Universidade Federal de Pernambuco

las4s e pelados

Icaro Copo Papel Nunes, Joao Pou Grangeiro, Pedro Grisi

2025-11-15

Contest (1)

template.cpp

14 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc

2 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
```

hash.sh

2 lines

```
# bash hash.sh file.cpp l1 l2
sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -
     fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Data structures (2)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null_type.
Time: $\mathcal{O}(\log N)$

782797, 17 lines

```
c4d  #include <bits/extc++.h>
0d7  using namespace __gnu_pbds;

4fc  template<class T>
c20  using Tree = tree<T, null_type, less<T>, rb_tree_tag,
3a1      tree_order_statistics_node_update>;

ad0  void example() {
c6f   Tree<int> t, t2; t.insert(8);
559   auto it = t.insert(10).first;
d28   assert(it == t.lower_bound(9));
969   assert(t.order_of_key(10) == 1);
d39   assert(t.order_of_key(11) == 2);
1b7   assert(*t.find_by_order(0) == 8);
a60   t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
9ad  }
```

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 8 lines

```
c4d  #include <bits/extc++.h>
     // To use most bits rather than just the lowest ones:
75f  struct chash { // large odd number for C
5d6   const uint64_t C = ll(4e18 * acos(0)) | 71;
2cf   ll operator()(ll x) const { return __builtin_bswap64(x*C)
     ; }
cdd  };
911  __gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{
     1<<16});
```

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
Time: $\mathcal{O}(\log N)$

0f4bdb, 20 lines

```
5ae  struct Tree {
ef4   typedef int T;
cbe   static constexpr T unit = INT_MIN;
e54   T f(T a, T b) { return max(a, b); } // (any associative
     fn)
6cd   vector<T> s; int n;
3d2   Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
6a3   void update(int pos, T val) {
56a     for (s[pos += n] = val; pos /= 2;)
326       s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
0e9   }
b4c   T query(int b, int e) { // query [b, e)
0f9     T ra = unit, rb = unit;
fbb     for (b += n, e += n; b < e; b /= 2, e /= 2) {
e83       if (b % 2) ra = f(ra, s[b++]);
064       if (e % 2) rb = f(s[--e], rb);
561     }
cb2     return f(ra, rb);
5b1   }
0f4  };
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: `Node* tr = new Node(v, 0, sz(v));`
Time: $\mathcal{O}(\log N)$.

"../various/BumpAllocator.h" 34ecf5, 51 lines

```
1a8  const int inf = 1e9;
bf2  struct Node {
d65   Node *l = 0, *r = 0;
938   int lo, hi, mset = inf, madd = 0, val = -inf;
e9c   Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of
     -inf
7ae   Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
cf3     if (lo + 1 < hi) {
7f0       int mid = lo + (hi - lo)/2;
c0a       l = new Node(v, lo, mid); r = new Node(v, mid, hi);
8da       val = max(l->val, r->val);
0ad     }
cb4     else val = v[lo];
34b   }
2dc   int query(int L, int R) {
7be     if (R <= lo || hi <= L) return -inf;
580     if (L <= lo && hi <= R) return val;
215     push();
8d7     return max(l->query(L, R), r->query(L, R));
f1d   }
b5b   void set(int L, int R, int x) {
d94     if (R <= lo || hi <= L) return;
7e2     if (L <= lo && hi <= R) mset = val = x, madd = 0;
4e6     else {
d2a       push(), l->set(L, R, x), r->set(L, R, x);
8da       val = max(l->val, r->val);
032     }
12a   }
634   void add(int L, int R, int x) {
d94     if (R <= lo || hi <= L) return;
60d     if (L <= lo && hi <= R) {
c27       if (mset != inf) mset += x;
61f       else madd += x;
c61       val += x;
a79     }
```

```
4e6     else {
fd7       push(), l->add(L, R, x), r->add(L, R, x);
8da       val = max(l->val, r->val);
1bf     }
aee   }
ecf   void push() {
268     if (!l) {
7f0       int mid = lo + (hi - lo)/2;
fde       l = new Node(lo, mid); r = new Node(mid, hi);
612     }
90f     if (mset != inf)
389       l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
5ce     else if (madd)
ab7       l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
4bc   }
079 };
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$ de4ad0, 22 lines

```
47a struct RollbackUF {
724   vi e; vector<pii> st;
f6f   RollbackUF(int n) : e(n, -1) {}
84b   int size(int x) { return -e[find(x)]; }
626   int find(int x) { return e[x] < 0 ? x : find(e[x]); }
49f   int time() { return sz(st); }
4db   void rollback(int t) {
314     for (int i = time(); i --> t;)
8d2       e[st[i].first] = st[i].second;
b04     st.resize(t);
30b   }
cf0   bool join(int a, int b) {
605     a = find(a), b = find(b);
5c2     if (a == b) return false;
745     if (e[a] > e[b]) swap(a, b);
bac     st.push_back({a, e[a]});
e6e     st.push_back({b, e[b]});
708     e[a] += e[b]; e[b] = a;
8a6     return true;
6c7   }
de4 };
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}(N^2 + Q)$ 337bb3, 69 lines

```
eaf int lcs_s[MAX], lcs_t[MAX];
a6d int dp[2][MAX];

    // dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
d12 void dp_top(int li, int ri, int lj, int rj) {
d13   memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
753   for (int i = li; i <= ri; i++) {
9aa     for (int j = rj; j >= lj; j--)
83b       dp[0][j - lj] = max(dp[0][j - lj],
741         (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 - lj] :
    0));
04c     for (int j = lj+1; j <= rj; j++)
939       dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1 -lj]);
09f   }
58f }

    // dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
ca0 void dp_bottom(int li, int ri, int lj, int rj) {
0dd   memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
3a2   for (int i = ri; i >= li; i--) {
49c     for (int j = lj; j <= rj; j++)
dbb       dp[1][j - lj] = max(dp[1][j - lj],
4da         (lcs_s[i] == lcs_t[j]) +  (j < rj ? dp[1][j+1 - lj] :
    0));
6ca     for (int j = rj-1; j >= lj; j--)
769       dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 - lj]);
19b   }
e8a }

93c void solve(vector<int>& ans, int li, int ri, int lj, int
    rj) {
2ad   if (li == ri){
49c     for (int j = lj; j <= rj; j++)
f5b       if (lcs_s[li] == lcs_t[j]){
a66         ans.push_back(lcs_t[j]);
c2b         break;
840       }
505     return;
126   }
534   if (lj == rj){
753     for (int i = li; i <= ri; i++){
88f       if (lcs_s[i] == lcs_t[lj]){
531         ans.push_back(lcs_s[i]);
c2b         break;
68a       }
a03     }
505     return;
76d   }
a57   int mi = (li+ri)/2;
ade   dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

d7a   int j_ = 0, mx = -1;

aee   for (int j = lj-1; j <= rj; j++) {
da8     int val = 0;
2bb     if (j >= lj) val += dp[0][j - lj];
b9e     if (j < rj) val += dp[1][j+1 - lj];

ba8     if (val >= mx) mx = val, j_ = j;
14e   }
6f1   if (mx == -1) return;
c2a   solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_+1, rj
    );
dd5 }

058 vector<int> lcs(const vector<int>& s, const vector<int>& t
    ) {
953   for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
577   for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
dab   vector<int> ans;
599   solve(ans, 0, s.size()-1, 0, t.size()-1);
ba7   return ans;
17c }
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`array<int, 3> vec = {1,2,3};`
`vec = (A^N) * vec;` 6ab5db, 27 lines

```
a95 template<class T, int N> struct Matrix {
db4   typedef Matrix M;
c89   array<array<T, N>, N> d{};
02c   M operator*(const M& m) const {
0f2     M a;
1c2     rep(i,0,N) rep(j,0,N)
a68       rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
3f5     return a;
7d2   }
01b   array<T, N> operator*(const array<T, N>& vec) const {
b58     array<T, N> ret{};
a29     rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
edf     return ret;
bfa   }
70f   M operator^(ll p) const {
5d8     assert(p >= 0);
ccf     M a, b(*this);
72e     rep(i,0,N) a.d[i][i] = 1;
d08     while (p) {
7ae       if (p&1) a = a*b;
e04       b = b*b;
8b8       p >>= 1;
12e     }
3f5     return a;
5ae   }
6ab };
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$ 8ec1c7, 31 lines

```
72c struct Line {
3e2   mutable ll k, m, p;
ca5   bool operator<(const Line& o) const { return k < o.k; }
abf   bool operator<(ll x) const { return p < x; }
7e3 };

781 struct LineContainer : multiset<Line, less<>> {
      // (for doubles, use inf = 1/.0, div(a,b) = a/b)
fd2   static const ll inf = LLONG_MAX;
33a   ll div(ll a, ll b) { // floored division
10f     return a / b - ((a ^ b) < 0 && a % b); }
a1c   bool isect(iterator x, iterator y) {
a95     if (y == end()) return x->p = inf, 0;
9cb     if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
591     else x->p = div(y->m - x->m, x->k - y->k);
870     return x->p >= y->p;
2fa   }
a0c   void add(ll k, ll m) {
116     auto z = insert({k, m, 0}), y = z++, x = y;
7b1     while (isect(y, z)) z = erase(z);
141     if (x != begin() && isect(--x, y)) isect(x, y = erase(y
    ));
57d     while ((y = x) != begin() && (--x)->p >= y->p)
774       isect(x, erase(y));
086   }
4ad   ll query(ll x) {
229     assert(!empty());
7d1     auto l = *lower_bound(x);
96a     return l.k * x + l.m;
d21   }
577 };
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$ 39494b, 20 lines

```
547 ll dp[MAX][2];

94b void solve(int k, int l, int r, int lk, int rk) {
de6   if (l > r) return;
```

```
109   int m = (l+r)/2, p = -1;
d2b   auto& ans = dp[m][k&1] = LINF;
6e2   for (int i = max(m, lk); i <= rk; i++) {
07b     ll at = dp[i+1][~k&1] + query(m, i);
57d     if (at < ans) ans = at, p = i;
8f5   }
1ee   solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
d3e }

cf1 ll DC(int n, int k) {
321   dp[n][0] = dp[n][1] = 0;
f27   for (int i = 0; i < n; i++) dp[i][0] = LINF;
b76   for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0, n-i);
8e7   return dp[0][k&1];
5e9 }
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

e62fac, 23 lines

```
066 struct FT {
cf7   vector<ll> s;
f03   FT(int n) : s(n) {}
cfe   void update(int pos, ll dif) { // a[pos] += dif
3e6     for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
a38   }
c6a   ll query(int pos) { // sum of values in [0, pos)
cd2     ll res = 0;
d2a     for (; pos > 0; pos &= pos - 1) res += s[pos-1];
b50     return res;
6de   }
6d8   int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
        // Returns n if no sum is >= sum, or -1 if empty sum is .
4b6     if (sum <= 0) return -1;
bec     int pos = 0;
888     for (int pw = 1 << 25; pw; pw >>= 1) {
4c6       if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
7a3         pos += pw, sum -= s[pos-1];
63f     }
d75     return pos;
ea7   }
e62 };
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 157f07, 23 lines

```
9a3 struct FT2 {
880   vector<vi> ys; vector<FT> ft;
6a4   FT2(int limx) : ys(limx) {}
5a4   void fakeUpdate(int x, int y) {
083     for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
01f   }
ca2   void init() {
a7a     for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
d5c   }
826   int ind(int x, int y) {
aee     return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()
      ); }
eb5   void update(int x, int y, ll dif) {
a1f     for (; x < sz(ys); x |= x + 1)
593       ft[x].update(ind(x, y), dif);
bb1   }
cdc   ll query(int x, int y) {
5ff     ll sum = 0;
14f     for (; x; x &= x - 1)
99b       sum += ft[x-1].query(ind(x-1, y));
e66     return sum;
833   }
157 };
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V| \log |V| + Q)$

510c32, 17 lines

```
4fc template<class T>
76a struct RMQ {
b0a   vector<vector<T>> jmp;
38e   RMQ(const vector<T>& V) : jmp(1, V) {
a1b     for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k)
        {
9d6       jmp.emplace_back(sz(V) - pw * 2 + 1);
939       rep(j,0,sz(jmp[k]))
d44         jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
288     }
e0a   }
0ad   T query(int a, int b) {
c7b     assert(a < b); // or return inf if a == b
e13     int dep = 31 - __builtin_clz(b - a);
7d3     return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
a3d   }
747 };
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}(N\sqrt{Q})$

a12ef4, 50 lines

```
ddb void add(int ind, int end) { ... } // add a[ind] (end = 0
    or 1)
291 void del(int ind, int end) { ... } // remove a[ind]
5dd int calc() { ... } // compute current answer

aed vi mo(vector<pii> Q) {
903   int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
e06   vi s(sz(Q)), res = s;
a09 #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk &
    1))
0af   iota(all(s), 0);
c43   sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
      });
476   for (int qi : s) {
7f7     pii q = Q[qi];
a3d     while (L > q.first) add(--L, 0);
a58     while (R < q.second) add(R++, 1);
6b7     while (L < q.first) del(L++, 0);
e4a     while (R > q.second) del(--R, 1);
806     res[qi] = calc();
0f7   }
b50   return res;
e37 }

c35 vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int
    root=0){
233   int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
ace   vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
74e   add(0, 0), in[0] = 1;
8e6   auto dfs = [&](int x, int p, int dep, auto& f) -> void {
a07     par[x] = p;
41b     L[x] = N;
2fe     if (dep) I[x] = N++;
86b     for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
340     if (!dep) I[x] = N++;
08a     R[x] = N;
329   };
219   dfs(root, -1, 0, dfs);
77f #define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk
    & 1))
0af   iota(all(s), 0);
c43   sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]);
      });
7b9   for (int qi : s) rep(end,0,2) {
ebe     int &a = pos[end], b = Q[qi][end], i = 0;
25d #define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
383                   else { add(c, end); in[c] = 1; } a = c;
    }
729     while (!(L[b] <= L[a] && R[a] <= R[b]))
efe       I[i++] = b, b = par[b];
dd2     while (a != b) step(par[a]);
82e     while (i--) step(I[i]);
1fc     if (end) res[qi] = calc();
c88   }
b50   return res;
ce9 }
```

Combinatorial (3)

3.1 Permutations

3.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

044568, 7 lines

```
aeb int permToInt(vi& v) {
fe8   int use = 0, i = 0, r = 0;
1d8   for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<
      x)),
7ca     use |= 1 << x;                  // (note: minus, not
      ~!)
4c1   return r;
044 }
```

3.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

3.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

3.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

3.2 Partitions and subsets

3.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

3.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod{p}$.

3.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$.

a0a312, 6 lines

```
f7d  ll multinomial(vi& v) {
015    ll c = 1, m = v.empty() ? 1 : v[0];
74f    rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
807    return c;
a0a  }
```

3.3 General purpose numbers

3.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able). $B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

3.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

3.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

3.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

3.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

3.3.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

3.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Various (4)

4.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

Time: $\mathcal{O}(\log N)$

edce47, 24 lines

```
d91  set<pii>::iterator addInterval(set<pii>& is, int L, int R)
      {
bb3   if (L == R) return is.end();
d4c   auto it = is.lower_bound({L, R}), before = it;
dc6   while (it != is.end() && it->first <= R) {
164     R = max(R, it->second);
1a5     before = it = is.erase(it);
fe9   }
1af   if (it != is.begin() && (--it)->second >= L) {
3ca     L = min(L, it->first);
164     R = max(R, it->second);
861     is.erase(it);
0de   }
aa0   return is.insert(before, {L,R});
d57  }

675  void removeInterval(set<pii>& is, int L, int R) {
17b   if (L == R) return;
bef   auto it = addInterval(is, L, R);
e14   auto r2 = it->second;
655   if (it->first == L) is.erase(it);
016   else (int&)it->second = L;
ee9   if (R != r2) is.emplace(R, r2);
059  }
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).

Time: $\mathcal{O}(N \log N)$

9e9d8d, 20 lines

```
4fc  template<class T>
dbe  vi cover(pair<T, T> G, vector<pair<T, T>> I) {
3d5   vi S(sz(I)), R;
d00   iota(all(S), 0);
591   sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
```

```
d10   T cur = G.first;
05e   int at = 0;
336   while (cur < G.second) { // (A)
438     pair<T, int> mx = make_pair(cur, -1);
f07     while (at < sz(I) && I[S[at]].first <= cur) {
032       mx = max(mx, make_pair(I[S[at]].second, S[at]));
69a       at++;
c42     }
c54     if (mx.second == -1) return {};
953     cur = mx.first;
fbf     R.push_back(mx.second);
dd1   }
b1a   return R;
b8d }
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

753a4c, 20 lines

```
fb4 template<class F, class G, class T>
d89 void rec(int from, int to, F& f, G& g, int& i, T& p, T q)
    {
6b6   if (p == q) return;
329   if (from == to) {
9a3     g(i, to, p);
c80     i = to; p = q;
956   } else {
0e5     int mid = (from + to) >> 1;
96c     rec(from, mid, f, g, i, p, f(mid));
695     rec(mid+1, to, f, g, i, p, q);
eff   }
fb5 }
f07 template<class F, class G>
06c void constantIntervals(int from, int to, F f, G g) {
783   if (to <= from) return;
522   int i = from; auto p = f(i), q = f(to-1);
691   rec(from, to-1, f, g, i, p, q);
b80   g(i, to, q);
8bf }
```

4.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

9155b4, 12 lines

```
044 template<class F>
20f int ternSearch(int a, int b, F f) {
25b   assert(a <= b);
329   while (b - a >= 5) {
924     int mid = (a + b) / 2;
c9e     if (f(mid) < f(mid+1)) a = mid; // (A)
ceb     else b = mid+1;
ce7   }
95e   rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
3f5   return a;
5d6 }
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

2932a0, 18 lines

```
8d3 template<class I> vi lis(const vector<I>& S) {
173   if (S.empty()) return {};
1d7   vi prev(sz(S));
085   typedef pair<I, int> p;
249   vector<p> res;
897   rep(i,0,sz(S)) {
        // change 0 -> i for longest non-decreasing subsequence
b69     auto it = lower_bound(all(res), p{S[i], 0});
ef6     if (it == res.end()) res.emplace_back(), it = res.end()
    -1;
df4     *it = {S[i], i};
6ce     prev[i] = it == res.begin() ? 0 : (it-1)->second;
147   }
629   int L = sz(res), cur = res.back().second;
bf5   vi ans(L);
ade   while (L--) ans[L] = cur, cur = prev[cur];
ba7   return ans;
293 }
```

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

b20ccc, 17 lines

```
4d3 int knapsack(vi w, int t) {
9af   int a = 0, b = 0, x;
50d   while (b < sz(w) && a + w[b] <= t) a += w[b++];
c8b   if (b == sz(w)) return a;
2b8   int m = *max_element(all(w));
754   vi u, v(2*m, -1);
0a2   v[a+m-t] = b;
564   rep(i,b,sz(w)) {
a68     u = v;
052     rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
605     for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
a42       v[x-w[j]] = max(v[x-w[j]], j);
ac5   }
4de   for (a = t; v[a+m-t] < 0; a--) ;
3f5   return a;
b20 }
```

4.3 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

d41d8c, 2 lines

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 19 lines

```
242 struct DP { // Modify at will:
178   int lo(int ind) { return 0; }
072   int hi(int ind) { return ind; }
f99   ll f(int ind, int k) { return dp[ind][k]; }
55e   void store(int ind, int k, ll v) { res[ind] = pii(k, v);
    }

105   void rec(int L, int R, int LO, int HI) {
d2c     if (L >= R) return;
c52     int mid = (L + R) >> 1;
a4e     pair<ll, int> best(LLONG_MAX, LO);
964     rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
af9       best = min(best, make_pair(f(mid, k), k));
4b0     store(mid, best.second, best.first);
ebc     rec(L, mid, LO, best.second+1);
ba2     rec(mid+1, R, best.second, HI);
541   }
26f   void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
d38 };
```

4.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

4.5 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

4.5.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

4.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

FastMod.h

Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod{b}$ in the range $[0, 2b)$.

751a02, 9 lines

```
f4c typedef unsigned long long ull;
7e2 struct FastMod {
634   ull b, m;
d2d   FastMod(ull b) : b(b), m(-1ULL / b) {}
683   ull reduce(ull a) { // a % b + (0 or b)
6fa     return a - (ull)((__uint128_t(m) * a) >> 64) * b;
f67   }
38e };
```

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: ./a.out < input.txt
Time: About 5x as fast as cin/scanf.

7b3c70, 18 lines

```
c30  inline char gc() { // like getchar()
0cd   static char buf[1 << 16];
0c8   static size_t bc, be;
a5a   if (bc >= be) {
bf4     buf[0] = 0, bc = 0;
842     be = fread(buf, 1, sizeof(buf), stdin);
d32   }
efa   return buf[bc++]; // returns 0 on EOF
026  }

e4d  int readInt() {
db8   int a, c;
169   while ((a = gc()) < 40);
0cc   if (a == '-') return -readInt();
17e   while ((c = gc()) >= 48) a = a * 10 + c - 480;
24d   return a - 48;
e04  }
```

BumpAllocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2, 9 lines

```
     // Either globally or in a single class:
2b9  static char buf[450 << 20];
a7c  void* operator new(size_t s) {
da1   static size_t i = sizeof buf;
3ca   assert(s < i);
663   return (void*)&buf[i -= s];
306  }
aa3  void operator delete(void*) {}
```

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

"BumpAllocator.h" 2dd6c9, 11 lines

```
0ca  template<class T> struct ptr {
949   unsigned ind;
185   ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
3d4     assert(ind < sizeof buf);
77e   }
e3c   T& operator*() const { return *(T*)(buf + ind); }
570   T* operator->() const { return &**this; }
618   T& operator[](int a) const { return (&**this)[a]; }
e0a   explicit operator bool() const { return ind; }
2dd  };
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.
Usage: vector<vector<int, small<int>>> ed(N);

bb66d4, 15 lines

```
30c  char buf[450 << 20] alignas(16);
cee  size_t buf_ind = sizeof buf;

ebc  template<class T> struct small {
d7b   typedef T value_type;
36e   small() {}
1ca   template<class U> small(const U&) {}
de2   T* allocate(size_t n) {
207     buf_ind -= n * sizeof(T);
df0     buf_ind &= 0 - alignof(T);
d25     return (T*)(buf + buf_ind);
e76   }
e28   void deallocate(T*, size_t) {}
164  };
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for _mm_ in /usr/lib/gcc/*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and #define __SSE__ and __MMX__ before including it. For aligned memory use _mm_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu.

551b82, 44 lines

```
ee8  #pragma GCC target ("avx2") // or sse4.1
492  #include "immintrin.h"

1b2  typedef __m256i mi;
8ca  #define L(x) _mm256_loadu_si256((mi*)&(x))

     // High-level/specific methods:
     // load(u)?_si256, store(u)?_si256, setzero_si256,
        _mm_malloc
     // blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of
        bytes)
     // i32gather_epi32(addr, x, 4): map addr[] over 32-b parts
         of x
     // sad_epu8: sum of absolute differences of u8, outputs 4
        xi64
     // maddubs_epi16: dot product of unsigned i7's, outputs 16
        xi15
     // madd_epi16: dot product of signed i16's, outputs 8xi32
     // extractf128_si256(, i) (256->128), cvtsi128_si32 (128->
        lo32)
     // permute2f128_si256(x,x,1) swaps 128-bit lanes
     // shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
     // shuffle_epi8(x, y) takes a vector instead of an imm

     // Methods that work with most data types (append e.g.
        _epi32):
     // set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and
        /or,
     // andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo
        |hi)

1e5  int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
6d0   int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
49f  mi zero() { return _mm256_setzero_si256(); }
1e1  mi one() { return _mm256_set1_epi32(-1); }
667  bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
382  bool all_one(mi m) { return _mm256_testc_si256(m, one());
     }

ff0  ll example_filteredDotProduct(int n, short* a, short* b) {
f37   int i = 0; ll r = 0;
766   mi zero = _mm256_setzero_si256(), acc = zero;
fe1   while (i + 16 <= n) {
25c     mi va = L(a[i]), vb = L(b[i]); i += 16;
2a9     va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
9d0     mi vp = _mm256_madd_epi16(va, vb);
1ee     acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
9d7       _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)
     ));
b3a   }
088   union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[
     i];
7b2   for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <-
     equiv
4c1   return r;
288  }
```

Techniques (A)

techniques.txt

159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```