競プロライブラリ

toku4388

目次

1 文字列

1.1 部分文字列として S を含む

長さ $N+K$ で部分文字列として $S(|S|=N)$ を含むのが何通りあるかを数えたい。$S=$ abbc とすると，

$$\text{(a 以外)a(b 以外)b(b 以外)c(任意)}$$

と考えることで，任意の部分文字列の長さ i を，$i=0,1,2,\dots K$ まで調べれば，うまいこと数えられる。

$$\sum_{i=0}^{K} 26^i 25^{K-i} \binom{N+K-i-1}{N-1}$$

1.2 ランレングス圧縮

同じ文字が隣り合うときとかなんとかいう問題で，一回ランレングス圧縮してしまうと実装が多少楽になることがある。

ソースコード 1　run length encode.cpp

```
template <typename T>
vector<pair<T, int>> runLength(const vector<T> &vec) {
  vector<pair<T, int>> res;
  for (const auto &p : vec) {
    if (res.size() == 0 || res.back().first != p) {
      res.push_back({p, 1});
    } else {
      res.back().second++;
    }
  }
  return res;
};
```

2 数列

2.1 座標圧縮

$(3,1,4,1,5,9)$ を $(1,0,2,0,3,4)$ にする。index の順序のみが大事なときに配列や BIT，セグ木に載せるために使う。compres(a) とすると a そのものが変わるが，返り値は $(1,3,4,5,9)$ のように圧縮された index からもとの値を復元するための配列が返ってくる。

ソースコード 2　compress.cpp

```
vector<ll> compress(vector<ll> &v) {
  vector<ll> uv = v;
  sort(uv.begin(), uv.end());
  uv.erase(unique(uv.begin(), uv.end()), uv.end());
```

```
  auto binSearch = [](ll key, const vector<ll> &arr) {
    int ok = -1, ng = arr.size(), mid;
    while (abs(ng - ok) > 1) {
      mid = (ng + ok) / 2;
      if (arr[mid] <= key)
        ok = mid;
      else
        ng = mid;
    }
    return ok;
  };
  for (int i = 0; i < (int)v.size(); i++) {
    v[i] = binSearch(v[i], uv);
  }
  return uv;
}
```

わざわざライブラリとして外に切り出すものをのせたが，べたで書いた方がむしろわかりやすいかもしれない。

```
auto tmp = a;
sort(tmp.begin(), tmp.end());
tmp.erase(unique(tmp.begin(), tmp.end()), tmp.end());
// 準備完了
for (auto v : a) {
    int i = tmp.lower_bound(tmp.begin(), tmp.end(), v) - tmp.begin()  //
    圧縮後のindexを知りたいときはこうすればよい
    // 処理
}
```

2.2 転倒数

転倒数は $\#\{(i,j) \mid i < j \wedge a_i > a_j\}$。バブルソートの交換回数。（必要であれば座標圧縮と）BIT を使うことで $O(N \log N)$ で転倒数を計算できる。前に出てきたやつの個数を記録しながら和を計算していくイメージ。

ソースコード 3　inversion.cpp

```
ll inversion(vector<ll> a) {
  const int n = a.size();
  compress(a);
  bit b(n);
  ll res = 0;
  for (int i = 0; i < n; i++) {
    res += i - b.sum(0, a[i] + 1);
    b.add(a[i], 1);
  }
  return res;
}
```

2.3 最長増加部分列 (LIS)

ソースコード 4　Longest Increasing Subsequence(LIS).cpp

```
int lis(vector<ll> x) {
  int n = x.size();
  compress(x);
  vector<int> dp(n + 1, INF);
  dp[0] = -1;
  for (int i = 0; i < n; i++) {
    int l = 0, r = n + 1, mid;
    while (abs(l - r) > 1) {
      mid = (l + r) / 2;
      (dp[mid] <= x[i] ? l : r) = mid;
    }
    dp[l + 1] = min(dp[l + 1], (int)x[i]);
  }
  int res = -1;
  for (int i = 0; i <= n; i++) {
    if (dp[i] != INF) res = i;
  }
  return res;
}
```

3 DP

3.1 最長共通部分列 (LCS)

文字列 s と t の共通する部分列のうち最長のもの。抜き出す要素が連続である必要はない。

$$dp[i][j] := s \text{ の最初 } i \text{ 文字, } t \text{ の最初 } j \text{ 文字だけ取ったときの LCS の長さ}$$

とする。

$s[i+1]$ と $t[j+1]$ が一致すれば

$$dp[i][j] = \max(dp[i][j], dp[i-1][j-1]+1)$$

となり，そうでなくても

$$dp[i][j] = \max(dp[i][j], dp[i-1][j])$$
$$dp[i][j] = \max(dp[i][j], dp[i][j-1])$$

になる。$dp[i][j]$ を基準にそこ**から**遷移していこうとすると端の更新がうまくいかないことがあるので注意。そこ**への**遷移を考える。

3.2 桁 DP

$$dp[i][smaller] \coloneqq \left\{ \begin{array}{l} i \text{ 桁まで } N \text{ より小さい } (smaller = 1) \\ i \text{ 桁まで } N \text{ と同じ } (smaller = 0) \end{array} \right\} \text{ときの最大値 (/最小値/場合の数)}$$

$N = 56243$ とすると，dp[1][1] は 00000〜49999 を考え，dp[1][0] は 50000〜56243 を考える。

- $dp[i][1]$ から $dp[i+1][1]$ に遷移 (0〜9 全部使える。ここまでで N より小さいのでどうやっても N と同じにはなれない)
- $dp[i][0]$ から $dp[i+1][1]$ に遷移 (0〜$(N[i]-1)$ が使える。ここまでで N と同じだが，ここで N の i 桁目より小さくする)
- $dp[i][0]$ から $dp[i+1][0]$ に遷移 ($N[i]$ が使える。ここまで N とまったく同じ)

3.3 ダブリング

頂点が N 個あり，1 回移動するとどの頂点に行くかということが定まっているとき，K 回移動したときにどこにいるかを，前処理 $O(N \log K)$，各クエリ $O(\log K)$ で求められる。

$$d[i][j] \coloneqq \text{頂点 } i \text{ から } 2^j \text{回移動したときの頂点}$$

とすると，遷移は，

$$d[i][j+1] = d[d[i][j]][j]$$

となる。これを各頂点につき $\log K$ 回行えばよい。また，各クエリについては，K を 2 進数展開して，c 桁目のビットが立っているなら，$now = d[now][c]$ として，now から 2^c 回移動後の頂点を求める。

4 二分探索

4.1 決め打ち二分探索

「最小値の最大化」や「最大値の最小化」に対して非常に有効。ある x を考えたときに，x が達成可能かで解がどちら側にあるかを判定する。x をどんどん大きくしていったときに，あるところまでは**全部** OK でそれ以降は**全部** NG，といった単調性があるときに使える。

4.2 平均値・中央値

平均値は K 以上にできるか？という決め打ち二分探索でとけることがある。中央値は K 以上が $N/2$ 個以上あるか？という二分探索でとけることがある。このとき，

$$\frac{1}{N}\sum_{i=1}^{N} a_i \geq K \Leftrightarrow \sum_{i=1}^{N}(a_i - K) \geq 0$$

という仮平均的な考え方を使って，総和の最大値が 0 以上という条件にすると，平均を求めるときに個数を保持する必要がなくなるので，計算量が改善することがある。

5 数え上げ

5.1 ${}_n\mathrm{C}_r$

二項係数。n 個のものから r 個とる組み合わせ。普通にやると $O(r)$ とかかかる。$\mathrm{mod}10^9+7$ とかで大量に求めたいときは，あらかじめ dp で $n!, n!^{-1}, n^{-1}$ のテーブルを $O(N)$ で作ってからやると各クエリ $O(1)$ でできる。

$$
\begin{aligned}
{}_n\mathrm{C}_r &= {}_{n-1}\mathrm{C}_{r-1} + {}_{n-1}\mathrm{C}_r \\
{}_n\mathrm{C}_r &= \frac{n!}{r!(n-r)!}
\end{aligned}
$$

ソースコード 5　nCr mod p(mint).cpp

```
mint comb(ll n, ll r) {
  if (n < r || n < 0 || r < 0) return 0;
  mint x = 1, y = 1;
  for (ll i = n; i > n - r; i--) x *= i;
  for (ll i = 2; i <= r; i++) y *= i;
  return (x / y);
}
```

以下はテーブルを作って前計算しておくもの。累積積を取っている。inv の計算は適当にやると log がつくが，拡張ユークリッドの互除法をいい感じに使ってやると線形になる（以下のコード中の MOD / i は普通の切り捨て除算である）。

書き方	操作	計算量
comb(int n)	前計算で n までテーブルを作る	$O(n)$
mint c(ll n, ll r)	${}_n\mathrm{C}_r$ を計算	$O(1)$

ソースコード 6　nCr mod p(table).cpp

```
struct comb {
  vector<mint> fact, finv, inv;
  comb(int n) : fact(n + 1), finv(n + 1), inv(n + 1) {
    fact[0] = finv[0] = inv[1] = fact[1] = finv[1] = 1;
    for (int i = 2; i < n; i++) {
      fact[i] = fact[i - 1] * i;
      inv[i] = -inv[MOD % i] * (MOD / i);
      finv[i] = finv[i - 1] * inv[i];
    }
  }
  mint c(ll n, ll r) {
    if (n < r || n < 0 || r < 0) return 0;
    return fact[n] * finv[r] * finv[n - r];
  }
};
```

5.2 写像 12 相

n 個の玉を k 個の箱に入れる場合の数は，12 パターンある。

玉の区別	箱の区別	1 個以下	制限なし	1 個以上
○	○	${}_k\mathrm{P}_n$	k^n	$\sum_{i=0}^{k}(-1)^{k-i}{}_k\mathrm{C}_i$
×	○	${}_k\mathrm{C}_n$	${}_{n+k-1}\mathrm{C}_n$	${}_{n-1}\mathrm{C}_{k-1}$
○	×	0 or 1	$B(n,k)$	$S(n,k)$
×	×	0 or 1	$P(n,k)$	$P(n-k,k)$

5.2.1 スターリング数

箱を区別する場合の結果（包除原理を使って求まる）から箱を区別しないようにして $k!$ で割ればよい。（第二種）スターリング数 $S(n,k)$ は,

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}{}_k\mathrm{C}_i i^n$$

漸化式は以下。

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

5.2.2 ベル数

ベル数 $B(n,k)$ は $O(\min(n,k)\log n)$ で計算できる。

$$\begin{aligned} B(n,k) &= \sum_{j=0}^{k} S(n,j) \\ &= \sum_{i=0}^{k}\frac{i^n}{i!}\sum_{j=0}^{k-i}\frac{(-1)^j}{j!} \end{aligned}$$

漸化式は $B(n,n)$ を単に $B(n)$ と書くと以下。

$$B(n+1) = \sum_{i=0}^{n}{}_n\mathrm{C}_i B(i)$$

5.2.3 分割数

分割数 $P(n,k)$ は以下の漸化式から $O(nk)$ で求まる。

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

5.3 部分集合の部分集合を $O(3^N)$ で列挙

```
int bit = 0b1010011;
for (int b = bit;; b = (b - 1) & bit) {
        // 処理
        if (b == 0) break;
    }
```

6 mod 関連

6.1 逆元と拡張ユークリッドの互除法

mod$p$ における a の逆元は a^{-1} と書き，$ax \equiv 1 \bmod p$ なる x のこと。a の逆元が mod$p$ で存在するためには，a と p が互いに素でなければならない（すなわち，$\gcd(a,p) = 1$）。a^{-1} は，

$$ax + py = 1$$

となる x であるから，これは拡張ユークリッドの互除法によって求めることができる。

ソースコード 7　extgcd.cpp と mod inverse.cpp

```
// (g,x,y) ax+by=gcd(a,b)
tuple<ll, ll, ll> extgcd(ll a, ll b) {
  if (b == 0) {
    return {a, 1, 0};
  }
  ll g, x, y;
  tie(g, x, y) = extgcd(b, a % b);
  return {g, y, x - (a / b * y)};
}
// a^-1 mod m
ll inv(ll a, ll m) {
  ll g, x, y;
  tie(g, x, y) = extgcd(a, m);
  if (g == 1) return (x % m + m) % m;
  return -1;
}
```

また，オイラー関数 $\varphi(n)$（n と互いに素である 1 以上 n 以下の自然数の個数）を用いて，

$$a^{\varphi(p)} \equiv 1 \bmod p$$

であるから，

$$a^{-1} \equiv a^{\varphi(p)-1} \bmod p$$

と表せる。特に，p が素数のとき，$\varphi(p) = p - 1$ であるから，

$$a^{-1} \equiv a^{p-2} \bmod p$$

が成り立つ。これは，繰り返し二乗法によって $O(\log p)$ で求められる。以下に繰り返し二乗法による逆元の計算を含む mint 構造体を示す。コピペできない環境では，modpow や inv などで累乗/逆元の計算だけ関数化して，あとはべたで書いた方がよいかと思われる。

ソースコード 8　mint.cpp

```
const int MOD = 1000000007;
// const int MOD = 998244353;
struct mint {
  ll x;
  mint(ll x = 0) : x((x % MOD + MOD) % MOD) {}
  mint operator-() { return mint(-x); }
  mint operator+=(mint rhs) {
    x += rhs.x;
    if (x >= MOD) x -= MOD;
    return *this;
  }
  mint operator-=(mint rhs) {
    x -= rhs.x;
    if (x < 0) x += MOD;
    return *this;
  }
  mint operator*=(mint rhs) {
    x = x * rhs.x % MOD;
    return *this;
  }
  mint operator+(mint rhs) { return mint(*this) += rhs; }
  mint operator-(mint rhs) { return mint(*this) -= rhs; }
  mint operator*(mint rhs) { return mint(*this) *= rhs; }
  mint pow(ll n) {
    mint r = 1;
    for (mint t = (*this); n; t *= t, n >>= 1)
      if (n & 1) r *= t;
    return r;
  }
  mint inv() { return (*this).pow(MOD - 2); }
  mint operator/=(mint rhs) { return *this *= rhs.inv(); }
  mint operator/(mint rhs) { return mint(*this) /= rhs; }
};
ostream &operator<<(ostream &os, const mint &dt) {
  os << dt.x;
  return os;
}
```

ソースコード 9　modpow.cpp

```
ll modpow(ll a, ll n, int m) {
  ll r = 1;
  for (ll t = a; n; t = t * t % m, n >>= 1)
    if (n & 1) r = r * t % m;
  return r;
}
```

6.2 オイラーの φ 関数

$\varphi(n)$ は n と互いに素である 1 以上 n 以下の自然数の個数を表す。$n = \prod_{i=1}^{k} {p_i}^{a_i}$ と素因数分解できるとき

$$\varphi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

と計算できる（直感的には p_i を追加することによって互いに素なものの個数が何分の 1 になるかを考えると分かる）。また，$\gcd(m, n) = 1$ のとき，$\varphi(mn) = \varphi(m)\varphi(n)$ が成り立つ。さらに約数に関しての和を取ると

$$\sum_{d|n} \varphi(d) = n$$

が成り立つ。

6.3 離散対数

$x = p\sqrt{m} + r$ として p, r を求める平方分割のような考え方で，$a^x \equiv b \bmod m$ なる x を $O(\sqrt{m} \log m)$ で求める。解がない場合は-1 を返す。unorderd_map を使えば log が落ちる。

$$a^x \equiv b \bmod m \Leftrightarrow b \left(a^{-\sqrt{m}}\right)^p \equiv a^r \bmod m$$

ソースコード 10　modlog.cpp

```
// a^x mod MOD = b
int modlog(int a, mint b) {
  ll low = -1, hi = MOD, mid;
  while (abs(low - hi) > 1) {
    mid = (low + hi) / 2;
    ((mid * mid < MOD) ? low : hi) = mid;
  }
  int sqp = hi;
  mint ap = 1;
  map<int, int> aps;
  for (int i = 0; i < sqp; i++) {
    if (!aps.count(ap.x)) aps[ap.x] = i;
    ap *= a;
  }
  mint rinv = ap.inv(), g = b;
  for (int i = 0; i <= sqp; i++) {
    if (aps.count(g.x)) return i * sqp + aps[g.x];
    g *= rinv;
  }
  return -1;
}
```

6.4 有理数 mod を復元

ソースコード 11 rational reconstruction.py

```
# find a,b s.t. xb = a (mod MOD)
MOD = 998244353
x = int(input())
v = [MOD, 0]
w = [x, 1].copy()
while w[0] * w[0] * 2 > MOD:
    q = v[0] // w[0]
    z = [v[0] - q * w[0], v[1] - q * w[1]]
    v = w.copy()
    w = z.copy()
if w[1] < 0:
    w[0] *= -1
    w[1] *= -1
assert (w[0] * pow(w[1], -1, MOD) % MOD) == x
print(f"{w[0]}/{w[1]}")
print(w[0] / w[1])
```

7 整数

7.1 GCD LCM

最大公約数はユークリッドの互除法で $O(\log(\max(A, B)))$ で計算できる。$\gcd(a, b)\,\mathrm{lcm}(a, b) = ab$ が成り立つが，これは $\min(x, y) + \max(x, y) = x + y$ という当たり前の式から理解できる。

ソースコード 12 gcd lcm.cpp

```
ll gcd(ll x, ll y) {
  if (x == 0) return y;
  if (y == 0) return x;
  ll r;
  while ((r = x % y) != 0) {
    x = y;
    y = r;
  }
  return y;
}
ll lcm(ll x, ll y) {
  return (x / gcd(x, y) * y);
}
```

7.2 約数列挙

$\sqrt{N}$ まで見れば相方でもう列挙し終わっている。

ソースコード 13　divisor.cpp

```
vector<ll> divisor(ll n) {
  vector<ll> res;
  for (ll i = 1; i * i <= n; i++) {
    if (n % i == 0) {
      res.push_back(i);
      if (i * i != n)
        res.push_back(n / i);
    }
  }
  sort(res.begin(), res.end());
  return res;
}
```

7.3　素因数分解

$O(\sqrt{N})$ でできる。

ソースコード 14　prime factorization.cpp

```
vector<pair<ll, int>> primeFac(ll n) {
  vector<pair<ll, int>> res;
  for (ll i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      int cnt = 0;
      while (n % i == 0) {
        cnt++;
        n /= i;
      }
      res.push_back(make_pair(i, cnt));
    }
  }
  if (n != 1) res.push_back(make_pair(n, 1));
  return res;
}
```

M 回 N 以下の数を素因数分解をするようなら，普通にやると $O(M\sqrt{N})$ だが，$O(M \log N)$ できる。エラトステネスの篩のようにあらかじめ，最小の素因数を見つけておく（このコンストラクションは素数の逆数和から $O(N \log \log N)$）であることが知られている)。この配列を s とする。そして x を素因数分解するときは $s[x]$ を見れば，x の最小の素因数がわかるので，$x/s[x]$ をする。そして，$s[x/s[x]]$ を見て... というように，x が 1 になるまで繰り返すと，素因数分解ができている。1 回あたりの計算量は，x の重複も含めた素因数の個数（すなわち，$x = \prod_{i=1}^{k} {p_i}^{a_i}$ と素因数分解できるときの $\sum_{i=1}^{k} a_i$）に比例するから，$O(\log x)$（はずれなし素因数ガチャをずっと引ける感じ)。

ソースコード 15　prime factorization(sieve).cpp

```
struct sieve {
  vector<int> s;
```

```
  sieve(int n) : s(n + 1) {
    for (ll i = 0; i <= n; i++) {
      s[i] = i;
    }
    for (ll i = 2; i <= n; i++) {
      if (s[i] == i) {
        for (ll j = i * i; j <= n; j += i) {
          if (s[j] == j) {
            s[j] = i;
          }
        }
      }
    }
  }
  vector<pair<int, int>> primeFac(int n) {
    vector<pair<int, int>> res;
    while (n != 1) {
      int p = s[n], cnt = 0;
      while (s[n] == p) {
        n /= p;
        cnt++;
      }
      res.push_back(pair<int, int>(p, cnt));
    }
    return res;
  }
  bool isPrime(int n) {
    if (n <= 1) return false;
    return (s[n] == n);
  }
};
```

7.4 切り上げ

x 以下の最大の整数を $\lfloor x \rfloor$ と表し，x 以上の最小の整数を $\lceil x \rceil$ と表す。$\lceil \frac{a}{b} \rceil$ を求めたいときは，`(a + b - 1) / b` とすればよい。場合分けがなくなって見通しがよくなる。

$$\left\lfloor \frac{a+b-1}{b} \right\rfloor = \left\lfloor \frac{a}{b} + \frac{b}{b} - \frac{1}{b} \right\rfloor = \left\lfloor \frac{a-1}{b} + 1 \right\rfloor = \left\lfloor \frac{a-1}{b} \right\rfloor + 1$$

であるから，$a \equiv 0 \pmod{b}$ のときは，

$$\left\lfloor \frac{a-1}{b} \right\rfloor + 1 = \left(\frac{a}{b} - 1 \right) + 1 = \frac{a}{b} = \left\lceil \frac{a}{b} \right\rceil$$

$a \not\equiv 0 \pmod{b}$ のときは，

$$\left\lfloor \frac{a-1}{b} \right\rfloor + 1 = \left\lfloor \frac{a}{b} \right\rfloor + 1 = \left\lceil \frac{a}{b} \right\rceil$$

となり，確かに切り上げを表している。

7.5 四捨五入

$\frac{a}{b}$ を小数点以下四捨五入して整数にするには `(a + b / 2) / b` とすればよい

$$\left\lfloor \frac{a + \lfloor \frac{b}{2} \rfloor}{b} \right\rfloor$$

7.6 N 進数

7.6.1 変換

下から順番にかけていく/割っていくことで log でできる。

ソースコード 16　toInt toStr.cpp

```
ll toInt(string s, ll b = 10) {
  assert(b > 1);
  ll res = 0;
  for (char c : s) {
    res *= b;
    res += c - '0';
  }
  return res;
}
string toStr(ll x, ll b = 10) {
  assert(b > 1 && x >= 0);
  if (x == 0) return "0";
  string res;
  while (x) {
    res += (x % b) + '0';
    x /= b;
  }
  reverse(res.begin(), res.end());
  return res;
}
```

8 幾何

8.1 内積と外積

平面では，$\vec{a} = (a_x, a_y), \vec{b} = (b_x, b_y)$ として，

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta = a_x b_x + a_y b_y$$
$$\vec{a} \times \vec{b} = |\vec{a}||\vec{b}| \sin\theta = a_x b_y - a_y b_x$$

また，空間では，$\vec{a} = (a_x, a_y, a_z), \vec{b} = (b_x, b_y, b_z)$ として，

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta = a_x b_x + a_y b_y + a_z b_z$$
$$\vec{a} \times \vec{b} = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$$

$\vec{a}, \vec{b}, \vec{a} \times \vec{b}$ はこの順で右手系になる。$\vec{a}$ から $\vec{b}$ に右ねじの方向。また，$A(a_x, a_y), B(b_x, b_y)$ として $\triangle OAB$ の面積 S は

$$S = \frac{1}{2}|\vec{a} \times \vec{b}|$$

8.2 平行と垂直判定

平面における判定。

$$\vec{a} \cdot \vec{b} = 0 \Leftrightarrow \vec{a} \perp \vec{b}$$
$$\vec{a} \times \vec{b} = 0 \Leftrightarrow \vec{a} \parallel \vec{b}$$

零ベクトルのときには注意。

8.3 2 線分の交差判定

線分 AB と線分 CD が交差しているかどうかの判定をしたい。$\vec{\mathrm{AB}} \times \vec{\mathrm{AC}}, \vec{\mathrm{AB}} \times \vec{\mathrm{AD}}$ が異符号（A から見て CD の間に B がある）かつ，$\vec{\mathrm{CD}} \times \vec{\mathrm{CA}}, \vec{\mathrm{CD}} \times \vec{\mathrm{CB}}$ が異符号（C から見て AB の間に D がある）かどうかを判定すればよい。二次元の点を扱うのに便利な構造体 vec とともに以下に示す。これでは 0 を含めていないので線分が両端を除いて交差しているかの判定になっている。

ソースコード 17　vector.cpp

```
struct vec {
    ll x, y;
    vec(ll x_, ll y_) : x(x_), y(y_) {}
    vec() : x(0), y(0) {}
    vec operator-() { return vec(-x, -y); }
    vec operator+=(vec rhs) {
        x += rhs.x;
        y += rhs.y;
        return *this;
      }
    vec operator-=(vec rhs) {
        x -= rhs.x;
        y -= rhs.y;
        return *this;
      }
    vec operator*=(ll rhs) {
        x *= rhs;
        y *= rhs;
        return *this;
      }
```

```
    vec operator+(vec rhs) { return vec(*this) += rhs; }
    vec operator-(vec rhs) { return vec(*this) -= rhs; }
    vec operator*(ll rhs) { return vec(*this) *= rhs; }
    bool operator==(vec rhs) {
        return x == rhs.x && y == rhs.y;
      }
    bool operator!=(vec rhs) {
        return !(*this == rhs);
      }
    ll dot(vec rhs) const { return x * rhs.x + y * rhs.y; }
    ll cross(vec rhs) const { return x * rhs.y - y * rhs.x; }
    ll norm2() const { return x * x + y * y; };
    double norm() const { return sqrt((*this).norm2()); };
    void print() {
        cout << x << " " << y << endl;
      }
  };
// 線分ABとCDが交差しているか
bool isCross(vec a, vec b, vec c, vec d) {
  return ((b - a).cross(c - a) * (b - a).cross(d - a) < 0 && (d - c).cross(a - c) * (d -
    c).cross(b - c) < 0);
}
```

8.4 偏角ソート

$(-\pi, \pi]$ で偏角ソートを行う。ただし，原点の偏角は 0 として，偏角が等しいものの順序は自由。まず $(-\pi, 0]$ と原点と $(0, \pi]$ の 3 つのエリアに分けて，エリアが同じものについては 180° 未満なので外積の符号で比較。

ソースコード 18　arg sort.cpp

```
void argSort(vector<vec> &v) {
  auto area = [&](const vec &p) {
    if (p.x == 0 && p.y == 0)
      return 0;
    else if (p.y < 0 || (p.y <= 0 && p.x > 0))
      return -1;
    else
      return 1;
  };
  sort(v.begin(), v.end(), [&](const vec &p, const vec &q) {
    return (area(p) != area(q) ? area(p) < area(q) : p.cross(q) > 0);
  });
}
```

8.5 行列

8.5.1 基本的な演算

足し算/引き算/掛け算ができる。特に，$m \times m$ 正方行列 A について A^n は，行列累乗で $O(m^3 \log n)$ で求められる。

ソースコード 19　matrix.cpp

```
template <typename T>
struct mat {
  int n, m;
  vector<vector<T>> a;
  mat(int n_, int m_) : mat(n_, m_, 0) {}
  mat(int n_, int m_, int e) : mat(vector<vector<T>>(n_, vector<T>(m_, e))) {}
  mat(const vector<vector<T>> &x) : a(x), n(x.size()), m(x[0].size()) {}
  mat operator-() { return mat(a) * -1; }
  mat operator+=(mat rhs) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        a[i][j] += rhs.a[i][j];
      }
    }
    return *this;
  }
  mat operator-=(mat rhs) { return *this += -rhs; }
  mat operator*=(int rhs) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        a[i][j] *= rhs;
      }
    }
    return *this;
  }
  mat operator*=(mat rhs) {
    assert(m == rhs.n);
    mat res(n, rhs.m);
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < rhs.m; j++) {
        T s = 0;
        for (int k = 0; k < m; k++) {
          s += a[i][k] * rhs.a[k][j];
        }
        res.a[i][j] = s;
      }
    }
    (*this) = res;
    return (*this);
  }
```

```
  mat operator+(mat rhs) { return mat(*this) += rhs; }
  mat operator-(mat rhs) { return mat(*this) -= rhs; }
  mat operator*(mat rhs) { return mat(*this) *= rhs; }
  mat operator*(int rhs) { return mat(*this) *= rhs; }
  mat pow(ll pn) {
    assert(n == m);
    mat r(n, m);
    for (int i = 0; i < n; i++) {
      r.a[i][i] = 1;
    }
    for (mat t = (*this); pn; t *= t, pn >>= 1)
      if (pn & 1) r *= t;
    return r;
  }
  void print() {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        cout << a[i][j] << " ";
      }
      cout << endl;
    }
  }
};
```

8.5.2 連立線形方程式

$A\vec{x} = \vec{b}$ を Gauss-Jordan の消去法により $O(N^3)$ で解く。誤差は小さくなるように気を付けているが十分注意されたい。

ソースコード 20　Linear Simultaneous Equation(LSE).cpp

```
template <typename T>
vector<T> solveLSE(mat<T> a, vector<T> b) { // ax=b
  assert(a.n == b.size() && a.m == b.size());
  int n = a.n;
  for (int i = 0; i < n; i++) {
    int p = i;
    for (int j = p; j < n; j++) {
      if (a.a[i][j] > a.a[i][p]) p = j;
    }
    swap(a.a[i], a.a[p]);
    swap(b[i], b[p]);
    for (int j = 0; j < n; j++) {
      if (j != i) a.a[i][j] /= a.a[i][i];
    }
    b[i] /= a.a[i][i];
    a.a[i][i] = 1;
    for (int k = 0; k < n; k++) {
      if (i == k) continue;
```

```
      for (int j = 0; j < n; j++) {
        if (j != i) a.a[k][j] -= a.a[i][j] * a.a[k][i];
      }
      b[k] -= b[i] * a.a[k][i];
      a.a[k][i] = 0;
    }
  }
  return b;
}
```

8.6 凸包

点の集合が与えられたときに，反時計回りに頂点の座標を配列にした形で凸包を返す。内角がちょうど 180 ° であるような頂点も許している。計算量は $O(N \log N)$

ソースコード 21　convex hull.cpp

```
vector<vec> convexHull(vector<vec> p) {
  sort(p.begin(), p.end(), [](const vec &lhs, const vec &rhs) {
    if (lhs.y == rhs.y) return lhs.x < rhs.x;
    return lhs.y < rhs.y;
  });
  vector<vec> res;
  int k = 0;
  for (int v = 0; v < 2; v++) {
    for (int i = 0; i < (int)p.size(); i++) {
      res.push_back(p[i]);
      k++;
      // < 0 は180度を許す
      while (k >= 3 && (res[k - 1] - res[k - 2]).cross(res[k - 3] - res[k - 2]) < 0) {
        swap(res[k - 2], res[k - 1]);
        res.pop_back();
        k--;
      }
    }
    res.pop_back();
    k--;
    reverse(p.begin(), p.end());
  }
  return res;
}
```

9 グラフ

9.1 完全グラフ

- どの 2 頂点間にも辺があるグラフ

- 頂点数 n の完全グラフを K_n と書いたりする
- K_n の辺の数は $\frac{n(n-1)}{2}$（各頂点から引ける辺の本数を考えると $(n-1)+(n-2)+\cdots+2+1$ となるから）

9.2 最短経路

9.2.1 ベルマンフォード法

負辺があっても使える。計算量は $O(VE)$。負の閉路が存在するとき，またかつそのときに限り true を返す。

ソースコード 22　bellman ford.cpp

```
bool bellmanFord(graph &G, int st) {
  for (int i = 0; i < n; i++) g.dist[i] = LINF;
  g.dist[st] = 0;
  for (int t = 0; t < n; t++) {
    bool f = false;
    for (int i = 0; i < n; i++) {
      for (const edge &e : G.g[i]) {
        if (g.dist[e.to] > g.dist[i] + e.cost) {
          g.dist[e.to] = g.dist[i] + e.cost;
          f = true;
        }
      }
    }
    if (!f) break;
    if (t == n - 1) return true;
  }
  return false;
}
```

9.2.2 BFS

辺の重みが全て 1 でないと使えない。辺の重みが 0 か 1 だけなら queue を dequeue に変更することによりいわゆる 01BFS が可能。これらは，dijkstra においては priority_queue で一般化しているところを，特殊な制約下で探索順を限定して高速化していると思えばよい。

ソースコード 23　BFS(graph).cpp

```
void bfs(graph &G, int st) {
  for (int i = 0; i < G.n; i++) G.dist[i] = LINF;
  G.dist[st] = 0;
  queue<int> q;
  q.push(st);
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    for (auto v : G.g[now]) {
```

```
      if (G.dist[v] == LINF) {
        G.dist[v] = G.dist[now] + 1;
        q.push(v);
      }
    }
  }
  return;
}
```

9.2.3 Dijkstra 法

辺の重みが非負のときに使える。計算量は $O(E \log V)$。

ソースコード 24　dijkstra.cpp

```
void dijkstra(graph &G, int st) {
  dist.assign(G.n, LINF);
  using qv = pair<ll, int>;
  priority_queue<qv, vector<qv>, greater<qv>> pq;
  dist[st] = 0;
  pq.push({0, st});
  while (!pq.empty()) {
    auto [d, v] = pq.top();
    pq.pop();
    if (dist[v] < d) continue;
    for (auto [nv, cost] : G.g[v]) {
      ll nd = dist[v] + cost;
      if (dist[nv] > nd) {
        dist[nv] = nd;
        pq.push({nd, nv});
      }
    }
  }
  return dist;
};
```

9.2.4 ワーシャルフロイド法

全ての 2 頂点間の最短路が求まる。負辺があっても動作する。kij の順。実は順番が分からなくなっても 3 回まわせば正しく求まっている。計算量は $O(V^3)$。

ソースコード 25　warshall floyd.cpp

```
for (int k = 0; k < N; k++) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            chmin(d[i][j], dp[i][k] + dp[k][j]);
        }
    }
}
```

9.3 最大流 (maxflow)

有向グラフにおいて，各辺に流せる最大の量が決まっていて，その容量を超えずに s から t に送ることのできる量の最大値。Dinic 法により $O(EV^2)(E$ は辺の本数,V は頂点数) であるが，実際はもっと少なく済むことが多い。

1. 流れを押し戻すための逆辺を考え，その逆辺と元々の辺とを合わせて考えたグラフ (残余グラフ) を作る。
2. 余分に流せる量が 0 より大きい辺のみを使い，s から BFS をして，各頂点までの距離を求める。
3. s から t への経路を，距離が $0, 1, 2, \cdots$ なる頂点をたどるように，DFS によってひとつ求める。このとき，各頂点についてどの辺まで調べたかを記録しておき，次はここから DFS を始める。
4. このとき，各辺について余分に流せる量の最小値を流す (最大限流せるだけ流す)。
5. 流せなくなるまで 3. に戻って繰り返す。
6. 流せなくなったらどの辺まで調べたかをリセットし，t にたどりつけなくなるまで 2. に戻って繰り返す。

ソースコード 26　max flow(dinic).cpp

```
struct edge {
  int to;
  ll cap;
  int rev;
  ll now;
};
struct graph {
  int n;
  vector<vector<edge>> g;
  vector<int> fin;
  vector<int> dist;
  graph(int n) : n(n), g(n), fin(n), dist(n) {}
  void addEdge(int from, int to, int cap) {
    g[from].push_back(edge{to, cap, (int)g[to].size(), 0});
    g[to].push_back(edge{from, 0, (int)g[from].size() - 1, 0});
  }
  void addEdgeBoth(int n1, int n2, int cap) {
    addEdge(n1, n2, cap);
    addEdge(n2, n1, cap);
  }
  void bfs(int s, int t) {
    queue<int> q;
    dist[s] = 0;
    q.push(s);
    while (!q.empty()) {
      int now = q.front();
      q.pop();
      for (edge &e : g[now]) {
        if (e.cap - e.now <= 0) continue;
```

```
        if (dist[e.to] >= 0) continue;
        dist[e.to] = dist[now] + 1;
        if (e.to == t) return;
        q.push(e.to);
      }
    }
  }
  ll dfs(int no, int t, ll mins) {
    if (no == t) return mins;
    for (int i = fin[no]; i < (int)g[no].size(); i++) {
      fin[no]++;
      edge &e = g[no][i];
      if (e.cap - e.now <= 0) continue;
      if (dist[e.to] != dist[no] + 1) continue;
      ll f = dfs(e.to, t, min(mins, e.cap - e.now));
      if (f <= 0) continue;
      g[e.to][e.rev].now -= f;
      e.now += f;
      return f;
    }
    return 0;
  }
  ll maxFlow(int s, int t) {
    ll r = 0;
    while (1) {
      for (int i = 0; i < n; i++) dist[i] = -1;
      bfs(s, t);
      if (dist[t] == -1) break;
      ll f = 1;
      for (int i = 0; i < n; i++) fin[i] = 0;
      while (f) {
        f = dfs(s, t, LINF);
        r += f;
      }
    }
    return r;
  }
};
```

9.3.1 最大 2 部マッチング問題

ある 2 種類のもの (黒/白，表/裏) と，ペアになってもいい組が与えられて，そこからペアを作るとき，できるだけ多くのペアを作る方法。Dinic 法で $O(EV^2)$ に思えるが，この問題の場合は $O(E\sqrt{V})$ で動作する。

1. ある 2 種類のもの (A, B とする) がある
2. ペアになってもよい組 (A_i, B_j) について，A_i から B_i に容量 1 の辺を張る
3. 別の頂点 s, t を作り，s から A_i，B_i から t に容量 1 の辺を張る

4. s から t への最大流が求める最大のマッチング数

9.3.2 mincut

$S \to T$ に行けなくなる辺の削除方法であって，その重みが最小のもの。S に 0，T に 1 を割り当て，各頂点に 01 を割り当てて，0 から 1 への辺があるとその辺のコストがかかるとすると燃やす埋めるなどの応用がききやすい。S から T にたどり着くには 0 から 1 へを少なくとも 1 回は通らないといけないので，これで確かに cut になっている。

9.4 最小費用流

書き方	操作	計算量
`void addEdge(int from, int to, ll cap, ll cost)`	from から to へ容量 cap コスト cost の辺を追加	$O(1)$
`ll minCostFlow(int s, int t, ll f)`	s から t へ流量 f を流したときの最小費用を求める	$O(\log N)$

なお，流量 f が流せないときは-1 を返す。

ソースコード 27　min cost flow.cpp

```
struct edge {
  int to;
  ll cap;
  int rev;
  ll cost;
};
struct graph {
  int n;
  vector<vector<edge>> g;
  vector<ll> d;
  vector<pair<int, int>> prev;
  graph(int n) : n(n), g(n), d(n), prev(n) {}
  void addEdge(int from, int to, ll cap, ll cost) {
    assert(cost >= 0);
    assert(cap >= 0);
    g[from].push_back({to, cap, (int)g[to].size(), cost});
    g[to].push_back({from, 0, (int)g[from].size() - 1, -cost});
  }
  ll minCostFlow(int s, int t, ll f) {
    ll r = 0;
    vector<ll> h(n, 0);
    while (f > 0) {
      using qv = pair<ll, int>;
      priority_queue<qv, vector<qv>, greater<qv>> q;
      d.assign(n, LINF);
```

```
      d[s] = 0;
      q.push({d[s], s});
      while (!q.empty()) {
        qv now = q.top();
        q.pop();
        int v = now.second;
        if (now.first > d[v]) continue;
        int i = 0;
        for (edge &e : g[v]) {
          ll nd = d[v] + e.cost + h[v] - h[e.to];
          if (e.cap > 0 && d[e.to] > nd) {
            d[e.to] = nd;
            prev[e.to] = {v, i};
            q.push({d[e.to], e.to});
          }
          i++;
        }
      }
      if (d[t] == LINF) return -1;
      for (int i = 0; i < n; i++) h[i] += d[i];
      ll d = f;
      for (int v = t; v != s; v = prev[v].first) {
        d = min(d, g[prev[v].first][prev[v].second].cap);
      }
      f -= d;
      r += d * h[t];
      for (int v = t; v != s; v = prev[v].first) {
        edge &e = g[prev[v].first][prev[v].second];
        e.cap -= d;
        g[v][e.rev].cap += d;
      }
    }
    return r;
  }
};
```

9.5 木

9.5.1 クラスカル法

最小全域木を $O(E \log E)$ で構築する。辺を，コストの小さい順にソートして，それによって閉路が生まれるかを UnionFind でチェックしながら，追加していく。このコードは全体のコストしか返さないが，木を返すことも可能。これをもとにした応用問題は多い。

ソースコード 28　kruskal.cpp

```
ll kruskal(graph G) {
  sort(G.g.begin(), G.g.end(), [](edge &l, edge &r) {
    return l.cost < r.cost;
```

```
  });
  unionFind t(n);
  ll res = 0;
  for (edge e : G.g) {
    if (t.same(e.from, e.to)) continue;
    t.unite(e.from, e.to);
    res += e.cost;
  }
  return res;
}
```

9.5.2 最小共通祖先 (LCA)

par[i][j] は i から 2^j 回根の方にたどっていったときにたどり着く頂点を表している。これはダブリングの要領で $O(N \log N)$ で計算していける。クエリにこたえるときは二分探索すると $O(\log N)$ で求まる。

ソースコード 29　LCA.cpp

```
struct LCA {
  vector<vector<int>> par;
  int n, k;
  LCA(graph &g) : n(g.n) {
    bfs(g, 0);
    int ex = 1, cnt = 0;
    while (n >= ex) {
      ex <<= 1;
      cnt++;
    }
    k = cnt + 1;
    par.resize(n, vector<int>(k, -1));
    for (int i = 0; i < n; i++) {
      for (int v : g.g[i]) {
        if (g.dist[v] < g.dist[i]) {
          par[i][0] = v;
        }
      }
      if (par[i][0] == -1) par[i][0] = i;
    }
    for (int j = 1; j < k; j++) {
      for (int i = 0; i < n; i++) {
        par[i][j] = par[par[i][j - 1]][j - 1];
      }
    }
  }
  int lca(graph &g, int a, int b) {
    assert(g.dist[a] != LINF);
    if (g.dist[a] > g.dist[b]) swap(a, b);
    int gap = g.dist[b] - g.dist[a];
    for (int i = 0; i < k; i++) {
```

```
      if (gap & 1) {
        b = par[b][i];
      }
      gap >>= 1;
    }
    if (a == b) return a;
    int ok = g.n, ng = -1, mid;
    int res = 0;
    while (abs(ok - ng) > 1) {
      mid = (ok + ng) / 2;
      int t = mid, ta = a, tb = b;
      for (int i = 0; i < k; i++) {
        if (t & 1) {
          ta = par[ta][i];
          tb = par[tb][i];
        }
        t >>= 1;
      }
      if (ta == tb) {
        ok = mid;
        res = ta;
      } else {
        ng = mid;
      }
    }
    return res;
  }
};
```

9.5.3 木の直径

1. 任意の頂点 s を選ぶ 2. s から最も遠くにある頂点 v を DFS(/BFS) で見つける 3. v から最も遠くにある頂点 u を DFS(/BFS) で見つける 4. u と v を結ぶのが木の直径

9.6 トポロジカルソート

DAG(有向無閉路グラフ) において，その順序を保ったまま一直線に並べる。厳密には，頂点 u から v にいくらかの辺をたどって到達できることを $f(u,v)$ と表すことにすると，$f(u,v)$ が成り立つすべての u と v について，ソートしたときの順序が $u < v$ となり，かつその裏が成り立たないことである。「DAG $\Leftrightarrow$ トポロジカルソートできる」は真。

1. 頂点の入次数を記録しておく。
2. 入次数が 0 の頂点を queue に入れる。
3. queue から頂点を一つ取り出す (u とする)。
4. u に隣接している頂点をすべて調べる (v とする)。
5. v の入次数を 1 減らす。このとき 0 なら queue に入れ，-1 ならこのグラフは DAG でない。

6. queue が空になるまで 3. に戻る。

ソースコード 30　topoligical sort.cpp

```
vector<int> topoSort(const graph &g) {
  vector<int> deg(g.n, 0);
  for (int i = 0; i < g.n; i++) {
    for (auto v : g.g[i]) {
      deg[v.to]++;
    }
  }
  queue<int> zero;
  for (int i = 0; i < g.n; i++) {
    if (deg[i] == 0) {
      zero.push(i);
    }
  }
  vector<int> res;
  while (!zero.empty()) {
    int node = zero.front();
    zero.pop();
    res.push_back(node);
    for (auto v : g.g[node]) {
      deg[v.to]--;
      if (deg[v.to] == 0) {
        zero.push(v.to);
      } else if (deg[v.to] < 0) {
        return {-1};
      }
    }
  }
  if (res.size() != g.n) return {-1};
  return res;
}
```

9.7　強連結成分部分分解 (SCC)

お互いに行き来できる ⇔ 同じグループ となるようにいくつかのグループに分ける。DFS を 2 回やれば SCC が求まる。分解に対応する 2 次元配列を返す。例えば，$\{\{v_1, v_2\}, \{v_3, v_4, v_5\}\}$ などと分解されているとき，res[0] は $\{v_1, v_2\}$ に，res[1] は $\{v_3, v_4, v_5\}$ になる。

ソースコード 31　strongly conneced component.cpp

```
vector<vector<int>> scc(graph &g) {
  for (int i = 0; i < g.n; i++) g.vis[i] = false;
  vector<int> label(g.n);
  int cnt = 0;
  auto dfsLabel = [&](auto self, int node) {
    if (g.vis[node]) return;
```

```
    g.vis[node] = true;
    for (auto v : g.g[node]) {
      self(self, v);
    }
    label[cnt] = node;
    cnt++;
    return;
  };
  for (int i = 0; i < g.n; i++) {
    dfsLabel(dfsLabel, i);
  }
  graph revg(g.n);
  for (int i = 0; i < g.n; i++) {
    for (auto v : g.g[i]) {
      revg.addEdge(v, i);
    }
  }
  for (int i = 0; i < revg.n; i++) revg.vis[i] = false;
  vector<int> nowDiv;
  vector<vector<int>> res;
  auto dfsDiv = [&](auto self, int node) {
    if (revg.vis[node]) return;
    revg.vis[node] = true;
    for (auto v : revg.g[node]) {
      self(self, v);
    }
    nowDiv.push_back(node);
    return;
  };
  for (int i = g.n - 1; i >= 0; i--) {
    dfsDiv(dfsDiv, label[i]);
    if (nowDiv.size() == 0) continue;
    res.push_back(nowDiv);
    nowDiv = {};
  }
  return res;
}
```

10 データ構造

10.1 std::set

追加と検索と削除が入り乱れるようなクエリに対して有効。

書き方	操作	計算量
s.insert(x)	追加	$O(\log N)$
s.count(x)	検索	$O(\log N)$
s.erase(x)	削除	$O(\log N)$

for でループした場合，小さい方から取り出される。また，最大値や最小値の取得は，*s.rbegin() や*s.begin() でできる。重要な注意として，二分探索をするときは upper_bound(s.begin(), s.end(), val); ではなく s.upper_bound(val); が正しい。計算量が大きく異なる（前者は $O(n)$，後者は $O(\log n)$）。

10.2 std::multiset

set では同じ値の要素は 1 つまでしか持てなかったが，これは複数持つことが可能。

書き方	操作	計算量
s.insert(x)	追加	$O(\log N)$
s.count(x)	検索 (個数)	$O(K + \log N)$（$K :=$ x の個数）
s.find(x)	検索	$O(\log N)$
s.erase(x)	x を**すべて**削除	$O(\log N)$
s.erase(itr)	itr が指す要素を削除	$O(\log N)$

count(x) は計算量に注意。もし multiset のすべての要素が x だったら，$O(N + \log N)$ になってしまう。その場合は後述のように find(x) を使って，s.end() と比較するとよい。また，erase(x) をしてしまうと，x がすべて削除されてしまうので要注意。1 つだけ削除したい場合は次のように書く。

```
s.erase(s.find(x)); // 確実にxがsにある場合
// xがsにないかもしれない場合はこのようにしてチェックする
auto itr = s.find(x);
if (itr != s.end()) {
    s.erase(itr);
}
```

10.3 std::priority_queue

追加と最大値 (最小値) の取得が入り乱れるようなクエリに対して有効。

操作	計算量
追加	$O(\log N)$
最大値の取得	$O(1)$

デフォルトだと降順 (top は最大値) になるが，昇順 (top は最小値) にすることも可能。

```
priority_queue<int> q1; // 降順
priority_queue<int, vector<int>, greater<int>> q2; // 昇順
```

10.4 UnionFind

連結判定/つなぐ/サイズ取得を動的に高速でできる。$\alpha(n)$ はアッカーマン関数 $A(n,n)$ の逆関数。$A(n,n)$ は $n \geq 4$ のときにめちゃくちゃに大きい値を取るため，その逆関数 $\alpha(n)$ はとても小さい（ほとんど定数)。辺を追加することはできても削除することはできない（そのため削除クエリに対しては逆から考えることにより追加クエリとみなす考え方が有効)。あとは，ある頂点につながる辺をすべて削除したいときは，その頂点は捨てて，別の新しい頂点をそれとみなす手法も有効。

書き方	操作	計算量
`unionFind (int n)`	n 頂点で構築	$O(n)$
`int root(int x)`	x の根を返す	ならし $O(\alpha(N))$
`bool same(int x, int y)`	x と y が連結かを返す	ならし $O(\alpha(N))$
`unite(int x, int y)`	x と y をつなぐ	ならし $O(\alpha(N))$
`int size(int x)`	x の連結成分の大きさを返す	$O(1)$

ソースコード 32　union find.cpp

```
struct unionFind {
  vector<int> ps;
  unionFind(int n) : ps(n, -1) {}
  int root(int x) {
    if (ps[x] < 0)
      return x;
    else
      return ps[x] = root(ps[x]);
  }
  bool same(int x, int y) { return root(x) == root(y); }
  void unite(int x, int y) {
    x = root(x);
    y = root(y);
    if (x == y) return;
    if (-ps[x] < -ps[y]) swap(x, y);
    ps[x] += ps[y];
    ps[y] = x;
    return;
  }
  int size(int x) { return -ps[root(x)]; }
};
```

10.5 Binary Indexed Tree(BIT, Fenwick Tree)

セグ木より定数倍が軽く，実装が簡単。ただ，逆演算がないと使えない。

書き方	操作	計算量
`bit (int n)`	要素数 n で初期化	$O(n)$
`void add(int p, ll x)`	$[p]$ に x を加える	$O(\log(N))$
`ll sum(int l, int r)`	$[l, r)$ の計算結果を返す	$O(\log(N))$
`ll s(int r)`	$[0, r)$ の計算結果を返す	$O(\log(N))$

ソースコード 33　Binary Indexed Tree(BIT).cpp

```
struct bit {
  int n;
  vector<ll> d;
  bit(int num) : n(num), d(num, 0) {}
  void add(int p, ll x) {
    p++;
    while (p <= n) {
      d[p - 1] += x;
      p += p & -p;
    }
    return;
  }
  ll sum(int l, int r) {
    return s(r) - s(l);
  }
  ll s(int r) {
    ll res = 0;
    while (r > 0) {
      res += d[r - 1];
      r -= r & -r;
    }
    return res;
  }
};
```

10.6　Segment Tree

結合法則が成り立っていて単位元がある演算 (足し算，最小値，gcd...) に対して

操作	計算量
区間の演算	$O(\log N)$
一点更新	$O(\log N)$

でできる。a と b の演算の結果を返す関数 `op(a, b)` と，単位元を返す関数 `e()` は用意する。例えば，このようにラムダ式で書ける

```
auto op = [](int a, int b) { return a + b; };
auto e = []() { return 0; };
```

書き方	操作	計算量
`segTree<T> (int n, op, e)`	要素数 N, 初期値 $e()$	$O(N)$
`segTree<T> (int n, int x, op, e)`	要素数 N, 初期値 x	$O(N)$
`segTree<T> (vector<T> arr, op, e)`	vector で初期化	$O(N)$
`set(int p, T x)`	$[p]$ に x を代入する。	$O(\log N)$
`add(int p, T x)`	$[p]$ に $op([p], x)$ を代入する。	$O(\log N)$
`T get()`	$[0, n)$ の $op([0], [1], [2], ..., [n-1])$ を返す。	$O(1)$
`T get(int p)`	$[p]$ を返す。	$O(1)$
`T get(int l, int r)`	$[l, r)$ の $op([l], [l+1], [l+2], ..., [r-1])$ を返す。	$O(\log N)$
`int getEndr(int l, function<bool(T)> f)`	$f([l, tr)$ の演算結果) が true になる最大の tr を返す。	$O(\log N)$
`int getEndl(int r, function<bool(T)> f)`	$f([tl, r)$ の演算結果) が true になる最小の tl を返す。	$O(\log N)$

ソースコード 34　segment tree.cpp

```
auto op = [](int a, int b) { return a + b; };
auto e = []() { return 0; };
template <typename T>
struct segTree {
  int n, size, hight;
  vector<T> d;
  function<T(T, T)> op;
  function<T()> e;
  segTree(int n, function<T(T, T)> op, function<T()> e) : segTree(vector<T>(n, e()), op,
    e) {}
  segTree(int n, int x, function<T(T, T)> op, function<T()> e) : segTree(vector<T>(n, x),
    op, e) {}
  segTree(const vector<T> &arr, function<T(T, T)> op, function<T()> e) :
    n((int)arr.size()), op(op), e(e) {
    hight = 0;
    int ex = 1;
    while (ex < n) {
      hight++;
      ex <<= 1;
    }
    size = ex;
    d = vector<T>(size * 2, e());
    for (int i = 0; i < n; i++) d[size + i] = arr[i];
    for (int i = size - 1; i > 0; i--) d[i] = op(d[i << 1], d[i << 1 | 1]);
  }
  void set(int p, T x) {
    int i = size + p;
    d[i] = x;
    while (i > 1) {
```

```
    i >>= 1;
    d[i] = op(d[i << 1], d[i << 1 | 1]);
  }
}
void add(int p, T x) { set(p, op(get(p), x)); }
T get() { return d[1]; }       // [0, n)
T get(int p) { return d[size + p]; } // [p]
T get(int l, int r) {        // [l, r)
  l += size;
  r += size;
  T resl = e(), resr = e();
  while (l < r) {
    if (l & 1) resl = op(resl, d[l++]);
    if (r & 1) resr = op(d[--r], resr);
    r >>= 1;
    l >>= 1;
  }
  return op(resl, resr);
}
int getEndr(int l, function<bool(T)> f) { // [l, maxr)
  if (f(get(l, n))) return n;
  l += size;
  T sum = e();
  while (1) {
    while (!(l & 1)) l >>= 1;
    if (!f(op(sum, d[l]))) {
      while (l < size) {
        l <<= 1;
        if (f(op(sum, d[l]))) {
          sum = op(sum, d[l]);
          l++;
        }
      }
      return l - size;
    }
    sum = op(sum, d[l]);
    l >>= 1;
    l++;
  }
}
int getEndl(int r, function<bool(T)> f) { // [minl, r)
  if (f(get(0, r))) return 0;
  r += size - 1;
  T sum = e();
  while (1) {
    while (r & 1) r >>= 1;
    if (!f(op(d[r], sum))) {
      while (r < size) {
        r <<= 1;
```

```
          r++;
          if (f(op(d[r], sum))) {
            sum = op(d[r], sum);
            r--;
          }
        }
        return r + 1 - size;
      }
      sum = op(d[r], sum);
      r >>= 1;
      r--;
    }
  }
};
```

10.6.1 デバッグ用コード

セグ木を渡すとセグ木っぽくきれいに表示してくれる。存在意義は不明。

```
template <typename T>
void segTreeShow(const segTree<T> &seg) {
  vector<int> s = {0}, m = {1};
  for (int i = 0; i < seg.hight; i++) {
    int ns = ((m.back() + 7) >> 1) + s.back() - 2;
    int nm = (m.back() << 1) + 3;
    s.push_back(ns);
    m.push_back(nm);
  }
  m.push_back(-1);
  int ex2 = 1;
  for (int i = 1; i < seg.size << 1; i++) {
    if (ex2 == i) {
      if (ex2 != 1) printf("\n");
      m.pop_back();
      for (int j = 0; j < s.back(); j++) {
        printf(" ");
      }
      s.pop_back();
      ex2 <<= 1;
    }
    printf("%03d", seg.d[i]);
    for (int j = 0; j < m.back(); j++) {
      printf(" ");
    }
  }
  printf("\n");
}
```

10.7 遅延セグメント木

区間加算もできるようになる。

10.8 2 次元 Sparse Table

更新は効率的にできないが，2 べきで前計算しておくことにより区間に対するクエリを高速に処理できる。

書き方	操作	計算量
`sparseTable2D(a, op, e)`	a で初期化	$O(HW \log H \log W)$
`T get(int si, int sj, int ti, int tj)`	$[si, ti) \times [sj, tj)$ の計算結果を返す	$O(1)$

```
auto op = [](ll a, ll b) { return max(a, b); };
auto e = []() { return 0; };
template <typename T>
struct sparseTable2D {
  int h, w;
  function<T(T, T)> op;
  function<T()> e;
  vector<vector<vector<vector<T>>>> table;
  vector<int> tLog;
  sparseTable2D(const vector<vector<T>> &a, function<T(T, T)> op_, function<T()> e_) :
    h(a.size()), w(a[0].size()), op(op_), e(e_), tLog(max(h, w) + 1, 0) {
    for (int i = 2; i < (int)tLog.size(); i++) tLog[i] = tLog[i / 2] + 1;
    int mk = 0, ml = 0;
    while ((1 << mk) <= h) mk++;
    while ((1 << ml) <= w) ml++;
    table.resize(mk, vector<vector<vector<T>>>(ml, vector<vector<T>>(h, vector<T>(w,
    e()))));
    for (int i = 0; i < h; i++) {
      for (int j = 0; j < w; j++) {
        table[0][0][i][j] = a[i][j];
      }
    }
    for (int k = 0; k < mk; k++) {
      for (int l = 0; l < ml; l++) {
        for (int i = 0; i + (1 << k) <= h; i++) {
          for (int j = 0; j + (1 << l) <= w; j++) {
            if (k > 0) {
              table[k][l][i][j] = op(table[k - 1][l][i][j], table[k - 1][l][i + (1 << (k
    - 1))][j]);
            } else if (l > 0) {
              table[k][l][i][j] = op(table[k][l - 1][i][j], table[k][l - 1][i][j + (1 <<
    (l - 1))]);
            }
          }
```

```
        }
      }
    }
  }
  T get(int si, int sj, int ti, int tj) { // [si,ti×)[sj,tj)
    int vi = tLog[ti - si];
    int vj = tLog[tj - sj];
    T r = e();
    for (int i : {si, ti - (1 << vi)}) {
      for (int j : {sj, tj - (1 << vj)}) {
        r = op(r, table[vi][vj][i][j]);
      }
    }
    return r;
  }
};
```

11 その他

11.1 小数出力

setprecision は，デフォルトは整数部も含めた桁数指定となる。std::fixed を同時に使用することで小数点以下の桁数指定になる。

```
double ans = 1e-8;
cout << fixed << setprecision(10);
cout << ans << endl;
```

11.2 乱数

uniform_int_distribution のコンストラクタは両端を含む。すなわち，$[a, b]$ の範囲で一様ランダムな値を返す。

```
mt19937 mt(time(NULL)); // シード値は時刻

uniform_int_distribution<int> rd(1, 100000); // 整数の乱数の生成器を作る
int x = rd(mt);

uniform_real_distribution<double> rd01(0, 1); // 実数の乱数の生成器を作る
double p = rd01(mt);
```

11.3 便利記法

11.3.1 chmin chmax

DP の更新などに便利。更新されたら，またかつそのときに限り，true を返す。

ソースコード 35　chmin chmax.cpp

```
template <typename T>
bool chmin(T &a, const T &b) {
  if (b < a) {
    a = b;
    return true;
  }
  return false;
}
template <typename T>
bool chmax(T &a, const T &b) {
  if (a < b) {
    a = b;
    return true;
  }
  return false;
}
```

11.4　degug

11.4.1　**範囲外アクセスチェック**

.at() してもいいけど，コンパイラオプションをつけると楽でよい。

```
#define _GLIBCXX_DEBUG
```

もしくはコンパイル時に

```
-D_GLIBCXX_DEBUG
```

11.4.2　show

こんなことしてるくらいならソースコードにらんでバグの波動を感じろと思ったが，沼から抜け出すには割と重要）チェックするときは-D_DEBUG をつけてコンパイルする。提出時は消さなくてよい（_DEBUG が定義されていないので)。

```
#ifdef _DEBUG
#define show(x)          \
  cerr << #x << " : "; \
  showVal(x)
template <typename T>
void showVal(const T &a) { cerr << a << endl;
}
template <typename T, typename U>
void showVal(const pair<T, U> &a) { cerr << a.first << " " << a.second << endl;
}
template <typename T>
void showVal(const vector<T> &a) {
  for (const T &v : a) cerr << v << " ";
```

```
  cerr << endl;
}
template <typename T, typename U>
void showVal(const vector<pair<T, U>> &a) {
  cerr << endl;
  for (const pair<T, U> &v : a) cerr << v.first << " " << v.second << endl;
}
template <typename T, typename U>
void showVal(const map<T, U> &a) {
  cerr << endl;
  for (const auto &v : a) cerr << "[" << v.first << "] " << v.second << endl;
}
template <typename T>
void showVal(const vector<vector<T>> &a) {
  cerr << endl;
  for (const vector<T> &v : a) showVal(v);
}
#else
#define show(x)
#endif
```

一つ注意は，`if (hoge) show(x)` のようにすると，define の展開上，hoge の真偽に関係なく showVal は実行されるので危ない。波括弧でくくるべき。

11.5 格言

- マンハッタン距離は 45 度回転
- 3 つ組は大小決めて真ん中固定
- 和の期待値は期待値の和
- 差の最小化は中央値
- 困難は分割せよ（DP）
- 桁ごと bit ごとに考える
- 上界下界が実は条件を満たす
- 実は二部グラフ
- 実は不変量がある

11.6 コンパイルや実行関連

- AtCoder のコンパイルは以下で行われる

```
g++-12 -std=gnu++20 -O2 -DONLINE_JUDGE -DATCODER \
-Wall -Wextra \
-mtune=native -march=native \
-fconstexpr-depth=2147483647 -fconstexpr-loop-limit=2147483647
    -fconstexpr-ops-limit=2147483647 \
-I/opt/ac-library -I/opt/boost/gcc/include -L/opt/boost/gcc/lib \
```

```
-o a.out Main.cpp \
-lgmpxx -lgmp \
-I/usr/include/eigen3
```

- ICPC2024 のコンパイル（Python はコンパイルではないが）は以下で行われる

ソースコード 36　C

```
gcc -x c -g -O2 -std=gnu11 -static -o"$DEST" "$@" -lm
```

ソースコード 37　C++

```
g++ -x c++ -g -O2 -std=gnu++20 -static -o"$DEST" "$@"
```

ソースコード 38　Python 3

```
/usr/local/py3-venv/bin/pypy3 -m py_compile "$@"
```

- 再帰が深すぎるとローカルでは動かないことがある。AtCoder のコードテストなどでは動く。`ulimit -S -s 1048576`（`ulimit -s unlimited` でもいいらしい）とすると深さの最大値を増やすことができる。 Python ならコード内に `sys.setrecursionlimit(10**6)` と書けばよい。