snippets in C++

外積、内積

２つのベクトルの成す角が反時計回りで 180 度未満なのかの判定には外積が使える。角度を常に１８０以下とした時に、鋭角なのか鈍角なのかの判定には、内積が使える。

また、二次元のベクトルを表すのに、complex を使うと便利だったりする（足し算、引き算が定義済みなので)。

参考： 037 - Intersection

浮動小数点数の出力

小数点以下 10 桁とか出したいとき：

```
cout << fixed << setprecision(10) << ans << endl;
```

64-bit 整数では溢れる場合

64-bit では足りないが、128-bit なら足りるというのであれば、`__int128_t`, `__uint128_t` を使ってしまうのも手である。

計算途中で一時的に 64-bit を溢れる場合があるが、そういうケースを判定したいというのであれば、double で概算して範囲内に入っていることを確認する方法がある。

参考： ABC250D - 250-like Number[*1]

Bellman-Ford

ABC061D - Score Attack[*2]

Binary Search, 二分探索

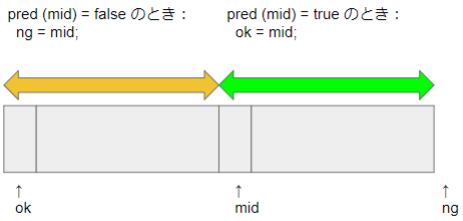
「めぐる式」[*3] がすばらしかった。

最大値を探す場合、および、最小値を探す場合、いずれも探索範囲を [ok, ng) または (ng, ok] のような半開区間として、イテレーションはまったく同じコードでおっけい：

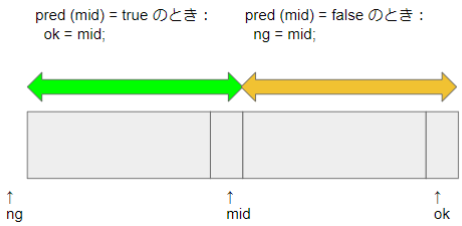
```
// 最大値を探す場合
int ok = 0;
```

[*1] https://uhideyuki.sakura.ne.jp/studs/index.cgi/ja/atcoder_drills6#2022-05-08p2
[*2] https://atcoder.jp/contests/abc061/submissions/40373762
[*3] https://twitter.com/meguru_comp/status/697008509376835584

```
    int ng = N;
    while (abs(ok - ng) > 1){
      int mid = (ok + ng) / 2;
      if (pred(mid))
ok = mid;
      else
ng = mid;
    }
```




```
    // 最小値を探す場合
    int ok = (N-1);
    int ng = -1;
    while (abs(ok - ng) > 1){
      int mid = (ok + ng) / 2;
      if (pred(mid))
ok = mid;
      else
ng = mid;
    }
```



Disjoint Set Union (Union Find)

ACL DSU[*4]
使用例：19830041[*5]

Fenwick Tree (BIT: Bit Indexed Tree)

ACL Fenwick Tree[*6]
使用例：19830137[*7]

ModInv

簡単なプログラムなので、どちらの形もその場で書けた方がいいと思う。
まずは、再帰をもちいて、以下をそのまま書いた形：

- $x^0 = 1$
- $x^{2N+1} = xx^{2N}$
- $x^{2N} = (x^2)^N$

```
long long modpow(long long x, long long n, long long p)
{
  if (n == 0) return 1;
  if (n & 1) return (x * modpow(x, n-1, p) % p);
  return modpow(x * x % p, n >> 1, p);
}
```

もうひとつは再帰せずにループで処理するもの。これは、n を下位ビットからみていって、i ビットめが立っていたら x を 2^i 乗したものを掛けていく。下位ビットからみていくので、x の 2^i 乗は x を毎ループ 2 乗していけば手に入る：

```
long long modpow(long long x, long long n, long long mod) {
    long long res = 1;
    while (n > 0) {
        if (n & 1) res = res * x % mod;
        x = x * x % mod;
```

[*4] https://github.com/atcoder/ac-library/blob/master/document_ja/dsu.md
[*5] https://atcoder.jp/contests/practice2/submissions/19830041
[*6] https://github.com/atcoder/ac-library/blob/master/document_ja/fenwicktree.md
[*7] https://atcoder.jp/contests/practice2/submissions/19830137

```
        n >>= 1;
    }
    return res;
}
```

Ratio

Binary Float では 0.01 すら正確に表現できないのですが、たまにこの辺をついてくる問題があって、double だけでいこうとすると嵌る。そこで、Haskell Prelude に習った有理数型 rat を実装してみた：Haskell-like Ratio[*8]

レポジトリには、ABC191D, ABC169C をこれを使って解くサンプルも置いてある。

計算のたびに約分をするせいか、ガツガツ計算するのには向いていない（TLE になってしまう）のだが、十進浮動小数点数として与えられた入力を正確に保持できる (cin >> a などで単純に読み込んでも double を経由しないので誤差が生じない）だけでも、便利な場面はあるかな。

なお、operator>> は実装してあるが、operator<< は実装していない。出力は、なんらか既存の型にキャストしてから行うことを想定しているので。

Segment Tree

ACL Segment Tree[*9]

使用例：19830596[*10]

Unique

unique は隣り合った重複を除いた要素を前の方に集め、重複のない要素の末尾の次のポインタを返す。unique 関数はコンテナから要素を取り除かないので、重複を取り除くには、erase などを用いる。

```
sort(v.begin(), v.end())
v.erase(unique(v.begin(), v.end()), v.end());
```

[*8] https://github.com/unnohideyuki/hlratio/

[*9] https://github.com/atcoder/ac-library/blob/master/document_ja/segtree.md

[*10] https://atcoder.jp/contests/practice2/submissions/19830596