update(int ql, int qr, lint val) {
		update(1, 0, S - 1, ql, qr, val);
	}
	lint range(lint ql, lint qr) {
		return range(1, 0, S - 1, ql, qr);
	}
};
```

Persistent Segment Tree

```
struct PersistentSegmentTree {
	struct node {
		int sum = 0;
		int L = -1, R = -1;
	};
	vector<node> tree;
	vector<int> root;
	int n;
	PersistentSegmentTree(int n) :n(n) {
		root.push_back(init(0, n - 1));
	}
	int init(int s, int e) {
		node p;
		if (s != e) {
			int m = (s + e) / 2;
			p.L = init(s, m);
			p.R = init(m + 1, e);
		}
		tree.push_back(p);
		return tree.size() - 1;
	}
	int add(int idx, int par, int a) {
		root.push_back(add(tree[root[par]], 0, n - 1, idx, a));
		return root.size() - 1;
	}
```

```
    int add(node p, int s, int e, int idx, int a) {
        int m = (s + e) / 2;
        node q = p;
        q.sum = a;
        if (s != e) {
            if (s <= idx && idx <= m) {
                q.L = add(tree[p.L], s, m, idx, a);
            }
            else {
                q.R = add(tree[p.R], m + 1, e, idx, a);
            }
            q.sum = tree[q.L].sum + tree[q.R].sum;
        }
        tree.push_back(q);
        return tree.size() - 1;
    }
    int getRange(int s, int e, int l, int r, int aidx, int bidx) {
        int m = (s + e) / 2;
        int ret = 0;
        if (r<s || e<l) {
            return 0;
        }
        if (l <= s && e <= r) {
            ret += tree[bidx].sum - tree[aidx].sum;
            return ret;
        }
        ret = getRange(s, m, l, r, tree[aidx].L, tree[bidx].L) + getRange(m + 1, e, l, r,
tree[aidx].R, tree[bidx].R);
        return ret;

    }
    int getRange(int l, int r, int a, int b) {
        if (a > b || l > r)return 0;
        return getRange(0, n - 1, l, r, root[a], root[b]);
    }
};
```

Removable Heap

```
#include <cstdint>
#include <cstdlib>

using namespace std;

/* max_heap */
template<typename Ty, typename Pr = less<Ty>>
struct removable_heap
{
    typedef int id_t;
    typedef ptrdiff_t pos_t;

    Pr comparator;
    vector<pair<Ty, id_t>> heap;
    id_t lastid;
    unordered_map<id_t, pos_t> pos;
    removable_heap():lastid(0){}
    /* returns id */
    id_t push(const Ty &val)
    {
        id_t id = lastid++;
        pos_t cur = heap.size();
        heap.emplace_back(move(val), id);
        while(cur > 0) {
            pos_t par = (cur-1)>>1;
            /* satisfies heap */
            if (!comparator(heap[par].first, heap[cur].first))
                break;
            swap(heap[par], heap[cur]);
            pos[heap[cur].second] = cur;
            cur = par;
        }
        pos[id] = cur;
        return id;
    }
    bool empty() { return heap.empty(); }
    Ty &top() { return heap.front().first; }
    void adjust(pos_t hole)
    {
        for(;;)
        {
            pos_t par = (hole-1)>>1;
            pos_t left = hole*2 + 1;
            pos_t right = hole*2 + 2;
            pos_t swpind = hole;
            if (hole > 0 && comparator(heap[par].first, heap[hole].first)){
                swpind = par;
            } else if (right < heap.size()) {
                if (comparator(heap[left].first, heap[right].first)) {
                    if (comparator(heap[hole].first, heap[right].first)) {
                        swpind = right;
                    }
                } else {
```

```
                    if (comparator(heap[hole].first, heap[left].first)) {
                        swpind = left;
                    }
                }
            } else if (left < heap.size()) {
                if (comparator(heap[hole].first, heap[left].first)) {
                    swpind = left;
                }
            }
            if (swpind == hole) break;
            swap(heap[swpind], heap[hole]);
            pos[heap[hole].second] = hole;
            hole = swpind;
        }
        pos[heap[hole].second] = hole;
    }
    void pop()
    {
        if (empty()) throw runtime_error("pop empty");
        if (heap.size() == 1) {
            heap.clear();
            pos.clear();
            return;
        }
        pos.erase(heap[0].second);
        swap(heap[0], heap.back());
        heap.pop_back();
        adjust(0);
    }
    bool remove(id_t id)
    {
        if (pos.count(id) == 0) return false;
        auto cur = pos[id];
        pos.erase(id);
        if (cur + 1 == heap.size()) {
            heap.pop_back();
            return true;
        }
        swap(heap[cur], heap.back());
        heap.pop_back();
        adjust(cur);
    }
    Ty find(id_t id)
    {
        if (pos.count(id) == 0) return Ty();
        return heap[pos[id]];
    }
    void modify(id_t id, const Ty &newValue)
    {
        if (pos.count(id) == 0) throw runtime_error("invalid id");
        auto cur = pos[id];
        heap[cur].first = move(newValue);
        adjust(cur);
    }
};
```

Rope

```
#include<ext/rope>
using __gnu_cxx::crope;
char tmp[100];
int main(){
    crope A,B;
    //std::cin>>A;
    //scanf("%s",tmp);
    //B=tmp;
    A="aaaa";
    A=crope("BBB")+A+"CCC";// const char[] + crope is error,but crope + const char[] is not.
so A="BBB"+A+"CCC" is error.
    B="BBBaaaaCCC";
    if(A.compare(B)==0){
        printf("%s %s is same₩n",A.c_str(),B.c_str());
    }
    if(A==B){// you can use this.
        std::cout << A << ' ' << B << " is same"<<std::endl;
    }

}
```

Red Black Tree

```
#include <cassert>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, std::less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

template <typename key,typename value>
using ordered_map =
tree<key,value,std::less<key>,rb_tree_tag,tree_order_statistics_node_update>;
int main()
```

```
{
	// Insert some entries into s.
	ordered_set<int> s;
	ordered_map<int,int> map;
	map[1]=1;
	printf("%d₩n",map[1]); // 1
	printf("%d₩n",map[0]); // default value : 0
	printf("%lu₩n",map.order_of_key(2)); // map has 0, 1, so print
	s.insert(12); s.insert(505); s.insert(30); s.insert(1000); s.insert(10000);
	s.insert(100);
	assert(*s.find_by_order(0) == 12);
	assert(s.order_of_key(10) == 0);
	// Erase an entry.
	s.erase(30);
	return 0;
}
```

String

KMP

```
//kmp
int pi[1000003];
void kmp(string O, string F) {

	for (int i = 1, j = 0; i < F.size(); i++) {
		while (j && F[i] != F[j]) j = pi[j - 1];
		if (F[i] == F[j]) j++;
		pi[i] = j;
	}

	for (int i = 0, j = 0; i < O.size(); i++) {
		while (j && O[i] != F[j]) j = pi[j - 1];
		if (O[i] == F[j]) j++;
		if (j == F.size()) { // 하나 찾음
			//시작 위치 i - j + 1
			j = pi[j - 1];
		}
	}
}
```

Aho-Corasick

```
using std::string;
const int N = 4;
char alpha[256];
struct aho {
	struct node {
		int vaild = 0;
		int ch;
		int next[N] = { -1,-1,-1,-1 };
		int parent = -1;
		node() {}
		inline int& operator[](int idx) {
			return next[idx];
		}
	};
	vector<node> tree;
	vector<int> idx;
	int cnt = 0;
	aho() {
		tree.push_back(node());
		cnt++;
	}
	void add_string(string str) {
		int p = 0;
		for (char ch : str) {
			if (tree[p][alpha[ch]] == -1) {
				tree.push_back(node());
				tree.back().parent = p;
				tree.back().ch = alpha[ch];
				tree[p][alpha[ch]] = cnt;
				cnt++;
			}
			p = tree[p][alpha[ch]];
		}
		tree[p].vaild++;
	}
	void preprocessing() {
		idx.resize(cnt, -1);
		std::queue<int> que;
		que.push(0);
		while (!que.empty()) {
			int id = que.front();
			que.pop();

			if (id == 0 || tree[id].parent == 0) {
				idx[id] = 0;
			}
			else {
				int p = idx[tree[id].parent];
```

```
                int c = tree[id].ch;
                while (p != 0 && tree[p][c] == -1) {
                    p = idx[p];
                }
                if (tree[p][c] != -1) {
                    p = tree[p][c];
                }
                idx[id] = p;
                tree[id].vaild += tree[p].vaild;
            }
            for (int i = 0; i < N; i++) {
                if (tree[id].next[i] != -1) {
                    que.push(tree[id][i]);
                }
            }
        }
    }
    int find_str(string str) {
        int p = 0;
        int ret = 0;
        for (char ch : str) {
            while (p != 0 && tree[p][alpha[ch]] == -1) {
                p = idx[p];
            }
            if (tree[p][alpha[ch]] != -1) {
                p = tree[p][alpha[ch]];
            }
            if (tree[p].vaild) {
                ret += tree[p].vaild;
            }
        }
        return ret;
    }
};
```

Suffix Array O(N log N) && LCP(kasai algorithm)

```
using std::string;
struct suffixArray {
    string str;
    vector<int> group;
    vector<int> sa;
    vector<int> tmp;
    vector<int> cnt;
    vector<int> cnt2;
    int len;
    suffixArray(string str) :str(str) {
        len = str.size();
        group.resize(len * 2+10, -1);
        sa.resize(len,0);
        tmp.resize(len,0);
        cnt.resize(len, 0);
        cnt2.resize(len, 0);
    }
    vector<int> getSA() {
        int len = str.size();
        int m = 255;
        for (int i = 0; i < len; i++) {
            sa[i] = i;
            group[i] = str[i];
        }

        group[len] = -1;
        for (int k = 1; k < len; k *= 2) {
            for (int i = 0; i <= m; i++) { cnt[i] = 0; cnt2[i] = 0; }
            for (int i = 0; i < len; i++) { cnt[group[sa[i] + k] + 1]++; }
            for (int i = 1; i <= m; i++) { cnt[i] += cnt[i - 1]; }
            for (int i = len - 1; i >= 0; i--) { tmp[--cnt[group[sa[i] + k] + 1]] = sa[i]; }
            for (int i = 0; i < len; i++) { cnt2[group[tmp[i]]]++; }
            for (int i = 1; i <= m; i++) { cnt2[i] += cnt2[i - 1]; }
            for (int i = len - 1; i >= 0; i--) { sa[--cnt2[group[tmp[i]]]] = tmp[i]; }
            tmp[sa[0]] = 0;
            tmp[len] = -1;
            for (int j = 1; j < len; j++) {
                if (group[sa[j - 1]] == group[sa[j]] && group[sa[j - 1] + k] == group[sa[j]
+ k]) {
                    tmp[sa[j]] = tmp[sa[j - 1]];
                }
                else {
                    tmp[sa[j]] = tmp[sa[j - 1]] + 1;
                }
            }
            int q = 0;
            for (int j = 0; j < len; j++) {
                group[j] = tmp[j];
                q = std::max(q, tmp[j]);
            }
            m = q + 1;
        }
        return sa;
    }
    vector<int> getLCP() {
        vector<int> pi(len, -1);
```

```
        vector<int> plcp(len);
        vector<int> lcp(len, -1);
        for (int i = 1; i < len; i++) { pi[sa[i]] = sa[i - 1]; }
        int k = 0;
        for (int i = 0; i < len; i++) {
            if (pi[i] == -1) {
                plcp[i] = 0;
                continue;
            }
            while (str[i + k] == str[pi[i] + k])k++;
            plcp[i] = k;
            k = std::max(k - 1, 0);
        }
        for (int i = 0; i < len; i++) {lcp[i] = plcp[sa[i]]; }
        return lcp;
    }
};
```

Manacher's algorithm (find palindrom)

```
char tmp[100010];
char str[200010];
int A[200010];// palindrome length
int main() {
    scanf("%s", tmp);
    int n = strlen(tmp);
    for (int i = 0; i<n; i++) { // for even palindrome
        str[i * 2] = '#';
        str[i * 2 + 1] = tmp[i];
    }
    str[n * 2] = '#';
    int r = -1, p = -1, ans = 0;
    for (int i = 0; i <= n * 2; i++) {
        if (r < i)A[i] = 0;
        else A[i] = std::min(A[2 * p - i], r - i);
        while (i - A[i] - 1 >= 0 && i + A[i] + 1 <= n * 2 && str[i - A[i] - 1] == str[i + A[i]
+ 1])A[i]++;
        if (i + A[i] > r) {
            r = i + A[i];
            p = i;
        }
    }
}
```

Math

Combination

```
#define MOD 1000000007
#define SIZE 4000000
int f[SIZE +1];
int inv[SIZE+1 ];
void make_com(){
    inv[0] = inv[1] = f[0] = f[1] = 1;
    for (int i = 2; i <=SIZE ; i++)f[i] = 1LL*f[i - 1] * i % MOD;
    for (int i = 2; i <= SIZE ; i++)inv[i] = -1LL * (MOD / i) * inv[MOD % i] % MOD;
    for (int i = 2; i <= SIZE ; i++)inv[i] = 1LL * inv[i - 1] * ((inv[i] + MOD) % MOD) % MOD;
}
int C(int n, int r){return (long long)f[n] * inv[r] % MOD * inv[n - r] % MOD;}
```

Extend Euclidean Algorithm

```
lint exdgcd(lint a, lint b, lint& x, lint& y) {
    lint d = a;
    if (b != 0) {
        d = exdgcd(b, a%b, y, x);
        y -= (a / b) * x;
    }
    else {
        x = 1; y = 0;
    }
    return d;
}
```

Eular Phi O(nlogn)

$$\varphi(n) = n\prod_{p|n}(1-\frac{1}{p})$$

```
const int N=10000;
long long phi[N+1] = { 1,1 };
long long up[N+1];
long long down[N+1];
bool chk[N+1] = { 1,1 };
int main() {
    for (int i = 0; i <= N; i++) {
        up[i] = i;
        down[i] = 1;
    }
```

```
	for (int i = 2; i <= N; i++) {
		if (!chk[i]) {
			up[i] *= i - 1;
			down[i] *= i;
			for (int j = i * 2; j <= N; j += i) {
				chk[j] = true;
				up[j] *= i - 1;
				down[j] *= i;
			}
		}
	}
	for (int i = 2; i <= N; i++) {
		phi[i] = up[i] / down[i];
	}

}
```

Eular Phi O(sqrt(n))

```
lint phi(lint n) {
	lint ret = n;
	for (lint i = 2; i * i <= n; i++) {
		if (n % i == 0) {
			while (n % i == 0) {
				n /= i;
			}
			ret -= (ret / i);
		}
	}
	if (n > 1) {
		ret -= ret / n;
	}
	return ret;
}
```

Miller Rabin Primality Test O(log^3)

```
ll modmul(ll a, ll b, ll m) {
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b & 1) r = (r + v) % m;
        b >>= 1;
        v = (v << 1) % m;
    }
    return r;
}
ll modpow(ll n, ll k, ll m) {
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = modmul(ret, n, m);
        n = modmul(n, n, m);
        k /= 2;
    }
    return ret;
}
bool test_witness(ull a, ull n, ull s) {
    if (a >= n) a %= n;
    if (a <= 1) return true;
    ull d = n >> s;
    ull x = modpow(a, d, n);
    if (x == 1 || x == n - 1) return true;
    while (s-- > 1) {
        x = modmul(x, x, n);
        if (x == 1) return false;
        if (x == n - 1) return true;
    }
    return false;
}
bool isprime(unsigned long long n) {
    if (n == 2 || n == 1) return true;
    if (n < 2 || n % 2 == 0) return false;
    ull d = n >> 1, s = 1;
    for (; (d & 1) == 0; s++) d >>= 1;
#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178) && T(450775) && T(9780504) &&
T(1795265022);
#undef T
}
```

교란 순열

```
d[n] = (n - 1) (d[n - 1] + d[n - 2]);
d[n] - n * d[n-1] = (-1) ^ n
```

뫼비우스 함수

```
lint dy[50003], chk[50003];
int main() {
	lint tc; scanf("%lld", &tc);
	dy[1] = 1;
	for (lint i = 1; i <= 50000; i++)
		chk[i] = 1;
```

```
	for (lint i = 2; i <= 50000; i++) {
		dy[i] = -chk[i];
		for (lint j = i; j <= 50000; j += i)
			chk[j] += dy[i];
		dy[i] += dy[i - 1];
	}

	while (tc--) {
		lint a, b, d; scanf("%lld %lld %lld", &a, &b, &d);

		lint ans = 0;
		a /= d, b /= d;
		d = 1;
		while (d <= a & d <= b) {
			lint ax = a / d;
			lint bx = b / d;
			lint idx = min((a / ax) + 1, (b / bx) + 1);

			ans += ax * bx * (dy[idx - 1] - dy[d - 1]);

			d = idx;
		}
		printf("%lld₩n", ans);
	}
}
```

카탈란 수

n쌍의 괄호로 만들 수 있는 올바른 괄호 구조의 개수
n + 2각형을 n개의 삼각형으로 나누는 방법의 수이다.
Cn은 n + 1개의 항에 괄호를 씌우는 모든 경우의 수이다.혹은 n + 1개의 항에 이항연산자를 적용하는 순서의 모든 가지수로도 볼 수 있다.
Cn은 n + 1개의 단말 노드를 갖는 이진 순서 트리의 개수
A[k] == 1 or A[k] == -1 일 때, A[1] + A[2] + .. + A[2n] = 0 일 때, 각각 부분합 A[1], A[1] + A[2], .., A[1] + A[2] + .. + A[n]이 모두 0 이상 되도록 하는 방법의 수
원탁에 있는 사람들이 엇갈리지 않고 악수하는 경우의 수

```
#include <stdio.h>
long long dp[10004];
int main()
{
	long long n;
	scanf("%lld", &n);
	n /= 2;
	dp[0] = 1; dp[1] = 1;
	for (int i = 2; i <= n; i++) {
		for (int j = 0; j < i; j++) {
			dp[i] += dp[j] * dp[i - j - 1];
			dp[i] %= 987654321;
		}
	}
	printf("%lld", dp[n]);
}
```

Chinese Remainder Theorem

x = a (mod n)가 되는 x를 찾는다.

Dependencies : gcd(a,b), modinverse(a,m)

```
long long chinese_remainder(long long *a, long long *n, int size) {
    if (size == 1) return *a;
    long long tmp = modinverse(n[0], n[1]);
    long long tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    long long ora = a[1];
    long long tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    long long ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

FFT

```
#include<complex>
using base = std::complex<double>;

void fft(vector<base>& a, bool inv) {
	int n = a.size();
	for (int i = 1, j = 0; i < n; i++) {
		int bit = (n >> 1);
		while (j >= bit) {
			j ^= bit; bit >>= 1;
		}
		j |= bit;
		if (i < j)std::swap(a[i], a[j]);
	}

	double ang = 2.0*acos(-1) / n * (inv ? -1.0 : 1.0);
	vector<base> W(n);
	for (int i = 0; i < n; i++)W[i] = base(cos(ang*i), sin(ang*i));
	for (int k = 2; k <= n; k <<= 1) {
		int step = n / k;
```

```
            for (int i = 0; i < n; i += k) {
                for (int j = 0; j < k / 2; j++) {
                    base u = a[i + j], v = a[i + j + k / 2];
                    a[i + j] = u + W[step*j] * v;
                    a[i + j + k / 2] = u - W[step*j] * v;
                }
            }
        }
        if (inv)for (int i = 0; i < n; i++)a[i] /= n;
}
vector<lint> multiply(vector<lint>& A, vector<lint>& B, lint mod) {
        int n = 1;
        while (n < std::max(A.size(), B.size()))n <<= 1;
        n <<= 1;
        vector<base> b1(n, 0), b2(n, 0), s1(n, 0), s2(n, 0);
        for (int i = 0; i < A.size(); i++) {
            b1[i] = A[i] >> 15;
            s1[i] = A[i] & ((1 << 15) - 1);
        }
        for (int i = 0; i < B.size(); i++) {
            b2[i] = B[i] >> 15;
            s2[i] = B[i] & ((1 << 15) - 1);
        }
        fft(b1, false);
        fft(s1, false);
        fft(b2, false);
        fft(s2, false);
        vector<base> b1b2(n, 0), b1s2(n, 0), s1b2(n, 0), s1s2(n, 0);
        for (int i = 0; i < n; i++) {
            b1b2[i] = b1[i] * b2[i];
            b1s2[i] = b1[i] * s2[i];
            s1b2[i] = s1[i] * b2[i];
            s1s2[i] = s1[i] * s2[i];
        }
        fft(b1b2, true);
        fft(b1s2, true);
        fft(s1b2, true);
        fft(s1s2, true);
        vector<lint> ret(n);
        for (int i = 0; i < n; i++) {
            lint ac = b1b2[i].real() + 0.5;
            lint ad = b1s2[i].real() + 0.5;
            lint bc = s1b2[i].real() + 0.5;
            lint bd = s1s2[i].real() + 0.5;
            ac %= mod; ad %= mod; bc %= mod; bd %= mod;
            ret[i] = (ac << 30) + ((ad + bc) << 15) + bd;
            ret[i] %= mod;
            ret[i] += mod;
            ret[i] %= mod;
        }
        return ret;
}
```

카라추바 & 스트라센

카라츠바(다항식 곱셈)O(n^log2(3))=O(n^1.58496)
x = x1Bm + x0
y = y1Bm + y0
(단, x0과 y0는 Bm보다 작다.)
z2 = x1y1
z1 = x1y0 + x0y1
z0 = x0y0
라고 할 때, x와 y의 곱은
xy = (x1Bm + x0)(y1Bm + y0) = z2 B2m + z1 Bm + z0
이 식은 4번의 곱셈을 해야한다. 카라추바는 덧셈을 몇 번 함으로써, xy를 3번의 곱셈을 통해 구할 수 있다는 걸 알았다.
z2 = x1y1 라 하자.
z0 = x0y0 라 하자.
z1 = (x1y1 + x1y0 + x0y1 + x0y0) - x1y1 - x0y0 = x1y0 + x0y1
이므로
z1 = (x1 + x0)(y1 + y0) − z2 − z0

슈트라센 알고리즘(행렬곱셈) O(n^2.807)
C[1][1]=A[1][1]B[1][1]+A[1][2]B[2][1]
C[1][2]=A[1][1]B[1][2]+A[1][2]B[2][2]
C[2][1]=A[2][1]B[1][1]+A[2][2]B[2][1]
C[2][2]=A[2][1]B[1][2]+A[2][2]B[2][2]

M1=(A[1][1]+A[2][2])*(B[1][1]+B[2][2])
M2=(A[2][1]+A[2][2])*B[1][1]
M3=A[1][1]*(B[1][2]-B[2][2])
M4=A[2][2]*(B[2][1]-B[1][1])
M5=(A[1][1]+A[1][2])*B[2][2]
M6=(A[2][1]-A[1][1])*(B[1][1]+B[1][2])
M7=(A[1][2]-A[2][2])*(B[2][1]+B[2][2])

C[1][1]=M1+M4-M5+M7
C[1][2]=M3+M5
C[2][1]=M2+M4
C[2][2]=M1-M2+M3+M6

C= (C[1][1],C[1][2])
   (C[2][1],C[2][2])

Dynamic Programming

Convex Hul Trick

$$dp[i] = min_{j<i}\{dp[j] + b[j] \star a[i]\}$$

조건 : $b[j] \geq b[j+1]$

~~optionally~~ $a[i] \leq a[i+1]$

$O(n^2)$-> $O(n)$

$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$

조건 : $b[k] \geq b[k+1]$

~~optionally~~ $a[j] \leq a[j+1]$

$O(kn^2)$-> $O(kn)$ // there isn't code

```
struct line {
	lint A, B;
};
lint dy[50010];
bool cmp(line& A, line& B, line& C) {
	lint u1 = (B.B - A.B);
	lint d1 = (A.A - B.A);

	lint u2 = (C.B - B.B);
	lint d2 = (B.A - C.A);

	return u1*d2 > u2*d1;
}
lint getVal(line& p, lint x) {
	return p.A*x + p.B;
}
int main(void) {
	memset(dy, 0x3f, sizeof(dy));
	dy[0] = 0;
	vector<pll> ract;
	/*  some of data increase gradient at ract  */
	vector<line> stack;
	vector<double> points;
	int n = ract.size() - 1;
	int idx2 = 0;
	for (int i = 1; i <= n; i++) {
		line p = { ract[i].second,dy[i - 1] };
		while (stack.size() >= 2 && cmp(*(stack.end() - 2), *(stack.end() - 1), p)) {
			stack.pop_back();
			if (!points.empty())points.pop_back();
		}
		if (!stack.empty()) {
			points.push_back((double)(p.B - stack.back().B) / (stack.back().A - p.A));
		}
		stack.push_back(p);
/*		idx2 = std::min<int>(idx2, stack.size() - 1);*/
		lint w = ract[i].first;

		int idx = std::lower_bound(points.begin(), points.end(), (double)w) - points.begin();
		dy[i] = std::min(dy[i], getVal(stack[idx], w));
		/* while (idx != stack.size() - 1 && getVal(stack[idx], w) > getVal(stack[idx + 1], w))
{
		idx++;
		}
		dy[i] = std::min(dy[i], getVal(stack[idx], w));*/
	}
	printf("%lld", dy[n]);
	return 0;

}
```

Divide and Conquer Optimiazation

$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$

조건 : **quadrangle inequality**

$$C[a][c] + C[b][d] \leq C[a][d] + C[b][c],\ a \leq b \leq c \leq d$$

p<q에 대해 W(a,d) + W(b, c) > W(a, c) + W(b, d)이면 C(i, p)≤C(i, q)이고, W(a,d) + W(b, c) < W(a, c) + W(b, d)이면 C(i, p)≥C(i, q)이다.

여기서 주목해야 될 사실은 a<b<c<d 일 때 w(a,d)+w(b,c) > w(a,c)+w(b,d) 라는 부등식이다.
이걸 w 가 볼록(convex)하다고 하고 부등호가 반대면 오목(concave)하다고 하는데 w 가 볼록하거나 오목할 때는 재미있는 사실이 있다.
dp[i-1][k]+w(k,j) (k<j) 들 중에서 가장 작아서 dp[i][j]의 값을 만든 k 를 opt[i][j]라고 하자.
만약 w 가 볼록하다면 임의의 a<b 인 a,b 에 대해서 opt[i][a]≤opt[i][b] 를 만족하고 오목하면 opt[i][a]≥opt[i][b]이다.

만족 : $A[i][j] \le A[i][j+1]$

$O(kn^2)$-> $O(knlogn)$

```
lint dy[810][8010];
int opt[810][8010];
lint C[8010];
lint sum[8010];
lint cost(int L, int R) {
	return (sum[R] - sum[L - 1])*(R - L + 1);
}
void go(int L, int R, int s, int e, int k) {
	if (L > R)return;
	int m = (L + R) / 2;
	for (int i = s; i <= e; i++) { // for example, for(int i=std::max(k,s);i<=std::min(m,e);i++)
		if (dy[k][m] > dy[k - 1][i - 1] + cost(i, m)) {
			dy[k][m] = dy[k - 1][i - 1] + cost(i, m);
			opt[k][m] = i;
		}
	}
	go(L, m - 1, s, opt[k][m], k);
	go(m + 1, R, opt[k][m], e, k);
}
int main(void) {
	memset(dy, 0x3f, sizeof(dy));
	int l, g;
	scanf("%d%d", &l, &g);
	for (int i = 1; i <= l; i++) {
		scanf("%lld", &C[i]);
		sum[i] = sum[i - 1] + C[i];
		dy[1][i] = cost(1, i);
	}
	dy[1][0] = 0;
	for (int i = 2; i <= g; i++) {
		go(1, l, 1, l, i);
	}
	printf("%lld", dy[g][l]);

	return 0;
}
```

Knuth Optimazation

$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$

조건 : **quadrangle**

inequality: $C[a][c] + C[b][d] \le C[a][d] + C[b][c],\ a \le b \le c \le d$

monotonicity: $C[b][c] \le C[a][d],\ a \le b \le c \le d$

C[i][j-1]<=C[i][j]<=C[i+1][j]

만족 : $A[i,j-1] \le A[i,j] \le A[i+1,j]$

$O(n^3)$-> $O(n^2)$

```
int opt[5010][5010];
lint dy[5010][5010];
int main(void) {
		for (int i = 1; i <= n; i++) {
			scanf("%lld", &C[i]);
			sum[i] = sum[i - 1] + C[i];
			for (int j = 1; j <= n; j++) {
				opt[i][j] = -1;
				dy[i][j] = 0x7fffffffffffffffLL / 2;
			}
			opt[i][i] = i;
			dy[i][i] = 0;
		}
		for (int d = 1; d < n; d++) {
			for (int i = 1; i + d <= n; i++) {
				int j = i + d;
				for (int k = opt[i][j - 1]; k <= opt[i + 1][j]; k++) {
					if (dy[i][j] > dy[i][k] + dy[k + 1][j] + sum[j] - sum[i - 1]) {
						dy[i][j] = dy[i][k] + dy[k + 1][j] + sum[j] - sum[i - 1];
						opt[i][j] = k;
					}
				}
			}
		}
		printf("%lld₩n", dy[1][n]);
}
```

```
.vimrc
syntax on
set expandtab
set ts=4
set autoindent
set nu
set mouse=a
```