agc

agc028_b

- N個のブロックが一列に並んでおり、取り除く際には隣接ブロック全ての値の和の力がいる。
- N回の操作により、全てのブロックが取り除かれるがこの時の力の総和
- 全ての取り除くパターンN!の総和を求める

ddcon

ddcon_b

期待値計算問題

- 数列（スライム）の並びに対して、選択されたスライムが右のスライムに合流する操作
    - スライムの数(N) - 1回行う

検討

- 一つ一つの期待値を導出する
    - 1体のスライムが歩く期待値
    - 右側のスライムとの選択順序によって歩く距離が変わる
    - **確率の計算を誤っていた** 下記の誤解

```
a b c dと並んでいたときのaの歩行距離
歩行距離がbまで=> 1/ 2
歩行距離がcまで=> 1/ 4
  bより選択順が遅い条件下でcより早い遅いかは等確率と考えていた。
歩行距離がdまで=> 1/ 4
```

解答

- 正しい確率の計算

```
a b c d と並んでいたときのaの歩行距離
全パターンは{abc, acb, bac, bca, cab, cba}
歩行距離がbまで => 1 / 2 (abc, acb, cab)
歩行距離がcまで => 1 / 6 (bac)
歩行距離がdまで => 1 / 3 (bca, cba)

上記の確率にも規則性はあり、
終端以外 => 1/n - 1/(n+1)
終端 => 1/n
となる。
```

- 確率の捉え方を変えると、

```
aがa-b間を通る => 1
aがb-c間を通る => 1 / 2 (a,bのうちaの選択順が遅い)
aがc-d間を通る => 1 / 3 (a,b,cのうちcの選択順が遅い)
```

これで計算可能！

抽象化

- 確率の計算を行う時は、何が「同様に確からしいか」を確認する必要がある。
- mintの除算は積算の前でも辻褄を合わせてくれる

類題

Removing Blocks
https://atcoder.jp/contests/agc028/tasks/agc028_b?lang=ja

abc126

abc126_d

木の問題 二色で塗った時の、同じ色に塗られた任意の二頂点の距離を偶数にしたい 辺は重み付き

検討

- 子を親と違う色で塗る！でＯＫ？
  - 辺は重み付きなのでダメ
- 探索の過程で親との距離が偶数なら親と同じ色、奇数なら違う色でＯＫ？
  - OKそう
- 隣接リストで目的地(v)と距離(w)を管理する
- fds(parent, now, parent_color)
  - 隣接リストと色はグローバルに保持

coding

- 色の更新をどこでやるか悩んだ
  - 親の関数中でやるか　⇒　こっちにした
    - 引数名はnow_colorにすべき
  - 自分の関数中でやるか

抽象化

- 木探索アルゴリズム（ほぼ基本形）

```
void dfs(int parent, int now, int now_color){
    if (root[now].size() == 1 && now != 0){
```

```
            return;
        }
        for (auto node : root[now])
        {
            if (node.first != parent){
                if (node.second%2 == 0){
                    color[node.first] = now_color;
                } else {
                    color[node.first] = -now_color;
                }
                dfs(now, node.first, color[node.first]);
            }
        }

    }
```

abc126_e

- N枚のカードには1 or 2が書かれている
- M個のヒントがある
    - A_x + A_y + Zが偶数
- 下記の魔法が使える
    - A_iの整数を知れる

検討

- 直観的にはUnion found treeっぽい

解法

- Union foundの典型問題
- 出てきたノードをmergeしていき、最終的なグループ数が答え
    - グループ数はsetにtree.root[i] を入れていき、set.size()を出力する

abc126_f

- 2^(M+1)の長さの数列
    - [0, 2^(M))の数字をそれぞれ2つずつ含む

検討

- [0, 2^(M))の排他的論理和は0
- 0以外の解作れなくね？？

解法

- [0, 2^(M))からkを引いた数の排他的論理和はkである
- [0, 2^(M))からkを引いた数列をc,その反転をdとし、
    - c k d k は条件をみたす

abc127

abc127_d

- borderが緑下位なのに解けなかった問題

ある数列Aに対し、以下の操作を行う

- 下位B_i個の数字をC_iに変える

検討

- 貪欲で解こうと思ったがO(NM)以上かかる
- BC情報のみで数列を組み立てるのも困難

解法

- 操作の順番を変えてみても答えは変わらない！
- BとCの情報から、Cの大きい順にソート
- 各CiをBi個ずつ並べた列をD1, D2, D3, ...として数列DとAのmaxを取る

abc127_e

- NxMマスのうちKマス選ぶ時のマンハッタン距離の和

検討

- 場合の数系統の問題なので何とか分割したい
- ある点1点を固定したときのマンハッタン距離を求めようとしたがうまく行かず

解法

- 「ある二点が何回足されるか」という視点で考える。
    - Choose(nm-2, k-2) 回
- ある二点の距離がxになる二点のパターン数は
    - M^2 * d * (N-d)
- d を0~N-1まで操作

抽象化

- マンハッタン距離の算出を行う際には、X座標とY座標を分けて考える
- 「A全体を調べたときのBの総和」⇒「Bの各要素bごとにA全体を調べた総和」を累計する
- mod combinationのライブラリは便利

abc127_f

- Q(<=2*10^5)のクエリが与えられる
- 更新クエリ
    - g(x) = f(x) + |x-a| + b
- 求値クエリ

    - f(x)の最小値を与えるxの最小値
    - f(x)の最小値

検討

- 今までのaのソート列Aに対してA[(N-1)/2]の要素がxとなる。
- 更新クエリによるAへの追加はlower_boundによってソートが維持されるようにしたい
- 問題はf(x)の最小値をどうやって計算するか
    - 各クエリに対してO(N)の計算をかけては行けない
    - 変則的な累積和
        - aの総和とINF-aの総和
        - 累積和自体がそもそも組めない
    - 追加時の増分を詳細検討したがWA
- そもそも、vectorのinsertは線形時間かかるのでTLE

解答

- データ構造
    - 中央値をlognで求めるために、aの値を二つのqriority_queで管理する
    - 最大値と最小値が逆転していたら、queueの値を入れ替える
    - 最大値がそのときのx
- f(x)参照
    - 中央値の範囲と挿入値の距離を随時足し合わせれば良いので、
    - 入れ替えが発生したら差分を足していく

抽象化

- 逐次値が挿入される状況下で、中央値をO(logn)で取り出したい場合は
    - prioryty_que二個(昇順、降順)使いがとても便利
        - 1, 7, 2, 6, 4の場合、二つのqueueに数字をいれて二種を比較
        - 昇順側の最小値と降順側の最大値を比較し、後者が大きければ入れ替える
    - vectorで行おうとすると、データ挿入(insert)時にO(N)かかる

coding

- priority_queueの昇順、降順に注意する 宣言時には、

```
priority_queue<int, vector<int>, less<int>> que_gre;//大きいものから取り出したい
priority_queue<int, vector<int>, greater<int>> que_less;//小さいものから取り出したい
```

abc128

abc128_c(AC)

- N個のスイッチとM個のライト
- M個のライトが複数のスイッチに繋がれている

- 点灯条件は、ONスイッチの数の余りがpに一致すること
- 全て点灯するパターン数

解答

- Nが小さいのでbitで全パターン列挙してカウント

coding

- N状態をbitで表すときは

```
rep(i, 1<<N)
```

で列挙可能

abc128_d(AC, 60min)

- 宝石の価値の最大値を求める
- queueに宝石がN個詰められており、下記のルールで取り出せる。(K回まで実行可能)
    - 左端/右端から取り出す
    - 左端/右端にpush

検討

- i回の操作でl,rまで掘った時の最大値を求めて、余りの操作で負の宝石を捨てれるだけ捨て
- 負の値を優先的に取り出したいのでset型で

解答（ほぼ同じ）

- 取り出す操作を先にやって、あとからpushしても同じ
- 取り出し方のパターンを全探索し、K-l-r回負の宝石を捨て、それらのパターンの最大値を求める。

abc128_e(AC, 50min)

- ある座標で、ある時間の区間道路工事が行われる
- 原点を任意の時間で出発し、+1/sで歩き、工事にぶち当たれば停止する

検討

- X座標での工事時間[s, t) => 0座標での[s-x, t-x)と変換でき、原点での議論に集約できる
- 工事の情報と人間の情報をヒープで管理すると、人間がどの工事でぶつかるかが計算できる
    - 工事が重複しているときは、近い座標が優先される
- 同じ時間でのソート順位は
    - 工事開始、人間、工事終了の順になるようpriority_queを組み立てる。

abc128-f(not AC)

- frogが蓮を乗り継いでジャンプし、その時のスコアの最大値を求める
- ジャンプの規則は、原点をスタート位置とし、+A,-B(両方自然数)を繰り返すような挙動を行う

- 一度乗った蓮は消えてしまい、二度乗ると実質アウト
- 蓮は3~10^5
- 蓮ごとのスコアは10^9

検討

- Aのジャンプでゴールするようなパターンしかない
- 全パターンをどのように考えればよいか、辿り着けず

解法

- 座標の軌跡は、
    - 0, A, A-B, 2A-B, 2A-2B,..., (x+1)A-xB
- C = A-Bとすると、
    - 0, A, C, A+C, 2C,...A+xCになる。
- A+xC = N-1なので、
    - 0, C, 2C,...,xC
    - N-1, N-1-C, N-1-2C,...,N-1-xC

coding

- setとpriority_queueの使い分け
    - 最大値（最小値）以外を取り出したい場合はset
    - 重複を許したい場合はpriority_que
- setの操作
    - insert() でデータ挿入
    - erase() でデータ削除
    - s.find() != s.end() で存在確認

abc129

abc129_d(AC, 15min)

- ./#が置かれた地図の.にライトを置く
- #に到達するまでの十字路を照らす、照らせるマスの最大値
- マスの大きさは2000*2000

検討

- 各列、各行の#の位置をvectorに格納していく
- 各座標に対し、vector二種のlower_boundを求め、範囲を求める
- vectorの最初と最後に番兵を仕込んでおくと実装が楽

abc129_e(AC, 40min)

- AxorB = A+B < lとなる場合の数
- lは2^100001くらいで、二進数で与えられる

検討

- あるビットに対して、lが1であるときa,bが(0,0)なら、下位ビットのパターンは3倍ずつ増える
- dp[100001][2]を作り、
    - dp[*][0]: {0,0}
    - dp[*][1]: {1,0}, {0,1}

抽象化

- xor 系の比較問題系は、前の数が比較値を下回っていたか否かでdp組むパターンが多い気がする

abc129_f

激ムズ

abc130

abc130_d

連続する部分列の和がk以上となる場合の数

検討

- 尺取り法。
    - N(N-1)/2 - k未満となる部分和

coding

- 尺取り法のコードはスクラッチで書くとミスりやすいので注意

abc130_e(not AC)

- 整数列二つを使って、部分列の組で等しいパターン（連続しなくても良い）
- 整数列としての余りの添え字の集合が違う場合はことなる部分列として扱う

検討

- dp[i][j] : S_i, T_j末尾の部分列の数(S_i == T_j)
    - dp[i+1][j+1] :dp[i][j]までの総和
    - これ計算しようとするとO(NMNM)かかる気がするが・・・・？

解答

- 総和をdpで求める
- sum[i][j] :dp[i][j]までの合計とするといい感じに立式できる
    - sum[i+1][j+1] = sum[i+1][j] + sum[i][j+1] - sum[i][j] + dp[i+1][j+1]
        - 一致のとき dp[i+1][j+1] = sum[i][j] + 1
        - 不一致の時 dp[i+1][j+1] = 0

抽象化

- 推移条件が組みにくいdp/計算量が多いdpは以下の考慮を行う

    - 累積和のdpテーブルが作れないか？
    - 漸化式が作れないか
    - 次元を増やせないか

abc131

abc131_c(AC)

- A以上B以下の整数のうち、CでもDでも割り切れないもの

検討

- B-A+1 - (Cで割れるもの) - (Dで割れるもの) + (CでもDでも割れるもの)
- cでもdでも割れるものはlcm(最小公倍数)を使う

abc131_d(AC)

- 〆切と所要時間が与えられた複数の仕事に対して、間に合うか否かを判定

検討

- 〆切が早いものから優先的にやって、間に合ってるか否かを順次判定

abc131_e(AC)

- N頂点のグラフに対して、最短距離が2となる組み合わせがM個作れるようなグラフを構成

検討

- 蜘蛛の巣状にするのが最も多い(N-1)(N-2)/2
- そこから葉同士を連結させると-1となるのでそうなるようにグラフを構成する

abc131_f(no AC)

- 3頂点から一つ頂点を追加して長方形を作成できる。
- 追加できる頂点の最大数

検討

- 少なくとも二点が存在する軸を求め、その軸数の積が答えになる？？？
    - 数えすぎだった。規則性を

解法

- (x1,y1)に対して頂点x-x1頂点y-y1を宣言し辺を引く
- 各連結成分に対して探索し、x群とy群の数をカウントし、各群の積をとる

coding

- vectorは再定義可能

```
vector<int> cnt;
cnt = vector<int>(2); //再定義可能らしい
```

- dfsはこれ

```
void dfs(int v){
    if (visited[v]) return;
    visited[v] = true;
    for(int u: to[v]) dfs(u);
    return;
}
```

abc132

abc132-e(AC)

- 有効グラフに対し最短距離が3の倍数になるかどうかを判定

検討

- グラフを3つ用意し、1->2 2->3 3->1となるよう3*Nのグラフを構成
- 最短距離を求める

抽象化

- 有効グラフであるとき、一つの辺情報に対して片方のみ貼ればよい
- 重み1の最短距離はO(V)オーダーで解ける(幅優先探索)

abc133

abc133-e(AC)

- 木に対して、二頂点の距離が2以下ならば、頂点の色は異なるような組み合わせ

検討

- 自分とも親とも違うような色を子に割り振る
    - 自分が根のとき、　P(k-1, child_num)を掛ける
    - 自分が根でないとき、 P(k-2, child_num)を掛ける

abc134

abc134-e(AC)

- 数列に対して、Ai < Aj となる部分列に分けたときの最小数

検討

- 数列を走査していき、
  - Bのあらゆる要素より最小であればBに追加する。
  - Bのある要素より大きければ、そのうちも最も大きな要素を塗り替える

coding

- upper_bound, lower_boundはソート規則を外から指定できる。

```
upper_bound(input_arr.begin(), input_arr.end(), query, greater<int>()) -
input_arr.begin();
```

abc135

abc135-e(no AC)

- ??2??5のうち13で割って5が余る場合の数

検討

- dp[i][j]: i桁までの数字において余りがjとなる数
- 立式で混乱
- 苦手

解法

- 各桁で分解する
  - ??3 = 3 + 10*A* + *100*B ≡ 3 + 10*A* + *9*B (mod 13)
- dp[i+1][j] = Σ dp[i][(j + 13 - (10*A)%13)%13] (0 <= A < 13)
  - 引き算のmodは面倒！
- dp[i+1][(j + mul * k)%13] += dp[i][j];
  - こんな感じでもかけるよ！

coding

- 13で割った数が難しい

abc136

abc136-e(AC)

- 数列aに対して、ある数字を+1,ある数字を-1する操作をk回以内で行った時の最大公約数

検討

- 数列aの総和の約数が解の候補となるので導出

coding

- 約数列挙

```
template<class X> void divisor(X input, vector<X>& Dnumber){
    for (X i = 2; i*i <= input; i++)
    {
        if (input % i == 0){
            Dnumber.push_back(i);
            if (i * i  != input) Dnumber.push_back(input / i);
        }
    }
    Dnumber.push_back(input);
    Dnumber.push_back(1);
    sort(Dnumber.begin(), Dnumber.end(), greater<X>());
}
```

abc137

abc137_c(AC,15min)

- 文字列Ｎ個のうち、アナグラムが一致する組み合わせ数を求める

検討

- 文字列を文字列内でソートすると、アナグラムが一致するものは同じ文字列に集約される
- さらに文字列間でソートすると、アルファベット順に並ぶ
- 0~N-1まで走査し、s[i] == s[i+1]になるときにカウントアップし、
- そうでないときに、cnt * (cnt + 1) / 2を足す

abc137_d(AC,25min)

- N種類の仕事があり、報酬が貰えるまでの期間Aiと報酬額Biが羅列されている。
- M日までに得られる報酬の合計の最大値

検討

- 制約条件から考えると、ＤＰではなさそう。
- 貪欲法っぽいが前からやろうとすると上手くいかない
- 日にちをM-1 => 0まで走査し、ある日にち時点で可能な仕事の内最大報酬を選んでいく

抽象化

- 貪欲法を考えるときは、走査順序を色々変えてみると上手くいくかもしれない。

abc137_e(no AC)

- N頂点M有向辺のグラフ、各辺にコインがある
- 移動の際にコインを消費する

検討

- 予め辺のコイン-消費コインをしておく
- 単一始点なので始点からのdfsを行う
- 頂点をCANNOTGOAL, UNVISITED, SEARCHING, CALCULATEDに分ける
- 各頂点からdfsして、goalに到達できるかを調査し、出来ない頂点をCANNOTGOAL
    - CANNOTGOALには進まないように探索
    - SEARCHINGに進んだ時、
        - 結果が+になれば最大値無
        - 結果が-になればreturn;
    - CALCULATEDに進んだ時、
        - 結果が+になれば最大値更新して通過
        - 結果が-になればreturn

解法

- 最長経路問題は重みに-1をかけることで最短経路問題にできる
- 重みが負の時の最短経路問題はベルマンフォード問題で解ける
- 始点- 終点の間に存在しえない頂点を省く必要がある
    - 始点からのdfsと終点からのdfsで解ける

```
void dfs(int v){
    if (reachableFrom1[v]) return;
    reachableFrom1[v] = true;
    for (int u: to[v]){
        dfs(u);
    }
}
```

coding

- 複数の数値を持つセットをvectorの管理する場合はtupleが便利

```
vector<tuple<int, int, int>> edges;
```

abc139

abc139_e

- N人で総当たり戦を行う
- 各人ごとに、守るべき対戦順が規定されている
- 何日で終わるかを出力

検討

- 試合を挑戦とおいて、優先順に対して有向辺を引く
- トポロジカルソートして、最長路が答え
    - 実装できねえ・・・

coding

- 二点の情報を元に頂点番号を割り振る時は、変換用のテーブルと関数を用意する

```

```

- トポロジカルソート(あってないかも)

```
void tsort(vector<vector<int>> &p, vector<int>& ans){

    stack<int> st;
    for (int i = 0; i < n*(n-1)/2; i++)
    {
        if (input[i] == 0) {st.push(i);}
    }
    while(!st.empty()){
        int i = st.top();
        st.pop(); //最後でpopするとpushしたものがpopされる
        ans.push_back(i);
        for(auto point : p[i]){
            input[point]--;
            if (input[point] == 0)
                st.push(point);
        }

    }
}
```

abc142

abc142_f(no AC)

- 有効グラフG(V,E)が存在
- 入次数1、出次数1となるような部分グラフを求める。

検討

- 閉路を求める問題では？？
    - トポロジカルソートの閉路検出アルゴリズムを元に作れそうだが、各ノードを覚えて置く必要がある

解法

- 最小サイズの閉路を見つける問題だった
- bfsである始点から探索し、始点にもどるノードを見つける
- 探索時には木構造なので親ノードを記録する変数を作っておくと、逆に辿れる

abc145

abc145-e(no AC)

All you can eat

- ラストオーダーに制限があるなか食べ物を最大の満足度のなかで食べれるだけ食べたい
- 注文したらすぐ提供されるが、食べている時間は注文できない
- 食べる時間と満足度が与えられる

検討

- T-1時間まででナップサック問題を解けばいいが、T-1時間より後に何を食べるかが定まらない

解法1

- まずT-1時間より後に何を食べるかを全探索で求めようと考える
    - i以外の料理を使うときのT-1時間での最大満足度(O(NNT)) ⇒間に合わない！
- 1~iまでの料理を使うときのdpとi~Nまでの料理を使う時のdpで分離する
- 最後は、dp1[i-1][j] + dp2[i+1][t-1-j] + b[i]のmaxで求める

解法2

- 最後の食べ物は食べる時間が最も長いもので良い
    - ≠候補の内、食べる時間が長いものを最後に持っていくと良い(誤り)
- 食べ物を時間で昇順ソート
    - dp1[i-1][t-1] + b[i] のmaxでOK

抽象化

- 計算量を考えず立式して、計算量を減らす工夫を考えるのがよい

coding

- ローカル変数で3000*3000にするとスタックオーバーフロー発生する。

abc147

abc147-c(WA, AC)

- N人が他の人の証言をして、矛盾が無い中での正直な最大人数を求める

coding

- bitカウント

```
int counter(int x) {
    if(x == 0) return 0;
    return counter(x >> 1) + (x & 1);
}
```

abc147-d(WA, AC)

- 数列に対して、二数のXORの全ての組み合わせの総和を求める

検討

- bitごとに全ての数字を舐めると、1がある数 ×　0がある数で加算される数が求まる

coding

- intのoverflowを只管警戒すべし(1WA)

抽象化

- 総和問題は総和の順序を帰る

abc147-e

検討

- ある地点x, yでの最小値を更新していけば、
    - ある地点x, yでの最小値がH,W地点での最小値とは限らない
    - 最小値を記録する二次元dpでは不可能

回答

- 三次元dp(T/F判定)

coding

- 前の要素がtrueであるかを確認する際、範囲外の参照とならないように注意(一敗)

抽象化

- 単一の値を記憶するn次元dpではダメ
    - 複数の値のT/Fを記憶するn+1次元dpで実現する！
- 「任意の数列を足すか引くかして出る和の最小値」の算出でも使用可能

abc147-f

検討

- num = iX + jDが成す数字において、

    - i = mのとき、j = 0+1+...+(m-1) ~ (N-1)+(N-2)+...+(N-m)
- ただ、i, jが別でもnumが一致してしまうパターンがある。
    - X => D, D=> Xに統合する？？
        - 場合分けが面倒だった。
        - 配列サイズが1になるとREとなる？？
- 区間の和集合の算出
    - カウンタを用意して、
        - leftでインクリメント/rightでデクリメント
        - カウンタが1以上のとき計算する

解答

- i = mごとの数列をmod D で管理する。
- f(m), f(m) + D , f(m)

coding

- map型で同一キーをpushした場合、値がvector型ならば複数追加される
- 範囲for文使用時には、コンテナ内の要素を変更できないパターンがある
    - 変更したい場合は、auto &i と書く
    - https://cpprefjp.github.io/lang/cpp11/range_based_for.html

abc148

abc148-e

- n * n-2 * n-4 * ...の時の0の数

検討

- 何回10(5*2)を使うかがポイント
- nが奇数であれば0
- nが偶数であれば5の倍数の数(重複含む)
    - 10の倍数であれば+1
    - 50の倍数であればさらに+1
    - 250の倍数であればさらに+1

coding

- 10^18はllの許容範囲

abc148-f

- 木の上で青木君と高橋君が鬼ごっこを行う。
- 青木君⇒高橋君の順番で1マス動く
- 木の形状と2人の初期位置は与えられる。

検討（解答）

- 各頂点に対して青木君からの距離、高橋君からの距離を算出する。
- 高橋君の距離＞青木君の距離となる頂点の内高橋君の距離の最大値-1が答え

coding

- queueを使った幅優先探索は意外と引っかかる
    - 通過した点を記憶しておかないとループする
    - q.front()で最初に入れたデータを参照する
        - q.back()ではない。
    - 親ノードに依存する処理はfor文中に行う。
        - 親ノードがpopされ次のwhile文になった時、選択したノードの親がどれか判別し難いため。

abc149

abc149-e

固有のパワーを持つ人間と両手で握手して幸福度を上げる。同じ組み合わせの握手はできない。

- 人数N ~100000 O(NlogN) まで
- 握手回数M ~N^2まで
- 幸福度 ~10^5

```
5 3
10 14 19 34 33
=> 202

9 14
1 3 5 110 24 21 34 5 3
=> 1837
```

真面目に解こうとすると、N^2の足し合わせをソートしてM回足すことになる。 O(N^2 * M) ⇒M回の繰り返しを避ける必要がある。

足し合わせ後の大きさに規則性はないか

```
一番大きい数 + 一番大きい数
二番大きい数 + 一番大きい数
一番大きい数 + 二番大きい数
↓移行の順序が曖昧
三番大きい数 + 一番大きい数
二番大きい数 + 二番大きい数
```

解法

- 定数X以上となる、握手の組み合わせ数を導出(O(N))
- 組み合わせ数がM以上の最小のXを二分探索で求める(O(NlogN))
- 定数Xからスコアを計算

抽象化

- ある条件(握手の回数はM回)下での、総和の最大値を求める問題 ⇒結果となるパラメータから条件を求める小問題を作成
- 二分探索で提示された条件を満たすような結果パラメータを導出

coding時の注意

- 中身が分かっている数列の二分探索はlower_bound(),upper_bound()を使うのが一番早い
    - 戻り値はlower_bound()-begin()でひくか、end()-lower_bound()から引くと良し
- 二分探索を書く時の初期値は注意。
    - leftの値は導出される解の最小値-1である必要がある。
    - rightの値は導出される解の最大値+1である必要がある。

所要時間

7時間

類題

Median of Median

abc149-f

頂点を黒or白（共に1/2の確立）で塗り、全ての黒頂点を含む最小の部分木の中の白頂点の個数の期待値

input

- 頂点数N
- 辺 ~N-1

output

- y/xに対して、xz === u (mod 1000000007)となるz

答えは木の形状で一意に定まる

検討

- ほぼ見当がつかず。
- 頂点数で一意に定まらないことは分かった。（形状によることがある）
- 最小の部分木であるため葉は黒となる。 ⇒(部分木の頂点数 - 葉の数)/2で求める・・・？

解法1

- 各辺が部分木Sに含まれる確率の合計+1が部分木Sの頂点数の期待値となる。
    - 空グラフ(黒点が0)でない場合の話
- 辺の両端の頂点に黒点が含まれる確率を求めれば良い

    - この確率を求める段階でグラフの形状は加味されている
    - 導出式は、(2^e - 1)(2^(n-e)-1)/(2^n): eは辺より下位の頂点数
- 確率の合計+1 -(空グラフとなる確率1/(2^n))が部分木Sの頂点数の期待値
- 頂点数の期待値 - 黒丸の期待値(n/2)が答え
- 有理数b/aに対してaz=b(mod 1e9+7)となるzを求めるためには、aの逆元を求めれば良さそう。

解法2(頂点に注目)

- 穴あき頂点の個数の期待値を求める。
    - ⇒各頂点が穴あき状態となる確率の総和を求める。
- ある頂点が穴あき状態となるパターン数の計算
    - 自信が白であり、接続先の頂点に黒点を含むような辺がふたつ以上ある。
    - パターン数 = 2^{n-1} - 1 - Σ(2^{e} - 1)
        - 自信が白: 2^{n-1}
        - 0の場合、1パターン
        - 1の場合、Σ(2^{e} -1) パターン(辺の数だけ和算)
- 上記のパターン数を前頂点に渡って計算し、2^{n}で割る

不明点

- (解法1)頂点数の期待値から黒丸の期待値(n/2)を引いているところ。
    - 頂点数/2を引くか、あるいは部分木Sの末端を除く頂点数/2を引くべきでは・・・？

coding

- mod計算のオンパレード:累乗、逆元
    - 演算子オーバーロードが強すぎる
    - b/aに対して、az=b(mod 1e9+7)の導出は普通にb/aの逆元の計算でできた。
- 再起関数を使った深さ優先探索
    - 辺に対して子要素の数を記録する必要があるので、stackでは実装できず。。

抽象化

- 期待値計算問題
    - 分割して、個々の確率を求める問題にするのは典型らしい

abc150

abc150_c

Count Order 1~N順列の組み合わせを辞書順でソートした時の要素Pと要素Qの差

検討

- 順列生成して、lower_boundで要素番号を求める

coding

- 順列生成
    - 再起関数 + bit情報で探索済を記録
    - next_permutation使う

abc150_d

abc150_e

abc150_f

- 長さNの数列a,bが存在する
- 数列aをkだけ巡回させ、xとのxorを取ったとき、bと一致するようなk,xを全て求める
- N^2を許容しない、a <2^30

検討

- 各bitごとに計算する？
    - 011100/111000 に対してO(N)でシフト量をもとめる必要があり、ここで頓挫

解法

- a' = b の全てが一致を証明するには、下記を示せばよい
    - a', bの要素の一つが一致
    - a'[i] ^ a'[i+1] = b[i] ^ b[i+1]　全てのiに対して
- a[i] ^ a[i+1]の数列を構成
- b[i] ^ b[i+1]の数列を構成
- aの数列がbの数列に含まれるかをリストアップ
    - bの数列を二倍にしたものを使う
- k, a[k]^b[0]を算出

abc151

---

abc151_c

- 問題とAC/WAの文字列が与えられる
- AC数とWA数を数える

検討

- 問題がACしているかどうかのフラグを持ち、
    - 未AC
        - AC => ac_cnt++, AC状態に
        - WA => wa_cnt++

解法

- 「未AC問題のWAはカウントされない」条件の検討もれ

抽象化

- 問題をよく読もう

abc151_d

- 迷路が与えられて、任意の点をスタート、ゴールとしたときの最長経路

検討

- 開始点を降って全探索
- 迷路は幅優先探索で解く

coding

- 迷路&幅優先探索
    - 迷路の距離をINFで初期化しておく
    - queueに開始位置を詰める
    - 次の位置を4パターン降って、下記を判定
        - 迷路の距離がINFか
        - 壁じゃないか
        - 端じゃないか
    - 判定OKなら迷路の距離を更新
- 壁から開始しないように注意

abc151_e

coding

- combination計算はTLEに注意する
    - これ凄すぎる
    - https://github.com/atcoder-live/library/blob/master/comb.cpp

abc151_f

- 最小包含円

検討

- 円の中心を導出して、半径を求める
    - できない

解法

- f(x, y) : (x, y)から最遠点までの距離を三分探索
- 最遠点の方向に少しずつ動かす

coding

- 三分探索
    - 単峰関数の極値を求める
- 平面上の三分探索は、

抽象化

- 単峰関数の極値を求める
    - 三分探索
    - 微分を二分探索

abc152

---

abc152_e

- 数列に対し、最小公倍数/個々の数列の足し合わせを求める
- 大きな数の最小公倍数を求める

検討

- mod計算しながら最小公倍数を求める作戦
- 無理、1000000007で割った瞬間素因数の情報は消失する
- とりあえず全ての数を書けて何らかの数で割る作戦
- TLEで出せず。。

解法

- 最小公倍数の管理を素因数で行う
- map<int, int>で素数、指数を管理する
- 最小公倍数をmapから算出(mint)
- 最小公倍数(mint) / 数列(mint)の総和

abc152_f

- N頂点N-1辺の木
- 下記条件m個を満たすものの個数
    - a, bの間を結ぶパスの内、黒く塗られているものが一つ以上存在しなければいけない

検討

- a,bの間を結ぶパスは一意に定まる
- abc149-fのように確率の問題に落とせないか
    - l_xが黒である確率⇒そっからどうやって求めるのだ
- for文をどう回す
    - ある辺に対して?
    - ある頂点に対して?

解法

- 場合の数を求める⇒余事象を計算する
- 条件が一つの場合
    - ある区間を白で塗った場合における、2^(残りの区間)
- 条件が二つの場合

    - 2^(区間A以外の頂点)　を足す
    - 2^(区間B以外の頂点)　を足す
    - 2^(区間AとB以外の頂点)　を引く
- 包除原理を使うと、複数条件のベンズの和集合（余事象の合計）が求まる
- 全ての条件の組み合わせ2^mに対して、以下を計算する
    - 各区間内の辺の和集合の数をcとして、2^(n-1-c)
    - 組み合わせ数に応じて正負を逆転（包除原理）
    - 組み合わせ0は全集合として2^n

抽象化

- 場合の数の算出⇒余事象の計算に落とし込む案
- 和集合を求めたい⇒包除原理
- 包除原理を議論するときは、2^mの組み合わせが必要となる
    - 0 ~ 1<<m の組み合わせが必要
    -

coding

- グラフ情報を格納するときは、tmp--を用いる
- 下記のコードは誤り。push_backは追加処理のため、4番目に追加される

```
vector<int> n(3)
n.push_back(0)
```

- __buildin_popcountll(i) iの二進数表記における1の数をカウント

abc154

abc154_d(WA)

数列が与えられ1~piが等確率でででるサイコロを人並びのk個降る時の最大期待値

検討

- 最大値となる区間を求めて、+kして2で割る

coding

- 小数を表記する際はprintf("%lf¥n", (double)aa)でやらないと、表示上WAとなってしまうことがある。

abc154_e(AC, 200min)

N以下の数字で非0を含む数がk個である場合の数を求める

検討

- 桁dpであることは直ぐ気付けた
- dp[i][j][k]: i桁目までの残り非ゼロ要素がjのときの場合の数(k=1のとき=となる)
- 実装でパニック

抽象化

- dpを組む時は、推移が+方向になるように要素を選ぶ
- 遷移のパターンが多い場合は、イテレータを計算し、
    - dp[ni][nj][nk] += dp[i][j][k]で集約する
    - ni, nj, nkを場合分け計算

abc154_f(no AC)

- 2次元平面上の経路の総数
    - startは0,0だが、goalはある矩形の範囲全て
    - 矩形のサイズは10^6

検討

- choose(i+j, i)で求まるが、この総和を計算するのをどうやって簡略化するか不明

解法

- chooseのsumの考え方
    - パスカルの三角形上で、横一列の和は2のべき乗で求められる
    - パスカルの三角形上で、斜め列の和は一つ下と同じ

abc155_d(no AC)

抽象化

- ある値を抽出したいが、候補が多く一つ一つ見ていくと間に合わない
    - 解を二分探索で求める
- k番目に大きい値を求める場合
    - 「x未満がk個未満」であるxの最大値を二分探索
- 少し面倒な条件の場合、lower_bound()を使わず、スクラッチで二分探索書いたほうが良し

coding

- めぐる式二分探索

abc155_e

- 1, 10, 100, ... のお札での最小の取引枚数

検討

- 上位bitから貪欲法っぽく解こうとするも上手く行かず。
- ぴったり払うか、お釣りを貰うかの二択を選び続けるようなイメージ？

解法

- dp[i][0]:上位i桁目でおつりをもらう前提での最小枚数
- dp[i][1];上位i桁目でおつりをもらわない前提での最小枚数

```
dp[0] = min(old_dp[0] + (10-num)-1, old_dp[1]+(10-num)+1);
dp[1] = min(old_dp[0], old_dp[1]) + num;
```

abc155_f

- 爆弾がある座標におかれており、ON/OFFになっている
- ある区間の爆弾のON/OFFを反転させるスイッチが存在
- 全てOFFにできるか判定、できるならスイッチを出力

検討

- 区間系の問題なので、区間情報と爆弾情報をデータ構造にぶちこんで貪欲法？
- 結局貪欲法から先が進まない

解法

- 爆弾のON/OFFの羅列を01で表し両端に0を挿入：bom(n+2)
- 差分列をとり、変化があった場合に1を立てる: mid(n+1)
- スイッチの情報を差分列のどこに対応するかをfrom, to, id で表す(tuple): range(m)
- midの情報を頂点V、rangeを辺Eとするグラフを構築する
- 各連結成分に対して、
    - 全域木を作成する(bfs)
    - dfsで深さ優先探索、子成分が1ならば辺をONにする。

coding

- visited[]を使うと、fdsしながら全域木が作れる
- 0->1, 1->0の操作を同じ式でやるには、a = a^bでよし
- pairで足りないときはtupleを使うと良し

abc156

abc156_c

- 複数の座標にN人の人間がいて、座標Xでパーティを行う際の移動距離の二乗合計

検討

- 二乗の式を展開すると、O(N)で計算することが分かる

abc156_d

- n本の花からできる花束の数
- a, b本の花束は除外(a,b本は10^5以下)

- n は10^9

検討

- nが大きいのでchooseのライブラリをそのまま使うとダメ
- nC1+nC2...は二項定理で計算可能
- nCa, nCbを引く必要があるので、これは手計算

```
mint comb(int n, int k){
    mint fact, ifact;
    fact = 1; ifact = 1;
    for (int i = n; i > n-k; i--) fact *= i;
    for (int i = 1; i <= k; i++) ifact *= i;
    return fact / ifact;
}
```

abc156_e(50min, AC)

- 各部屋に1名ずついる、k回の移動であり得る組み合わせの総数

検討

- 0の数がk個以下の場合ならOK
- 余事象で解いた
    - 合計nとなる要素列(n)の組み合わせはC(2n-1, n)
    - そこから、0の数がk個より多い場合を引く
- 0以外の数がk個の求め方
    - n-1の候補に棒をk-1個入れる　C(n-1, k-1)
    - 両端ork-1の部分に残りの棒(n-1)-(k-1)個を重複を許して入れる
        - k+1の箱にn-k個入れる C(n, k)

抽象化

- n個のものをk個の箱に重複を許して入れる　C(n+k-1, k)
- n個のものをk個の箱に重複を許さず入れる　C(n, k)

abc156_f(no AC)

- ある数列Aはある数列Bを循環するように足していく
- ある数列Aの要素のmodが増える数

検討

- 数列Bは予めmodで持っておいて良さそう
- 数列Bの要素をそれぞれ何回足すかは算出可能なのでそれでなんとか・・・

解法

- 余事象で考えると求めたい解は、

    - 数列Aの長さ(-1) - 数列Aのmodが不変の場合 - 数列Aのmodが減る場合
- 数列Aのmodが不変: 0要素×巡回数
- 数列Aのmodが減少: 数列Aの最初と最後の割るmodでの商の差

抽象化

- 条件下での場合の数系は余事象も考慮

Tips

グラフ理論

ある地点からの目的地の辺の番号を返す

```
/* 目的地までの経路を返す */
bool fds(int parent, int to, int from, vector<int>& root){
    if (from == to){
        return true;
    }
    for (auto edge : a[from])
    {
        if (parent == edge.to) continue;
        if (fds(from, to, edge.to, root)){
            root.push_back(edge.id);
            return true;
        }
    }
    return false;
}
```

最小公倍数

最大公約数

```
ll gcd(int x, int y){ return y?gcd(y, x%y):x;}
ll lcm(ll x, ll y){ return (x*y)/gcd(x,y);}
```

約数

素因数分解

```
template<class X> void factorization(X input, vector<X>& Pnumber){
    for(X i = 2; i*i <= input; i++)
    {
        if (input % i == 0){
            while (input % i == 0){
```

```
                input /= i;
                Pnumber.push_back(i);
            }
        }
    }
    if (input != 1) Pnumber.push_back(input);
}
```

約数列挙

```
template<class X> void divisor(X input, vector<X>& Dnumber){
    for (X i = 2; i*i <= input; i++)
    {
        if (input % i == 0){
            Dnumber.push_back(i);
            if (i * i  != input) Dnumber.push_back(input / i);
        }
    }
    sort(Dnumber.begin(), Dnumber.end());
}
```

累積和

添字等で混乱しがち！ 累積和を何も考えず書けるようにする
https://qiita.com/drken/items/56a6b68edef8fc605821

- s[i] : [0, i)区間の和
    - s[3] = a[0] + a[1] + a[2]
    - [2, 7)区間の和はs[7] - s[2]

しゃくとり法

https://qiita.com/drken/items/ecd1a472d3a0e7db8dce

```
ll ans_ = 0;
int right = 0;

ll sum = 0;
rep(left, n){

    while(right < n && (sum + a[right]) < k){
        sum += a[right];
        ++right;
    }
    ans_ += (right - left);
    if (right == left) ++right;
    else sum -= a[left];
}
```

ユースシーン

長さnの数列において、

- 「条件」を満たす区間のうち、最小・最大の長さを求める
- 「条件」を満たす区間を数え上げよ

方法

- 区間の開始地点を固定し、終了地点を条件を満たす範囲で動かす
- 区間の開始地点を1増やす。

再起関数の組み方のポイント

- 小問題に分解していき、最終的にreturnさせるようにする
    - 関数の最初にreturnを置く
    - fdsを呼び出さないようにして、関数の最後にreturnを置く
- 何らかの処理を行うとき、親側で行うか、小側で行うかを明確にする
    - 色を塗る
- 子から親に情報を引き継ぐ必要があるときは、戻り値で受け取るか、グローバル変数で受け取るか
- 具体例から考察するときは、葉とその親から考えてだんだん上に上ってくイメージで考えるといいかも

二分探索

ある条件に対して単調的にtrue/falseが推移する数列や数字列に対し、
true/falseの境界を求めるための手法。

- 計算量は区間[l, r]のサイズの対数時間
- lの初期値,rの初期値は、確実にtrue/false, false/trueになる値にする必要がある。 *（あるいは参照されない値にする必要がある）
- l,rによりtrue,falseの境界が定まる

```
ll l = MIN_INI; //最小値(確実にtrue/falseとなる)
ll r = MAX_INI; //最大値(確実にfalse/trueとなる)
while(abs(l-r) > 1){
    ll mid = (l + r) /2;
    bool ok = [&]{
        //条件
        return tf; //true/false
    };
    //midがtrueだった場合、true側を更新
    //midがfalseだった場合、false側を更新
    if (ok) l = mid; else r = mid;
}
// l, rが最終的に境界となる。
```

思考ポイントとしては、条件をどう設定すべきか？という点

- ある候補(要素数大)の中からk番目の数字を選ぶ
    - => xより小さい数がk個未満の中で最大値を求めたい
    - => xより小さい数がk個未満か否かでT/F判定
- abs(l-r)>1 のお陰でl,rの初期値での処理は行われない
    - l + r / 2は正負によって切り上げ、切り捨てが変わるので考慮が面倒だが、 abs(l-r)>1のときにl,rと一致する可能性はない

lower_bound, upper_bound

共に、**昇順ソートされた**数列に対して適応する。

- lower_boundは探索したいkey以上のイテレータを返す。
- upper_boundは探索したいkeyより大きいイテレータを返す。
- 要素を取得するためにはアスタリスクを付ける
    - *Iter //Iter = 4(値)
- 要素番号を取得するためには.begin()で引く
    - Iter - a.begin() // 3(要素番号)

binary_search

ソートされた配列やvectorの中に、keyがあるかどうかをTrue/Falseで返す。

部分的にWAになるとき

オーバーフロー

int ⇒ llに変えよう

想定範囲外

- 配列のサイズ誤り
- 二分探索の開始点の誤り
    - left/rightは答えとなり得る範囲外の値からスタートする必要がある。

sort時のルールを定義したいとき

第一要素のみを降順、第二要素はtrueが最初に来るように

```
bool cmp(const pair<ll, bool> &a, const pair<ll, bool> &b){
    if (a.first != b.first){
        return a.first < b.first;
    } else {
        return a.second > b.second;
    }
}

sort(range[i].begin(), range[i].end(), cmp);
```

mapの使い方

宣言/代入

```
map<ll, vector<P>> mp;
mp.insert(make_pair(i, p));
mp.emplace(i, p);
```

順列生成ライブラリ

next_permutation(v.begin(), v.end())

priority_queueの使い方

宣言時には、

```
priority_queue<int, vector<int>, less<int>> que_gre;//大きいものから取り出したい
priority_queue<int, vector<int>, greater<int>> que_less;//小さいものから取り出したい
```

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007
#define rep(i, n) for(int i = 0; i < (int)(n); i++)
typedef long long ll;

typedef struct{
    int mem1;
    int mem2;
} test;

int main(){

    auto compare = [](const test& a, const test& b){
        if (a.mem1 != b.mem1){
            return a.mem1 > b.mem1;
        } else {
            return a.mem2 > b.mem2;
        }
    };
    priority_queue<test, vector<test>, decltype(compare)> p(compare);

    p.push({1,2});
    p.push({7,2});
    p.push({3,2});
    p.push({4,2});
    p.push({6,2});
    p.push({3,2});
```

```
    p.push({1,5});

    while(!p.empty()){
        printf("%d %d\n", p.top().mem1, p.top().mem2);
        p.pop();
    }

    /*
    1 2
    1 5
    3 2
    3 2
    4 2
    6 2
    7 2
    */

}
```

グラフ系アルゴリズム

幅優先探索

- 重み無最短経路
- O(V)

トポロジカルソート

有向非巡回グラフ(DAG)の各ノードを順位付けして、 どのノードの出力も先のノードになるように並べる。

入次数を用いたアルゴリズム

- 事前に入次数を管理するデータ構造が必要！
- 順序を格納するデータ構造(vector)を用意
- 入次数が0の頂点を探し、stackに入れる(stackでなくとも、集合ならなんでもＯＫ)
- 集合からデータを抜き出して下記を行う（popはここでやっておかないとループ内の物と混ざる）
    - vectorにデータを格納
    - 子ノードの入次数をマイナス
    - 子ノードの入次数が0となればstackに入れる
- 上を繰り返す
- vectorの要素数が頂点数を満たしていないとき、閉路が存在する

DFSを用いたアルゴリズム

- 全ての点から深さ優先探索を行い、その帰りがけ順がトポロジカルソートの結果になる。
- A -> B -> Cの依存関係があったとき、深さ優先探索を抜けるのはC -> B -> Aの順番になる
- DFSは訪問済か否かを表すフラグを用意し、下記のように行う
    - 訪問済ならreturn
    - 未訪問なら訪問済にする

    - 子ノードを再起関数で呼び出す
    - 関数末尾でノードを追加する

```
void dfs(int u) {
  if(used[u]) return;
  used[u] = true;
  for(auto& i: g[u]) dfs(i);
  // 帰りがけ順で追加
  ans.push_back(u);
}
```

- 得られたノード順序を反転させる
- 予め有向辺を逆向きに張っておき、上記の処理を行ってもよい
    - この手法を用いると最長路を同時に求めることができる。
- 閉路検出
    - 訪問済かつ、未計算のものに到達してしまった場合
    - 閉路を検出すると-1を親に返す

```
int dfs(int v) {
    if (visited[v]) {
        if (calculated[v]) return dp[v];
        else return -1;
    }
    visited[v] = true;
    dp[v] = 1;
    for (int u: to[v]){
        int res = dfs(u);
        if (res == -1) return -1;
        dp[v] = max(dp[v], res + 1);
    }
    calculated[v] = true;
    return dp[v];
}
```

ダイクストラ法(単一始点最短経路)

http://kuuso1.hatenablog.com/entry/2015/12/20/212620

ベルマンフォード法(負辺ありの最短経路)

- 計算量がO(V*M)
- 全ての辺に対して頂点数N回を下記の操作を行う
    - 辺u->v(重みw)に対して、 d[v] = min(d[v], d[u] + w)
- 負辺があってもＯＫ
- N回後に更新があると負閉路がある

```
    int step = 0;
    while(upd){
        upd = false;
        rep(i, m){
            int a,b,c;
            //tuple unpack
            tie(a, b, c) = edges[i];
            if (!ok[a]) continue;
            if (!ok[b]) continue;
            int newD = d[a] + c;
            if (newD < d[b]){
                upd = true;
                d[b] = newD;
            }
        }
        step++;
        if (step > n){
            puts("-1");
            return 0;
        }
    }
```

ワーシャルフロイド法(全頂点間最短経路)

mint

```
long mod = 1000000007;
/* C++の構造体 メンバがpublicなclass */
typedef struct mint
{
    ll x;
    mint(ll x = 0) : x(x % mod) {} //コンストラクタ
    /* 演算子オーバーロード */
    mint &operator+=(const mint a)
    {
        if ((x += a.x) >= mod)
            x -= mod;
        return *this;
    }
    mint &operator-=(const mint a)
    {
        if ((x += mod - a.x) >= mod)
            x -= mod;
        return *this;
    }
    mint &operator*=(const mint a)
    {
        (x *= a.x) %= mod;
        return *this;
    }
```

```
    mint operator+(const mint a) const
    {
        mint res(*this);
        return res += a;
    }
    mint operator-(const mint a) const
    {
        mint res(*this);
        return res -= a;
    }
    mint operator*(const mint a) const
    {
        mint res(*this);
        return res *= a;
    }
    mint pow(ll t) const
    {
        if (!t)
            return 1;
        mint a = pow(t >> 1);
        a *= a;
        if (t & 1)
            a *= *this;
        return a;
    }
    // for prime mod
    mint inv() const
    {
        return pow(mod - 2);
    }
    mint &operator/=(const mint a)
    {
        return (*this) *= a.inv();
    }
    mint operator/(const mint a) const
    {
        mint res(*this);
        return res /= a;
    }
} mint_t;

ll intpow(mint_t m, int n)
{
    if (n == 0) return 1;
    mint_t ans(1);
    while (n >= 1)
    {
        if ((n % 2) == 0){
            m *= m.x;
            n /= 2;
        }else {
            ans *= m.x;
            n -= 1;
        }
```

```
    }
    return ans.x;
}

struct combination {
  vector<mint> fact, ifact;
  combination(int n):fact(n+1),ifact(n+1) {
    assert(n < mod);
    fact[0] = 1;
    for (int i = 1; i <= n; ++i) fact[i] = fact[i-1]*i;
    ifact[n] = fact[n].inv();
    for (int i = n; i >= 1; --i) ifact[i-1] = ifact[i]*i;
  }
  mint operator()(int n, int k) {
    if (k < 0 || k > n) return 0;
    return fact[n]*ifact[k]*ifact[n-k];
  }
};
```

切り上げの割り算

```
// -3 / 5 = -2, 4 / 3 = 2
bool divabsfloor (ll x, ll y, ll& ans) {
    if(!y) return false;
    int mut = 1;
    if ((x < 0 && y >= 0) || (x >= 0 && y < 0)) mut = -1;
    ans = (abs(x)+abs(y)-1)/abs(y) * mut;
    return true;
}

// -3 / -4 = 0, 3 / 4 = 1
bool divfloor (ll x, ll y, ll& ans) {
    if(!y) return false;
    ans = x/y;
    if (x%y && ans > 0) ans++;
    return true;
}
```

3つ以上のデータの配列管理したいとき

- tupleが便利

```
vector<tuple<int, int, int>> edges; //宣言する
edges.emplace_back(tmpa,tmpb,tmpc); //入れる
tie(a,b,c) = edges[i]; // 受け取る
```

printfを楽にしたい

これでできるの？

```
inline int isum(){ return 0;}
template<typename First, typename... Rest>
int isum(const First& first, const Rest&... rest){  return first + isum(rest...);}
```