**STL**

<iostream>, <iomanip>

cout, cin, while (cin >> ...)

getline, while + getline, getline after cin

<iomanip>: fixed, setprecision, setw, setfill, hex / dec

<iomanip>: noskipws / skipws

cin / cout.tie (0); cin / cout.sync\_with\_stdio (false); endl vs '\ n'

<cstdio>: printf / scanf for bold inputs

<vector>

clean! = empty

begin, end, rbegin, rend

push\_back, pop\_back, emplace\_back, front, back

insert

size, resize, capacity, reserve, swap hack / shrink\_to\_fit

vector <bool>

<string>

string (10, ''), std :: string ("a") + "b"

length / size,

find, rfind, find \_ \* \_ of, string :: npos

< sstream >: stringstream ss ("str"), ss.str ()

<cctype>

isalpha, isalnum, isblank, isdigit, islower, isupper, isxdigit

tolower, toupper, use together with transform

<deque>

<queue>: priority\_queue

<tuple>: pair, make\_pair, .first / .second; tuple, make\_tuple, get <#> ();

Lexicographic comparison

<map>, <set>

map, key sorting

[key] = vs at, for (auto kv: mapa) {}

count, erase

set, insert

<unordered\_set>, <unordered\_map>

std :: hash <T>: : operator ()

<algorithm>

min, max, minmax, max\_element,

sort, predicate with tie, stable\_sort, is\_sorted

sort / iota + next\_permutation

unique / remove / remove\_if + .erase

reverse

fill, copy, copy\_n, <iterator>: back\_inserter, istream\_iterator

most vexing parse

find, find\_if, count, count\_if

search

includes, set\_union , set\_intersection, set\_difference, set\_symmetric\_difference

lower\_bound / upper\_bound

<iterator>: begin (cont), end (cont), size (cont)

<numeric>: accumulate, partial\_sum, iota

<cmath>

hypot, atan2, pi = atan (1) \* 4

round, floor, ceil

abs

<complex>

<limits>: numeric\_limits <int> :: max ()

<random>

<utility>: swap

<bitset>

<chrono>: std :: chrono :: high\_resolution\_clock :: now ( )

<functional>

Compiler-specific: \_\_builtin\_popcount, \_\_builtin\_clz, \_\_builtin\_ctz, \_\_gcd, \_\_int128

# <iostream>, <iomanip> cout, cin, while (cin >> ...)

I / O streams are preferable to sysnichny functions because they do not need a format string and are free from errors in it.

int n = 123123123;

char ch = 'a' + 3;

long long ll = 92233720368547758LL;

cout << n << " " << ch << " " << ll << endl; // 123123123 d 92233720368547758

Similarly, input does not require a format string and the use of pointers.

int k;

cin >> k;

cout << k << endl;

In some tasks, the exact size of the input is not given and it is suggested to enter before EOF ( run.exe <file or Ctrl-Z in stdin). Overloading the >> operator on input streams returns istream & , which can be implicitly cast to bool , the cast returns ! Fail () , and fail () returns true when trying to read after EOF.

int k, sum = 0;

while (cin >> k) {

sum += k;

}

cout << sum << endl;

## getline, while + getline, getline after cin

Some tasks require you to enter the entire line before the line feed, a simple string s; cin >> s; Enter up to the first whitespace character. Then you should use getline , which also returns istream & with the same EOF processing method. The final line feed is not written to the string variable.

aa bb

cc ddddd^Z

string s1, s2;

while (getline(cin, s1)) {

s2 += s1;

}

cout << s2.length() << endl; // 13

It should be added that cin >> var; does not enter a line feed, so you may need to make an additional call to getline to eat it.

stringstream ss("5\na b c");

int n;

string s;

ss >> n;

getline(ss, s);

getline(ss, s);

cout << n << " : " << s << endl;

## <iomanip>: fixed, setprecision, setw, setfill, hex / dec

The <iomanip> header contains formatting options, which in most cases replace printf format strings .

The setprecision and fixed manipulators allow you to set the precision of the output of real numbers for all subsequent << , indicating the number of all digits or digits after the decimal point.

double n = 3.141592653;

cout << setprecision(5) << n << endl; // 3.1416

cout << fixed << setprecision(5) << n << endl; // 3.14159

A pair of setw and setfill allows you to output a string or number to a specified number of positions, possibly filling them with something non-whitespace.

cout << setw(8) << "hello" << endl; // " hello"

cout << setfill('0') << setw(3) << 7 << endl; // "007"

Manipulators hex and dec allow you to display integers in hexadecimal or decimal notation.

int n;

stringstream("2A") >> hex >> n;

cout << dec << n << endl; // 42

cout << hex << 42 << endl; // 2a

## <iomanip>: noskipws / skipws

By default, cin enters a few char , skipping whitespace. The noskipws / skipws manipulators allow you to switch this behavior.

char s1, s2, s3;

stringstream("a b c") >> s1 >> s2 >> s3;

cout << s1 << ", " << s2 << ", " << s3 << endl; // a, b, c

stringstream("a b c") >> noskipws >> s1 >> s2 >> s3;

cout << s1 << "," << s2 << "," << s3 << endl; // a, ,b

## cin / cout.tie (0); cin / cout.sync\_with\_stdio (false); endl vs '\ n'

I / O streams were designed to intensively expand and modify their state. Overhead caused by this can lead to TL. In such cases, view magic may help.

cin.tie(0);

cin.sync\_with\_stdio(false);

for volume input, or the same with cout for volume output.

As a rule, in a joint venture, you should use << endl to line feed the output. It is similar to << '\ n' << flush . The flush manipulator outputs everything from the buffer to the output; when outputting a very large number of lines, this pretty much slows down. In this case, replace all endl with "\ n" , leaving endl or flush at the end .

# <cstdio>: printf / scanf for bold inputs

If you can’t get I / O streams into the TL in any way (only if this is the case and only because of this), you can use the sysh functions.

int n;

long long n\_l;

double d;

float f;

char s[5];

char ch;

scanf("%d %lld %lf %f %s %c", &n, &n\_l, &d, &f, &s, &ch);

printf("%d %lld %lf %f %s %c", n, n\_l, d, f, s, ch);

# <vector>

Instead of arrays in C ++ programs, you should always use the serial container vector . Its size is set in the constructor. You can set the value of each element, the default will be default (empty vector , 0 , "" , false ).

vector<int> v(5); // {0,0,0,0,0}

vector<vector<int>> v(2, vector<int>(3, -1)); // {{-1,-1,-1}, {-1,-1,-1}}

If you need to set the size and fill value after creation

vector<double> v;

v.assign(4, 1.0); // {1.0, 1.0, 1.0, 1.0}

A vector can be initialized with initialization lists or from a pair of iterators.

vector<int> v{1, 2, 3, 4, 5};

vector<int> v2(v.begin() + 2, v.begin() + 4); // {3, 4}

An appeal to an element of a vector like an array.

cout << v2[0]; // 3

A vector has almost the same performance as an array, so in release it does not check the bounds of the array. You can force him to do this (and throw an exception in violation):

cout << v2.at(2); // Exception: out\_of\_range

## clean! = empty

Clean (not to be confused with check v.empty () (but are there any who confuse?)), Works for any container.

v.clear();

## begin, end, rbegin, rend

There are several ways to iterate over the container (example for vector ):

vector<int> v{1, 2, 3}

or (int i = 0; i < v.size(); ++i) {

cout << v[i] << endl;

}

for (auto it = v.begin(); it != v.end(); ++it) {

cout << \*it << endl;

}

for (int item : v) {

cout << item << endl;

}

The container can also be bypassed in the reverse order.

for (auto it = v.rbegin(); it != v.rend(); ++it) {

cout << \*it << endl;

}

## push\_back, pop\_back, emplace\_back, front, back

A vector is a dynamic array, at the end you can add new elements ( push\_back ) for the amortized O (1 ), changing the size ( size ). In order to keep this complexity, the vector allocates more memory than necessary (how much is capacity ), and when the limit is reached ( .size () == .capacity () ) it increases the size immediately by 1.5-2 times (example for MSVC), copying old elements (which spoils all iterators).

vector<int> v;

for (int i = 0; i < 16; ++i) {

v.push\_back(123);

cout << v.size() << "-" << v.capacity() << " ";

}

// 1-1 2-2 3-3 4-4 5-6 6-6 7-9 8-9 9-9 10-13 11-13 12-13 13-13 14-19 15-19 16-19

If you need to first create an object to be inserted (for example, a pair), the constructor and push\_back can be combined into one call, saving copying.

vector<pair<int, string>> v;

v.emplace\_back(5, "hello");

In addition to adding to the end, you can also delete elements from the end.

vector<int> v{1, 2, 3};

v.pop\_back(); // {1, 2}

The first and last element can be accessed by reference.

vector<int> v{2, 4, 8, 16, 32};

cout << v.front() << endl; // 2

cout << v.back() << endl; // 32

v.back() = -1; // {2, 4, 8, 16, -1}

## insert

On an arbitrary iterator, you can insert a value or a segment from a pair of iterators, while all elements after the iterator are copied for O (N).

vector<int> v{1, 2, 3};

vector<int> w{5, 6};

v.insert(v.begin() + 1, w.begin(), w.end()); // {1, 5, 6, 2, 3}

v.insert(v.begin() + 2, 8); // {1, 5, 8, 6, 2, 3}

## size, resize, capacity, reserve, swap hack / shrink\_to\_fit

An existing vector can be resized

vector<int> v{1, 2, 3};

v.resize(5, -1); // {1, 2, 3, -1, -1}

or capacity (useful to avoid the time-consuming large number of copies of the elements of the vector, if the upper estimate of their number is known in advance).

cout << v.size() << " " << v.capacity() << endl; // 5 5

v.reserve(1000);

cout << v.size() << " " << v.capacity() << endl; // 5 1000

With reserve, you can only change the capacity up, but you can recreate a vector with existing elements and a minimum capacity of one line (the so-called swap hack). This can be useful if the total size of the vectors at each moment is limited, but the total maximum size is too large.

v.swap(vector<int>(v));

cout << v.size() << " " << v.capacity() << endl; // 5 5

The .shrink\_to\_fit () method introduced in C ++ 11 has the same effect .

## vector <bool>

The vector is specialized for bool , so that the element takes up 1 bit, this saves memory, but is slower and does not allow the use of pointers to the element.

# <string>

Instead of zero-terminated C-lines in C ++ programs, you should always use string .

string s = "hello, world!";

char c = s[6];

cout << s << endl;

s += c; s += s;

## string (10, ''), std :: string ("a") + "b"

The string constructor , like vector , allows you to specify the number of elements and the element itself.

cout << string(5, '!') << endl;

For C compatibility, the string literal "a" is char const \* , + does not concatenate them.

string s = "AaBb";

s += string("C") + "c";

cout << s << endl;

In C ++ 14, a literal is created that creates strings.

string s = "a"s + "b";

## 

## length / size, substr

Similarly, vector, string has the methods assign , clear , empty , begin , end , rbegin , rend , size , resize , capacity , reserve . A synonym for size is length .

string t = "Hedgehog is gonna get out from that jail";

cout << t.length() << endl;

Substring (start, length):

cout << t.substr(18, 3) << endl;

## push\_back

Like vector, string has methods, push\_back , pop\_back , front , back , insert .

string s;

for (int i = 0; i < 10; ++i) {

s.push\_back('a' + i);

}

cout << s << endl; // abcdefghij

## to\_string, stoi

Numeric types can be converted to the string to\_string and vice versa with the family stoi / stoll / stoull / stof / stod .

double pi = 3.14159265;

string s = "Pi = " + to\_string(pi);

cout << s << endl; // Pi = 3.141593

s = "12345678901";

//cout << stoi(s) << endl; // Exception: out\_of\_range

cout << stoll(s) << endl; // 12345678901

## find, rfind, find \_ \* \_ of, string :: npos

In string, you can search for substrings (a trivial algorithm for O (NM)). An optional second argument is the offset from which to start the search. If there is no substring, string :: npos is returned .

string s = "hello, world!";

// 0123456789012

cout << s.find("wo") << endl; // 7

cout << boolalpha << (s.find("hi") == string::npos) << endl; // true

Search from the end (the found substring will have a beginning not to the right of the offset):

cout << s.rfind("l", 9) << endl; // 3

Such a loop will find all occurrences of the substring

size\_t off = 0;

while (true) {

off = s.find("l", off);

if (off == string::npos) { break; }

cout << off << endl; // 2 3 10

off += 1;

}

In addition to substrings, you can search for characters from some set using the find\_first\_of method . Similarly, find\_first\_not\_of searches for characters that do not belong to the set, and find\_last\_of and find\_last\_not\_of look for such characters from the end of the line.

cout << s.find\_last\_of("aeiou") << endl;

# <sstream>: stringstream ss ("str"), ss.str ()

The input / output string stream is similar to cin / cout , but it does not work with stdin / stdout, but with a string.

stringstream ss;

ss << 2 << " " << 4 << " " << 8;

string s = ss.str();

cout << s << endl; // "2 4 8"

stringstream ss("1 2 3");

int n1, n2, n3;

ss >> n1 >> n2 >> n3;

cout << n1 << " " << n2 << " " << n3 << endl; // 1 2 3

# <cctype>

## isalpha, isalnum, isblank, isdigit, islower, isupper, isxdigit

In the JV, these character classes are most often useful, a list of 0-127:

isalpha - A-Za-z

isalnum - 0-9A-Za-z

isblank - \ t and space

isdigit - 0-9

islower - az

isupper - AZ

isxdigit - 0-9A-Fa-f

## tolower, toupper, use with transform

Expectedly convert AZ to az and vice versa. Strings can be converted by these functions using transform from <algorithm> .

string s = "where is STL?!";

transform(s.begin(), s.end(), s.begin(), toupper);

cout << s << endl; // WHERE IS STL?!

# <deque>

The deck is similar to a vector, but you can add / remove elements from both ends in O (1). Inside it is a cyclic buffer of variable size with pointers to segments of elements of equal length.

deque<int> d;

d.push\_back(3);

d.push\_front(2);

d.push\_back(4);

d.push\_front(1);

d.push\_back(5);

for (int i = 0; i < d.size(); ++i) {

cout << d[i] << " ";

}

cout << endl; // 1 2 3 4 5

Based on deque , std :: stack and std :: queue are built , which simply leave the wrapped deque without unnecessary methods.

# <queue>: priority\_queue

The priority queue allows inserting elements for O (logN), and for O (1) to get the maximum element in the collection, for O (logN) to delete it. Inside, it is a binary heap on a vector that supports the invariant for all i A [i]> = A [2 \* i + 1] && A [i]> = A [2 \* i + 2] .

priority\_queue<int> pq;

for (int i = 0; i < 5; ++i) {

pq.push(i);

}

while (pq.size()) {

int item = pq.top();

cout << item << " ";

pq.pop();

}

cout << endl; // 4 3 2 1 0

Usually, a priority queue is used over pairs (priority, task), from where it got its name.

priority\_queue<pair<int, char>> pq;

for (int i = 0; i < 5; ++i) {

pq.emplace(i, 'a' + i);

}

cout << pq.top().second << endl; // e

# <tuple>: pair, make\_pair, .first / .second; tuple, make\_tuple, get <#> ();

Two or more related values ​​can be paired or a tuple. Access to the elements of the pair through the fields:

pair<int, int> p(1, 2);

cout << p.first << " " << p.second << endl; // 1 2

p = make\_pair(14, 28);

cout << p.first << " " << p.second << endl; // 14 28

to tuple elements by a template function:

tuple<int, int, int> tp(1, 2, 3);

cout << get<0>(tp) + get<1>(tp) + get<2>(tp) << endl; // 6

tp = make\_tuple(14, 28, 42);

cout << get<0>(tp) + get<1>(tp) + get<2>(tp) << endl; // 84

# Lexicographic comparison

For sequential containers ( vector , string , deque , pair , tuple ), the operator <is defined in STL , comparing them lexicographically (first by the first element, if they are equal, by the second, and so on), which means that they can be compared, and the containers from sort them.

Comparison of vectors:

vector<int> box1(3), box2(3);

for (int i = 0; i < 3; ++i) { cin >> box1[i]; }

for (int i = 0; i < 3; ++i) { cin >> box2[i]; }

sort(box1.begin(), box1.end());

sort(box2.begin(), box2.end());

if (box1 <= box2) {

cout << "You can put the first box into the second one" << endl;

} else if (box2 <= box1) {

cout << "You can put the second box into the first one" << endl;

} else {

cout << "You can't put one box into another" << endl;

}

String Comparison:

cout << boolalpha << (string("a") < string("abcd")) << endl; // true

cout << boolalpha << (string("a") < string("ABCD")) << endl; // false

Comparison of pairs:

cout << boolalpha << (make\_pair(1, 2) < make\_pair(1, 3)) << endl; // true

# <map>, <set>

## map, key sorting

The STL has an associative map container (dictionary) that allows you to map to a key of any type that supports operator <a value of an arbitrary type. Inside, it is implemented on the basis of an almost balanced binary red-black tree, so key search takes O (logN) comparisons, inserts and deletes O (logN). The red-black tree is ordered by keys, the keys are unique.

## [key] = vs at, for (auto kv: mapa) {}

Calling the operator [] automatically creates a default value if the constructor did not have such a key before. Therefore, for constant maps, instead of the [] operator, you must use the at method , throwing an out\_of\_range exception in the absence of a key.

Iterating over the map gives pair <TKey, TValue> .

map<char, int> m;

for (char ch : string("hello, world!")) {

m[ch] += 1;

}

cout << m.size() << endl; // 10

for (auto kv : m) {

cout << kv.first << "-" << kv.second << " ";

} // -1 !-1 ,-1 d-1 e-1 h-1 l-3 o-2 r-1 w-1

## count, erase

You can check for a key in map either by comparing m.find (key) (returns an iterator) with m.end () , or by using the count method .

map<char, int> m{{'a', 1}, {'c', 1}, {'b', 0}, {'d', 2}, {'f', 1}};

cout << m.count('a') << " " << m.count('e') << endl; // 1 0

You can remove from map by the erase method , which can take both a key, and an iterator, and a segment specified by two iterators.

m.erase('c'); // {'a': 1, 'b': 0, 'd': 2, 'f': 1}

m.erase(m.find('f')); // {'a': 1, 'b': 0, 'd': 2}

## set, insert

The set container (set) is similar to map , only without values ​​and, therefore, without [] . Insertion is done by the insert method (both an element and a segment of iterators).

set<int> s{1, 2, 3, 5, 6};

cout << s.count(1) << " " << s.count(4) << endl; // 1 0

cout << s.size() << endl; // 5

s.insert(2);

cout << s.size() << endl; // 5

s.insert(0);

cout << s.size() << endl; // 6

vector<int> add{6, 7, 8};

s.insert(add.begin(), add.end());

s.erase(7);

for (int item : s) {

cout << item << " "; // 0 1 2 3 5 6 8

}

# <unordered\_set>, <unordered\_map>

STL has an open hash table, which allows inserting and searching for a key for amortized O (1). Its implementation is based on std :: list , which stores all inserted elements in the traversal order of the container, and pointers inside this list to the head and tail of b = 2 n buckets in a vector of length 2b. The number of buckets is supported by no less than the number of elements. When the size is exhausted, the number of buckets changes by a factor of 2 (in MSVC with a small hash table size immediately by 8 times) and the std :: hash hash is recalculated for all elements.

The unordered\_map and unordered\_set containers are similar to map and set , but the keys are sorted by hash% u.bucket\_count ().

unordered\_set<char> u;

for (int i = 0; i < 26; ++i) {

u.insert('a' + i);

}

for (char ch : u) {

cout << ch;

} // iabcdefghjklmnopqrstuvwxyz

## std :: hash <T> :: operator ()

namespace std {

template<typename T1, typename T2> struct hash<pair<T1, T2>> {

size\_t operator () (pair<T1, T2> const& arg) const {

return hash<T1>()(arg.first) ^ hash<T2>()(arg.second);

}

};

}

# <algorithm>

STL implements some simple and frequently used generic algorithms. They are generalized because they usually do not care what container they work with, they accept a couple of iterators.

## min, max, minmax, max\_element, min\_element

Minimum, maximum, both at once:

cout << min(123, 456) << " " << min('a', 'b') << endl; // 123 a

cout << max(2.5, 2.6) << " " << max('k', 'o') << endl; // 2.6 o

pair<int, int> res = minmax({ 2, 4, 8, 16, 32 });

cout << res.first << ' ' << res.second << endl; // 2 32

The maximum in the sequence (similar to min\_element ), returns an iterator:

vector<int> v{1, 2, 3, 4, 5};

cout << max\_element(v.begin(), v.end()) - v.begin() << endl;

cout << \*max\_element(v.begin(), v.end()) << endl;

## sort, predicate with tie, stable\_sort, is\_sorted

Sorting:

vector<int> v{2342, 1231, 87978, 123, 789};

sort(v.begin(), v.end()); // 123 789 1231 2342 87978

in reverse order:

sort(v.rbegin(), v.rend()); // 87978 2342 1231 789 123

with a predicate for comparison:

vector<int> v{-2, -1, 0, 1, 2, 3};

sort(v.begin(), v.end(), [](int a, int b) {

return abs(a) < abs(b);

}); // 0 -1 1 -2 2 3

with a predicate for lexicographic comparison ( tie creates a tuple of links); The example sorts letters first by frequency (descending), then by character code (ascending):

string s = "hello, world!";

map<char, int> m;

for (char ch : s) { m[ch] += 1; }

vector<pair<char, int>> v(m.begin(), m.end());

sort(v.begin(), v.end(), [](pair<char, int> const& a, pair<char, int> const& b) {

return tie(-a.second, a.first) < tie(-b.second, b.first);

}); // {{'l', 3}, {'o', 2}, {' ', 1}, {'!', 1}, {',', 1}, {'d', 1}, {'e', 1}, {'h', 1}, {'r', 1}, {'w', 1}}

Stable sorting does not change the original order of characters equal in terms of the predicate.

vector<pair<char, int>> v(m.rbegin(), m.rend());

stable\_sort(v.begin(), v.end(), [](pair<char, int> const& a, pair<char, int> const& b) {

return -a.second < -b.second;

}); // {{'l', 3}, {'o', 2}, {'w', 1}, {'r', 1}, {'h', 1}, {'e', 1}, {'d', 1}, {',', 1}, {'!', 1}, {' ', 1}}

Check that the sequence is already sorted:

vector<int> v{1, 2, 3, 1};

cout << boolalpha << is\_sorted(v.begin(), v.end()) << endl; // false

sort(v.begin(), v.end());

cout << boolalpha << is\_sorted(v.begin(), v.end()) << endl; // true

## sort / iota + next\_permutation

The next\_permutation function allows you to iterate over permutations (returning false if the current permutation is lexicographically maximal). In a joint venture, as a rule, iterates over permutation of indices, in this case the initial sequence is usually created by iota . If not, you should almost always take lexicographically minimal permutation of objects first, using sort .

vector<int> v(4);

iota(v.begin(), v.end(), 1);

do {

cout << v[0] << v[1] << v[2] << v[3] << endl;

} while (next\_permutation(v.begin(), v.end()));

// 1234; 1243; 1324; 1342; 1423; 1432; 2134; 2143; 2314; 2341; 2413; 2431;

// 3124; 3142; 3214; 3241; 3412; 3421; 4123; 4132; 4213; 4231; 4312; 4321;

## unique / remove / remove\_if + .erase

Generalized algorithms can interchange elements, but they cannot delete them; the container itself is responsible for this. The unique algorithm does not search for the first repeating elements and rearranges them at the end. In order for unique to give truly unique elements, the sequence must first be sorted. The remove / remove\_if algorithms rearrange a specific value or values ​​that satisfy a predicate. They return an iterator to the beginning of the end, which can be used to remove by erase container.

vector<int> v = {1, 1, 2, 3, 3, 3, 1, 1, 2, 2};

sort(v.begin(), v.end());

v.erase(unique(v.begin(), v.end()), v.end()); // 1 2 3

v.erase(remove\_if(v.begin(), v.end(), [](int x) {

return x % 2 == 0;

}), v.end()); // 1 3

## reverse

Invert elements of a sequential container:

string s = "desserts";

reverse(s.begin(), s.end());

cout << s << endl; // stressed

## fill, copy, copy\_n, <iterator>: back\_inserter, istream\_iterator

A number of functions are designed to fill containers with elements from some source. Every time you use memset , a kitten dies in the world. The back\_inserter and inserter functions create an insert iterator that calls push\_back and insert, respectively, on the argument container when trying to write to it.

vector<int> a(5); // {0, 0, 0, 0, 0}

fill(a.begin() + 1, a.end(), -1); // {0, -1, -1, -1, -1}

copy(a.begin(), a.begin() + 2, a.begin() + 2); // {0, -1, 0, -1, -1}

vector<int> b;

copy\_n(a.rbegin(), 3, back\_inserter(b)); // {-1, -1, 0}

For versatility, the STL contains an iterator class (read-only or write-only) for input / output streams. The default constructor istream\_iterator allows you to read to EOF.

stringstream ss("5\n1 2 3 4 5\n1 1 2 3 5");

int n;

ss >> n;

vector<int> v, w;

copy\_n(istream\_iterator<int>(ss), n, back\_inserter(v)); // v = {1, 2, 3, 4, 5}

copy(istream\_iterator<int>(ss), istream\_iterator<int>(), back\_inserter(w)); // w = {1, 1, 2, 3, 5}

copy(w.begin(), w.end(), ostream\_iterator<int>(cout, ", ")); cout << endl; // 1, 1, 2, 3, 5,

## most vexing parse

It’s rare, but there may be an unpleasant ambiguity (the so-called most vexing parse) when a programmer thinks that w is a variable declaration that takes two objects into the constructor, and the compiler that it is a function prototype.

vector<int> w(istream\_iterator<int>(ss), istream\_iterator<int>());

MSVC issues view warning

warning C4930: prototyped function not called (was a variable definition intended?)

You can convince the compiler that this is a variable declaration with parentheses.

stringstream ss("1 2 3 4 5");

vector<int> w((istream\_iterator<int>(ss)), istream\_iterator<int>());

copy(w.begin(), w.end(), ostream\_iterator<int>(cout, "\_")); // 1\_2\_3\_4\_5\_

## find, find\_if, count, count\_if

The find / find\_if algorithms allow you to find the first value in the sequence that is equal to the given value or satisfies the predicate, and count / count\_if to calculate how many there are.

vector<int> v{0, 1, 1, 2, 3, 5, 8, 13, 21, 34};

cout << find(v.begin(), v.end(), 8) - v.begin() << endl; // позиция 6

cout << \*find\_if(v.begin(), v.end(), [](int i) { return i > 10; }) << endl; // 13

cout << count(v.begin(), v.end(), 1) << endl; // 2

cout << count\_if(v.begin(), v.end(), [](int i) { return i % 2 == 0; }) << endl; // 4

## search

The search algorithm extends string :: find to arbitrary sequences.

vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6};

auto seq = {3, 5};

cout << search(v.begin(), v.end(), seq.begin(), seq.end()) - v.begin() << endl; // 9

## includes, set\_union, set\_intersection, set\_difference, set\_symmetric\_difference

Set-theoretic operations are calculated on sorted segments (most often in vectors, because the overhead is lower).

Inclusion:

vector<int> v1{2, 8, 16, 32};

vector<int> v2{2, 4, 8};

vector<int> v3{8, 16};

cout << boolalpha << includes(v1.begin(), v1.end(), v2.begin(), v2.end()); // false

cout << boolalpha << includes(v1.begin(), v1.end(), v3.begin(), v3.end()); // true

union, intersection, difference, symmetric difference (require an iterator to write the result):

vector<int> v1{10, 20, 30, 40};

vector<int> v2{40, 50, 60, 70};

vector<int> res0111, res0001, res0100, res0110;

set\_union(v1.begin(), v1.end(), v2.begin(), v2.end(), back\_inserter(res0111));

// 10 20 30 40 50 60 70

set\_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), back\_inserter(res0001));

// 40

set\_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), back\_inserter(res0100));

// 10 20 30

set\_symmetric\_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), back\_inserter(res0110));

// 10 20 30 50 60 70

## lower\_bound / upper\_bound

An integer binary search searches in the segment sorted in ascending order (in the vector after O (logN)) the segment [begin, end [containing this value.

vector<int> v{1, 1, 1, 1, 2, 2, 2, 3, 3, 4};

cout << "2 in [" <<

lower\_bound(v.begin(), v.end(), 2) - v.begin() << "; " <<

upper\_bound(v.begin(), v.end(), 2) - v.begin() << "[" << endl; // 2 in [4; 7[

If you want to calculate the "elements" on the fly by the "index", you can use your own iterator class.

struct int\_iterator : iterator<random\_access\_iterator\_tag, int> {

int n;

function<int(int)> pred;

int\_iterator(int n, function<int(int)> pred) : n(n), pred(pred) { }

int operator \* () const { return pred(n); }

operator int () const { return n; }

int\_iterator& operator ++ () { return \*this += 1; }

int\_iterator& operator += (int rhs) { n += rhs; return \*this; }

};

function<int(int)> pred = [](int x) {

if (x < 100500) { return -1; }

if (x > 100600) { return 1; }

return 0;

};

int\_iterator it\_begin(0, pred), it\_end(1000000, pred);

cout << "[" <<

lower\_bound(it\_begin, it\_end, 0) << ", " <<

upper\_bound(it\_begin, it\_end, 0) << "[" << endl; // [100500, 100601[

# <iterator>: begin (cont), end (cont), size (cont)

Instead of the container methods .begin () / .end () / .size (), you can use the global functions begin (cont) / end (cont) / size (cont) , which also support C-arrays and C-strings.

cout << size("abcde") << endl;

# <numeric>: accumulate, partial\_sum, iota

The sum of the segment specified by a pair of iterators. The type of the return value is determined by the initial value of the sum (sum vector <int64\_t> in 0LL ).

vector<int> v{1, 2, 3, 4};

cout << accumulate(v.begin(), v.end(), 0) << endl; // 10

Prefix amounts:

vector<int> v{1, 2, 3, 4, 5}, res(5);

partial\_sum(v.begin(), v.end(), res.begin()); // 1 3 6 10 15

Serial value generation (prefix ++ ):

vector<int> v(5);

iota(v.begin(), v.end(), -2); // -2 -1 0 1 2

# <cmath>

There are many mathematical functions, a subset popular in the joint venture and non-obvious

## hypot, atan2, pi = atan (1) \* 4

L 2 -distance (hypotenuse of a right triangle), more precisely, but slower than the naive method:

cout << hypot(4, 3) << endl; // 5

The expression atan2 (y, x) considers the angle of direction by (x, y) in] −π, + π], the arc tangent y / x, which takes into account the signs and is not afraid of dividing by 0:

cout << atan2(10, -10) << " radians" << endl; // 2.35619

Pi is still not in the standard. Instead of using compiler-specific macros or knocking handles, you can use

double const pi = atan(1) \* 4; // 3.1415926535897931

cout << setprecision(20) << pi << " " // 3.141592653589793116

<< nexttoward(pi, 3.0) << " " // 3.1415926535897926719

<< nextafter(pi, 4.0) << endl; // 3.1415926535897935601

## round, floor, ceil

Rounding to the nearest integer down (antier down):

cout << floor(2.7) << " " << floor(-2.5) << endl; // 2 -3

up:

cout << ceil(2.7) << " " << ceil(-2.5) << endl; // 3 -2

to the nearest (half-integer further from 0):

cout << round(2.7) << " " << round(-2.5) << endl; // 3 -3

## abs

Absolute value:

cout << abs(-10) << endl; // 10

# <complex>

Complex numbers in SPs can reduce some 2D geometric calculations. For example, generating coordinate offsets on a square grid to 4 neighbors

complex<int> dir(1, 0);

for (int i = 0; i < 4; ++i) {

cout << dir << " " << dir.real() << " + " << dir.imag() << "j" << endl;

// (1,0) 1 + 0j; (0, 1) 0 + 1j; (-1, 0) -1 + 0j; (0, -1) 0 + -1j

dir \*= complex<int>(0, 1);

}

or area of ​​a triangle.

stringstream ss("1 1\n1 4\n5 1");

double x, y;

ss >> x >> y; complex<double> a(x, y);

ss >> x >> y; complex<double> b(x, y);

ss >> x >> y; complex<double> c(x, y);

complex<double> A = b - a, B = c - a;

double S = abs((conj(A) \* B).imag()) / 2.0;

cout << S << endl; // 6

Do not forget that in wartime the value of the sine can reach four.

cout << sin(1.570796327 - 2.063437069i) << endl; // (4,7.94362e-10)

# <limits>: numeric\_limits <int> :: max ()

For any INF-values, instead of pretended numbers like 100500, you can use the maximum and minimum values ​​of the type.

cout << numeric\_limits<int64\_t>::min() << endl; // -9223372036854775808

cout << numeric\_limits<int64\_t>::max() << endl; // 9223372036854775807

# <random>

PRNG is rarely used in a joint venture, but sometimes a randomized solution is difficult to fix. By default, C ++ uses mt19937 .

default\_random\_engine generator;

generator.seed(0);

uniform\_real\_distribution<double> distribution(0.0, 1.0); // аналогично uniform\_int\_distribution<int>

double rnd = distribution(generator);

cout << rnd << endl; // 0.592845

# <utility>: swap

Swap objects:

int x = 5, y = 3;

swap(x, y);

cout << x << " " << y << endl; // 3 5

# <bitset>

A known value bit value can be conveniently inferred using bitset .

cout << bitset<10>(777) << endl; // 1100001001

# <chrono>: std :: chrono :: high\_resolution\_clock :: now ( )

Accurately measure time:

auto t0 = chrono::high\_resolution\_clock::now();

this\_thread::sleep\_for(1.0s);

auto t1 = chrono::high\_resolution\_clock::now();

cout << chrono::duration\_cast<chrono::milliseconds>(t1 - t0).count() << endl; // 999

cout << chrono::duration<double>(t1 - t0).count() << endl; // 0.999376

# <functional>

With the advent of lambda functions in C ++ 11, it became very convenient to write callbacks. To save them into variables or accept in functions, it is useful to use the function template .

stringstream ss("6 5\n0 1\n0 2\n1 3\n1 4\n4 5");

int n, m;

ss >> n >> m;

vector<vector<int>> adjlist(n);

for (int i = 0; i < m; ++i) {

int a, b;

ss >> a >> b;

adjlist[a].push\_back(b);

adjlist[b].push\_back(a);

}

int time = 0;

function<void(int, int, function<void(int, int)>)> dfs =

[&adjlist, &time, &dfs](int v, int from, function<void(int, int)> enter) {

enter(v, time++);

for (int nn : adjlist[v]) {

if (nn != from) {

dfs(nn, v, enter);

}

}

time++;

};

dfs(0, -1, [](int v, int time) {

cout << "Entered " << v << " at time " << time << endl;

});

/\*

Entered 0 at time 0

Entered 1 at time 1

Entered 3 at time 2

Entered 4 at time 4

Entered 5 at time 5

Entered 2 at time 9

\*/

# Compiler-specific: \_\_builtin\_popcount, \_\_builtin\_clz, \_\_builtin\_ctz, \_\_gcd, \_\_int128

Intrinsics are often useful in a joint venture, counting as one assembler opcode the number of set bits in a number ( popcount ), the number of zeros in a binary notation of a number on the left ( clz - count leading zeroes) and on the right ( ctz - count trailing zeroes). Intrinsics are compiler-specific; here are implementations for MSVC and GCC.

#if defined(\_MSC\_VER)

#include <intrin.h>

#define popcount \_\_popcnt

#define popcount64 \_\_popcnt64

#elif defined(\_\_GNUC\_\_)

#define popcount \_\_builtin\_popcount

#define popcount64 \_\_builtin\_popcountll

#endif

#if defined(\_MSC\_VER)

#include <intrin.h>

int clz(uint32\_t x) { unsigned long result = -1; \_BitScanReverse(&result, x); return 31 - result; }

int ctz(uint32\_t x) { unsigned long result = -1; \_BitScanForward(&result, x); return result; }

int clz64(uint64\_t x) { unsigned long result = -1; \_BitScanReverse64(&result, x); return 63 - result; }

int ctz64(uint64\_t x) { unsigned long result = -1; \_BitScanForward64(&result, x); return result; }

#elif defined(\_\_GNUC\_\_)

#define clz \_\_builtin\_clz

#define ctz \_\_builtin\_ctz

#define clz64 \_\_builtin\_clzll

#define ctz64 \_\_builtin\_ctzll

#endif

uint64\_t num = 0x000000F000000000ULL;

cout << "popcount: " << popcount64(num) << endl; // 4

cout << "leading 0s: " << clz64(num) << " trailing: " << ctz64(num) << endl; // 24 36

uint32\_t num2 = 0x000F0000;

cout << "popcount: " << popcount(num2) << endl; // 4

cout << "leading 0s: " << clz(num2) << " trailing: " << ctz(num2) << endl; // 12 16

In C ++ 17 , std :: gcd appeared in <numeric> , but it wasn't there before. In GCC, it was a detail of the std :: rotate implementation , so it is available under the name \_\_gcd . In MSVC, you can take it in boost.

#if defined(\_MSC\_VER)

#include <boost/math/common\_factor.hpp>

using boost::math::gcd;

#elif defined(\_\_GNUC\_\_)

#define gcd \_\_gcd

#endif

cout << gcd(2983479376572795LL, 29837483726583645LL) << endl; // 15

In 64-bit GCC there is a compiler extension \_\_int128 , in MSVC it ​​will also have to be taken from boost.

#if defined(\_MSC\_VER)

#include <boost/multiprecision/cpp\_int.hpp>

typedef boost::multiprecision::int128\_t int128\_t;

#elif defined(\_\_GNUC\_\_)

typedef \_\_int128 int128\_t;

#endif

int128\_t result = int128\_t(2983479376572795LL) \* 29837483726583645LL;

cout << int(result % 1000000007) << endl; // 493398412

**Some Formula**

* Perimeter of a Square = P = 4a

Where a = Length of the sides of a Square

* Perimeter of a Rectangle = P = 2(l+b)

Where, l = Length ; b = Breadth

* Area of a Square = A = a2

Where a = Length of the sides of a Square

* Area of a Rectangle = A = l×b

Where, l = Length ; b = Breadth

* Area of a Triangle = A = ½×b×h

Where, b = base of the triangle ; h = height of the triangle

* Area of a Trapezoid = A = ½×(b1 + b2)×h

Where, b1 & b2 are the bases of the Trapezoid ; h = height of the Trapezoid

* Area of a Circle = A = π×r2
* Circumference of a Circle = A = 2πr

Where, r = Radius of the Circle

* Surface Area of a Cube = S = 6a2

Where, a = Length of the sides of a Cube

* Surface Area of a Cylinder=S=2πrh
* Volume of a Cylinder = V = πr2h

Where, r = Radius of the base of the Cylinder ; h = Height of the Cylinder

* Surface Area of a Cone = S = πr[r+√(h2+r2)]
* Volume of a Cone = V = ⅓×πr2h

Where, r = Radius of the base of the Cone, h = Height of the Cone

* Surface Area of a Sphere = S = 4πr2
* Volume of a Sphere = V = 4/3×πr3

Where, r = Radius of the Sphere

### Area and Perimeter of a Hexagon

Perimeter = a + b1 + b2 + c

Area = (3√3/2 )r2

### Area and Perimeter of an Octagon

* Perimeter = 8a
* Area = ( 2 + 2√2 )a2

**BFS**

#include <bits/tdc++.h>

using namespace std;

vector<int>adj[105];

int vis[105],dis[105];

int bfs(int s,int d){

vis[s] = 1;

dis[s]=0;

queue<int>Q;

Q.push(s);

while(!Q.empty()){

int u = Q.front();

cout<<u<<endl;

if(u==d) return dis[u];

Q.pop();

for(int i=0; i<adj[u].size(); i++){

if(!vis[adj[u][i]]){

vis[adj[u][i]] = 1;

dis[adj[u][i]]=dis[u]+1;

Q.push(adj[u][i]);

}

}

}

// cout<<endl;

}

int main()

{

int n;

cin>>n;

for(int i=1; i<=n-1; i++){

int u,v;

cin>>u>>v;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

int a;

cin>>a;

cout<<bfs(1,a)<<endl;

}

**DFS**

#include <bits/stdc++.h>

using namespace std;

vector<int>adj[105];

int vis[105],dis[105];

void dfs(int s){

vis[s] = 1;

//dis[s]=0;

stack<int>Q;

Q.push(s);

while(!Q.empty()){

int u = Q.top();

cout<<u<<endl;

//if(u==d) return dis[u];

Q.pop();

for(int i=0; i<adj[u].size(); i++){

if(!vis[adj[u][i]]){

vis[adj[u][i]] = 1;

//dis[adj[u][i]]=dis[u]+1;

Q.push(adj[u][i]);

}

}

}

// cout<<endl;

}

int main()

{

int n;

cin>>n;

for(int i=1; i<=n-1; i++){

int u,v;

cin>>u>>v;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

dfs(1);

}

**Divisor**

//inteligent prime factor

#include<bits/stdc++.h>

using namespace std;

int mark[105];

int main()

{

int t,cas=0;

cin>>t;

while(t--){

int x;

cin>>x;

int num=x,c=0,p=0;

for(int j=x;j>1;j--){

c=0;

int n=j;

if(n%2==0) p++;

while(n%2==0){

n=n/2;

c++;

}

mark[2]+=c;

for(int k=3;k<=sqrt(n);k++){

c=0;

if(n%k==0){

p++;

while(n%k==0){

n=n/k;

c++;

}

}

mark[k]+=c;

}

if(n>1) {mark[n]+=1; p++;}

//cout<<mark[3]<<" "<<j<<endl;

}

// cout<<p<<endl;

cout<<"Case "<<++cas<<": "<<num<<" ="<<" "<<2<<" "<<"("<<mark[2]<<")";

for(int m=3;m<105;m++){

if(mark[m]!=0){

cout<<" \*"<<" "<<m<<" "<<"("<<mark[m]<<")";

}

}

cout<<endl;

memset(mark,0,sizeof(mark));

}

}

**Sieve of Eratosthenes**

void SieveOfEratosthenes(int n)

{

// Create a boolean array "prime[0..n]" and initialize

// all entries it as true. A value in prime[i] will

// finally be false if i is Not a prime, else true.

bool prime[n+1];

memset(prime, true, sizeof(prime));

for (int p=2; p\*p<=n; p++)

{

// If prime[p] is not changed, then it is a prime

if (prime[p] == true)

{

// Update all multiples of p

for (int i=p\*2; i<=n; i += p)

prime[i] = false;

}

}

// Print all prime numbers

for (int p=2; p<=n; p++)

if (prime[p])

cout << p << " ";

}

**Phi Algorithm**

#include <bits/stdc++.h>

#define ll long long

**using** **namespace** std;

#define MAXN 5000007

unsigned ll phi[MAXN+1];

void sieve()

{

for(ll i=2; i<=MAXN; i++)

phi[i] = i;

for(ll i =2; i<=MAXN; i++)

{

if(phi[i]==i)

{

for(ll j=i; j<=MAXN; j+=i)

phi[j]-=phi[j]/i;

}

}

}

int main()

{

ios\_base::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

sieve();

phi[0]=0;

for(ll i=1;i<MAXN;i++) {

phi[i]=phi[i-1]+(phi[i]\*phi[i]);

}

int t,m,n,cas=0;

scanf("%d",&t);

while(t--){

scanf("%d %d",&n,&m);

printf("Case %d: %llu**\n**",++cas,phi[m]-phi[n-1]);

}

}

**Knapsack**

#include<bits/stdc++.h>

using namespace std;

long long dp[101][100001],n,w,value[9999],weight[9999];

long long ans(long long pos, long long we){

long long profit,profit1;

if(pos==n) return 0;

if(dp[pos][we]!=-1) return dp[pos][we];

if(we+weight[pos]<=w){

profit=value[pos]+ans(pos+1,we+weight[pos]);

}

else profit=0;

profit1=ans(pos+1,we);

return dp[pos][we]=max(profit,profit1);

}

int main()

{

memset(dp,-1,sizeof(dp)); cin>>n>>w;

for(int i=0;i<n;i++){

cin>>weight[i]>>value[i];

}

cout<<ans(0,0)<<endl;

}

**Pair Sort**

#include<bits/stdc++.h>

using namespace std;

bool hemel(pair<int,int> a,pair<int,int> b)

{

return (a.second<b.second);

}

bool sortbysec(const pair<int,int> &a, const pair<int,int> &b) {

if(a.first==b.first) return(a.second < b.second);

else return (a.first<b.first);

}

int main()

{

vector<pair<int,int> >v;

for(int i=0;i<10;i++)

{

int x; cin>>x;

v.push\_back(make\_pair(i,x));

}

sort(v.begin(),v.end(),hemel);

for(int i=0;i<v.size();i++)

{

cout<<v[i].first<<" "<<v[i].second<<endl;

}

}

**Bitmask**

int w[20][20];

int n;

int dp[(1<<15)+2];

int call(int mask)

{

if(mask==(1<<n)-1) return 0;

if(dp[mask]!=-1) return dp[mask];

int mn=1<<28;

for(int i=0;i<n;i++)

{

if(check(mask,i)==0)

{

int price=w[i][i];

for(int j=0;j<n;j++)

{

if(i!=j and check(mask,j)!=0)

{

price+=w[i][j];

}

}

int ret=price+call(Set(mask,i));

mn=min(mn,ret);

}

}

return dp[mask]=mn;

}

int main()

{

mem(dp,-1);

cin>>n;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

scanf("%d",&w[i][j]);

}

}

int ret=call(0);

printf("%d\n",ret);

return 0;

}

**Neighbor house**

#include<bits/stdc++.h>

#define ll long long

#define dou double

#define big 0

#define pi 2\*acos(0.0)

using namespace std;

ll arr1[25],arr2[25],arr3[25];

ll dp[1009][25];

ll x,m1=0,m2=0;

ll ans(ll pos, ll rev){

if(pos>x-1){

return 0;

}

if(dp[pos][rev]!=0) return dp[pos][rev];

ll m1=0,m2=0;

if(rev==1){

m1=m1+arr1[pos]+ans(pos+1,2);

m2=m2+arr1[pos]+ans(pos+1,3);

}

else if(rev==2){

m1=m1+arr2[pos]+ans(pos+1,1);

m2=m2+arr2[pos]+ans(pos+1,3);

}

else{

m1=m1+arr3[pos]+ans(pos+1,1);

m2=m2+arr3[pos]+ans(pos+1,2);

}

return dp[pos][rev]=min(m1,m2);

}

int main()

{

ll t;

cin>>t;

ll cas=0;

while(t--){

cin>>x;

for(int i=0;i<x;i++){

cin>>arr1[i]>>arr2[i]>>arr3[i];

}

ll s1=ans(0,1);

m1=0;

memset(dp,0,sizeof(dp));

ll s2=ans(0,2);

memset(dp,0,sizeof(dp));

m1=0;

ll s3=ans(0,3);

memset(dp,0,sizeof(dp));

//cout<<s1<<" "<<s2<<" "<<s3<<endl;

cout<<"Case "<<++cas<<": "<<min(s1,min(s2,s3))<<endl;

memset(dp,0,sizeof(dp));

m1=0;

}

}

**Big Mod**

long long bigmod ( long long a, long long p, long long m )

{

    long long res = 1;

    long long x = a;

    while ( p ){

        if ( p & 1 ) //p is odd

        {

            res = ( res \* x ) % m;

        }

        x = ( x \* x ) % m;

        p = p >> 1;

    }

    return res;

}

[**Last non-zero digit of factorial**](http://alavolacodes.blogspot.com/2013/06/last-non-zero-digit-of-factorial.html)

#include<stdio.h>

int main()

{

    long long int ans;

        int num;

        scanf("%d",&num);

        ans=1;

        for(int i=2;i<=num;i++)

        {

                ans=ans\*i;

                while(ans%10==0)

                {

                        ans/=10;

                }

                ans%=10000000;

        }

        printf("%lld\n",ans%10);

        return 0;

}

[**Trailing zero in factorial**](http://alavolacodes.blogspot.com/2013/06/trailing-zero-in-factorial.html)

#include <stdio.h>

int main()

{

    int n,total,deno;

        scanf("%d",&n);

    total=0;

    deno=5;

    while(deno<=n)

    {

        total+=n/deno;

        deno\*=5;

    }

    printf("%d\n",total);

    return 0;

}

[**Sum of Divisors of a Number**](http://alavolacodes.blogspot.com/2013/06/sum-of-divisors-of-number.html)

#define i64 long long

i64 power(i64 N,i64 P)

{

    i64 sum=1,i;

    if(P==0)return 1;

    else

    {

        for(i=1;i<=P;i++)

            sum=sum\*N;

        return sum;

    }

}

i64 sumofdivisor(i64 n)

{

    i64 sum=1,i,count;

    i64 sq=(i64)sqrt(n);

    for(i=0;prime[i]<=sq;i++)

    {

        count=0;

        while(n%prime[i]==0)

        {

            count++;

            n/=prime[i];

        }

        sum\*=(i64)(power(prime[i],count+1)-1)/(prime[i]-1);

    }

    if(n>1)

        sum\*=(n+1);

    return sum;

}

**Number Of Divisors**

int number\_of\_divisor(int num)

{

 int j,count,div=1;

 for(j=0;prime[j]<=sqrt(num);j++) //prime array holds the prime numbers

 {

  count=0;

  while(num%prime[j]==0)

  {

   count++;

   num/=prime[j];

  }

  div\*=(count+1);

 }

 if(num>1)

  div<<=1;

 return div;

}

**Number of Digits in Factorial**

#include <stdio.h>

#include <math.h>

int main()

{

    int number,i,base,digit;

    double c,a=0;

    printf("Give the base and the number: ");

    scanf ("%d%d",&base,&number);

    c=log10(base);

    for (i=number;i>=1;i--) {

        a+=(log10(i))/c;

    }

    digit=a+1;

    printf ("%d\n",digit);

    return 0;

}

**Factorial of a Large Number**

#include<iostream>

using namespace std;

// Maximum number of digits in output

#define MAX 500

int multiply(int x, int res[], int res\_size);

// This function finds factorial of large numbers

// and prints them

void factorial(int n)

{

int res[MAX];

// Initialize result

res[0] = 1;

int res\_size = 1;

// Apply simple factorial formula n! = 1 \* 2 \* 3 \* 4...\*n

for (int x=2; x<=n; x++)

res\_size = multiply(x, res, res\_size);

cout << "Factorial of given number is \n";

for (int i=res\_size-1; i>=0; i--)

cout << res[i];

}

// This function multiplies x with the number

// represented by res[].

// res\_size is size of res[] or number of digits in the

// number represented by res[]. This function uses simple

// school mathematics for multiplication.

// This function may value of res\_size and returns the

// new value of res\_size

int multiply(int x, int res[], int res\_size)

{

int carry = 0; // Initialize carry

// One by one multiply n with individual digits of res[]

for (int i=0; i<res\_size; i++)

{

int prod = res[i] \* x + carry;

// Store last digit of 'prod' in res[]

res[i] = prod % 10;

// Put rest in carry

carry = prod/10;

}

// Put carry in res and increase result size

while (carry)

{

res[res\_size] = carry%10;

carry = carry/10;

res\_size++;

}

return res\_size;

}

// Driver program

int main()

{

factorial(100);

return 0;

}

**Binary Exponentiation**

ll binpow(ll base, ll power, ll mod){

base=base%mod;

ll res=1;

while(power>0){

if(power&1) res=(res\*base)%mod;

base=(base\*base)%mod;

power>>=1;

}

return res;

}

**Extended GCD**

ll ext\_gcd(ll a,ll b,ll &x,ll &y){

if(a==0){

x=0;

y=1;

return b;

}

ll x1,y1;

ll d=ext\_gcd(b%a,a,x1,y1);

x=y1-(b/a)\*x1;

y=x1;

return d;

}

**Inverse Mod**

ll modInv (ll a,ll m){

ll x,y;

ext\_gcd(a,m,x,y);

x%=m;

if(x<0) x+=m;

return x;

}

int mod\_exp(int n, int p)

{

if(p==-1) p=mod-2;

int ret = 1;

while(p){

if(p&1) ret = (ret\*1LL\*n)%mod;

p>>=1;

n=(n\*1LL\*n)%mod;

}

return ret;

}

**Finding frequency of a number**

//globally

unordered\_map< int, vector<int> > store;

int findFrequency(int arr[],int n,int left,int right,int element)

{

int a = lower\_bound(store[element].begin(),store[element].end(),left)- store[element].begin();

int b = upper\_bound(store[element].begin(),store[element].end(),right)- store[element].begin();

return b-a;

}

//inside main function

for(int i=0;i<n;++i)store[arr[i]].push\_back(i);

cout<< findFrequency(arr, n, b, p, v[0]) <<endl;

store.clear();

# **Find the smallest window in a string containing all characters of another string**

const int no\_of\_chars = 256;

string findSubString(string str, string pat)

{

int len1 = str.length();

int len2 = pat.length();

if (len1 < len2) cout << "No such window exists"; return "";

int hash\_pat[no\_of\_chars] = {0};

int hash\_str[no\_of\_chars] = {0};

for (int i = 0; i < len2; i++) hash\_pat[pat[i]]++;

int start = 0, start\_index = -1, min\_len = INT\_MAX;

int count = 0; // count of characters

for (int j = 0; j < len1 ; j++) {

hash\_str[str[j]]++;

if (hash\_pat[str[j]] != 0 &&

hash\_str[str[j]] <= hash\_pat[str[j]] )

count++;

if (count == len2){.

while ( hash\_str[str[start]] > hash\_pat[str[start]] || hash\_pat[str[start]] == 0){

if (hash\_str[str[start]] > hash\_pat[str[start]])

hash\_str[str[start]]--;

start++;

}

int len\_window = j - start + 1;

if (min\_len > len\_window) {

min\_len = len\_window; start\_index = start;

}

}

}

if (start\_index == -1)cout << "No such window exists"; return "";

}

return str.substr(start\_index, min\_len);

}

// Driver code

int main()

{

string str = "this is a test string";

string pat = "tist";

cout << "Smallest window is : \n"

<< findSubString(str, pat);

return 0;

}

# **Smallest number with at least n trailing zeros in factorial**

#include<bits/stdc++.h>

using namespace std;

bool check(int p, int n){

int temp = p, count = 0, f = 5;

while (f <= temp){

count += temp/f;

f = f\*5;

}

return (count >= n);

}

int findNum(int n){

if (n==1) return 5;.

int low = 0;

int high = 5\*n;

while (low <high){

int mid = (low + high) >> 1;

if (check(mid, n)) high = mid;

else low = mid+1;

}

return low;

}

int main()

{

int n = 6;

cout << findNum(n) << endl;

return 0;

}

# **Prime Factorization using Sieve O(log n) for multiple queries**.

#include "bits/stdc++.h"

using namespace std;

#define MAXN 100001

int spf[MAXN];

void sieve(){

spf[1] = 1;

for (int i=2; i<MAXN; i++) spf[i] = i;

for (int i=4; i<MAXN; i+=2) spf[i] = 2;

for (int i=3; i\*i<MAXN; i++){

if (spf[i] == i) {

for (int j=i\*i; j<MAXN; j+=i)

if (spf[j]==j)

spf[j] = i;

}

}

}

vector<int> getFactorization(int x){

vector<int> ret;

while (x != 1){

ret.push\_back(spf[x]);

x = x / spf[x];

}

return ret;

}

int main(int argc, char const \*argv[])

{

sieve();

int x = 12246;

cout << "prime factorization for " << x << " : ";

vector <int> p = getFactorization(x);

for (int i=0; i<p.size(); i++)

cout << p[i] << " ";

cout << endl;

return 0;

}

**Inverse Factorial / nCr of two number**

#include<bits/stdc++.h>

using namespace std;

const int maxn = 1022, mod = 1e9 + 7;

int fac[maxn];

int mod\_exp(int n, int p)

{

if(p==-1) p=mod-2;

int ret = 1;

while(p){

if(p&1) ret = (ret\*1LL\*n)%mod;

p>>=1;

n=(n\*1LL\*n)%mod;

}

return ret;

}

void init()

{

fac[0] = 1;

for(int i=1;i<maxn;i++) fac[i] = (fac[i-1]\*1LL\*i)%mod;

}

int nCr(int n, int r)

{

return ((fac[n]\*1LL\*mod\_exp(fac[r], -1))%mod)\*mod\_exp(fac[n-r], -1)%mod;

}

int main()

{

int n, m;

init();

cin>>n>>m;

cout << nCr(n+2\*m-1, n-1);

return 0;

}

# **count of divisor of a factorial (also ncr)**

//first we have to find the prime factorization count of that number

//we have to found out count after ncr calculation

//then add 1 with the count

//calculate if need more

#include<stdio.h>

#define ll long long

int prime[90], range=0;

int vis[500];

void inti(){

int i,j;

for(i=2; i\*i<=431; i++){

if(vis[i]==0){

for(j=i\*i;j<=431;j+=i){

vis[j]=1;

}

}

}

for(i=2;i<=431;i++){

if(vis[i]==0){

//printf("%d\n", i);

prime[range++]=i;

}

}

}

int cal(int n, int p){

if(n<p) return 0;

else return(n/p + cal(n/p, p));

}

int main(){

//inti();

int n,k,i;

\_\_int64 result =1,pre;

inti();

//printf("%d\n", range);

while(scanf("%d%d",&n,&k)==2){

result=1;

if(k==0||k==n){

printf("1\n"); continue;

}

for(i=0;i<range&&prime[i]<=n;i++){

pre=cal(n,prime[i]) - cal(n-k,prime[i]) - cal(k,prime[i]);

result = result \* (pre+1);

}

printf("%I64d\n", result);

}

return 0;

}

**How many pair gcd greater than 1**

#include<bits/stdc++.h>

#define ll long long

#define maxx 100009

using namespace std;

ll phi[maxx];

ll mark[maxx];

ll cu\_phi[maxx];

void phi\_sum(){

cu\_phi[1]=1;

cu\_phi[2]=1;

for(int i=2;i<=maxx;i++) {cu\_phi[i]=phi[i]+cu\_phi[i-1];}

}

void sieve\_phi(){

for(int i=1;i<=maxx;i++) phi[i]=i;

phi[1]=1;

mark[1]=1;

for(int i=2;i<=maxx;i++){

if(!mark[i]){

for(int j=i;j<=maxx;j+=i){

mark[j]=1;

phi[j]=phi[j] / i\*(i-1);

}

}

}

}

int main()

{

sieve\_phi(); phi\_sum();

int t,cas=0; cin>>t;

while(t--){

ll x; cin>>x;

cout<<"Case "<<++cas<<": "<<((x\*(x+1))/2)-cu\_phi[x]<<endl;

}

}

**Divisor of a number using Prime Factorization (NOD)**

int main()

{

prime\_number();

int test\_case,case\_no=0;

scanf("%d",&test\_case);

while(test\_case--){

long long number; cin>>number;

int counting,total=1;

for(int i=0;i<storage.size() and storage[i]<=sqrt(number);i++){

counting=0;

if(number<storage[i]) break;

while(number%storage[i]==0){

number=number/storage[i];

counting++;

}

total=total\*(counting+1);

}

if(number>1) total=total\*2;

printf("Case %d: %d\n",++case\_no,total-1);

}

}

**Formula of Sum of Divisor of a Number (SOD)**

SOD = (((prime1)^(prime1\_power+1))-1)/(prime1-1) \* (((prime2)^(prime2\_power+1))-1)/(prime2-1) \* ……\* (((primeN)^(primeN\_power+1))-1)/(primeN-1)

N = last prime number that divides the number.

**Trie (prefix checking of string)**

#include<bits/stdc++.h>

using namespace std;

struct trie{

int stop; trie \*next[10];

trie(){

stop=0;

for(int i=0;i<10;i++) next[i]=NULL;

}

}\*root;

bool insert(string s, trie \*cur){

int l=s.length();

bool test=0;

for(int i=0;i<l;i++){

int now=s[i]-'0';

if(cur->next[now]==NULL) cur->next[now]=new trie();

if(cur->stop){

test=1; break;

}

cur=cur->next[now];

}

cur->stop=1;

return test;

}

void del(trie \*cur){

for(int i=0;i<10;i++){

if(cur->next[i]) del(cur->next[i]);

}

delete(cur);

}

int main()

{

int t,cas=0; cin>>t;

while(t--){

root = new (trie);

string s;

int x; cin>>x;

vector<string> v;

bool test = 0;

for(int i=0;i<x;i++){

cin>>s;

v.push\_back(s);

}

sort(v.begin(),v.end());

for(int i=0;i<x;i++){

test = insert(v[i],root);

if(test) break;

}

cout<<"Case "<<++cas<<": ";

if(test) cout<<"NO"<<endl;

else cout<<"YES"<<endl;

del(root);

}

}

**Trie++**

#include<bits/stdc++.h>

#define ll long long

**using** **namespace** std;

int res=1;

struct trie{

int stop;

trie \*next[4];

int cnt[4];

**trie(){**

stop = 0;

for(int i=0;i<4;i++){

next[i] = **NULL**;

}

**for(int i=0;i<4;i++) cnt[i]=0;**

}

}\*root;

bool insert(string s, trie \*cur){

**int l=s.length();** bool test = 0;

for(int i=0;i<l;i++){

*// int now = s[i]-'A';*

*// if(cur->next[now]==NULL){*

***// cur->next[now]=new trie();***

*// }*

*// if(cur->stop){*

*// test=1;*

*// break;*

***// }***

int point;

if(s[i]=='A') point=0;

else if(s[i]=='C') point=1;

if(s[i]=='G') point=2;

**if(s[i]=='T') point=3;**

cur->cnt[point]++;

if(cur->next[point]==**NULL**){

cur->next[point]=new trie();

}

**res=max(res, cur->cnt[point]\*(i+1));**

cur=cur->next[point];

}

cur->stop=1;

return test;

**}**

void del(trie \*cur){

for(int i=0;i<4;i++){

if(cur->next[i]) **del(cur->next[i]);**

}

for(int i=0;i<4;i++) if(cur->cnt[i]) cur->cnt[i]=0;

delete (cur);

**}**

**template** <**typename** Container>

bool LexCompare(const Container& a, const Container& b) {

return std::lexicographical\_compare(a.begin(), a.end(),

b.begin(), b.end());

**}**

**template** <**typename** ContainerIterator>

void sort\_by\_lexicographical\_comapre(ContainerIterator beg,

ContainerIterator end)

**{**

std::sort(beg, end, LexCompare<**typename** ContainerIterator::value\_type>);

}

**int main()**

{

int t,cas=0;

cin>>t;

while(t--){

**res=1;**

root = new (trie);

string s;

int x;

cin>>x;

**getchar();**

std::vector<string> v;

bool test = 0;

for(int i=0;i<x;i++){

cin>>s;

**v.push\_back(s);**

}

*//sort(v.begin(),v.end(),);*

sort\_by\_lexicographical\_comapre(v.begin(), v.end());

for(int i=0;i<x;i++){

**test = insert(v[i],root);**

if(test) break;

}

cout<<"Case "<<++cas<<": "<<res<<endl;

del(root);

**}**

}

# **Smallest subarray with k distinct numbers**

//n= size of array

//k= size of the length/unique element number/the size where you want to find most of the unique number

void minRange(long long arr[], long long n, long long k){

ll l = 0, r = n;

for (ll i = 0; i < n; i++) {

unordered\_set<ll> s;

ll j;

for (j = i; j < n; j++) {

s.insert(arr[j]);

if (s.size() == k) {

if ((j - i) < (r - l)) {

r = j;

l = i;

}

break;

}

}

if (j == n)

break;

}

cout << l << " " << r<<endl;

}

int main()

{

int t,cas=0;

cin>>t;

while(t--){

int x; cin>>x;

long long arr[x];

map<long long , long long> m;

long long k=0;

for(int i=0;i<x;i++){

cin>>arr[i];

if(m[arr[i]]==0){

k++;

m[arr[i]]=1;

}

}

cout<<"Case "<<++cas<<": ";

minRange(arr, x, k);

}

return 0;

}