Contents

1. C++ Macros and useful hearders 3
2. Data Structures 4
2.1 SUFFIX ARRAY 4
2.2A BIT / FENWICK TREE (LIGHTOJ Curious Robin Hood) 6
2.2B BIT / FENWICK TREE (RANGE UPDATE – SPOJ HORRIBLE) 7
2.3 2D BIT (SPOJ MATSUM) 8
2.4 SEGMENT TREE (SPOJ GSS3) 10
2.5 SEGMENT TREE WITH LAZY PROPAGATION (SPOJ MULTQ3) 14
3. Dynamic Programming 14
3.1 1/0 Knapsack Top Down 14
3.2 LIS (NON DP, N LOG N) 14
3.3 LIS RETURN RESULT 14
3.4 Longest Common Subsequence: 15
3.5 Matrix Chain Multiplication: 15
3.6 Longest Common Increasing Subsequence (LCIS) 16
4. Mathematics 17
4.1 Sieve of Eratosthenes 17
4.2 Segmented Sieve 17
4.3 Prime Factors, Divisors & Numbder of Divisors 18
4.4 GCD, LCM, EGCD 19
4.5 Euler Phi/Totient Function 20
4.6 Permutation nPr, Combination nCr 21
4.7 Chinese Remainder Theorem 24
4.8 POLLARD’s RHO (integer factorization algorithm) 24
4.9 Millar Rabin 27
4.10 Miscellaneous Topics 29
5. Geometry 30
5.1 Header/define 30
5.2 Structures 30

5.3 Function list 31
5.3 Point 31
5.4 Vectors 32
5.5 Point and Line 33
5.7 Lines 34
5.8 Triangle 35
5.9 Circle 35
5.10 Monotone Chain Convex Hull + Points inside hull check 36
6. Graph 38
6.1 Topological sort 38
6.2 Bipartite Graph 40
6.3 Articulation point 40
6.4 Strongly Connected Components (Tarjan’s): 41
6.5 Prim’s MST 41
6.6 Kruskal’s MST 42
6.7 Dijkstra – All shortest paths 45
6.8 Bellman Ford 47
6.9 Dinic’s Algorithm 47
6.10 Bipartite Matching 49
7. String 50
7.1 Hasing 50
7.2 Rabin – Karp 51

1. C++ Macros and useful hearders

```
//#pragma warning (disable: 4786)
//#pragma comment (linker, "/STACK:0x800000")
//#define _CRT_SECURE_NO_WARNINGS 1

#include<bits/stdc++.h>
using namespace std;

template<class T > T _abs(T n) { return (n < 0 ? -n : n); }
template<class T > T _max(T a, T b) { return (!(a < b) ? a :
b);}
template<class T > T _min(T a, T b) { return (a < b ? a : b); }
template<class T > T sq(T x) { return x * x; }

#define ALL(p) p.begin(),p.end()
#define MP(x, y) make_pair(x, y)
#define SET(p) memset(p, -1, sizeof(p))
#define CLR(p) memset(p, 0, sizeof(p))
#define MEM(p, v) memset(p, v, sizeof(p))
#define CPY(d, s) memcpy(d, s, sizeof(s))
#define READ(f) freopen(f, "r", stdin)
#define WRITE(f) freopen(f, "w", stdout)
#define SZ(c) (int)c.size()
#define PB(x) push_back(x)
#define ff first
#define ss second
#define i64 long long
#define ld long double
#define pii pair< int, int >
#define psi pair< string, int >

#define cpp_io() {ios_base::sync_with_stdio(false);cin.tie(0);}
#define FOR(i,N) FORR(i,0,N)
#define FORR(i,a,b) FOTR(i,a,b,1)
#define FOTR(i,a,b,c) for(int i=(a);i<(b);i+=(c))
#define ll long long
#define MAX 100000000
#define  MOD 1000003

const double EPS = 1e-9;
const int INF = 0x7f7f7f7f;
const double PI = 3.141592653589793;
const double E = 2.718281828459045;


/** -------------------END TEMPATE--------------------- **/
```

2. Data Structures

2.1 SUFFIX ARRAY

```
/*sa:
The suffix array. Contains the n suffixes of s sorted in
lexicographical order. Each suffix is represented as a single
integer (the position in the string where it starts).

rank:
The inverse of the suffix array. rank[i] = the index of the
suffix s[i..n) in the pos array. (In other words, sa[i] = k
<==> rank[k] = i). With this array, you can compare two
suffixes in O(1):
    Suffix s[i..n) is smaller than s[j..n) if and only if
rank[i] < rank[j].

lcp:
The length of the longest common prefix between two consecutive
suffixes:
    lcp[i] = lcp(s + sa[i], s + sa[i-1]). lcp[0] = 0.

lcp(i,j) = min(lcp[i],lcp[i + 1], ..., lcp[j - 1]). */

int cmp(double x, double y = 0, double tol = EPS) {
     return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;}
const int MAXN = 100005;
char s[MAXN];
int t, n, rnk[MAXN], sa[MAXN], lcp[MAXN];
// n = strlen(s) + 1
bool compare(int i, int j) {
     return rnk[i + t] < rnk[j + t];}
void setLCP() {
     int size = 0, i, j;
     for (i = 0; i < n; i++)
          if (rnk[i] > 0) {
               j = sa[rnk[i] - 1];
               while (s[i + size] == s[j + size])++size;
               lcp[rnk[i]] = size;
               if (size > 0)--size;}
     lcp[0] = 0;}
int distinctSubstringsCount() {
     int ans = n - 1 - sa[1], i;
     for (i = 2; i < n; i++) {
          int ln = n - 1 - sa[i];
          ans += (ln - lcp[i]);}
     return ans;}
```

```
void distinctSubstrings() {
     vector<pair<int, int>> substrs;
     int i, j, ln;
     for (i = 1; i < n; i++) {
          ln = n - 1 - sa[i];
          for (j = sa[i] + lcp[i]; j < n - 1; j++)
            substrs.push_back(make_pair(sa[i], j - sa[i] + 1));
     }
     for (i = 0; i < substrs.size(); i++) {
          for (j = substrs[i].first; j < substrs[i].first +
                 substrs[i].second;j++)printf("%c", s[j]);
          printf("\n");}}
void build() {
     int bc[256];
     for (int i = 0; i < 256; ++i)
          bc[i] = 0;
     for (int i = 0; i < n; ++i)
          ++bc[s[i]];
     for (int i = 1; i < 256; ++i)
          bc[i] += bc[i - 1];
     for (int i = 0; i < n; ++i)
          sa[--bc[s[i]]] = i;
     for (int i = 0; i < n; ++i)
          rnk[i] = bc[s[i]];
     for (t = 1; t < n; t <<= 1) {
          for (int i = 0, j = 1; j < n; i = j++) {
               while (j < n && rnk[sa[j]] == rnk[sa[i]])
                    j++;
               if (j - i == 1)
                    continue;
               int *start = sa + i, *end = sa + j;
               sort(start, end, compare);
               int first = rnk[*start + t], num = i, k;
               for (; start < end; rnk[*start++] = num) {
                    k = rnk[*start + t];
                    if (k != first and (i > first or k >= j))
                         first = k, num = start - sa;
               }
          }
     }
     setLCP();
}
void search(char *pat) {
     int m = strlen(pat);
     int l = 1, r = n - 1;
```

```
	while (l <= r) {
		int mid = l + (r - l) / 2;
		int res = strncmp(pat, s + sa[mid], m);
		if (res == 0) {
			cout << s + sa[mid] << endl;
			return;
		}
		if (res < 0)
			r = mid - 1;
		else
			l = mid + 1;
	}
	cout << "Pattern not found\n";
}
void printSuffixStrings() {
	//prints suffixes in lexicographical order
	for (int i = 1; i < n; ++i) {
		printf("%s\n", (s + sa[i]));
	}
}
int main() {
	scanf("%s", s);
	n = strlen(s) + 1;
	build();
	//printSuffixStrings();
	distinctSubstrings();
	//cout<<distinctSubstringsCount() ;
	//	for (int i = 0; i<5; i++) {
	//		char st[10];
	//		scanf("%s", st);
	//		search(st);
	//	}
}
```

2.2A BIT / FENWICK TREE (LIGHTOJ Curious Robin Hood)

```
/// LIGHTOJ Curious Robin Hood
// SET N ** IMPORTANT
ll bit[MAX]; int N;
//returns sum from 1 to p
ll get(int p) {
	ll res = 0;
	for (; p >= 0; p = (p & (p + 1)) - 1) res += bit[p];
	return res;
}
```

```
void update(int p, ll val) {
	for (; p < N + 1; p = (p | (p + 1)))
		bit[p] += val;
}
ll rangeSum(int low, int high) {
	return get(high) - (low == 0 ? 0 : get(low - 1));
}
int main() {
	//read();
	int T, m, x, y, z;
	scanf("%d", &T);
	FORR(cs, 1, T + 1) {
		MEM(bit, 0);
		scanf("%d%d", &N, &m);
		FOR(i, N) {
			scanf("%d", &x);
			update(i + 1, x);
		}
		printf("Case %d:\n", cs);
		FOR(LLL, m) {
			scanf("%d", &x);
			if (x == 1) {
				scanf("%d", &y);
				int v = rangeSum(y + 1, y + 1);
				printf("%d\n", v);
				update(y + 1, -v);
			}
			else if (x == 2) {
				scanf("%d %d", &y, &z);
				update(y + 1, z);
			}
			else {
				scanf("%d %d", &y, &z);
				printf("%lld\n", rangeSum(y + 1, z + 1));
			}}}}
```

2.2B BIT / FENWICK TREE (RANGE UPDATE – SPOJ HORRIBLE)

```
//range update, solution of spoj horrible

ll bit1[MAX], bit2[MAX];
int N;
ll get(ll *bit, int p) {
	ll res = 0;
	for (; p >= 0; p = (p & (p + 1)) - 1)
		res += bit[p];
	return res;
}
```

```
void update(ll *bit, int p, ll val) {
     for (; p < N + 1; p = (p | (p + 1)))
          bit[p] += val;
}
// adds val to each elements in [low, high]
void rangeUpdate(int low, int high, ll val) {
     update(bit1, low, val);
     update(bit1, high + 1, -val);
     update(bit2, low, val * (low - 1));
     update(bit2, high + 1, -val * high);
}
ll get_p(int pos) {
     return get(bit1, pos) * pos - get(bit2, pos);
}
// returns sum in range [low, high]
ll rangeSum(int low, int high) {
     return get_p(high) - (low == 0 ? 0 : get_p(low - 1));
}
int main() {
     cpp_io();
     int t, c, m, p, q; ll v;
     cin >> t;
     while (t--) {
          cin >> N >> c; // importent set N!
          memset(bit1, 0, sizeof(bit1));
          memset(bit2, 0, sizeof(bit2));
          while (c--) {
               cin >> m;
               if (m == 0) {
                    cin >> p >> q >> v;
                    rangeUpdate(p, q, v);
               }
               else {
                    cin >> p >> q;
                    cout << rangeSum(p, q) << "\n";}}}}
```

2.3 2D BIT (SPOJ MATSUM)

//http://spoj-solutions.blogspot.com/2014/10/matsum-matrix-summation.html

**** REMARKS ****

1. update function adds given value with already set value.
2. BIT can not start with index 0. That's why input is added with 1.

```
ll tree[1050][1050];
```

```
void update(int x, int y, ll val, int MAX) {
	while (x <= MAX) {
		int ty = y;
		while (ty <= MAX) {
			tree[x][ty] += val;
			ty += (ty & -ty);
		}
		x += (x & -x);
	}
}
ll read(int x, int y) {
	ll sum = 0;
	while (x) {
		int ty = y;
		while (ty) {
			sum += tree[x][ty];
			ty -= (ty & -ty);
		}
		x -= (x & -x);
	}
	return sum;
}
ll readRegtangle(int x0, int y0, int x1, int y1) {
	return read(x1, y1) + read(x0 - 1, y0 - 1) - read(x0 - 1,
		y1)- read(x1, y0 - 1);}
int main() {
	int t; scanf("%d", &t);
	while (t--) {
		int n;
		scanf("%d", &n);memset(tree, 0, sizeof tree);
		while (1) {
			char s[10];scanf("%s", s);
			if (s[1] == 'E') {
				int x, y, val;
				scanf(" %d%d%d", &x, &y, &val);
				ll p_val = readRegtangle(x + 1, y + 1, x +
				1, y + 1);
				update(x + 1, y + 1, val - p_val, n + 9);
			} else if (s[1] == 'U') {
				ll sum = 0;
				int x1, y1, x, y;
				scanf(" %d%d%d%d", &x, &y, &x1, &y1);
				sum = readRegtangle(x + 1, y + 1, x1 + 1,
				y1 + 1);
				printf("%lld\n", sum);
```

```
                } else
                    break;
            }
            printf("\n");
        }
}
```

2.4 SEGMENT TREE (SPOJ GSS3)

```
const ll MAX = 500000;
const ll MIN = -50000000000000000LL;
struct NODE {
    ll rmax, lmax, tmax, sum;
} tree[MAX];
int inp[MAX];
int max3(ll a, ll b, ll c) {
    ll mx = a;
    if (b > mx)
        mx = b;
    if (c > mx)
        mx = c;
    return mx;
}
void setNode(int node) {
    int ch1 = node << 1, ch2 = ch1 | 1;
    tree[node].sum = tree[ch1].sum + tree[ch2].sum;
    tree[node].lmax = max(tree[ch1].lmax, tree[ch1].sum +
tree[ch2].lmax);
    tree[node].rmax = max(tree[ch2].rmax, tree[ch1].rmax +
tree[ch2].sum);
    tree[node].tmax = max3(tree[ch1].tmax, tree[ch2].tmax,
            tree[ch1].rmax + tree[ch2].lmax);
}
NODE join(NODE n1, NODE n2) {
    NODE n;
    n.sum = n1.sum + n2.sum;
    n.lmax = max(n1.lmax, n1.sum + n2.lmax);
    n.rmax = max(n2.rmax, n1.rmax + n2.sum);
    n.tmax = max3(n1.tmax, n2.tmax, n1.rmax + n2.lmax);
    return n;
}

void build(int node, int first, int last) {
    if (first == last) {
        tree[node].rmax = tree[node].tmax = tree[node].sum =
        tree[node].lmax =inp[first];}
```

```
else {
        int mid = (first + last) >> 1, ch1 = node << 1, ch2 =
        ch1 | 1;
        build(ch1, first, mid);
        build(ch2, mid + 1, last);
        setNode(node);
    }
}
void update(int node, int first, int last, int p, int val) {
    if (first == last) {
        tree[node].rmax = tree[node].tmax = tree[node].sum =
        tree[node].lmax =val;
        return;
    }
    int mid = (first + last) >> 1;
    if (p <= mid)
        update(node << 1, first, mid, p, val);
    else
        update((node << 1) | 1, mid + 1, last, p, val);
    setNode(node);
}
NODE query(int node, int first, int last, int l, int r) {
    if (first > last || first > r || last < l)
        return {MIN,MIN,MIN,MIN};
    if (first >= l && last <= r)
        return tree[node];
    int mid = (first + last) >> 1;
    NODE n1 = query(node << 1, first, mid, l, r);
    NODE n2 = query((node << 1) | 1, mid + 1, last, l, r);

    if (n1.tmax == MIN)
        return n2;
    else if (n2.tmax == MIN)
        return n1;

    return join(n1, n2);
}
int main() {
    cpp_io();
    int n, m, x, y, z;
    cin >> n;
    FORR(i, 1, n + 1)cin >> inp[i];
    build(1, 1, n);cin >> m;
    FOR(i, m) {
        cin >> x >> y >> z;
```

```
            if (x == 1) {
                  NODE nd = query(1, 1, n, y, z);
                  cout << nd.tmax << "\n";
            }
            else {
                  update(1, 1, n, y, z);
            }
      }
}
```

2.5 SEGMENT TREE WITH LAZY PROPAGATION (SPOJ MULTQ3)

```
const ll MAX = 500000;
struct NODE {
      int zero, one, two;
      NODE() {
            zero = one = two = 0;
      }
}tree[MAX];

int lazy[MAX];
void setNode(int node) {
      int ch1 = node << 1, ch2 = ch1 ^ 1;
      tree[node].zero = tree[ch1].zero + tree[ch2].zero;
      tree[node].one = tree[ch1].one + tree[ch2].one;
      tree[node].two = tree[ch1].two + tree[ch2].two;
}
void lazyUpdate(int node, int lazyVal, bool push) {
      int v[3];
      v[0] = tree[node].zero, v[1] = tree[node].one,
      v[2] = tree[node].two;
      tree[node].zero = v[lazyVal];
      tree[node].one = v[(lazyVal + 1) % 3];
      tree[node].two = v[(lazyVal + 2) % 3];
      if (push) {
            lazy[node << 1] += lazyVal;
            lazy[(node << 1) | 1] += lazyVal;
      }
      lazy[node] = 0;
}
void build(int node, int first, int last) {
      if (first == last) tree[node].zero++;
      else {
            int mid = (first + last) >> 1;
            build(node << 1, first, mid);
            build((node << 1) | 1, mid + 1, last);
            setNode(node);
           }
}
```

```
void update(int node, int first, int last, int l, int r, int val) {
	if (lazy[node] != 0) {
		lazyUpdate(node, lazy[node] % 3, first != last);
	}
	if (first > last || first > r || last < l) return;

	if (first >= l && last <= r) { //update
		lazyUpdate(node, val % 3, first != last);
		return;
	}
	int mid = (first + last) >> 1;
	update(node << 1, first, mid, l, r, val);
	update((node << 1) | 1, mid + 1, last, l, r, val);

	setNode(node);
}
int query(int node, int first, int last, int l, int r) {
	if (first > last || first > r || last < l) return 0;
	if (lazy[node] != 0) {
		lazyUpdate(node, lazy[node] % 3, first != last);
	}
	if (first >= l && last <= r) return tree[node].zero;

	int mid = (first + last) >> 1;
	int p1 = query(node << 1, first, mid, l, r);
	int p2 = query((node << 1) | 1, mid + 1, last, l, r);

	return (p1 + p2);
}

int main() {
	//read();
	cpp_io();
	int n, m, x, y, z;
	cin >> n >> m;
	build(1, 1, n);
	FOR(i, m) {
		cin >> x >> y >> z;
		if (x == 1) {
			cout << query(1, 1, n, y + 1, z + 1) << "\n";
		}
		else {
			update(1, 1, n, y + 1, z + 1, 1);
		}
	}
}
```

3. Dynamic Programming

3.1 1/0 Knapsack Top Down

```
int dp[MAX][MAX], P[]={60, 100, 120},  W[] = {10, 20, 30},N;
int knps(int i, int w) {
     if (i >= N || w == 0)return 0;
     if (dp[i][w] != -1)return dp[i][w];
     int p1 = 0, p2 = 0;
     if (W[i] <= w)p1 = P[i] + knps(i + 1, w - W[i]);
     p2 = knps(i + 1, w);
     return dp[i][w] = max(p1, p2);
    }
```

3.2 LIS (NON DP, N LOG N)

```
int n;int v[] = {50, 3, 10, 7, 40, 80};

int LISLength() {

     multiset<int> s; //use set if input doesn't have duplicates
     multiset<int>::iterator it;
     for (int i = 0; i<n; i++) {
          s.insert(v[i]);
          it = upper_bound(s.begin(), s.end(), v[i]);
         /* for strictly increasing:
         it = lower_bound(s.begin(), s.end(), x); it++; */
          if (it != s.end()) s.erase(it);
     }
     return s.size();
}
```

3.3 LIS RETURN RESULT

```
vector<int> LISResult(vector<int> input) {
     vector<int>L(input.size());
     vector<int>L_id(input.size());
     vector<int>P(input.size());
     vector<int>output;
     int lis = 0, lis_end = 0, i;
     for (int i = 0; i < input.size(); ++i) {
          int pos = lower_bound(L.begin(), L.begin() + lis,
       input[i]) - L.begin();
       //use upper_bound for non decreasing
          L[pos] = input[i];
          L_id[pos] = i;
          P[i] = pos ? L_id[pos - 1] : -1;
          if (pos + 1 > lis) {
               lis = pos + 1; lis_end = i;
          }}
     i = lis_end;
     for (; P[i] >= 0; i = P[i]) output.push_back(input[i]);
     output.push_back(input[i]);
     reverse(output.begin(), output.end());
     return output;}
```

3.4 Longest Common Subsequence:

```
char str1[] = "AGGTAB", str2[] = "GXTXAYB";
int lcs(int n1, int n2) {
	int lcs[n1 + 1][n2 + 1];
	for (int i = 0; i <= n1; i++)
		for (int j = 0; j <= n2; j++) {
			if (i == 0 || j == 0)
				lcs[i][j] = 0;
			else {
				lcs[i][j] = max(lcs[i - 1][j], lcs[i][j - 1]);
				if (str1[i - 1] == str2[j - 1])
					lcs[i][j] = max(lcs[i][j],
					lcs[i - 1][j - 1] + 1);
			}
		}
	return lcs[n1][n2];
}
```

3.5 Matrix Chain Multiplication:

```
int m[MAX][MAX], s[MAX][MAX];
int MatrixChainOrder(int p[], int n){
	n = n - 1;  //length of p -1
	int L, i, j, k, q;
	for (int i = 1; i <= n; i++) m[i][i] = 0;
	for ( L = 2; L <= n; L++) {
		for (i = 1; i <= n - L + 1; i++) {
			j = i + L - 1;
			m[i][j] = INT_MAX;
			for (k = i; k <= j - 1; k++) {
				q = m[i][k] + m[k + 1][j]
				+ p[i - 1] * p[k] * p[j];
				if (q < m[i][j]) {
					m[i][j] = q;
					s[i][j] = k;
				}
			}
		}
	}
	return  m[1][n];
}
void print(int p[], int i , int j) {
	if (i == j) cout << p[i]<< " ";
	else {
		cout << "(";
		print(p, i, s[i][j]);
		print(p, s[i][j]+1, j);
		cout << ")";
	}
}
```

3.6 Longest Common Increasing Subsequence (LCIS)

```
#define N 2000
int n, a[N], m, b[N], dp[N], pr[N], l;
void trace(int x, bool m) {
	if (!x)
		return;
	trace(pr[x], 1);
	printf("%d", b[x]);
	putchar(m ? ' ' : '\n');
}
void lcis() {
	int i, j;
	memset(dp, 0, sizeof(dp));
	memset(pr, 0, sizeof(pr));
	for (i = 1; i <= n; i++) {
		int c = 0, p = 0;
		for (j = 1; j <= m; j++) {
			if (a[i] == b[j] && dp[j] < c + 1) {
				dp[j] = c + 1;
				pr[j] = p;
			}
			if (a[i] > b[j] && c < dp[j]) {
				c = dp[j];
				p = j;
			}
		}
	}
	l = -1; // l = lcis length
	for (i = 1; i <= m; i++)
		l = max(l, dp[i]);
}
int main() {
	scanf("%d", &n); /// let n = 4 int i, j;
	for (i = 1; i <= n; i++)
		scanf("%d", &a[i]); /// let  a[] = {3, 4, 9, 1};
	scanf("%d", &m); /// let m = 7
	for (i = 1; i <= m; i++)
		scanf("%d", &b[i]); /// let b[] = {5, 3, 8, 9, 10, 2, 1};
	lcis();
	printf("Length of LCIS is %d\n\n", l); /// Length of LCIS is 2
	if (l) {
		for (i = 1; i <= m; i++)
			if (dp[i] == l) {
				cout << "The LCIS is : ";
				trace(i, 0);
				break;
			}
	}
}
```

4. Mathematics

4.1 Sieve of Eratosthenes

```
#define MAX 46656
#define LMT 216
#define LEN 4830
#define RNG 100032
#define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))
#define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))
unsigned base[MAX / 64], segment[RNG / 64], primes[LEN];
void sieve() {
	unsigned i, j, k;
	for (i = 3; i < LMT; i += 2)
		if (!chkC(base, i))
			for (j = i * i, k = i << 1; j < MAX; j += k)
				setC(base, j);
	primes[0] = 2;
	j = 1;
	for (i = 3; i < MAX; i += 2)
		if (!chkC(base, i))
			primes[j++] = i;
}
bool isPrime(int n) {
	if (n == 2 || (n > 2 && n & 1 && !chkC(base, n)))
		return true;
	return false;
}
```

4.2 Segmented Sieve

```
/// NOT OK
#define sq(x) ((x)*(x))
#define mset(x,v) memset(x,v,sizeof(x))
int primes[100],segment[100]; /// ADDED by SunW
void segmented_sieve(int a, int b) {
	unsigned i, j, k, cnt = (a <= 2 && 2 <= b) ? 1 : 0;
	if (b<2) return;
	if (a<3) a = 3;
	if (a % 2 == 0) a++;

	mset(segment, 0);
	for (i = 1; sq(primes[i]) <= b; i++) {
		j = primes[i] * ((a + primes[i] - 1) / primes[i]);
		if (j % 2 == 0) j += primes[i];
		for (k = primes[i] << 1; j <= b; j += k)
			if (j != primes[i]) setC(segment, (j - a));
	}
}
bool isPrimeSeg(int n, int a) {
	if (n == 2)return true;
	else if (!(n & 1) || n<2) return false;
```

```
	if (!chkC(segment, n - a))return true;
	return false;
}
```

4.3 Prime Factors, Divisors & Numbder of Divisors

```
vector<int> v;
int primes[] = { 2, 3, 5, 7, 11, 13, 17, 19 }; /// ADDED by SunW
void primeFactors(int n) { //Ex. When n = 12, v[] = {2,2,3}
	for (int i = 0; primes[i] * primes[i] <= n; i++) {
		int x = primes[i];
		while (n % x == 0) {
			v.push_back(x);
			n = n / x;
		}
	}
	if (n >= 2)
		v.push_back(n);
}
int noOfDivisors(int n) { //When n = 12, returns 6
	int c = 0, ans = 1;
	for (int i = 0; primes[i] <= sqrt(n); i++) {
		c = 0;
		int x = primes[i];
		while (n % x == 0) {
			c++;
			n = n / x;
		}
		if (c > 0)
			ans *= (c + 1);
	}
	if (n >= 2)
		ans *= 2;
	return ans;
}

vector<int> divisors;
void allDivisors(int n) { //all divisors for small n
	divisors.clear();
	int i;
	for (i = 1; i < sqrt(n); i++) {
		if (n % i == 0) {
			divisors.push_back(i);
			divisors.push_back(n / i);
		}
	}
	if (i * i == n)
		divisors.push_back(i);
}

vector<pii> factors;
```

```
// When n = 12, factors[] = {(2,2), (3,1)}
void primeFactorsPair(int n) {
	int c = 0;
	for (int i = 0; primes[i] <= sqrt(n); i++) {
		c = 0;
		int x = primes[i];
		while (n % x == 0) {
			c++;
			n = n / x;
		}

		if (c > 0)
			factors.push_back(make_pair(x, c));
	}
	if (n >= 2)
		factors.push_back(make_pair(n, 1));
}
//call primeFactorsPair with n(12)
//call allDivisors with (1, 0), divisors[] = {2,6,4,12,3}
//also needs sieve()
void allDivisors(int n, int i) {
	int j, x, k;
	for (j = i; j < factors.size(); j++) {
		x = factors[j].first * n;
		for (k = 0; k < factors[j].second; k++) {
			divisors.push_back(x);
			allDivisors(x, j + 1);
			x *= factors[j].first;
		}
	}
}
///Divisors of numbers from 1 to N
/* When MAX  = 500007 it takes about 600ms/51MB in codeforces
 MAX = 10^6 takes 1.4s in uva*/
#define MAX 500007
vector<int> divisorz[MAX + 1];
void divisors_1_to_n(int n) {
	for (int i = 1; i <= n; i++) {
		for (int j = i; j <= n; j += i)
			divisorz[j].push_back(i);
		// divisors[j]+=i for Sum of divisors
		// divisors[j]++ for Number of divisors
	}
}
```

4.4 GCD, LCM, EGCD

```
ll gcd(ll a, ll b) {
	return b ? gcd(b, a % b) : a;
}
```

```
ll lcm(ll a, ll b) {
	return a * (b / gcd(a, b));
}
//egcd solves a.x + b.y = gcd(a,b)
//if used to solve for ax+by=1, where a & b are coprime,
//check if egdc returns -1. If it does, swap x, y
ll egcd(ll a, ll b, ll & x, ll & y) {
	if (a == 0) {
		x = 0;
		y = 1;
		return b;
	}
	ll x1, y1;
	ll d = egcd(b % a, a, x1, y1);
	x = y1 - (b / a) * x1;
	y = x1;
	return d;
}
```

4.5 Euler Phi/Totient Function

```
long long phi[MAX + 2], mark[MAX + 2];
//Phi from 1 to n
void ETF(int n) {
	int i, j;
	for (i = 0; i <= n; i++)
		phi[i] = i;
	mark[1] = 1;
	for (i = 2; i <= n; i++) {
		if (!mark[i]) {
			for (j = i; j <= n; j += i) {
				mark[j] = 1;
				phi[j] = phi[j] / i * (i - 1);
			}
		}
	}
}
int phif(int n) { //phi for n
	int ret = n;
//iterate through primes[] for faster result
	for (int 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;
}
```

4.6 Permutation nPr, Combination nCr

```
///NOT OK
long long F[MAX];
void initF() {
	F[1] = 1;
	F[0] = 1;
	for (int i = 2; i < MAX; i++)
		F[i] = (F[i - 1] * i) % MOD;
}
long long POW(int a, int b) { // a^b)%MOD
	long long x = 1, y = a;
	while (b > 0) {
		if (b & 1)
			x = (x * y) % MOD;
		y = (y * y) % MOD;
		b >>= 1;
	}
	return x;
}
long long inversePow(int n) {
	return POW(n, MOD - 2);
}
long long C(int n, int r) {
	return (F[n] * ((inversePow(F[r]) * inversePow(F[n - r])) %
MOD)) % MOD;
}
long long P(int n, int r) {
	return (F[n] * inversePow(F[n - r])) % MOD;
}
```

Some Formulas:

** C(n, k) = C(n-1, k-1) + C(n-1, k)

1	Permutation: Order ✓ | Repetition ✖	$\frac{n!}{(n-r)!}$
2	Order ✓ | Repetition ✓	n^r
3	Combination: Order ✖ | Repetition ✖	$\frac{n!}{(r!)(n-r)!}$
4	Permutation when P1 are of same kind, P2 are of same kind and so on	$\frac{n!}{p_1!\ p_2!\ \cdots\ p_r!}$
5	Circular permutation of n distinct object	(n-1)!

6	Circular permutation when clockwise and anti-clockwise arrangements are not different	.5*(n-1)!
7	Number of **permutations** of **n** distinct things taking **r** at a time, when **s** particular things are always to be **included** in each arrangement	$^{(n-s)}C_{(r-s)} \times r!$
8	Number of **permutations** of **n** distinct things taking **r** at a time, when **s** particular things are **never included**	$^{(n-s)}C_{r} \times r!$
9A	Number of **combinations** of **n** distinct objects taking **r** at a time when each object may be **repeated any number of times**	$^{(n+r-1)}C_{r}$
9B	Number of ways in which **n** identical things can be distributed among **r** persons, each one of them can receive **0,1,2** or more items	$^{(n+r-1)}C_{(r-1)}$
9C	Number of ways in which **n** identical things can be distributed among **r** persons, each one of them can receive **1,2** or more items	$^{(n-1)}C_{(r-1)}$
10	Number of **combinations** of **n** distinct things taking **r** at a time, when **s** particular things are **always included** in each selection	$^{(n-s)}C_{(r-s)}$
11	Number of **combinations** of **n** distinct things taking **r** at a time, when **s** particular things are **never included**	$^{(n-s)}C_{r}$
12	Number of **combinations** of **n** distinct things taking **r** at a time, when **m** particular things **never come together**	$^{n}C_{r} - {}^{(n-m)}C_{(r-m)}$
13	Number of ways in which **n** distinct things can be divided into **r unequal non-distinct groups** containing a_1, a_2, a_3, ..., a_r things	$\frac{n!}{a_1!\, a_2!\, a_3! \cdots a_r!}$
14	Number of ways in which **n** distinct things can be divided into **r unequal distinct groups** containing a_1, a_2, a_3, ….., a_r things	$\frac{n!\, r!}{a_1!\, a_2!\, a_3! \ldots a_r!}$
15	Number of ways in which **m×n** distinct things can be divided equally into **n** groups ***(each group will have m things and the groups are unmarked)***	$\frac{(mn)!}{(m!)^n\, n!}$

16	Number of ways in which **m×n** distinct things can be divided equally into **n** groups ***(each group will have m things and the groups are distinct)***	$\frac{(mn)!}{(m!)^n}$

Distribution of		How many balls boxes can contain			
k Balls	into n Boxes	No Restrictions	≤ 1 (At most one)	≥ 1 (At least one)	= 1 (Exactly one)
Distinct	Distinct	n^k *(formula 1)*	${}^nP_k$ *(formula 2)*	S(k,n) × n! *(formula 3)* *(more info)*	${}^nP_n = n!$ if k = n 0 if k ≠ n *(formula 4)*
Identical	Distinct	${}^{(k+n-1)}C_{(n-1)}$ *(formula 5)*	${}^nC_k$ *(formula 6)*	${}^{(k-1)}C_{(n-1)}$ *(formula 7)*	1 if k = n 0 if k ≠ n *(formula 8)*
Distinct	Identical	$\sum_{i=1}^{n} S(k,i)$ *(formula 9)* *(more info)*	1 if k ≤ n 0 if k > n *(formula 10)*	S(k,n) *(formula 11)* *(more info)*	1 if k = n 0 if k ≠ n *(formula 12)*
Identical	Identical	$\sum_{i=1}^{n} P(k,i)$ *(formula 13)* *(more info)*	1 if k ≤ n 0 if k > n *(formula 14)*	P(k, n) *(formula 15)* *(more info)*	1 if k = n 0 if k ≠ n *(formula 16)*

4.7 Chinese Remainder Theorem

```
/* Solves equations of the format x % mods[i] = r[i],
( 0<=i<n, where n is the number of equations ) */
///NO IDEA ??
long long CRT(vector<long long> &r, vector<long long> &mods) {
      long long M = 1, ret = 0;
      for (int i = 0; i < int(mods.size()); i++)
            M *= mods[i];
      vector<long long> m, s;
      for (int i = 0; i < int(mods.size()); i++) {
            m.push_back(M / mods[i]);
            long long temp = m[i] % mods[i], k = 0;
            /*if there is a possibility of k being very big, then
             prime factorize m[i], find modular inverse of 'temp'
             of each of the factors 'k' equals to the multiplication
             (modular mods[i]) of modular inverses */
```

```
            while (true) {
                  if ((k * temp) % mods[i] == 1)
                        break;
                  k++;
            }
            s.push_back(k);
      }
      for (int i = 0; i < int(s.size()); i++) {
            ret += ((m[i] * s[i]) % M * r[i]) % M;
            if (ret >= M)
                  ret -= M;
      }return ret;}
```

4.8 POLLARD's RHO (integer factorization algorithm)

```
const long long LIM = LLONG_MAX;
uint64_t gcd(uint64_t a, uint64_t b) {
      return b ? gcd(b, a % b) : a;
}
long long mul(long long a, long long b, long long m) {
      long long x, res;
      if (a < b)
            swap(a, b);
      if (!b)
            return 0;
      if (a < (LIM / b))
            return ((a * b) % m);
      res = 0, x = (a % m);
      while (b) {
            if (b & 1) {
                  res = res + x;
                  if (res >= m)
                        res -= m;
            }
            b >>= 1;
            x <<= 1;
            if (x >= m)
                  x -= m;
      }
      return res;
}
long long expo(long long x, long long n, long long m) {
      long long res = 1;
      while (n) {
            if (n & 1)
                  res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
      }
      return (res % m);
}
```

```
const int small_primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43,47, 51, 53, 59, 61, 67, 71 };

bool miller_rabin(long long p, int lim) {
	long long a, s, m, x, y;
	s = p - 1, y = p - 1;
	while (!(s & 1))
		s >>= 1;

	while (lim--) {
		x = s;
		a = (rand() % y) + 1;
		m = expo(a, x, p);

		while ((x != y) && (m != 1) && (m != y)) {
			m = mul(m, m, p);
			x <<= 1;
		}
		if ((m != y) && !(x & 1))
			return false;
	}
	return true;
}
void brent_pollard_rho(uint64_t n, vector<uint64_t> &v) {
	if (miller_rabin(n, 10)) {
		v.push_back(n);
		return;
	}
	uint64_t a, g, x, y;
	y = 1;
	a = rand() % n;
	x = rand() % n;

	for (int i = 0; ((i * i) >> 1) < n; i++) {
		x = mul(x, x, n);
		x += a;
		if (x < a)
			x += (ULLONG_MAX - n) + 1;
		x %= n;
		g = gcd(n, y - x);
		if ((g != 1) && (g != n)) {
			n /= g;
			brent_pollard_rho(g, v);
			if (n != g)
				brent_pollard_rho(n, v);
			else if (miller_rabin(n, 10))
				v.push_back(n);
			return;
		}
```

```
            if (!(i & (i - 1)))
                y = x;
        }
        brent_pollard_rho(n, v);
}
void factorize(uint64_t n, vector<uint64_t> &v) {
        srand(time(0));
        int i, j, x;
        for (i = 0; i < 21; i++) {
            x = small_primes[i];
            while ((n % x) == 0) {
                n /= x;
                v.push_back(x);
            }
        }
        if (n > 1)
            brent_pollard_rho(n, v);
        sort(v.begin(), v.end());
}
vector<pair<uint64_t, int> > fc;
vector<uint64_t> dv;
void divs(int n, int i) {
        uint64_t j, x, k;
        for (j = i; j < fc.size(); j++) {
            x = fc[j].first * n;
            for (k = 0; k < fc[j].second; k++) {
                dv.push_back(x);
                divs(x, j + 1);
                x *= fc[j].first;
            }
        }
}
void divisors(vector<uint64_t> &v) {
        dv.clear();
        fc.clear();
        int c = 0;
        for (int i = 0; (i + 1) < v.size(); i++) {
            if (v[i] == v[i + 1])
                c++;
            else {
                if (c)
                    fc.push_back(make_pair(v[i], c + 1));
                else
                    fc.push_back(make_pair(v[i], 1));
                c = 0;
            }
        }
        divs(1, 0);
        sort(dv.begin(), dv.end());  }
```

```
int main() {
	uint64_t n, i, t, x;
	cin >> t;
	while (t--) {
		cin >> n;
		vector<uint64_t> v;
		factorize(n, v);
		sort(v.begin(), v.end());
		v.push_back(-1); /***IMPORTANT***/
		int len = v.size(), c = 0, counter = 0;
		printf("%llu = ", n);
		for (i = 0; (i + 1) < len; i++) {
			if (v[i] == v[i + 1])
				counter++;
			else {
				if (c)
					printf(" * ");
				if (counter)
					printf("%llu^%d", v[i], ++counter);
				else
					printf("%llu", v[i]);
				c++, counter = 0;
			}
		}
		puts("");
		divisors(v);
		for (auto i : dv)
			cout << i << " ";
		cout << endl;
	}
}
```

4.9 Millar Rabin

```
const long long LIM = 10000000000000000;
const int small_primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43,
		47, 51, 53, 59, 61, 67, 71 };

long long mul(long long a, long long b, long long m) {
	long long x, res;
	if (a < b)
		swap(a, b);
	if (!b)
		return 0;
	if (a < (LIM / b))
		return ((a * b) % m);
	res = 0, x = (a % m);
	while (b) {
		if (b & 1) {
			res = res + x;
```

```
                if (res >= m)
                        res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m)
                x -= m;
    }
    return res;
}
long long expo(long long x, long long n, long long m) {
    long long res = 1;

    while (n) {
        if (n & 1)
                res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}
bool miller_rabin(long long p, int lim) {
    if (p < 2)
        return false;
    if (p == 2)
        return true;
    if (!(p & 1))
        return false;
    int i, val;
    long long a, s, m, x, y;
    for (i = 0; i < 20; i++) {
        val = small_primes[i];
        if (p == val)
                return true;
        if ((p % val) == 0) {
                return false;
        }
    }
    srand(time(0));
    s = p - 1, y = p - 1;
    while (!(s & 1))
        s >>= 1;

    while (lim--) {
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);
```

```
            while ((x != y) && (m != 1) && (m != y)) {
                  m = mul(m, m, p);
                  x <<= 1;
            }
            if ((m != y) && !(x & 1))
                  return false;
      }return true;}
```

4.10 Miscellaneous Topics

a. Java BigInteger count digits:

```
public static int getDigitCount(BigInteger number) {
    double factor = Math.log(2) / Math.log(10);
    int digitCount = (int) (factor * number.bitLength() + 1);
    if (BigInteger.TEN.pow(digitCount - 1).compareTo(number) >
0) {
        return digitCount - 1;
    }
    return digitCount;
}
```

BigDecimal Division:

```
a.divide(b, 1000, RoundingMode.CEILING);      //java.math.*
```

b. ax + by = c

if (c % gcd(a,b) == 0), then equation has a solution.

c.

x	primes(*x*)
10	4
100	25
1,000	168
10,000	1,229
100,000	9,592
1,000,000	78,498
10,000,000	664,579
100,000,000	5,761,455
1,000,000,000	50,847,534
10,000,000,000	455,052,511

d.

Gray Code

```
int gray(int n) { return n ^ (n >> 1); }
int rev_g(int g) {
     int n = 0;
     for (; g; g >>= 1) n ^= g;
     return n;
}
```

```
e. //roots of eqn ax^2 + bx + c. a!=0
void quadroots (double a, double b, double c, double &r1,
double &r2) {
     if ((b*b - 4 * a*c) >= 0.0) {
          double d = sqrt(b*b - 4 * a*c);
          r1 = (-b - d) / (2 * a);
          r2 = (-b + d) / (2 * a);
          return;
     }
     //imaginary root
     r1 = -b / (2 * a);
     r2 = sqrt(-(b*b - 4 * a*c)) / (2 * a);
```

f. Log base change formula

$\text{Log}_b x = \text{Log}_a x \;/\; \text{Log}_a b$

5. Geometry

5.1 Header/define

```
#include <cmath>
#include <iomanip>
#include <cfloat>   //DBL_MAX, DBL_MIN
#define EPS (1e-9)
#define PI 3.14159265359 // acos(-1);
```

5.2 Structures

```
struct point {
     double x, y;
     point() { x = y = 0.0; }
     point(double _x, double _y) : x(_x), y(_y) {}
     bool operator <(const point & p) const {
          return ((x < p.x) || (abs(x - p.x) < EPS &&
               (y < p.y)));
     }
     point operator-(point v) { return point(x - v.x, y - v.y);
}
     point operator+(point v) { return point(x + v.x, y + v.y);
}
};
struct line{ double a, b, c; };

struct vec {
       double x, y;
       vec( double _x, double _y) : x( _x), y( _y) {} };
```

5.3 Function list

1. dist	15. pointAtDist	29. tarea1
2. DEGtoRAD	16. linePointDist1	30. tarea2
3. ROTATE	17. linePointDist2	31. tarea3
4. rotatep	18. linePointDist3	32. rInCircle
5. toVec	19. linePointDist4	33. rInCircleP
6. scale	20. toLine	34. rCircumCircle
7. translate	21. parallel	35. rCircumCircleP
8. dot	22. sameLine	
9. norm_sq	23. lineIntersect1	
10. len	24. lineIntersect2	
11. dot2	25. cross3	
12. cross	26. lineIntersect3	
13. onLine	27. angle	
14. onLineL	28. s_perm	

5.3 Point

1. Distance of 2 points

```
double dist(point p1, point p2) {
        return hypot(p1.x - p2.x, p1.y - p2.y); }
```

2. Degree to radian

```
double DEGtoRAD(double theta){
        return (theta * ( PI / 180.0)); }
```

3. Rotate p by theta degrees CCW w.r.t origin (0, 0)

```
point ROTATE(point p, double theta) {
        double rad = DEGtoRAD(theta);
        return point( p.x * cos(rad) - p.y * sin(rad),
              p.x * sin(rad) + p.y * cos(rad));
}
```

4. Rotate a point angle radian about center, CCW

```
point rotatep(point a , point center , double angle ){
       point p;
       double x1 = a.x - center.x;
       double y1 = a.y - center.y;
       double x2 = x1 * cos( angle) - y1 * sin( angle);
       double y2 = x1 * sin( angle) + y1 * cos( angle);
       p.x = x2 + center.x;
       p.y = y2 + center.y;
       return p; }
```

5.4 Vectors

```
5. vec toVec(point a , point b ) { return vec( b.x - a.x, b.y -
a.y); }
```

```
6. vec scale(vec v, double s) { return vec(v.x * s, v.y * s); }

7. point translate(point p, vec v) { return point(p.x+v.x,
p.y+v.y); }

8. double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

9. double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

10. double len(point a ) { return sqrt(a.x*a.x + a.y*a.y); }

11. double dot2(point a , point b ) { return a.x*b.x + a.y*b.y;
}

12. double cross(point a , point b ) { return a.x * b.y - a.y *
b.x; }
```

5.5 Point and Line

13. Is distance ab = ar + rb ?

```
bool onLine(point a , point b , point r ){
        if (fabs(dist( a, b) - (dist( a, r)
                + dist( b, r))) < EPS) return true;
        return false;
}
```

14. Is point b on line x?

```
bool onLineL(line x , point b ){
     if ( fabs(( x.a * b.x + x.b * b.y) + x.c) < EPS)
          return true;
     return false;
}
```

15. Find a point p which is on line ab and ap = d

```
point pointAtDist(point a , point b , double d ){
     point p;
     double _d = dist( a, b);
     p.x = ( a.x + ( b.x - a.x)*( d/_d) );
     p.x = ( a.y + ( b.y - a.y)*( d/_d) );
     return p;
}
```

16. Distance of AB to C (When point on AB is not needed)

```
double linePointDist1
     (point A, point B, point C, bool isSegment) {
     double dis = cross(B - A, C - A) / len(B - A);
     if (isSegment) {
          if (dot2(C - B, B - A) > EPS)
```

```
                return len(B - C);
           if (dot2(C - A, A - B) > EPS)
                return len(A - C);
     }
     return fabs(dis);
}
```

17. Distance ab to p. Closest point saved in c

```
double linePointDist2(point p, point a, point b, point &c) {
     vec ap = toVec(a, p), ab = toVec(a, b);
     double u = dot(ap, ab) / norm_sq(ab);
     c = translate(a, scale(ab, u));
     return dist(p, c);
}
```

18. Distance of p to ab(Segment). Closest point saved in c

```
double linePointDist3
     (point p, point a, point b, point &c) {
     vec ap = toVec(a, p), ab = toVec(a, b);
     double u = dot(ap, ab) / norm_sq(ab);
     if (u < 0.0) {
          c = point(a.x, a.y);
          return dist(p, a);
     }
     if (u > 1.0) {
          c = point(b.x, b.y);
          return dist(p, b); }
     return distToLine(p, a, b, c);
}
```

19.

```
double linePointDist4(line l, point p) {
     return fabs(p.x * l.a + p.y * l.b - l.c) /
          sqrt(l.a* l.a + l.b* l.b);
}
```

5.7 Lines

20.

```
line toLine(point p1 , point p2 ) {
       line x;
       x.a = p1.y - p2.y; x.b = p2.x - p1.x;
       x.c = -(x.a * p1.x + x.b* p1.y);
       return x;
}
```

21. l1 parallel of l2?

```
bool parallel(line l1, line l2) {
     return (fabs(l1.a * l2.b - l2.a * l1.b) < EPS); }
```

22. line l1 == l2?

```
bool sameLine(line l1, line l2) {
     return (fabs(l1.a * l2.b - l1.b * l2.a) < EPS &&
          fabs(l1.b * l2.c - l1.c * l2.b)<EPS);  }
```

23. Intersect of 2 lines(return false if there is no intersect)

```
bool lineIntersect1 (line l1 , line l2 , point &p ) {
        double a = l1.a * l2.b - l2.a * l1.b;
        if (fabs(a) < EPS) return false;
        double b = l1.b * l2.c - l2.b * l1.c;
        double c = l1.c * l2.a - l2.c * l1.a;
        p.x = b / a;  p.y = c / a;  return true;
}
```

24. Intersect of 2 line segments (return false if there is no intersect)

```
bool lineIntersect2(point a, point b, point x, point y, point
&m) {
     line l1 = toLine(a, b), l2 = toLine(x, y);
     if (sameLine(l1, l2)){
     //Check what's said about same lines
          if (onLine(a, b, x) || onLine(a, b, y) ||
               onLine(x, y, a) || onLine(x, y, b))
                    return true;
          return false;
     }
     if ( lineIntersect1 (l1, l2, m)){
          if (onLine(a, b, m) && onLine(x, y, m))
               return true;
     }
     return false;
}
```

25.

```
double cross3(point p1, point p2, point p){
     return (p2.x - p1.x) * (p.y - p1.y) - (p2.y - p1.y) *(p.x
     - p1.x);
}
```

26. Checks if 2 line segment intersect, returns false if segments touch

```
bool lineIntersect3(point p1, point p2, point q1, point q2){
     return cross3(p1, p2, q1) * cross3(p1, p2, q2) < 0 &&
          cross3(q1, q2, p1) * cross3(q1, q2, p2) < 0;
}
```

5.8 Triangle

27.

```
double angle(point a, point o, point b) {
     double ang = (fabs(atan2(a.y - o.y, a.x - o.x) -
          atan2(b.y - o.y, b.x - o.x)) * 180) / PI;
     //if (ang > 180.0)return(360.0 - ang);
     return ang; }
```

28. Semi perimeter

```
double s_perm(double a, double b, double c){
     return (a + b + c) / 2; }
```

29.

```
double tarea1(double h, double b){
     return h*b*0.5; }
```

30. Given length of all sides

```
double tarea2(double ab, double bc, double ca){
     double s = (ab + bc + ca) / 2;;
     return sqrt(s*(s-ab)*(s-bc)*(s-ca));
}
```

31.

```
double tarea3(point a , point b , point c ) {
      return fabs(.5 * ( a.x * ( b.y - c.y) + b.x * ( c.y -
      a.y) + c.x *( a.y - b.y))); }
```

Any triangle's area T can be expressed in terms of its medians m_a, m_b, and m_c as follows. Denoting their semi-sum (ma + mb + mc)/2 as σ, we have

$T = \frac{4}{3}\sqrt{\sigma(\sigma - m_a)(\sigma - m_b)(\sigma - m_c)}.$

5.9 Circle:

```
(x - h)^2 + (y - k)^2 = r^2
X^2 + y^2 + 2gx + 2fy + c = 0
```

h=-g, k=-f, c = h^2 + k^2 – r^2
h & k are the centre of the circle

Given 3 points(x1,y1; x2,y2; x3,y3)find circle’s radius/centre:

```
double x1,x2,x3,y1, y2, y3, a1, a2, b1, b2, c1, c2, d, h, k, r;
a1 = 2*(x2-x1); a2 = 2*(x3-x2);
b1 = 2*(y2-y1); b2 = 2*(y3-y2);
c1 = x1*x1+y1*y1-x2*x2-y2*y2;
c2 = x2*x2+y2*y2-x3*x3-y3*y3;
d = a1*b2-a2*b1;
```

```
h=(b1*c2-b2*c1)/d;
k=(a2*c1-a1*c2)/d;
r=sqrt((h-x1)*(h-x1)+(k-y1)*(k-y1));
```

32. Radius of Incircle = Area / semi-perimeter

```
double rInCircle(double ab, double bc, double ca) {
	return tarea2(ab, bc, ca) / s_perm(ab, bc, ca);
}
```

33.

```
double rInCircleP(point a, point b, point c) {
	return rInCircle(dist(a, b), dist(b, c), dist(c, a));
}
```

34. Radius of circumscribed circle = R = a × b × c/(4 × A).

```
double rCircumCircle(double ab, double bc, double ca) {
	return ab * bc * ca / (4.0 * tarea2(ab, bc, ca));
}
```

35.

```
double rCircumCircleP(point a, point b, point c) {
	return rCircumCircle(dist(a, b), dist(b, c),
		dist(c, a));
}
```

5.10 Monotone Chain Convex Hull + Points inside hull check

```
* Returns a list of convex hull in counter-clockwise order
* In case of collinear points, only end-points are taken
* The last point is the same as the first one
/// OK chkd by SunW
#define ll64 long long
struct point {
	ll64 x, y;
	bool operator <(const point &p) const {
		return (x < p.x || (x == p.x && y < p.y));
	}
};
vector<point> CH, P;
ll64 cross(point p1, point p2, point p) {
	return (p2.x - p1.x) * (p.y - p1.y) - (p2.y - p1.y) * (p.x -
p1.x);
}
void convexHull() {
	sort(P.begin(), P.end());
	int n = P.size(), k = 0;
	vector<point> H(2 * n);
	for (int i = 0; i < n; i++) {
		while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) <= 0)
			k--;
		H[k++] = P[i];
	}
```

```
	for (int i = n - 2, t = k + 1; i >= 0; i--) {
		while (k >= t && cross(H[k - 2], H[k - 1], P[i]) <= 0)
			k--;
		H[k++] = P[i];
	}
	H.resize(k);
	CH = H;
}
ll64 ar(point a, point b, point c) {
	return abs(a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y
- b.y));
}
bool insideCH(point p) {

	point pi = CH[0];

	int low = 1, up = CH.size() - 2, mid;
	while (up - low > 1) {
		mid = (low + up) >> 1;
		ll64 t = cross(pi, CH[mid], p);
		if (t < 0)
			up = mid;
		else
			low = mid;
	}
	if (ar(pi, CH[low], CH[up])== (ar(pi, p, CH[low]) + ar(pi, p,
		CH[up]) + ar(CH[low], p, CH[up]))) return true;
	return false;
}
int main() {
	//rd();
	int x, y, l, s;
	while (scanf("%d", &l) == 1) {
		P.clear();
		CH.clear();
		point pp;
		for (int i = 0; i < l; i++) {
			scanf("%d %d", &x, &y);
			pp.x = x;
			pp.y = y;
			P.push_back(pp);
		}
		convexHull();
		int ans = 0;
		scanf("%d", &s);
		for (int i = 0; i < s; i++) {
			scanf("%d %d", &x, &y);
			pp.x = x;
			pp.y = y;
```

```
                if (insideCH(pp)) {
                    ans++;
                }
            }
            printf("%d\n", ans);
        }
}
```

6. Graph

6.1 Topological sort

```
/// Topological Sort with cycle check: (inside main)
// if ans.size()!= n   TOPOSORT FAILS
//Memory efficient
int col[MAX], fl;
vector<int> G[MAX], ans;
void toposort(int u) {
    col[u] = 1;
    int i, y;
    //col, 0 = non visited, 1 = not finished, 2 = finished
    for (i = 0; i < G[u].size(); i++) {
        y = G[u][i];
        if (col[y] == 0)
            toposort(y);
        else if (col[y] == 1) {    //cycle
            fl = 1;
            return;
        }
    }
    if (fl == 1)
        return;
    col[u] = 2;
    ans.push_back(u);
// ans contains elements in reversed topo order
}

/// All possible Topo sort:
int vals[MAX], ANS[MAX], N;  //vals = input
bool taken[MAX];
vector<int> GE[MAX];
void toposort(int u, int in[]) {
    if (u == N) {
    }
// Possible sort found print, return

    int i, j, x;
    vector<int> v;
    int _in[MAX];
    for (i = 0; i < MAX; i++)
        _in[i] = in[i];
```

```
	for (i = 0; i < N; i++) {
		x = vals[i];
		if (in[x] == 0 && !taken[x])
			v.push_back(x);
	}
	for (i = 0; i < v.size(); i++) {
		x = v[i];
		taken[x] = true;
		ANS[u] = x;
		for (j = 0; j < GE[x].size(); j++) {
			if (_in[GE[x][j]] > 0)
				_in[GE[x][j]]--;
		}
		toposort(u + 1, _in);
		taken[v[i]] = false;
		for (j = 0; j <= N; j++)
			_in[j] = in[j];
	}
}
int main() {
	int n = 5, x, y;
	priority_queue<int, vector<int>, greater<int> > q;
	int *indeg = new int[n + 2] { };
	for (int i = 1; i <= n; i++)
		if (indeg[i] == 0)
			q.push(i);
	if (q.empty()) {
	}		// TopoSort Fail
	else {
		while (!q.empty()) {
			x = q.top();
			q.pop();
			ans.push_back(x);
			for (int i = 0; i < G[x].size(); i++) {
				y = G[x][i];
				if (indeg[y] == 0)
					break;
				//cycle found … break with flag
				else if (indeg[y] == 1) {
					indeg[y]--;
					q.push(y);
				} else
					indeg[y]--;
			}
		}
	}
}
```

6.2 Bipartite Graph

** Sometimes even if isBip is false, the while loop shouldn't break because all the vertices in that graph needs to be set as visited.
** Easier problems don't need number of colors

```
int colour[MAX]; bool visited[MAX]; vector <int> G[MAX];
bool bipartite(int &no, int u) {
	queue<int> q;
	q.push(u); colour[u] = 0;
	int x, i, tot = 0, zr = 1;
	bool isBip = true;
	visited[u] = true;
	while (!q.empty() /* && isBip */) {
		tot++;
		x = q.front(); q.pop();
		for (i = 0; i<G[x].size(); i++) {
			int y = G[x][i];
			if (!visited[y]) {
				colour[y] = 1 - colour[x]; q.push(y);
 visited[y] = true;
				if (colour[y] == 0)zr++;
			}
			else if (colour[y] == colour[x]) { isBip = false; }
		}
	}
if (!isBip) { no = 0; return false; }
	if (tot <= 2) { no = 1; return true; }
	else { no = min(zr, tot - zr); }  return true;
// min or max
}
```

6.3 Articulation point

```
/// OK
int low[MAX], num[MAX], parent[MAX], root, rootc, counter, n;
bool visited[MAX], artiVer[MAX];
vector<int> G[MAX];
void artculPoint(int u) {
	low[u] = num[u] = counter++;
	for (int i = 0; i < G[u].size(); i++) {
		int j = G[u][i];
		if (!visited[j]) {
			parent[j] = u;
			visited[j] = true;
			if (u == root)
				rootc++;
			artculPoint(j);
			if (low[j] >= num[u])
				if (!artiVer[u] && u != root)
					artiVer[u] = true;
```

```
                //if (low[j] > num[u]) { Bridge
                //    int k = u, l = j;
                //    if (k>l)swap(k, l);
                //    edges.push_back(pii(k, l));
                //}
                low[u] = min(low[u], low[j]);
          } else if (j != parent[u])
                low[u] = min(low[u], num[j]);
     }
}
** in main after calling articulPoint
if (rootc >= 2) artiVer[root] = true;
```

6.4 Strongly Connected Components (Tarjan's):

```
///NOT OK
vector<int> G[MAX], output[MAX];
int stck[MAX], num[MAX], component[MAX], low[MAX], onstack[MAX],
top;
int idx, components, N;
void tarjan(int u) {
     num[u] = low[u] = idx++;
     stack[top++] = u;
     onstack[u] = 1;
     for (int i = 0; i < G[u].size(); i++) {
          int v = G[u][i];
          if (num[v] == -1) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
          } else if (onstack[v])
                low[u] = min(low[u], num[v]);
     }
     int v;
     if (low[u] == num[u]) {
          do {
                v = stack[--top];
                onstack[v] = 0;
                component[v] = components;
                output[components].push_back(v);
// Storing SCCs in array of vector
          } while (u != v);
          components++;
     }
}
```

6.5 Prim's MST

```
struct node {
     int n, e;
     node() {
          n = 0;
          e = 0;
     }
```

```
	node(int _a, int _b) :
			n(_a), e(_b) {
	}
	bool operator<(const node &B) const {
		return e > B.e;
	}
};
vector<node> G[MAX];
bool visited[MAX];
int parent[MAX], C[MAX], root;

long long mst() {
	C[root] = 0;
	priority_queue<node> Q;
	Q.push(node(root, 0));
	long long cost = 0;
	while (!Q.empty()) {
		node u = Q.top();
		Q.pop();
		if (!visited[u.n]) {
			cost += u.e;
			visited[u.n] = true;
			for (int i = 0; i < G[u.n].size(); i++) {
				node v = G[u.n][i];
				if (!visited[v.n] && C[v.n] > v.e) {
					Q.push(v);
					C[v.n] = v.e;
					parent[v.n] = u.n;
				}
			}
		}
	}
	return cost;
}
```

6.6 Kruskal's MST

```
struct edl {
	int a, b;
	long long w;
	bool operator <(const edl &a) const {
		return w < a.w;
	}
} edlist[MAX * MAX];// Edgelist
int P[MAX];
int R[MAX], NE, NV; // R[]=Rank
void makeset(int n) {
	for (int i = 1; i <= n; i++) {
		P[i] = i;
	}
}
```

```
int findP(int n) {
	if (P[n] == n)
		return n;
	P[n] = findP(P[n]);
	return P[n];
}
void Union(int a, int b) {
	int x = findP(a);
	int y = findP(b);
	if (x == y)
		return;
	else {
		P[x] = y;
	}
}
long long kruskal( /* bool flg */) {
//Should kruskal add MST edges to mark[]?
	makeset(NE);
	sort(edlist, edlist + NE);
	int i, j, k = 0;
	long long cost = 0;
	edl top;
	for (i = 0; i < NE; i++) {
		top = edlist[i];
		if (findP(top.a) != findP(top.b)) {
			cost += top.w;
			k++;
			Union(top.a, top.b);
// if (flg)mark.push_back(i);
//Add MST edges to vector mark
			if (k == NV - 1)
				return cost;
		}
	}
// return -1;
}
```

2nd best MST(Kruskal, UVa 10600):

```
///
struct edl {
	int a, b;
	long long w;
	bool operator <(const edl &a) const {
		return w < a.w;
	}
} edlist[MAX * MAX], edlist2[MAX * MAX];
int P[MAX];
int R[MAX], NE, NV; // R[]=Rank
```

```
void makeset(int n) {
	for (int i = 1; i <= n; i++) {
		P[i] = i;
	}
}
int findP(int n) {
	if (P[n] == n)
		return n;
	P[n] = findP(P[n]);
	return P[n];
}
void Union(int a, int b) {
	int x = findP(a);
	int y = findP(b);
	if (x == y)
		return;
	else {
		P[x] = y;
	}
}
long long kruskal(bool flg) {
//Should kruskal add MST edges to mark[]?
	makeset(NE);
	sort(edlist, edlist + NE);
	int i, j, k = 0;
	long long cost = 0;
	edl top;
	for (i = 0; i < NE; i++) {
		top = edlist[i];
		if (findP(top.a) != findP(top.b)) {
			cost += top.w;
			k++;
			Union(top.a, top.b);
// if (flg)mark.push_back(i);
//Add MST edges to vector mark
			if (k == NV - 1)
				return cost;
		}
	}
// return -1;
}
vector<int> mark;
int main() {
	int n, i, j, k;
	cin >> n;
	while (cin >> NV >> NE) {
		edl a;
		k = 0;
		for (i = 0; i < NE; i++) {
```

```
            cin >> a.a >> a.b >> a.w;
            edlist[k++] = a;
        }
        cout << kruskal(true) << " ";
        int mi = 9999999;
        for (i = 0; i < NE; i++)
            edlist2[i] = edlist[i];
        for (i = 0; i < mark.size(); i++) {
            edlist[mark[i]].w = 100000;
            j = kruskal(false);
            if (j < mi)
                mi = j;
            for (j = 0; j < NE; j++)
                edlist[j] = edlist2[j];
        }
        cout << mi << "\n"; //Cost of 2nd best MST
        mark.clear();
    }
    }
```

6.7 Dijkstra – All shortest paths

```
#define MEM(p, v) memset(p, v, sizeof(p))
struct Node {
    int node, cost, par;
    Node() {
        node = 0;
        cost = 0;
        par = 0;
    }
    Node(int _a, int _b, int _c) :
            node(_a), cost(_b), par(_c) {
    }

    bool operator<(const Node &A) const {
        return cost > A.cost;
    }
};
int dist[MAX], visited[MAX], inf;
vector<Node> G[MAX];
vector<int> P[MAX];
vector<pair<int, int> > spedges;   //shortest path edges

void btrack(int n) {
    if (P[n].size() == 0 || visited[n])
        return;
    visited[n] = 1;
    for (int i = 0; i < P[n].size(); i++) {
        int pr = P[n][i];
        spedges.push_back(make_pair(n, pr));
        btrack(pr);}}
```

```
void dijkstra(int s, int d) {
	MEM(visited, 0);
	MEM(dist, 127);
	dist[s] = 0;
	Node nd(s, 0, -1);
	priority_queue<Node> Q;
	Q.push(nd);
	while (!Q.empty()) {
		int cn = Q.top().node;
		int dst = dist[cn];
		Node tp = Q.top();
		Q.pop();
		if (dst == tp.cost && cn != s) {
			P[cn].push_back(tp.par);
		}
		if (visited[cn])
			continue;
		for (int i = 0; i < G[cn].size(); i++) {
			int n = G[cn][i].node;
			int c = G[cn][i].cost;
			if (dst + c <= dist[n]) {
				dist[n] = dst + c;
				Q.push(Node(n, dist[n], cn));
			}
		}
		visited[cn] = 1;
	}
}
int main() {
	cpp_io();
	int n, e, a, b, c, s, d;
	cin >> n >> e;
	for (int i = 0; i<e; i++) {
		cin >> a >> b >> c;
		G[a].push_back(Node(b, c, -1));
		G[b].push_back(Node(a, c, -1));
	}
	cin >> s >> d;
	dijkstra(s, d);
	memset(visited, 0, sizeof(visited));
	btrack(d);
	for (int i = 0; i<spedges.size(); i++)
	cout << spedges[i].first << " " << spedges[i].second <<endl;
}
```

6.8 Bellman Ford

```
struct node {
	int n, e;
	node() {n = 0;e = 0;}
	node(int _a, int _b) :n(_a), e(_b) {}
};
int dist[MAX], NV;
vector<node> G[MAX];
void init(int n) {
	memset(dist, 127, sizeof(dist));
	for (int i = 0; i <= n; i++)
		G[i].clear();
}
bool bellman_ford(int s) {
	dist[s] = 0;
	for (int k = 0; k < NV; k++) {
		for (int i = 0; i < NV; i++) {
			for (int j = 0; j < G[i].size(); j++) {
				int n = G[i][j].n;
				int e = G[i][j].e;
				if (dist[i] + e < dist[n]) {
					dist[n] = dist[i] + e;
					if (k == NV - 1)
						return false;
				}}}}
	return true;
}
```

6.9 Dinic's Algorithm

```
const int MAXN = 1000;
struct edge {
	int a, b, cap, flow;
};

int n, s, t, d[MAXN], ptr[MAXN], q[MAXN];
vector<edge> e;
vector<int> g[MAXN];   // Residual Graph

void add_edge(int a, int b, int cap) {
	edge e1 = { a, b, cap, 0 };
	edge e2 = { b, a, 0, 0 };
	g[a].push_back((int) e.size());
	e.push_back(e1);
	g[b].push_back((int) e.size());
	e.push_back(e2);                }
```

```
bool bfs() {
	int qh = 0, qt = 0;
	q[qt++] = s;
	memset(d, -1, sizeof(d));
	d[s] = 0;
	while (qh < qt && d[t] == -1) {
		int v = q[qh++];
		for (size_t i = 0; i < g[v].size(); ++i) {
			int id = g[v][i], to = e[id].b;
			if (d[to] == -1 && e[id].flow < e[id].cap) {
				q[qt++] = to;
				d[to] = d[v] + 1;
			}
		}
	}
	return d[t] != -1;
}
int dfs(int v, int flow) {
	if (!flow)
		return 0;
	if (v == t)
		return flow;
	for (; ptr[v] < (int) g[v].size(); ++ptr[v]) {
		int id = g[v][ptr[v]], to = e[id].b;
		if (d[to] != d[v] + 1)
			continue;
		int pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
		if (pushed) {
			e[id].flow += pushed;
			e[id ^ 1].flow -= pushed;
			return pushed;
		}
	}
	return 0;
}
int dinic() {
	int flow = 0;
	for (;;) {
		if (!bfs())
			break;
		memset(ptr, 0, sizeof(ptr));
		while (int pushed = dfs(s, INF))
			flow += pushed;
	}
	return flow;
}
```

6.10 Bipartite Matching

```
///
//Graph G strats from index 0
vector<int> G[MAX];
bool visited[MAX];
int Left[MAX], Right[MAX];

bool dfs(int u) {
	if (visited[u])
		return false;
	visited[u] = true;
	int len = G[u].size(), i, v;
	for (i = 0; i < len; i++) {
		v = G[u][i];
		if (Right[v] == -1) {
			Right[v] = u, Left[u] = v;
			return true;
		}
	}
	for (i = 0; i < len; i++) {
		v = G[u][i];
		if (dfs(Right[v])) {
			Right[v] = u, Left[u] = v;
			return true;
		}
	}
	return false;
}
int match(int n) {		// n = size of G
	int i, ret = 0;
	bool done;
	memset(Left, -1, sizeof Left);
	memset(Right, -1, sizeof Right);
	do {
		done = true;
		memset(visited, 0, sizeof visited);
		for (i = 0; i < n; i++) {
			if (Left[i] == -1 && dfs(i)) {
				done = false;
			}
		}
	} while (!done);
	for (i = 0; i < n; i++)
		ret += (Left[i] != -1);
	return ret;}
```

7. String

/// OK chkd by SunW

7.1 Hasing

http://e-maxx-eng.appspot.com/string/string-hashing.html

```
typedef unsigned long long ull;
const int P = 31, SIZE = 100000;
// P = 53 if for both uppercase & lowercase
vector <ull> p_pow(SIZE);
vector <ull> h(SIZE);       //hash of prefixes

void init_pow() {
     p_pow[0] = 1;
     for (int i = 1; i < SIZE; i++)
          p_pow[i] = p_pow[i - 1] * P;
}
void hash_pre(string s) { //hash of prefixes
     for (int i = 0; i < s.length(); i++){
          h[i] = (s[i] - 'a' + 1) * p_pow[i];
          if (i) h[i] += h[i - 1];
     }
}
ull hash_s(string s) { //hash of a string
     ull h_s = 0;
     for (int i = 0; i < s.length(); i++)
          h_s += (s[i] - 'a' + 1) * p_pow[i];
     return h_s;
}
bool comp_substr(int l1, int l2, int r1, int r2) {
     //compares hashes of 2 substring in a string(0 based)

     ull h1 = h[r1];
     if (l1) h1 -= h[l1 - 1];
     ull h2 = h[r2];
     if (l2) h2 -= h[l2 - 1];

     // Get the two hashes multiplied by the same power of P
     // and then compare them
     if (l1 <l2 && h1 * p_pow[l2 - l1] == h2 ||
          l1> l2 && h1 == h2 * p_pow[l1 - l2])
          return true;      // substrings are equal
     else
          return false;
}
```

```
int count_substr(int n) {
     // Count number of differetn substrings in a string
     // n = length of string

     int result = 0;
     for (int l = 1; l <= n; l++) {
          // Need to find the number of distinct substrings of
          // length l. Get the hashes for all substrs of length
     l

          vector <ull> hs(n - l + 1);
          for (int i = 0; i <n - l + 1; i++){
               ull cur_h = h[i + l - 1];
               if (i) cur_h -= h[i - 1];
               cur_h *= p_pow[n - i - 1];
               hs[i] = cur_h;
          }
          sort(hs.begin(), hs.end());
          hs.erase(unique(hs.begin(), hs.end()), hs.end());
          result += (int)hs.size();
     }
     return result;
}
```

7.2 Rabin – Karp

```
/// OK chkd by SunW
vector<int> rabin_karp (string s, string t) {
     // s = pattern, t = text
     vector<int> res; //returns 0 based starting indexs of
matches

     init_pow();
     ull h_s = hash_s(s);
     hash_pre(t);

     for (int i = 0; i + s.length() - 1 < t.length(); i++){
          ull cur_h = h[i + s.length() - 1];
          if (i) cur_h -= h[i - 1];

          if (cur_h == h_s * p_pow[i])
               res.push_back(i);
     }
     return res;
}
```