1 全体のテンプレ

別にこれに固定しなくてもよい

```
/*
input data
*/

void init(){
}

bool input(){
}


/*
dp とか
*/
void init_solve(){
}

void q_input(){
}

int solve(){
  return 0;
}


int main(){
  int m;
  for(int i=0;i<m;i++){
    cout<<solve()<<endl;
  }
}
```

2 include

```
#include<iostream>
#include<iomanip>
#include<cmath>
#include<string>
#include<functional>
#include<vector>
#include<list>
#include<queue>
#include<stack>
#include<set>
#include<map>
#include<algorithm>
#include<utility>
#include<bitset>
#include<climits>
```

```
16 #include<cstdio>
17 #include<cassert>
18 using namespace std;
```

3 素数

```
vector<int> prime;
vector<int> prime_list;
void prime_set(int n){
  n+=100;
  prime.resize(n);
  for(int i=0;i<n;i++){
    prime[i]=1;
  }
  prime[0]=prime[1]=0;
  for(int i=0;i*i<n;i++){
    if(prime[i]){
      for(int j=i*2;j<n;j+=i){
        prime[j]=0;
      }
    }
  }
  for(int i=0;i<n;i++){
    if(prime[i])prime_list.push_back(i);
  }
}
```

4 文字列

```
string revStr(string s){
    return string(s.rbegin(),s.rend());
}
```

5 二次元幾何

```
typedef long double ld;
typedef long long ll;
typedef ld P_type;
typedef complex<P_type> P;

const ld INF = 1e39;
const ld EPS = 1e-8;
const ld  PI = acos(-1);
```

- operator

```
namespace std{
  bool operator<(P a, P b) {
    if(a.X!=b.X){
      return a.X < b.X;
    }else{
      return a.Y < b.Y;
    }
  }
}
```

- 外積

```
P_type out_pro(P a,P b) {
  return (a.X * b.Y - b.X * a.Y);
}
```

- 内積

```
P_type in_pro(P a,P b){
  return (a.X * b.X + a.Y *b.Y);
}
```

- 長さの二乗

```
P_type pow_len(P a) {
  return a.X * a.X + a.Y * a.Y;
}
```

- 長さ::小数点以下が必要なことがあるので ld にする

```
ld len(P a){
  return sqrt(pow_len(a));
}
```

- 凸法::依存::外積,operator-

```
vector<P> convex(vector<P> list) {
  int m=0;
  vector<P> res(list.size()*2);
  sort(list.begin(),list.end());
  for(int i=0; i<list.size(); res[m++]=list[i++]){
    for(;m>1&&out_pro(res[m-1]-res[m-2],list[i]-res[m-2])<=EPS;--m);
```

```
7     }
8     for(int i=list.size()-2,r=m; i>=0; res[m++]=list[i--]){
9       for(;m>r&&out_pro(res[m-1]-res[m-2],list[i]-res[m-2])<=EPS;--m);
10    }
11    res.resize(m-1);
12    return res;
13  }
```

6 UnionFind

```
class union_find{
private:
  vector<int> parents;
  vector<int> rank;
public:
  union_find(int n){
    parents.resize(n);
    rank.resize(n);
  }
  void init(){
    for(int i=0;i<parents.size();i++){
      parents[i]=i;
      rank[i]=0;
    }
  }
  int find(int x){
    if(parents[x]==x){
      return x;
    }else{
      return parents[x]=find(parents[x]);
    }
  }
  void unite(int x,int y){
    x=find(x);
    y=find(y);
    if(x==y)return;
    if(rank[x]<rank[y]){
      parents[x]=y;
    }else{
      parents[y]=x;
      if(rank[x]==rank[y])rank[x]++;
    }
  }
  bool same(int x,int y){
    return (find(x)==find(y));
  }
};
```

7　チェックリスト

1. 問題読む時のチェックリスト
    - 問題のジャンルは？
    - 自分はそのジャンル得意？他の人が得意じゃない？
    - 設定ちゃんと理解してる？誰かと確認とった？
        - 各変数の範囲は？
        - 何を求めるの？
            - 求めるべきものの独自設定が問題にある？文字列とか？
            - 何かの値？個数？距離？
            - 最大？最小？
            - 確率？期待値？確率ならちゃんと $0 \leq out \leq 1$ が出力される？
            - 場合分け？
        - 思い込んでない？（それは問題に書いてある設定？）
        - その解釈以外に解釈できない？できるなら審判に質問した？

2. 解法考える時のチェックリスト
   - 自分はそのジャンル得意？他の人が得意じゃない？
   - アルゴリズムの前提条件と問題の条件はあってる？
     - 負の閉路とかない？
     - 絶対に「ほげほげ」ができるの？（ex.. 全域木、経路、）できないときどうするの？
     - 「ほげほげができない」、ケースはそれだけ？途中で変わることはない？
     - 一般性があることを確認した？分割されるのが二個までとか思い込んでない？
   - アルゴリズムの正当性は？常に正しい答えを出せる？
     - 設定から読み取れるパターン全部に対応できてる？
     - 例外処理は存在する？対応できる？
     - サンプル紙の上で通した？コーナーケースサンプルにあるかもよ？
     - 全探索？ほんとに全探索になってる？（無意識に削ってない？）
     - 探索範囲削ってる？削る必要ある？削った範囲に答えは絶対にない？
       どうせ定数倍ぐらい（要確認）だから削らなくていいんじゃない？
     - 必要な機能は列挙した？ STL 使える？使えないなら実装できる？
   - オーダーは？計算量 $O(10^9)$ は？配列サイズは $O(10^6)$？
     - 指数とか階乗とかでてない？出てるとしたら係数大丈夫？
       - 組み合わせ全部とか危なそうだよ？
     - 大体 $O(10^8)$ 以下っぽい？
       - 典型アルゴリズムはあり本とかで調べた？勘違いしてない？
     - 逆に余裕あったりする？
       - 簡単に実装できる方法ない？
       - 難しいなら考えたほうがいいよ！
   - 実装大丈夫そう？
     - バグりそうなところは関数に分けれる？分ける関数の引数と、期待する返り値は紙に書いた？
     - バグってたら結果はどうずれる？
       - サンプルでバグチェックできる？
       - サンプルにないなら、自作してチェックできるようにできない？
     - バグりそうなところは設定？
     - アルゴリズムが複雑？ライブラリ利用できない？使ったことある人はチームにいない？
       やったことがある人がいるならその人にその部分の実装を頼めない？
     - 別の実装じゃダメ？
       - 計算量増えても、同等の効果があるならそっちのほうがいいんじゃない？
       - 他の部分の計算量削ってここに回せない？
       - 最悪 $O(10^9)$ とかでも大丈夫だよ？

3. 実装時チェックリスト
    - 初期化してる？ INF?0?-1?
    - operator は正しい？ priority queue は注意！
    - 入力の終わりは？
    - 出力に余計な空白とかない？
    - ライブラリは正しいと信じよう（タイポはチェックしても良い) コンテスト中に訂正はしない！ ライブラリがバグってるっぽい時にはそれしかすることがないときしか修正しないこと
    - バグりそうなところは大丈夫？
        - 設定の小分けのチェックは誰かに手伝ってもらった？
        - サンプルでチェックできる？
        - チェックできないパターンのサンプル作って通した？
        - その関数だけ分けてチェックできるならした？
        - 小さい機能を他の関数に分けた？
        - 下位関数は思ってる機能を持ってる？

8 ヒント集

- ジャンル分け
  他のジャンルと複合してる時もある
  - 全探索
    - DP できるかも？
      - hogehoge までなにかを詰め込む（ナップサック）
      - 共通資源をとりあう (スケジューリング)
      - 再帰的ならメモ化できる
      - その他、間に合わないならがんばれ
    - もしかして：：：：：：：：貪欲（できれば避けたい）
  - グラフ
    - 最短経路
      - 全点（ワーシャルフロイド）
      - 始点終点（ダイクストラ、ベルマンフォード (負の閉路)）
    - 全域木
      - クラスカル、プリム
    - フロー :::::出たら死にますごめんなさい
  - 構文解析
    - 数式
    - それ以外
  - データ構造系
  - 計算
    - 場合の数
      - 数え上げ
      - 確率
      - 期待値
    - 素数とか、定義の実装がある
      - エラトステネスの篩
        こいつ（亜種含む）で先にテーブル作っちゃえたり
  - ゲーム
    - 分割統治とか
    - シミュレーション？サイコロ、文中で設定されてる
    - Nim に帰着？まだよくわかってない
  - 幾何
    多分他のジャンルと複合してる グラフに落としたりするかも
    - 二次元
    - 三次元
    - 線

* 円
* 図の方程式.. 計算カテゴリに放り込めるかも
* 交差
* 凸包（すべてを包むほげほげ）