|  |  |
| --- | --- |
| **Basic**  #include <bits/stdc++.h>  using namespace std;  using ll = long long;  const int mx\_sz  = (int) 2e6+3;  void idea() {    }  int main() {      ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);      int T = 1;      // cin >> T;      for(int C = 1; C <= T; C++) {          // cout << "Case " << C << ": " << '\n';          idea();      }      return 0;  }  **Elementary Things**  // freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);  #define PI 3.14159265358979323846  #define toLowerCase(s) transform(s.begin(), s.end(), s.begin(), ::tolower);  #define toUpperCase(s) transform(s.begin(), s.end(), s.begin(), ::toupper);  int dx[] = {+1, -1, 0, 0, +1, +1, -1, -1};  int dy[] = {0, 0, -1, +1, +1, -1, +1, -1};  bool check\_power\_of\_two(ll n){ return !(n & (n - 1)); }  bool check\_perfect\_square(ll n){ if (n < 0) return false; ll root = sqrt(n); return (root \* root == n); }  bool check\_fibonacci(int n) { return check\_perfect\_square(5\*n\*n + 4) or check\_perfect\_square(5\*n\*n - 4); }  bool check\_parity(ll n) { return \_\_builtin\_parityll(n); } // returns 1 if the number has odd parity  **Bit Manipulation**  int check\_kth\_bit\_on\_or\_off(int x, int k) {      return (x >> k) & 1;  }  int turn\_on\_kth\_bit(int x, int k) {      return (x | (1 << k));  }  int turn\_off\_kth\_bit(int x, int k) {      return (x & (~(1 << k)));  }  int toggle\_kth\_bit(int x, int k) {      return (x ^ (1 << k));  }  void print\_on\_and\_off\_bits(int x) {      for (int k = 0; k <= 31; k++) {          if (check\_kth\_bit\_on\_or\_off(x, k)) {              cout << 1 << " ";          }          else {              cout << 0 << " ";          }      }      cout << '\n';  }  void grey\_code\_sequence() {      for (int i = 0; i < (1 << n); i++) {          for (int k = 0; k < n; k++) {              if ((i >> k) & 1) cout << 1 << ' ';              else cout << 0 << ' ';          }          cout << '\n';      }  }  **Standard Sieve: Sieve of Eratosthenes**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e8 + 3;  vector<bool> is\_prime(N + 1, true);  vector<long long> saved\_primes;  void standard\_sieve() {  is\_prime[0] = is\_prime[1] = false;  for (int i = 3; i \* i < N; i += 2)  if (is\_prime[i])  for (int j = i \* i; j < N; j += i + i)  is\_prime[j] = false;  saved\_primes.push\_back(2);  for (int i = 3; i < N; i += 2)  if (is\_prime[i]) saved\_primes.push\_back(i);  }  int main() {  standard\_sieve();  cout << saved\_primes.size() << '\n' << saved\_primes.back() << '\n';  return 0;  }  **Linear Sieve**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e8 + 3;  vector<int> spf(N + 1, 0);  vector<long long> saved\_primes;  void linear\_sieve() {  for (int i = 2; i <= N; i += 2) {  if (spf[i] == 0) {  spf[i] = 2;  if (i == 2) saved\_primes.push\_back(2);  }  }  for (int i = 3; i <= N; i += 2) {  if (spf[i] == 0) {  spf[i] = i;  saved\_primes.push\_back(i);  }  for (int j = 0; j < saved\_primes.size() && saved\_primes[j] <= spf[i] && i \* saved\_primes[j] <= N; j++)  spf[i \* saved\_primes[j]] = saved\_primes[j];  }  }  int main() {  linear\_sieve();  cout << saved\_primes.size() << '\n' << saved\_primes.back() << '\n';  return 0;  }  **Segmented Sieve**  #include <bits/stdc++.h>  using namespace std;  #define MAXSIEVE 100000001  #define MAXSIEVEHALF (MAXSIEVE >> 1)  #define MAXSQRT 5000  #define isprime(n) ((is\_prime[n >> 4] & (1 << ((n >> 1) & 7))) && ((n & 1) || (n == 2)))  char is\_prime[MAXSIEVE / 16 + 2];  vector<int> Yarin\_primes;  void Yarin() {  memset(is\_prime, (1 << 8) - 1, sizeof(is\_prime));  is\_prime[0] = 0xFE;  for (int i = 1; i < MAXSQRT; i++) if (is\_prime[i >> 3] & (1 << (i & 7)))  for (int j = 2 \* i \* (i + 1); j < MAXSIEVEHALF; j += (i << 1) + 1)  is\_prime[j >> 3] &= ~(1 << (j & 7));  }  void nPrime() {  for (int i = 2; i < MAXSIEVE; i++) if (isprime(i)) Yarin\_primes.push\_back(i);  }  int main() {  Yarin(); nPrime();  cout << "Number of primes found: " << Yarin\_primes.size() << '\n';  if (!Yarin\_primes.empty()) cout << "Last prime: " << Yarin\_primes.back() << '\n';  for (int prime : Yarin\_primes) cout << prime << " ";  cout << '\n';  return 0;  }  **All divisor of a number**  vector<long long> all\_divisors(long long n) {  vector<long long> divisor;  for (long long i = 1; i \* i <= n; i++) {  if (n % i == 0) {  divisor.push\_back(i);  if (i \* i != n) divisor.push\_back(n / i);  }  }  return divisor;  }  **Divisor List and Divisor Count of a number**  const int MAX\_LIMIT = 1e7 + 3;  // Store lists of divisors for all numbers from 1 to MAX\_LIMIT  vector<vector<int>> divisorLists(MAX\_LIMIT);  // Count of divisors for all numbers from 1 to MAX\_LIMIT  vector<int> divisorCounts(MAX\_LIMIT, 0);  void computeDivisors() {      for (int num = 1; num < MAX\_LIMIT; num++) {          for (int multiple = num; multiple < MAX\_LIMIT; multiple += num){              divisorLists[multiple].push\_back(num);              divisorCounts[multiple]++;          }      } // (O(N log N))  }  **Single Query Prime Factors**  #include <bits/stdc++.h>  using namespace std;  vector<unsigned long long> Factorization(unsigned long long n) {  vector<unsigned long long> Factors;  for (unsigned long long ii = 2; ii \* ii <= n; ii++) {  if (n % ii == 0) Factors.push\_back(ii);  while (n % ii == 0) n /= ii;  }  if (n > 1) Factors.push\_back(n);  return Factors;  }  void idea() {  unsigned long long n = (1ULL << 63) - 1 + (1ULL << 63);  for (auto i : Factorization(n)) cout << i << ' ';  cout << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  for (int T = 1; T <= 1; T++) idea();  return 0;  }  **Multiple Query Prime Factors**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e8 + 3;  vector<bool> is\_prime(N + 1, true);  vector<long long> saved\_primes;  void standard\_sieve() { // TC: O(N log log N)  is\_prime[0] = is\_prime[1] = false;  for (int i = 3; i \* i < N; i += 2) if (is\_prime[i])  for (int j = i \* i; j < N; j += i + i) is\_prime[j] = false;  saved\_primes.push\_back(2);  for (int i = 3; i < N; i += 2) if (is\_prime[i]) saved\_primes.push\_back(i);  }  vector<unsigned long long> Prime\_Factorization(unsigned long long n) {  vector<unsigned long long> prime\_factors;  for (size\_t i = 0; i < saved\_primes.size() && saved\_primes[i] \* saved\_primes[i] <= n; i++) {  if (n % saved\_primes[i] == 0) {  prime\_factors.push\_back(saved\_primes[i]);  while (n % saved\_primes[i] == 0) n /= saved\_primes[i];  }  }  if (n > 1) prime\_factors.push\_back(n);  return prime\_factors;  }  int main() {  standard\_sieve();  cout << "Number of primes found: " << saved\_primes.size() << '\n'  << "Last prime: " << saved\_primes.back() << '\n';    unsigned long long n = (1ULL << 63) - 1 + (1ULL << 63);  vector<unsigned long long> factors = Prime\_Factorization(n);    cout << "Prime factors of " << n << ": ";  for (auto factor : factors) cout << factor << ' ';  cout << '\n';    return 0;  }  **Modular Arithmatic**  ll Modular\_Exponentiation(ll base, ll exp, ll mod) {      ll res = 1LL;      base %= mod;      while (exp) {          if (exp % 2) res = res \* base % mod;          base = base \* base % mod;          exp /= 2;      }      return res;  }  ll Modular\_Addition(ll x, ll y, ll mod) {      return ((x % mod + y % mod) % mod + mod) % mod;  }  ll Modular\_Subtraction(ll x, ll y, ll mod) {      return ((x % mod - y % mod) % mod + mod) % mod;  }  ll Modular\_Multiplication(ll x, ll y, ll mod) {      return ((x % mod \* y % mod) % mod + mod) % mod;  }  ll Modular\_Inverse(ll x, ll mod) {      return Modular\_Exponentiation(x, mod - 2, mod);  }  **Combinatorics**  // nPr % MOD calculation  ll nPr(ll n, ll r, ll mod) { // O(log(MOD))      if (r > n) return -1;      ll numerator = fact[n] % mod;      ll denominator = fact[n - r] % mod;      numerator = (numerator \* Modular\_Exponentiation(denominator, mod - 2, mod)) % mod;      return numerator;  }  // nCr % MOD calculation  ll nCr(ll n, ll r, ll mod) { // O(log(MOD))      if (r == 0) return 1;      if (r > n) return -1;      ll numerator = fact[n] % mod;      ll denominator = (fact[n - r] \* fact[r]) % mod;      numerator = (numerator \* Modular\_Exponentiation(denominator, mod - 2, mod)) % mod;      return numerator;  }  // Precompute factorials up to n % mod  void cal\_fact(ll n, ll mod) {      fact.resize(n + 1);      fact[0] = 1;      for (ll i = 1; i <= n; i++) {          fact[i] = (fact[i - 1] \* i) % mod;      }  }  **BigInteger**  #include <bits/stdc++.h>  #define debug(x) cout << #x << " = "; cout << x << '\n';  using namespace std;  typedef long long ll;  const int ARRAY\_SIZE = (int)2e6 + 3;  // BigInt class for large integer support  class BigInt{  string digits;  public:  // Constructors  BigInt(unsigned long long n = 0);  BigInt(string &);  BigInt(const char \*);  BigInt(BigInt &);  BigInt(const BigInt &);    // Helper Functions  friend void divide\_by\_2(BigInt &a);  friend bool Null(const BigInt &);  friend int Length(const BigInt &);  int operator[](const int) const;  // Operators  BigInt &operator=(const BigInt &);  BigInt &operator++();  BigInt operator++(int temp);  BigInt &operator--();  BigInt operator--(int temp);  friend BigInt &operator+=(BigInt &, const BigInt &);  friend BigInt operator+(const BigInt &, const BigInt &);  friend BigInt operator-(const BigInt &, const BigInt &);  friend BigInt &operator-=(BigInt &, const BigInt &);  friend bool operator==(const BigInt &, const BigInt &);  friend bool operator!=(const BigInt &, const BigInt &);  friend bool operator>(const BigInt &, const BigInt &);  friend bool operator>=(const BigInt &, const BigInt &);  friend bool operator<(const BigInt &, const BigInt &);  friend bool operator<=(const BigInt &, const BigInt &);  friend BigInt &operator\*=(BigInt &, const BigInt &);  friend BigInt operator\*(const BigInt &, const BigInt &);  friend BigInt &operator/=(BigInt &, const BigInt &);  friend BigInt operator/(const BigInt &, const BigInt &);  friend BigInt operator%(const BigInt &, const BigInt &);  friend BigInt &operator%=(BigInt &, const BigInt &);  friend BigInt &operator^=(BigInt &, const BigInt &);  friend BigInt operator^(BigInt &, const BigInt &);  // Additional Functions  friend BigInt sqrt(BigInt &a);  friend BigInt NthCatalan(int n);  friend BigInt NthFibonacci(int n);  friend BigInt Factorial(int n);  // I/O  friend ostream &operator<<(ostream &, const BigInt &);  friend istream &operator>>(istream &, BigInt &);  };  // Constructor: BigInt from string  BigInt::BigInt(string &s) {  digits = ""; int n = s.size();  for (int i = n - 1; i >= 0; i--) {  if (!isdigit(s[i])) throw("ERROR");  digits.push\_back(s[i] - '0');  }  }  // Constructor: BigInt from unsigned long long  BigInt::BigInt(unsigned long long nr) {  do { digits.push\_back(nr % 10); nr /= 10; } while (nr);  }  // Constructor: BigInt from char\*  BigInt::BigInt(const char \*s) {  digits = "";  for (int i = strlen(s) - 1; i >= 0; i--) {  if (!isdigit(s[i])) throw("ERROR");  digits.push\_back(s[i] - '0');  }  }  // Copy constructor  BigInt::BigInt(BigInt &a) { digits = a.digits; }  BigInt::BigInt(const BigInt &a) { digits = a.digits; }  // Helper Functions  bool Null(const BigInt &a) {  return (a.digits.size() == 1 && a.digits[0] == 0);  }  int Length(const BigInt &a) { return a.digits.size(); }  int BigInt::operator[](const int index) const {  if (digits.size() <= index || index < 0) throw("ERROR");  return digits[index];  }  // Comparison operators  bool operator==(const BigInt &a, const BigInt &b) { return a.digits == b.digits; }  bool operator!=(const BigInt &a, const BigInt &b) { return !(a == b); }  bool operator<(const BigInt &a, const BigInt &b) {  int n = Length(a), m = Length(b);  if (n != m) return n < m;  while (n--) if (a.digits[n] != b.digits[n]) return a.digits[n] < b.digits[n];  return false;  }  bool operator>(const BigInt &a, const BigInt &b) { return b < a; }  bool operator>=(const BigInt &a, const BigInt &b) { return !(a < b); }  bool operator<=(const BigInt &a, const BigInt &b) { return !(a > b); }  // Assignment operator  BigInt &BigInt::operator=(const BigInt &a) { digits = a.digits; return \*this; }  // Increment/Decrement  BigInt &BigInt::operator++() {  int i, n = digits.size();  for (i = 0; i < n && digits[i] == 9; i++) digits[i] = 0;  if (i == n) digits.push\_back(1);  else digits[i]++;  return \*this;  }  BigInt BigInt::operator++(int temp) { BigInt aux; aux = \*this; ++(\*this); return aux; }  BigInt &BigInt::operator--() {  if (digits[0] == 0 && digits.size() == 1) throw("UNDERFLOW");  int i, n = digits.size();  for (i = 0; digits[i] == 0 && i < n; i++) digits[i] = 9;  digits[i]--;  if (n > 1 && digits[n - 1] == 0) digits.pop\_back();  return \*this;  }  BigInt BigInt::operator--(int temp) { BigInt aux; aux = \*this; --(\*this); return aux; }  // Addition and Subtraction  BigInt &operator+=(BigInt &a, const BigInt &b) {  int t = 0, s, i;  int n = Length(a), m = Length(b);  if (m > n) a.digits.append(m - n, 0);  n = Length(a);  for (i = 0; i < n; i++) {  s = (i < m ? (a.digits[i] + b.digits[i]) : a.digits[i]) + t;  t = s / 10;  a.digits[i] = s % 10;  }  if (t) a.digits.push\_back(t);  return a;  }  BigInt operator+(const BigInt &a, const BigInt &b) { BigInt temp; temp = a; temp += b; return temp; }  BigInt &operator-=(BigInt &a, const BigInt &b) {  if (a < b) throw("UNDERFLOW");  int n = Length(a), m = Length(b), t = 0, s, i;  for (i = 0; i < n; i++) {  s = a.digits[i] - (i < m ? b.digits[i] : 0) + t;  if (s < 0) s += 10, t = -1;  else t = 0;  a.digits[i] = s;  }  while (n > 1 && a.digits[n - 1] == 0) a.digits.pop\_back(), n--;  return a;  }  BigInt operator-(const BigInt &a, const BigInt &b) { BigInt temp; temp = a; temp -= b; return temp; }  // Multiplication  BigInt &operator\*=(BigInt &a, const BigInt &b) {  if (Null(a) || Null(b)) { a = BigInt(); return a; }  int n = a.digits.size(), m = b.digits.size();  vector<int> v(n + m, 0);  for (int i = 0; i < n; i++)  for (int j = 0; j < m; j++) v[i + j] += (a.digits[i]) \* (b.digits[j]);  n += m;  a.digits.resize(v.size());  for (int s, i = 0, t = 0; i < n; i++) {  s = t + v[i]; v[i] = s % 10; t = s / 10; a.digits[i] = v[i];  }  for (int i = n - 1; i >= 1 && !v[i]; i--) a.digits.pop\_back();  return a;  }  BigInt operator\*(const BigInt &a, const BigInt &b) { BigInt temp; temp = a; temp \*= b; return temp; }  // Division and Modulo  BigInt &operator/=(BigInt &a, const BigInt &b) {  if (Null(b)) throw("Arithmetic Error: Division By 0");  if (a < b) { a = BigInt(); return a; }  if (a == b) { a = BigInt(1); return a; }  int i, lgcat = 0, cc;  int n = Length(a), m = Length(b);  vector<int> cat(n, 0);  BigInt t;  for (i = n - 1; t \* 10 + a.digits[i] < b; i--) { t \*= 10; t += a.digits[i]; }  for (; i >= 0; i--) {  t = t \* 10 + a.digits[i];  for (cc = 9; cc \* b > t; cc--);  t -= cc \* b;  cat[lgcat++] = cc;  }  a.digits.resize(cat.size());  for (i = 0; i < lgcat; i++) a.digits[i] = cat[lgcat - i - 1];  a.digits.resize(lgcat);  return a;  }  BigInt operator/(const BigInt &a, const BigInt &b) { BigInt temp; temp = a; temp /= b; return temp; }  BigInt &operator%=(BigInt &a, const BigInt &b) {  if (Null(b)) throw("Arithmetic Error: Division By 0");  if (a < b) { return a; }  if (a == b) { a = BigInt(); return a; }  int i, lgcat = 0, cc;  int n = Length(a), m = Length(b);  vector<int> cat(n, 0);  BigInt t;  for (i = n - 1; t \* 10 + a.digits[i] < b; i--) { t \*= 10; t += a.digits[i]; }  for (; i >= 0; i--) {  t = t \* 10 + a.digits[i];  for (cc = 9; cc \* b > t; cc--);  t -= cc \* b;  cat[lgcat++] = cc;  }  a = t; return a;  }  BigInt operator%(const BigInt &a, const BigInt &b) { BigInt temp; temp = a; temp %= b; return temp; }  // Power operator  BigInt &operator^=(BigInt &a, const BigInt &b) {  BigInt Exponent, Base(a); Exponent = b; a = 1;  while (!Null(Exponent)) {  if (Exponent[0] & 1) a \*= Base;  Base \*= Base;  divide\_by\_2(Exponent);  }  return a;  }  BigInt operator^(BigInt &a, BigInt &b) { BigInt temp(a); temp ^= b; return temp; }  // Helper function for dividing BigInt by 2  void divide\_by\_2(BigInt &a) {  int add = 0;  for (int i = a.digits.size() - 1; i >= 0; i--) {  int digit = (a.digits[i] >> 1) + add;  add = ((a.digits[i] & 1) \* 5);  a.digits[i] = digit;  }  while (a.digits.size() > 1 && !a.digits.back()) a.digits.pop\_back();  }  // Square root function for BigInt  BigInt sqrt(BigInt &a) {  BigInt left(1), right(a), v(1), mid, prod; divide\_by\_2(right);  while (left <= right) {  mid += left; mid += right; divide\_by\_2(mid); prod = (mid \* mid);  if (prod <= a) { v = mid; ++mid; left = mid; }  else { --mid; right = mid; }  mid = BigInt();  }  return v;  }  // Catalan number  BigInt NthCatalan(int n) {  BigInt a(1), b;  for (int i = 2; i <= n; i++) a \*= i;  b = a;  for (int i = n + 1; i <= 2 \* n; i++) b \*= i;  a \*= a; a \*= (n + 1); b /= a;  return b;  }  // Fibonacci sequence  BigInt NthFibonacci(int n) {  BigInt a(1), b(1), c;  if (!n) return c;  n--;  while (n--) { c = a + b; b = a; a = c; }  return b;  }  // Factorial of n  BigInt Factorial(int n) {  BigInt f(1);  for (int i = 2; i <= n; i++) f \*= i;  return f;  }  // Input stream for BigInt  istream &operator>>(istream &in, BigInt &a) {  string s; in >> s; a.digits.clear();  for (int i = s.size() - 1; i >= 0; i--) {  if (!isdigit(s[i])) throw("INVALID NUMBER");  a.digits.push\_back(s[i] - '0');  }  return in;  }  // Output stream for BigInt  ostream &operator<<(ostream &out, const BigInt &a) {  for (int i = a.digits.size() - 1; i >= 0; i--) out << (short)a.digits[i];  return out;  }  // Main function with test cases  void idea() {  // take input  BigInt first\_num, Second\_num;  cin >> first\_num >> Second\_num;    // check equality  if (first\_num == Second\_num) cout << "Equal" << '\n';  else cout << "Not Equal" << '\n';    // comparison  if (first\_num > Second\_num) cout << "Greater" << '\n';  else cout << "Smaller" << '\n';    // printing  cout << first\_num << ' ' << Second\_num << '\n';    // vector input  vector <BigInt> vec = {first\_num, Second\_num};  for (auto val : vec) { cout << val << ' '; cout << '\n'; }    BigInt Fib = NthFibonacci(6); // 6th Fibonacci is 8  BigInt Cat = NthCatalan(10); // 10th Catalan is 16796  BigInt Fact = Factorial(5); // Factorial of 5 is 120  cout << Fib << ' ' << Cat << ' ' << Fact << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  int T = 1;  // cin >> T;  for (int C = 1; C <= T; C++) {  // cout << "Case " << C << ": " << '\n';  idea();  }  return 0;  }  **Binary Search**  #include <bits/stdc++.h>  using namespace std;  int main() {  ios::sync\_with\_stdio(false); cin.tie(NULL);  int n, key; cin >> n >> key;  vector<int> a(n);  for (int i = 0; i < n; i++) cin >> a[i];  int l = 0, r = n - 1, idx = -1;  while (l <= r) {  int mid = l + (r - l) / 2;  if (a[mid] == key) { idx = mid; break; }  if (a[mid] < key) l = mid + 1;  else r = mid - 1;  }  cout << (idx == -1 ? "Element not found" : "Element found at index " + to\_string(idx)) << '\n';  return 0;  }  **Maximize the Median**  #include <bits/stdc++.h>  using namespace std;  int main() {  ios::sync\_with\_stdio(false); cin.tie(NULL);  int n, k; cin >> n >> k;  vector<int> a(n);  for (int &x : a) cin >> x;  sort(a.begin(), a.end());  auto ok = [&](long long mid) {  long long cnt = 0;  for (int i = n / 2; i < n; i++) {  cnt += max(0LL, mid - a[i]);  }  return cnt <= k;  };  long long l = 1, r = 2e9, ans = 0;  while (l <= r) {  long long mid = l + (r - l) / 2;  if (ok(mid)) {  ans = mid;  l = mid + 1;  } else {  r = mid - 1;  }  }  cout << ans << '\n';  return 0;  }  **Policy Based Data Structure**  // count\_elements\_less\_or\_equal  #include <bits/stdc++.h>  #include <ext/pb\_ds/assoc\_container.hpp>  #include <ext/pb\_ds/tree\_policy.hpp>  using namespace std;  using namespace \_\_gnu\_pbds;  template <typename T> using pbds = tree<T, null\_type, less\_equal<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;  int main() {      ios::sync\_with\_stdio(false); cin.tie(NULL);      int n, q;      cin >> n >> m;      pbds<int> p;      for (int i = 1; i <= n; i++) {          int x;          cin >> x;          p.insert(x);      }      for (int i = 1; i <= q; i++) {          int x;          cin >> x;          cout << p.order\_of\_key(x + 1) << " ";      }      cout << '\n';      return 0;  }  **Sliding Window Median**  #include <bits/stdc++.h>  #include <ext/pb\_ds/assoc\_container.hpp>  #include <ext/pb\_ds/tree\_policy.hpp>  using namespace \_\_gnu\_pbds;  using namespace std;  template <typename T>  using pbds = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;  int main() {  ios::sync\_with\_stdio(false); cin.tie(NULL);  int n, k;  cin >> n >> k;  vector<int> a(n);  for (int &x : a) cin >> x;  int l = 0;  pbds<pair<int, int>> p;  for (int r = 0; r < n; r++) {  p.insert({a[r], r});  if (r - l + 1 == k) {  int pos = (k - 1) / 2;  auto it = p.find\_by\_order(pos);  cout << it->first << " ";  p.erase({a[l], l});  l++;  }  }  cout << '\n';  return 0;  }  **Segment Tree**  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const ll N = 2e5 + 5;  ll arr[N];  // Input array  ll segTree[4 \* N];  // Segment Tree  ll lazy[4 \* N];  // Lazy propagation array  // Propagate the pending updates to child nodes  void propagate(int node, int start, int end) {      if (lazy[node] != 0) {          segTree[node] += lazy[node] \* (end - start + 1);  // Apply the pending update to this node          if (start != end) {  // Not a leaf node              lazy[2 \* node] += lazy[node];  // Mark left child for lazy propagation              lazy[2 \* node + 1] += lazy[node];  // Mark right child for lazy propagation          }          lazy[node] = 0;  // Clear the lazy value      }  }  // Build the segment tree  void build(int node, int start, int end) {      if (start == end) {          segTree[node] = arr[start];  // Leaf node stores the actual value      } else {          int mid = (start + end) / 2;          build(2 \* node, start, mid);  // Left child          build(2 \* node + 1, mid + 1, end);  // Right child          segTree[node] = segTree[2 \* node] + segTree[2 \* node + 1];  // Merge the results      }  }  // Range query: get the sum of elements in the range [L, R]  ll query(int node, int start, int end, int L, int R) {      propagate(node, start, end);  // Ensure any pending updates are applied      if (start > R || end < L) {  // No overlap          return 0;  // For sum queries, return 0 for no overlap      }      if (start >= L && end <= R) {  // Total overlap          return segTree[node];      }      // Partial overlap      int mid = (start + end) / 2;      ll leftQuery = query(2 \* node, start, mid, L, R);      ll rightQuery = query(2 \* node + 1, mid + 1, end, L, R);      return leftQuery + rightQuery;  // Merge the results  }  // Point update: update the value at index 'idx' by 'val'  void update(int node, int start, int end, int idx, ll val) {      propagate(node, start, end);  // Ensure any pending updates are applied      if (start == end) {          segTree[node] += val;  // Point update      } else {          int mid = (start + end) / 2;          if (idx <= mid) {              update(2 \* node, start, mid, idx, val);  // Update left child          } else {              update(2 \* node + 1, mid + 1, end, idx, val);  // Update right child          }          segTree[node] = segTree[2 \* node] + segTree[2 \* node + 1];  // Recalculate the sum for this node      }  }  // Range update: add 'val' to all elements in the range [L, R]  void rangeUpdate(int node, int start, int end, int L, int R, ll val) {      propagate(node, start, end);  // Ensure any pending updates are applied      if (start > R || end < L) {  // No overlap          return;      }      if (start >= L && end <= R) {  // Total overlap          segTree[node] += val \* (end - start + 1);  // Apply the update          if (start != end) {  // Not a leaf node              lazy[2 \* node] += val;  // Mark left child for lazy propagation              lazy[2 \* node + 1] += val;  // Mark right child for lazy propagation          }          return;      }      // Partial overlap      int mid = (start + end) / 2;      rangeUpdate(2 \* node, start, mid, L, R, val);      rangeUpdate(2 \* node + 1, mid + 1, end, L, R, val);      segTree[node] = segTree[2 \* node] + segTree[2 \* node + 1];  // Recalculate the sum for this node  }  void solve() {      ll n, q;      cin >> n;      for (ll i = 1; i <= n; i++) {          cin >> arr[i];      }      build(1, 1, n);  // Build the segment tree      cin >> q;      while (q--) {          int type;          cin >> type;          if (type == 1) {  // Query operation: Sum in range [L, R]              int L, R; cin >> L >> R;              cout << query(1, 1, n, L, R) << '\n';          } else if (type == 2) {  // Point update: Update arr[idx] by value              int idx, val; cin >> idx >> val;              update(1, 1, n, idx, val);          } else if (type == 3) {  // Range update: Add value to range [L, R]              int L, R, val; cin >> L >> R >> val;              rangeUpdate(1, 1, n, L, R, val);          }      }  }  int main() {      ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);      solve(); return 0;  }  **Debugger**  #include <bits/stdc++.h>  using namespace std;  #define bug(x) debug(x, #x)  // Pair Input  template <typename X, typename Y>  istream &operator>>(istream &cin, pair<X, Y> &a) {  return cin >> a.first >> a.second;  }  // Pair Output  template <typename X, typename Y>  ostream &operator<<(ostream &cout, const pair<X, Y> &a) {  return cout << a.first << ' ' << a.second;  }  // Vector of Pairs Input  template <typename X, typename Y>  istream &operator>>(istream &cin, vector<pair<X, Y>> &vec) {  for (auto &x : vec) cin >> x; return cin;  }  // Vector of Pairs Output  template <typename X, typename Y>  ostream &operator<<(ostream &cout, const vector<pair<X, Y>> &vec) {  for (const auto &x : vec) cout << x << '\n';  return cout;  }  // Tuple Input  template <typename X, typename Y, typename Z>  istream &operator>>(istream &cin, tuple<X, Y, Z> &a) {  return cin >> get<0>(a) >> get<1>(a) >> get<2>(a);  }  // Tuple Output  template <typename X, typename Y, typename Z>  ostream &operator<<(ostream &cout, const tuple<X, Y, Z> &a) {  return cout << '(' << get<0>(a) << ", " << get<1>(a) << ", " << get<2>(a) << ')';  }  // Vector of Tuples Input  template <typename X, typename Y, typename Z>  istream &operator>>(istream &cin, vector<tuple<X, Y, Z>> &vec) {  for (auto &t : vec) cin >> t; return cin;  }  // Vector of Tuples Output  template <typename X, typename Y, typename Z>  ostream &operator<<(ostream &cout, const vector<tuple<X, Y, Z>> &vec) {  for (const auto &t : vec) cout << t << '\n'; return cout;  }  // Vector Input  template <typename X>  istream &operator>>(istream &cin, vector<X> &a) {  for (auto &x : a) cin >> x; return cin;  }  // Vector Output  template <typename X>  ostream &operator<<(ostream &cout, const vector<X> &a) {  int n = a.size();  if (n == 0) return cout; cout << a[0];  for (int i = 1; i < n; i++) cout << ' ' << a[i]; return cout;  }  // Matrix (Nested Vector) Input  template <typename X>  istream &operator>>(istream &cin, vector<vector<X>> &mat) {  for (auto &row : mat) { for (auto &elem : row) { cin >> elem; } } return cin;  }  // Matrix (Nested Vector) Output  template <typename X>  ostream &operator<<(ostream &cout, const vector<vector<X>> &mat) {  for (const auto &row : mat) {  for (const auto &elem : row) { cout << elem << ' '; } cout << '\n';  } return cout;  }  // Map Input  template <typename X, typename Y>  istream &operator>>(istream &cin, map<X, Y> &m) {  size\_t n; cin >> n;  for (size\_t i = 0; i < n; ++i) {  X key; Y value; cin >> key >> value;  m[key] = value;  } return cin;  }  // Map Output  template <typename X, typename Y>  ostream &operator<<(ostream &cout, const map<X, Y> &m) {  for (const auto &[x, y] : m) cout << x << ' ' << y << '\n'; return cout;  }  // Set Input  template <typename X>  istream &operator>>(istream &cin, set<X> &s) {  size\_t n; cin >> n;  for (size\_t i = 0; i < n; ++i) { X value; cin >> value; s.insert(value); } return cin;  }  // Set Output  template <typename X>  ostream &operator<<(ostream &cout, const set<X> &s) {  for (const auto &x : s) cout << x << ' '; return cout;  }  // Stack Input  template <typename X>  istream &operator>>(istream &cin, stack<X> &s) {  size\_t n; cin >> n;  for (size\_t i = 0; i < n; ++i) {  X value; cin >> value; s.push(value);  } return cin;  }  // Stack Output  template <typename X>  ostream &operator<<(ostream &cout, stack<X> s) {  while (!s.empty()) { cout << s.top() << ' '; s.pop(); } return cout;  }  // Queue Input  template <typename X>  istream &operator>>(istream &cin, queue<X> &q) {  size\_t n; cin >> n;  for (size\_t i = 0; i < n; ++i) { X value; cin >> value; q.push(value); } return cin;  }  // Queue Output  template <typename X>  ostream &operator<<(ostream &cout, queue<X> q) {  while (!q.empty()) { cout << q.front() << ' '; q.pop(); } return cout;  }  // Deque Input  template <typename X>  istream &operator>>(istream &cin, deque<X> &dq) {  for (auto &x : dq) cin >> x; return cin;  }  // Deque Output  template <typename X>  ostream &operator<<(ostream &cout, const deque<X> &dq) {  for (const auto &x : dq) cout << x << ' '; return cout;  }  // Priority Queue Input  template <typename X>  istream &operator>>(istream &cin, priority\_queue<X> &pq) {  size\_t n; cin >> n;  for (size\_t i = 0; i < n; ++i) {  X value; cin >> value; pq.push(value);  } return cin;  }  // Priority Queue Output  template <typename X>  ostream &operator<<(ostream &cout, priority\_queue<X> pq) {  while (!pq.empty()) { cout << pq.top() << ' '; pq.pop(); } return cout;  }  // Debugger: Finding Bug  template <typename X>  void debug(const X &x, const string &name) { cout << name << " = " << x << '\n'; }  void Run\_Time() {  auto start = chrono::high\_resolution\_clock::now();  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> elapsed = end - start;  cout << "Code Execution Time: " << elapsed.count() << " seconds.\n";  }  int main() {  int num = 42;  double pi = 3.14159;  string msg = "Hello, World!";  vector <int> vec = {1, 2, 3, 4, 5};  bug(num); // Output: num = 42  bug(pi); // Output: pi = 3.14159  bug(msg); // Output: msg = Hello, World!  bug(vec); // Output: vec = 1 2 3 4 5  return 0;  } | **Prims Algorithm**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 5;  vector<pair<int, int>> mat[N];  bool vis[N];  class cmp {  public:  bool operator()(pair<int, int> a, pair<int, int> b) {  return a.second > b.second;  }  };  int prims(int src) {  priority\_queue<pair<int, int>, vector<pair<int, int>>, cmp> pq;  pq.push({src, 0});  int totalCost = 0;  while (!pq.empty()) {  pair<int, int> current = pq.top();  int pnode = current.first;  int pcost = current.second;  pq.pop();  if (!vis[pnode]) {  totalCost += pcost;  vis[pnode] = true;    for (pair<int, int> child : mat[pnode]) {  int cnode = child.first;  int ccost = child.second;  if (!vis[cnode]) {  pq.push({cnode, ccost});  }  }  }  }  return totalCost;  }  int main() {  int n, e; cin >> n >> e;  // Reading edges  for (int i = 0; i < e; i++) {  int a, b, c; cin >> a >> b >> c;  mat[a].push\_back({b, c});  mat[b].push\_back({a, c});  }  memset(vis, false, sizeof(vis));  int totalCost = prims(1);  // Output total cost of MST  cout << "Total cost of MST: " << totalCost << endl;  return 0;  }  **Kruskal’s Algorithm**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 5;  int par[N], level[N];  class Edge {  public:  int u, v, w;  Edge(int x, int y, int z) : u(x), v(y), w(z) {}  };  int dsu\_find(int node) {  if (par[node] == -1) return node;  return par[node] = dsu\_find(par[node]);  }  void dsu\_union\_by\_rank(int a, int b) {  int leaderA = dsu\_find(a), leaderB = dsu\_find(b);  if (leaderA != leaderB) {  if (level[leaderA] < level[leaderB]) par[leaderA] = leaderB;  else if (level[leaderA] > level[leaderB]) par[leaderB] = leaderA;  else {  par[leaderB] = leaderA;  level[leaderA]++;  }  }  }  bool cmp(Edge &e1, Edge &e2) {  return e1.w < e2.w;  }  int main() {  int n, e;  cin >> n >> e;  vector<Edge> edges;  for (int i = 0; i < e; i++) {  int a, b, c;  cin >> a >> b >> c;  edges.push\_back(Edge(a, b, c));  }  sort(edges.begin(), edges.end(), cmp);  memset(par, -1, sizeof(par));  memset(level, 0, sizeof(level));  int totalCost = 0;  for (Edge ed : edges) {  int leaderA = dsu\_find(ed.u), leaderB = dsu\_find(ed.v);  if (leaderA != leaderB) {  dsu\_union\_by\_rank(ed.u, ed.v);  totalCost += ed.w;  cout << ed.u << " " << ed.v << " " << ed.w << endl;  }  }  cout << "Total cost of MST: " << totalCost << endl;  return 0;  }  **Dynamic Programming**  **Problem Statement**: A frog is on Stone 1 and needs to reach Stone N. From Stone `i`, it can jump to Stone `i+1` or `i+2`, incurring a cost of `|h[i] - h[j]|` for each jump. Find the minimum cost for the frog to reach Stone N.  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  void solve() {  int N;  cin >> N;  vector<int> h(N + 1);  for (int i = 1; i <= N; i++) cin >> h[i];  vector<ll> dp(N + 1, LLONG\_MAX);  dp[N] = 0;  for (int i = N - 1; i >= 1; i--) {  if (i + 1 <= N) dp[i] = min(dp[i], abs(h[i] - h[i + 1]) + dp[i + 1]);  if (i + 2 <= N) dp[i] = min(dp[i], abs(h[i] - h[i + 2]) + dp[i + 2]);  }  cout << dp[1] << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0);  solve(); return 0;  }  **Problem Statement**: There are N stones, numbered 1, 2, ..., N. For each i (1 ≤ i ≤ N), the height of Stone i is h[i]. A frog starts on Stone 1 and can jump to one of the next K stones: Stone i+1, i+2, ..., i+K. The cost of jumping from Stone i to Stone j is |h[i] - h[j]|. Find the minimum cost for the frog to reach Stone N.  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  void solve() {  int N, K; cin >> N >> K;  vector<int> h(N + 1);  for (int i = 1; i <= N; i++) cin >> h[i];  vector<ll> dp(N + 1, LLONG\_MAX);  dp[N] = 0;  for (int i = N - 1; i >= 1; i--) {  for (int j = 1; j <= K && i + j <= N; j++) {  dp[i] = min(dp[i], abs(h[i] - h[i + j]) + dp[i + j]);  }  }  cout << dp[1] << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0);  solve(); return 0;  }  **Problem Statement**: Taro's vacation consists of N days. On each day, he can choose one of three activities:  1. Swim in the sea, gaining a[i] points of happiness.  2. Catch bugs in the mountains, gaining b[i] points of happiness.  3. Do homework at home, gaining c[i] points of happiness.  Taro cannot repeat the same activity on consecutive days. Find the maximum total happiness Taro can gain during the vacation.  Constraints: 1 ≤ N ≤ 10^5 and 1 ≤ a[i], b[i], c[i] ≤ 10^4  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  void solve() {  ll N; cin >> N;  vector<ll> a(N + 1), b(N + 1), c(N + 1);  for (ll i = 1; i <= N; i++) {  cin >> a[i] >> b[i] >> c[i];  }  vector<ll> dp1(N + 1, 0), dp2(N + 1, 0), dp3(N + 1, 0);  dp1[1] = a[1]; dp2[1] = b[1]; dp3[1] = c[1];  for (ll i = 2; i <= N; i++) {  dp1[i] = max(dp2[i - 1] + a[i], dp3[i - 1] + a[i]);  dp2[i] = max(dp1[i - 1] + b[i], dp3[i - 1] + b[i]);  dp3[i] = max(dp1[i - 1] + c[i], dp2[i - 1] + c[i]);  }  cout << max({dp1[N], dp2[N], dp3[N]}) << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0);  solve(); return 0;  }  **Problem Statement**: There are N items, numbered 1, 2, ..., N. For each i (1 ≤ i ≤ N), Item i has a weight w[i] and a value v[i]. Taro wants to choose a subset of items and carry them in a knapsack with a capacity W. The total weight of the items chosen should not exceed W. The task is to find the maximum possible sum of values of the items that Taro can take home.  Constraints: 1 ≤ N ≤ 100, 1 ≤ W ≤ 10^5, 1 ≤ w[i] ≤ W, 1 ≤ v[i] ≤ 10^9  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const int MAX\_N = 105;  const int MAX\_W = 100005;  ll dp[MAX\_N][MAX\_W]; // dp[i][w] represents the max value for the first i items with total weight <= w  void solve() {  int n, W; cin >> n >> W;  vector<int> weight(n + 1), value(n + 1);  for (int i = 1; i <= n; i++) { cin >> weight[i] >> value[i]; }  // Initialize dp table with 0 (base case: 0 items, 0 weight)  for (int i = 0; i <= n; i++) {  for (int w = 0; w <= W; w++) { dp[i][w] = 0; }  }  // Fill the dp table using bottom-up approach  for (int i = 1; i <= n; i++) {  for (int w = 0; w <= W; w++) {  // If we do not take the current item  dp[i][w] = dp[i-1][w];  // If we take the current item, check if the weight fits  if (w >= weight[i]) {  dp[i][w] = max(dp[i][w], dp[i-1][w - weight[i]] + value[i]);  }  }  }  // The answer is in dp[n][W] which is the max value with the full capacity  cout << dp[n][W] << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  solve(); return 0;  }  **Problem Statement**: There are N items, numbered 1, 2, ..., N. For each i (1 ≤ i ≤ N), Item i has a weight w[i] and a value v[i]. Taro wants to choose some of the N items and carry them in a knapsack with a capacity W. The total weight of the items chosen should not exceed W. The task is to find the maximum possible sum of values of the items that Taro can take home.  Constraints: 1 ≤ N ≤ 100, 1 ≤ W ≤ 10^9, 1 ≤ w[i] ≤ W, 1 ≤ v[i] ≤ 10^3  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const int MAX\_N = 100; // Maximum number of items  const int MAX\_W = 100000; // Maximum weight for dp array (since weight can be up to W)  void solve() {  int n, W; cin >> n >> W;  vector<int> weight(n + 1), value(n + 1);  for (int i = 1; i <= n; i++) { cin >> weight[i] >> value[i]; }  vector<ll> dp(W + 1, 0); // DP array to store maximum value for each weight capacity  // Process each item  for (int i = 1; i <= n; i++) {  for (int w = W; w >= weight[i]; w--) { // Traverse from W down to weight[i] to prevent overwriting results  dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);  }  }  // The answer is the maximum value that can be obtained with any weight ≤ W  cout << dp[W] << '\n';  }  int main() {  ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);  solve(); return 0;  }  **Longest Increasing Subsequence**  #include <bits/stdc++.h>  using namespace std;  vector<long long> LIS\_Path(vector<long long>& seq) {  long long n = seq.size();  vector<long long> sub, subIndex, path(n, -1);  for (long long i = 0; i < n; ++i) {  if (sub.empty() || sub.back() < seq[i]) {  path[i] = sub.empty() ? -1 : subIndex.back();  sub.push\_back(seq[i]);  subIndex.push\_back(i);  } else {  long long idx = lower\_bound(sub.begin(), sub.end(), seq[i]) - sub.begin();  path[i] = (idx == 0) ? -1 : subIndex[idx - 1];  sub[idx] = seq[i];  subIndex[idx] = i;  }  }  vector<long long> result;  for (long long t = subIndex.back(); t != -1; t = path[t])  result.push\_back(seq[t]);  reverse(result.begin(), result.end());  return result;  }  int main() {  vector<long long> v = {1, 3, 5, 4, 6, 2, 8};  vector<long long> lis = LIS\_Path(v);  for (long long i : lis) cout << i << ' ';  }  **Longest Common Subsequence**  #include <bits/stdc++.h>  using namespace std;  int LCS(string& s1, string& s2) {  int len1 = s1.length();  int len2 = s2.length();  vector<int> dp(len2 + 1, 0);  for (int i = 1; i <= len1; i++) {  int prevDiagonal = 0;  for (int j = 1; j <= len2; j++) {  int temp = dp[j];  if (s1[i - 1] == s2[j - 1]) { dp[j] = prevDiagonal + 1; }  else { dp[j] = max(dp[j], dp[j - 1]); }  prevDiagonal = temp;  }  }  return dp[len2];  }  int main() {  string s1 = "AGGTAB"; string s2 = "GXTXAYB";  int lcsLength = LCS(s1, s2);  cout << "Length of Longest Common Subsequence: " << lcsLength << '\n';  return 0;  }  **Longest Common Substring**  #include <bits/stdc++.h>  using namespace std;  int LongestCommonSubstring(string& s1, string& s2) {  int len1 = s1.length(); int len2 = s2.length();  vector<int> dp(len2 + 1, 0);  int maxLength = 0;  for (int i = 1; i <= len1; i++) {  int prevDiagonal = 0;  for (int j = 1; j <= len2; j++) {  int temp = dp[j];  if (s1[i - 1] == s2[j - 1]) {  dp[j] = prevDiagonal + 1;  maxLength = max(maxLength, dp[j]);  } else { dp[j] = 0; }  prevDiagonal = temp;  }  }  return maxLength;  }  int main() {  string s1 = "ABABC";  string s2 = "BABCAB";  int lcsLength = LongestCommonSubstring(s1, s2);  cout << "Length of Longest Common Substring: " << lcsLength << '\n';  return 0;  }  **BFS Graph Traversal**  #include <bits/stdc++.h>  using namespace std;  const int mx = 2e5 + 5;  vector<int> v[mx];  bool vis[mx];  void bfs(int src) {  queue<int> q;  q.push(src);  vis[src] = true;  while (!q.empty()) {  int par = q.front();  q.pop();  cout << par << endl;  for (int child : v[par]) {  if (!vis[child]) {  q.push(child);  vis[child] = true;  }  }  }  }  int main() {  int n, e; cin >> n >> e;  while (e--) {  int a, b; cin >> a >> b;  v[a].push\_back(b); v[b].push\_back(a);  }  int src; cin >> src;  memset(vis, false, sizeof(vis));  bfs(src);  return 0;  }  **BFS Graph Levels**  #include <bits/stdc++.h>  using namespace std;  const int mx = 2e5 + 5;  vector<int> adj[mx];  bool visited[mx];  int level[mx];  void bfs(int src) {  queue<int> q;  q.push(src);  visited[src] = true;  level[src] = 0;  while (!q.empty()) {  int par = q.front();  q.pop();  for (int child : adj[par])  if (!visited[child]) {  q.push(child);  visited[child] = true;  level[child] = level[par] + 1;  }  }  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  int n, e; cin >> n >> e;  for (int i = 0; i < e; i++) {  int a, b; cin >> a >> b;  adj[a].push\_back(b);  adj[b].push\_back(a);  }  int src; cin >> src;  memset(visited, false, sizeof(visited));  memset(level, -1, sizeof(level));  bfs(src);  for (int i = 0; i < n; i++) cout << "Node: " << i << ", Level: " << level[i] << '\n';  return 0;  }  **BFS Cycle Detection**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 5;  bool vis[N];  vector<int> adj[N];  int parentArray[N];  bool ans;  void bfs(int s) {  queue<int> q;  q.push(s);  vis[s] = true;  while (!q.empty()) {  int parent = q.front(); q.pop();  for (int child : adj[parent]) {  if (vis[child] == true && parentArray[parent] != child) { ans = true; }  if (vis[child] == false) {  vis[child] = true;  parentArray[child] = parent;  q.push(child);  }  }  }  }  int main() {  int n, e; cin >> n >> e;  while (e--) {  int a, b; cin >> a >> b;  adj[a].push\_back(b); adj[b].push\_back(a);  }  memset(vis, false, sizeof(vis));  memset(parentArray, -1, sizeof(parentArray));  ans = false;  for (int i = 0; i < n; i++) {  if (!vis[i]) { bfs(i); }  }  if (ans) { cout << "Cycle found"; } else { cout << "Cycle not found";}  return 0;  }  **BFS Shortest Path**  #include <bits/stdc++.h>  using namespace std;  const int mx = 2e5 + 5;  vector<int> adj[mx];  bool visited[mx];  void bfs(int src, int des) {  queue<pair<int, int>> q;  q.push({src, 0});  visited[src] = true;  bool found = false;  while (!q.empty()) {  pair<int, int> parent = q.front();  q.pop();  int node = parent.first;  int level = parent.second;  if (node == des) {  cout << "Shortest path length: " << level << '\n';  found = true;  break;  }  for (int child : adj[node]) {  if (!visited[child]) {  q.push({child, level + 1});  visited[child] = true;  }  }  }  if (!found) { cout << "Destination not reachable" << '\n'; }  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  int n, e; cin >> n >> e;  for (int i = 0; i < e; i++) {  int a, b; cin >> a >> b;  adj[a].push\_back(b); adj[b].push\_back(a);  }  int src, des; cin >> src >> des;  memset(visited, false, sizeof(visited));  bfs(src, des);  return 0;  }  **BFS Shortest Path with Path Printing**  #include <bits/stdc++.h>  using namespace std;  const int mx = 2e5 + 5;  vector<int> adj[mx];  bool visited[mx];  int level[mx];  int parent[mx];  void bfs(int src) {  queue<int> q;  q.push(src);  visited[src] = true;  level[src] = 0;  parent[src] = -1;  while (!q.empty()) {  int par = q.front();  q.pop();  for (int child : adj[par]) {  if (!visited[child]) {  q.push(child);  visited[child] = true;  level[child] = level[par] + 1;  parent[child] = par;  }  }  }  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  int n, e; cin >> n >> e;  for (int i = 0; i < e; i++) {  int a, b; cin >> a >> b;  adj[a].push\_back(b); adj[b].push\_back(a);  }  int src, des; cin >> src >> des;  memset(visited, false, sizeof(visited));  memset(level, -1, sizeof(level));  memset(parent, -1, sizeof(parent));  bfs(src);  if (level[des] == -1) {  cout << "Destination not reachable" << '\n';  } else {  vector<int> path;  int x = des;  while (x != -1) {  path.push\_back(x);  x = parent[x];  }  reverse(path.begin(), path.end());  cout << "Shortest path length: " << level[des] << '\n';  cout << "Path: ";  for (int val : path) { cout << val << " "; }  cout << '\n';  }  return 0;  }  **BFS Grid Shortest Distance**  #include <bits/stdc++.h>  using namespace std;  const int MAX = 20;  bool visited[MAX][MAX];  int level[MAX][MAX];  vector<pair<int, int>> directions = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};  int row, col;  char grid[MAX][MAX];  bool isValid(int i, int j) {  return (i >= 0 and i < row and j >= 0 and j < col);  }  void bfs(int startX, int startY) {  queue<pair<int, int>> q;  q.push({startX, startY});  visited[startX][startY] = true;  level[startX][startY] = 0;  while (!q.empty()) {  auto [x, y] = q.front();  q.pop();  for (auto [dx, dy] : directions) {  int newX = x + dx;  int newY = y + dy;  if (isValid(newX, newY) and !visited[newX][newY]) {  q.push({newX, newY});  visited[newX][newY] = true;  level[newX][newY] = level[x][y] + 1;  }  }  }  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  cin >> row >> col;  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  cin >> grid[i][j];  }  }  int startX, startY; cin >> startX >> startY;  memset(visited, false, sizeof(visited));  memset(level, -1, sizeof(level));  bfs(startX, startY);  cout << "Distance to (2, 3): " << level[2][3] << '\n';  return 0;  }  **DFS Graph Traversal**  #include <bits/stdc++.h>  using namespace std;  const int MAX = 20;  char grid[MAX][MAX];  bool visited[MAX][MAX];  vector<pair<int, int>> directions = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};  int n, m;  bool isValid(int row, int col) {  return (row >= 0 && row < n && col >= 0 && col < m);  }  void dfs(int row, int col) {  cout << "Visited cell: (" << row << ", " << col << ")\n";  visited[row][col] = true;  for (auto [dRow, dCol] : directions) {  int newRow = row + dRow;  int newCol = col + dCol;  if (isValid(newRow, newCol) && !visited[newRow][newCol]) {  dfs(newRow, newCol);  }  }  }  int main() {  ios::sync\_with\_stdio(false); cin.tie(NULL);  cin >> n >> m;  for (int i = 0; i < n; ++i) {  for (int j = 0; j < m; ++j) { cin >> grid[i][j]; }  }  int startRow, startCol; cin >> startRow >> startCol;  memset(visited, false, sizeof(visited));  dfs(startRow, startCol);  return 0;  }  **DFS Cycle Detection**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 5;  bool vis[N];  bool pathVisit[N];  vector<int> adj[N];  bool ans;  void dfs(int parent) {  vis[parent] = true;  pathVisit[parent] = true;  for (int child : adj[parent]) {  if (pathVisit[child]) { ans = true; }  if (!vis[child]) { dfs(child); }  }  pathVisit[parent] = false;  }  int main() {  int n, e;  cin >> n >> e;  while (e--) {  int a, b; cin >> a >> b;  adj[a].push\_back(b);  // Uncomment the following line for an undirected graph  // adj[b].push\_back(a);  }  memset(vis, false, sizeof(vis));  memset(pathVisit, false, sizeof(pathVisit));  ans = false;  for (int i = 0; i < n; i++) {  if (!vis[i]) { dfs(i); }  }  if (ans) cout << "Cycle detected";  else cout << "Cycle not detected";  return 0;  }  **DFS Connected Components**  #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 5;  vector<int> adj[N];  bool visited[N];  void dfs(int src) {  cout << "Visited node: " << src << '\n';  visited[src] = true;  for (int child : adj[src]) {  if (!visited[child]) {  dfs(child);  }  }  }  int main() {  ios::sync\_with\_stdio(false);  cin.tie(NULL);  int n, e; cin >> n >> e;  for (int i = 0; i < e; i++) {  int a, b; cin >> a >> b;  adj[a].push\_back(b);  adj[b].push\_back(a);  }  memset(visited, false, sizeof(visited));  int componentCount = 0;  for (int i = 0; i < n; i++) {  if (!visited[i]) {  cout << "Starting DFS at component: " << componentCount + 1 << " starting from node " << i << '\n';  dfs(i);  componentCount++;  }  }  cout << "Number of components: " << componentCount << '\n';  return 0;  }  **Dijkstra**  #include <bits/stdc++.h>  using namespace std;  const int N = 100;  vector<pair<int, int>> v[N];  int dis[N];  class cmp {  public:  bool operator()(pair<int, int> a, pair<int, int> b) {  return a.second > b.second;  }  };  void dijkstra(int src) {  priority\_queue<pair<int, int>, vector<pair<int, int>>, cmp> pq;  pq.push({src, 0});  dis[src] = 0;  while (!pq.empty()) {  pair<int, int> parent = pq.top();  pq.pop();  int node = parent.first;  int cost = parent.second;  for (pair<int, int> child : v[node]) {  int childNode = child.first;  int childCost = child.second;  if (cost + childCost < dis[childNode]) {  dis[childNode] = cost + childCost;  pq.push({childNode, dis[childNode]});  }  }  }  }  int main() {  int n, e; cin >> n >> e;  while (e--) {  int a, b, c; cin >> a >> b >> c;  v[a].push\_back({b, c});  v[b].push\_back({a, c});  }  for (int i = 0; i < n; i++) { dis[i] = INT\_MAX; }  dijkstra(0);  for (int i = 0; i < n; i++) {  cout << i << " -> " << dis[i] << endl;  }  return 0;  }  **Adhoc**  ll All\_Possible\_Substring\_Sum(string s) {      int n = (int)s.size();      vector<ll> digit\_sum(n);      digit\_sum[0] = s[0] - '0';      ll totalSum = digit\_sum[0];      for (int i = 1; i < n; i++) {          int cur\_val = s[i] - '0';          digit\_sum[i] = (i + 1) \* cur\_val + 10 \* digit\_sum[i - 1];          totalSum += digit\_sum[i];      }      return totalSum; // TC: O(N)  }  void GenerateAndPrintAllSubstrings(string s) {      int n = s.size();      for (int i = 0; i < n; i++) {          string currentSubstring;          for (int j = i; j < n; j++) {              currentSubstring += s[j];              cout << currentSubstring << '\n';          }      } // TC: O(N x N)  }  **Maximum Sum Subarray in fixed length**  ll maximumSumSubarray(int k, vector<int>& a, int n) {      int l = 0, r = 0;      long long sum = 0, ans = 0;      while (r < n) {          sum += a[r];          if ((r - l + 1) == k) {              ans = max(ans, sum);              sum -= a[l];              l++;          }          r++;      }      return ans;  }  **Kadane’s Algo**  ll maximum\_subarray\_sum(vector <ll> &v) {      int n = v.size();      ll maxSum = v[0], currentSum = v[0];      for (int i = 1; i < n; i++) {          currentSum = max(currentSum + v[i], v[i]);          maxSum = max(maxSum, currentSum);      }      return maxSum; // TC: O(N)  }  **String**  void String\_Permutations(string s) { // TC: O(n x n!)      sort(s.begin(), s.end());      do { cout << s << '\n'; } while (next\_permutation(s.begin(), s.end()));  }  ll longestSubstringWithKUniqueChars(string s, ll k) {      ll start = 0; ll end = 0; ll maxLength = -1; ll uniqueCount = 0;      vector<ll> charFrequency(26, 0);      while (end < s.size()) {          if (charFrequency[s[end] - 'a']++ == 0) { uniqueCount++; }          while (uniqueCount > k) {              if (--charFrequency[s[start] - 'a'] == 0) { uniqueCount--; }              start++;          }          if (uniqueCount == k) { maxLength = max(maxLength, end - start + 1); }          end++;      }      return maxLength; // O(N)  }  bool is\_subsequence(string& s1, string& s2) {      int n = s1.length(), m = s2.length();      int i = 0, j = 0;      while (i < n && j < m) { if (s1[i] == s2[j]) i++; j++; }      return i == n; // TC: O(len(s2))  }  bool is\_substring(string child, string mother) {      if (mother.find(child) != string::npos) return true;      return false; // TC: O(N)  }  **KMP**  vector<int> constructTempArray(string pattern) {  vector<int> lps(pattern.size());  int index = 0;  for (int i = 1; i < (int)pattern.size();) {  if (pattern[i] == pattern[index]) {  lps[i] = index + 1; ++index; ++i;  } else {  if (index != 0) { index = lps[index - 1] } else { lps[i] = index; ++i; }  }  }  return lps;  }  bool KMPMultipleTimes(string text, string pattern) {  vector<int> lps = constructTempArray(pattern);  int j = 0, i = 0;  while (i < (int)text.size()) {  if (text[i] == pattern[j]) { ++i; ++j; }  else {  if (j != 0) j = lps[j - 1];  else i++;  }  if (j == (int)pattern.size()) { return true; }  }  return false; // O(N + M)  }  int main() {  string text = "ababcabcabababd";  string pattern = "ababd";  if (KMPMultipleTimes(text, pattern)) { cout << "Pattern found in the text.\n"; }  else { cout << "Pattern not found in the text.\n"; }  return 0;  }  // (graph revise kora) Bellman ford + seg tree  **Geometry**  Triangle:  Area = 0.5 x base x height  = This is Heron’s formula  = 0.5 ab sin C = 0.5 bc sin A = 0.5 ac sin B  Perimeter, 2S = a + b + c Semi perimeter, S =  A VT with its inscribed circle   Triangle Area = S . r  = r .  = 0.5 ar + 0.5 br + 0.5 cr  Radius, r =    **Stress Testing Code:**  // Bash Script for Stress Testing: (checker.sh)  /\*--------------------------------------------  for((i = 1; ; ++i)); do  echo $i  ./gen $i > in.txt  diff -w <(./a < in.txt) <(./b < in.txt) || break  done  --------------------------------------------\*/  // Random Integer Number Generator:  #using ll = long long  mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());  inline ll gen\_random(ll l, ll r) {  return uniform\_int\_distribution<ll>(l, r)(rng);  }  // Random Real Number Generator:  mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());  inline double gen\_random(double l, double r) {  return uniform\_real\_distribution<double>(l, r)(rng);  }  **Environment Setup:**  {  "C\_Cpp.default.cppStandard": "c++23",  "C\_Cpp.default.cStandard": "c11",  "terminal.integrated.defaultProfile.windows": "Command Prompt",  "code-runner.runInTerminal": false,  "code-runner.saveAllFilesBeforeRun": true,  "code-runner.terminalRoot": "/",  "code-runner.executorMap": {  "c": "cd $dir && gcc $fileName -o $fileNameWithoutExt.exe && $dir$fileNameWithoutExt.exe <input.txt> output.txt",  "cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt.exe && $dir$fileNameWithoutExt.exe <input.txt> output.txt",  // "cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt.exe && $dir$fileNameWithoutExt.exe",  },  "extensions.ignoreRecommendations": true,  "terminal.integrated.enableMultiLinePasteWarning": false,  "settingsSync.ignoredExtensions": [  "formulahendry.code-runner"  ],  "code-runner.defaultLanguage": "cpp",  "editor.largeFileOptimizations": false,  "editor.fontSize": 17,  "files.autoSave": "afterDelay",  "editor.minimap.enabled": false,  "workbench.iconTheme": "material-icon-theme",  "workbench.colorTheme": "GitHub Light",  "[cpp]": {  "editor.defaultFormatter": "ms-vscode.cpptools"  }  } |