Algorithms (2020 Summer)
#8：「難しい問題」とは，
整数関連

矢谷 浩司

授業アンケートのご協力お願いします.

slackにも流してありますので，どうぞお願いいたします.
https://forms.gle/vepzjJTgtQZy9P388

今日のお題は2つ.

コンピュータにとって「難しい問題」とは？

そういう問題にぶち当たったときは？

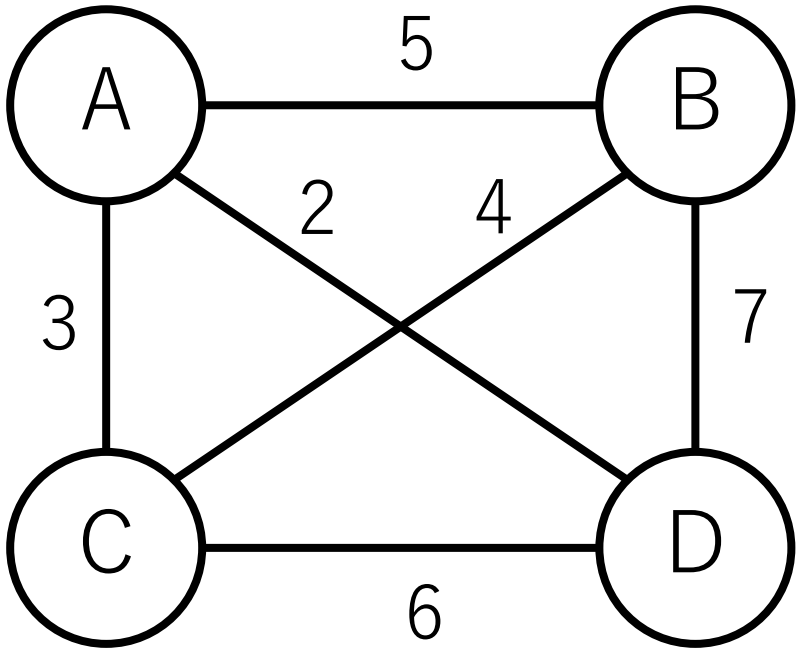
本当は最後の回で紹介する予定でしたが，時間の関係で今日軽くお話としてご紹介.

整数関連のアルゴリズム

整数ならではの性質に関係するお話.

巡回セールスパーソン問題

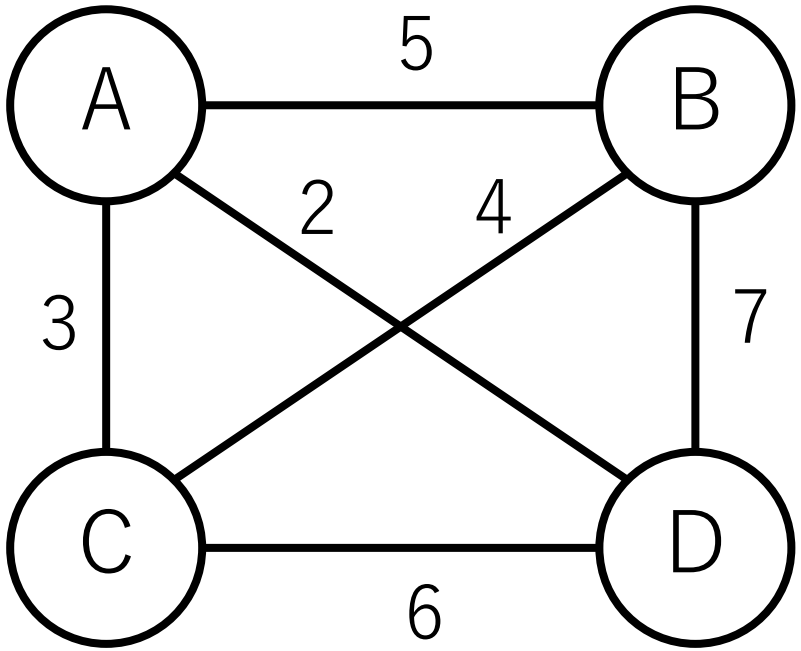
「複数の都市を全て必ず1回だけ通るように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」



巡回セールスパーソン問題

「複数の都市を全て必ず1回だけ通るように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

A->C->B->D->A（逆でも可）と回って，16.



巡回セールスパーソン問題

この問題が解けると嬉しいことがいっぱいありそう！

指定した観光地を最も安い金額で周るツアーの設計
配達物を届けるための経路選択

などなど．．．

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

単なる最短経路（全て必ず通る必要はない）であれば，
なんとかなるんだが．．．（10回目で紹介）

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

全探索するなら都市の順列を考えて，$O(n!)$.

巡回セールスパーソン問題

「複数の都市を**全て必ず1回だけ通る**ように巡回し，
開始都市に戻ってくる時，最短の経路はなにか？」

全探索するなら都市の順列を考えて，$O(n!)$.

コンピュータにとっての「難しい問題」

コーディングが複雑（コードに落とすのが難しい），
という意味ではない.

現時点において「効率的に」解くアルゴリズムが
知られていない問題.

「難しい問題」とはどんなものか？

以下のスライドでは，直感的な理解を重視して，ざっくりした説明をしています．（チューリングマシンとかの説明を完全に無視しております．．．）

厳密な解説は他の講義や資料におまかせしたいと思いますので，興味ある人はぜひご自身で探してみてください．

決定問題（判定問題）

ある入力に対してyesかnoの答えを求める問題.

答えの出力にかかる時間が入力に依存しない.

「与えられた整数が素数であるかどうか．」
「与えられた文字列が回文かどうか．」

P問題とNP問題

P = Polynomial（多項式）Time Solvable

多項式時間で解くアルゴリズムが存在する判定問題.

多項式時間：$O(n), O(n^k), O(\log n)$などなど.
指数関数よりは時間のかからないものを指す.

P問題とNP問題

P = Polynomial（多項式）Time Solvable

アルゴリズムの世界でいう「易しい問題」.
（現実的にはめちゃ遅いものも含まれるが. . .）

「効率的に」解くことのできる方法が知られている
問題とされる.

P問題とNP問題

NP = Non-deterministic Polynomial（非決定的多項式）
Time Solvable

Non-polynomialではないことに注意！

出力がyesとなる証拠があった時，その証拠を確認する
多項式時間のアルゴリズムが存在する判定問題.

P問題とNP問題

NP = Non-deterministic Polynomial（非決定的多項式）

「多項式時間で問題を解くアルゴリズムが存在しない」ではない！

無限に並列計算できるなら，動かしている中のある組み合わせが，yesの出力になることはあり得るので，多項式時間で問題を絶対に解けないというわけではない.

P問題とNP問題

P問題は必ずNP問題でもある．ただし，その逆は必ずしも成立しない．

P問題：多項式時間で答えを出す方法がある

NP問題：多項式時間で答えを確認する方法がある

NPだけどPではない問題は，「難しい問題」として扱われる．

多項式時間帰着

ある問題から，別の問題へ変換することが多項式時間で可能であること．

問題Aを多項式で解く方法がわからないが，問題Bは多項式で解く方法が知られているとする．

この時，問題Aを問題Bに多項式時間で帰着できれば，
問題Aは「帰着＋問題Bを解く」ことで解くことができる．
すなわち，全体として多項式時間で処理できる．

NP完全（NP-complete）

全てのNP問題を多項式時間帰着できるNP問題.

NP問題の中でも最も難しい問題とされる.

このようなNP完全問題が存在することは知られている.

NP困難（NP-hard）

多項式時間帰着でNP問題の1つに変換できる問題.

決定問題でなくても良い.

問題自身はNPに属していなくても良い.

NP困難であり，かつNPに属する問題はNP完全となる.

NP困難（NP-hard）

NP困難は，NP完全と同等かそれ以上に難しいとされる.

多項式時間帰着すればNP完全になるので，元の問題はそれと同等かそれ以上に難しい.

NP（完全）「よりも困難」と覚えると良い.

NP完全，NP困難かがわかる意義

効率的に解けないことがわかる悪い知らせ？

NP完全，NP困難かがわかる意義

効率的に解けないことがわかる悪い知らせ？

世の中の他の人も効率的に解けない.

-> 落ち込む必要なし！

気持ちを切り替えて，「少しでもマシ」な方に向かうことができないか，を考えれば良い.

部分点を積極的に取りに行くイメージ.

NP完全，NP困難の例をみてみよう．

どんな問題がNP完全，NP困難であるかを知っていれば，「この問題はやばそうだぞ」というセンスを養うことができる．

NP完全：充足可能性問題（SAT）

入力：n個のBoolean変数に対する論理式
出力：論理式全体を真にする変数の組み合わせは存在するか？

例：$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3)$という論理式において
真になる変数の組み合わせはあるか？

答え：yes．例えば，$x_1 = T, x_2 = F, x_3 = F.$

NP完全：充足可能性問題（SAT）

もし変数の組み合わせがわかれば，それを実際に代入して真になるかどうかを確認するのは，多項式時間でできる．

それを見つけようとすると単純には$O(2^n)$となる．．．

Steve Cook先生によりSATがNP完全であることが証明されている（世界で一番最初にNP完全であることが示された）．

NP完全：部分和問題

「n個の整数a[0]，a[1]，…，a[n-1]が与えられたとき，そのいくつかを組み合わせて総和がMにできるかどうかを判定せよ．」

ん？どこかでみたような．．．？

矢谷式DPの考え方：部分和問題の場合（再掲）

#4 コード化

2通りのパターンがある.

a[i]を入れる：dp[i][j] = dp[i-1][j-a[i]]

a[i]を入れない：dp[i][j] = dp[i-1][j]

この内，どちらか一方でもTrueならdp[i][j]もTrue.
そうでなければ，False.

（ただし，j-a[i]は0以上．そうでない場合，自動的に dp[i][j] = dp[i-1][j] ）

矢谷式DPの考え方：部分和問題の場合（再掲）

#4 コード化

2通りのパターンがある．

a[i]を入れる：dp[i][j] = dp[i-1][j-a[i]]

a[i]を入れない：dp[i][j] = dp[i-1][j]

この内，どちらか一方でもTrueならdp[i][j]もTrue.
そうでなければ，False.

（ただし，j-a[i]は0以上．そうでない場合，自動的に dp[i][j] = dp[i-1][j] ）

解けてるやん！

部分和問題ってDPで解けるのでは？

DPを使えば$O(nM)$なので，多項式なのでは？

実際それなりの時間で解ける？

部分和問題ってDPで解けるのでは？

DPを使えば$O(nM)$なので，多項式なのでは？
　　実際それなりの時間で解ける？

$M = 2^n$のような場合には，結局$O(n2^n)$となり，そのままでは多項式時間で解くことにならない.

このようなものは擬多項式時間（Pseudo-polynomial time）アルゴリズムと呼ばれている.

部分和問題ってDPで解けるのでは？

第1回目の講義でも説明したが，計算量の議論は入力が$O(n)$規模であることを前提としている．

指数的に変化するような場合などは考えない．

一方Mは入力の個数ではなく，こちらが勝手に決めることのできる数字（今回の例では部分和）．よって，これを2^nにしてしまうと，入力のサイズに応じて計算量が指数的に増加してしまうことになる．

NP完全：ハミルトン閉路問題

「与えられたグラフにおいて，全ての頂点を一度だけ通る閉路が存在するか」

グラフに向きがあるときは有向ハミルトン閉路問題，向きがないときは無向ハミルトン閉路問題と呼ばれる．

NP困難：巡回セールスパーソン問題

「与えられたグラフにおいて，全ての頂点を一度だけ通る最短距離の閉路は何か」

もし判定問題（距離がLより短い閉路があるか）であれば，NP完全として扱える．

巡回セールスパーソン問題は多項式時間でハミルトン閉路問題に帰着できる．

NP困難：ナップサック問題

DPの回で扱いましたが，こちらもNP困難.

部分和問題と同じく，擬多項式時間.

NP困難：スーパーマリオ

「ステージの最初からゴールまで辿り着けるかどうか」

ファミコンのゲームのいくつかは同様にNP困難であることが証明されている.

Classic Nintendo Games are (Computationally) Hard

Greg Aloupis, Erik D. Demaine, Alan Guo, Giovanni Viglietta

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokemon. Our results apply to generalized versions of Super Mario Bros. 1-3, The Lost Levels, and Super Mario World; Donkey Kong Country 1-3; all Legend of Zelda games; all Metroid games; and all Pokemon role-playing games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

https://arxiv.org/abs/1203.1895
https://www.technologyreview.com/s/427197/super-mario-bros-proved-np-hard/

NP完全，NP困難

以上の問題の多くは「組み合わせ最適化問題」と呼ばれるカテゴリに入る．

「XXXをうまく選んで，YYYになるか」などのような問題は雲行きが怪しい可能性あり．

ただ，世の中の現実問題はこういうものが結構多い．

NP完全，NP困難にぶち当たったとき

DPで解ける or 少し楽になる場合もある.

巡回セールスマン問題もDPを使うと，$O(2^n n^2)$に計算量を落とせる.

近似アルゴリズムで行く方針に切り替え.

様々な方法が知られており，また問題ごとに特化したアルゴリズムも存在する.

NP完全，NP困難かがわかる意義（再掲）

効率的に解けないことがわかる悪い知らせ？

世の中の他の人も効率的に解けない.

-> 落ち込む必要なし！

気持ちを切り替えて，「少しでもマシ」な方に向かうことができないか，を考えれば良い.

部分点を積極的に取りに行くイメージ.

お次のお題は

整数関連の処理についてのお話.

今回は今までの話とほぼ完全に切り離しができるトピック.

DPで心が折れた人も是非ここで一度気持ちを取り返してください. 😊

最大公約数

「2つの整数a，bの最大公約数を求めよ．」

例)
14と30 -> 2
786,240と76,608 -> 4032

単純なやり方

2から順に調べる？

最悪のケースでは$O(\min(a,b))$かかる.

ユークリッドの互除法

「$a, b\ (a \geq b)$の最大公約数は$a\%b$（剰余）とbの最大公約数に等しい．」

明記されている最古のアルゴリズムだそうです．
（紀元前300年くらい）

a，bの剰余を交互に繰り返して行き，どちらかが0になった時点で終了．

ユークリッドの互除法の実装例

```
def gcd(a, b):
    if b==0:
        return a
    else:
        # 値の小さい方が常に2番目に来るようにする
        return gcd(b, a%b)
```

ユークリッドの互除法の計算量

1回のgcdでa, bのどちらかは半分以下になる.

よって大まかには$O(\log(\max(a, b)))$かかる.

厳密にはラメの定理により，小さい方の整数の桁数の5倍が上限であることが知られている.

ちなみに最悪のケースはどんなもの？その時の計算量は？

最小公倍数

最大公約数がわかれば，$a * b/gcd(a, b)$で求める．

（言語によっては，数字が大きい場合は計算順序に注意する必要あり．）

例）

14と30 -> 210

786,240と76,608 -> 14,938,560

拡張ユークリッドの互除法

一次不定方程式の整数解の1つを求める.

例）$14x + 6y = 4$ -> $x = 2, y = -4$

拡張ユークリッドの互除法

一次不定方程式$\mathrm{a}x + by = c$が整数解を持つ必要十分条件はcが$\gcd(\mathrm{a}, \mathrm{b})$で割り切れることである.

（証明はここでは省略．．．）

つまり，　$c = d * \gcd(\mathrm{a}, \mathrm{b})$となるので，　$\mathrm{a}x + by = \gcd(\mathrm{a}, \mathrm{b})$を計算できれば，元の式に対する答えもわかる.

拡張ユークリッドの互除法

$14x + 6y = 4$の例で考えてみると，$14x + 6y = 2$を解けば良い.

つぎに14と6の最大公約数をユークリッドの互除法を使って求めると,

14, 6 -> 2, 6 -> 2, 0

になる.

拡張ユークリッドの互除法

$14x + 6y = 2$を分解すると，$(12 + 2)x + 6y = 2$であるから，$2x + 6(y + 2x) = 2$となる．

つまり，$2x + 6y' = 2$を解けば，$y = y' - 2x$から元の解が求まる．

さらに$2(x + 3y') + 0y' = 2 \rightarrow 2x' = 2$を解けば，元の解が求まる．

この場合，$x' = 1, y' = 0, x = 1, y = -2$と順々に求まる．

拡張ユークリッドの互除法

ユークリッドの互除法を利用することで，x, yの係数をどんどん小さくすることが出来，最終的には明示的に求まる形になる．

再帰を使って実装すれば良い．

拡張ユークリッドの互除法の実装例

```
# gcd(a, b), x, yが返る.
def ext_gcd(a, b):
    if b == 0:
        return a, 1, 0
    else:
        d, x, y = ext_gcd(b, a%b)
        return d, y, x - (a//b)*y
```

拡張ユークリッドの互除法の計算量

こちらも大まかには$O(\log(\max(a,b)))$.

素数判定

「与えられた整数nが素数であるかを判定せよ．」

例)
13 -> Yes
25 -> No
1,000,000,007（10^9 + 7）-> Yes

素数判定

ナイーブな方法
2からn/2まで順番に割っていき，割り切ることができればNo．そうでなければYes.

この場合，計算量は$O(n)$.

素数判定

もし，dがnの約数だとすると，n/dもnの約数.
（例：n=30, d=3なら，n/d=10もnの約数）

つまり，$(d, n/d)$が必ずペアになっている.

$\min(d, n/d)$の最大値は$\sqrt{n}$となるので，2から$\sqrt{n}$まで調べれば良いことになる.

こうすると，計算量が$O(\sqrt{n})$まで削減できる.

素数判定アルゴリズムの実装例

```
def prime(n):
    if n <= 1: False
    i = 2
    while i*i <= n:
        if n%i == 0: return False
        i += 1
    return True
```

素数数え上げ

「1以上n以下の素数の数を求めよ．」
「1以上n以下の素数を全て求めよ．」

例)
13 -> 6 (2, 3, 5, 7, 11, 13)
25 -> 9 (2, 3, 5, 7, 11, 13, 17, 19, 23)
1,000,000 -> 78498

素数判定アルゴリズムを単純に使うと

1からnまで素数判定を繰り返す．よって，計算量は$\sum_{i=1}^{n}\sqrt{i}$.

$$\int_1^n \sqrt{x}\,dx = \frac{2}{3}\left(n^{\frac{3}{2}} - 1\right)$$

であることを考えれば，全体の計算量はおおよそ$O(n\sqrt{n})$.
（単純に，$O(\sqrt{n})$がn回あると考えても良い．）

nが大きいと，ちょっと遅い．．．

改良案

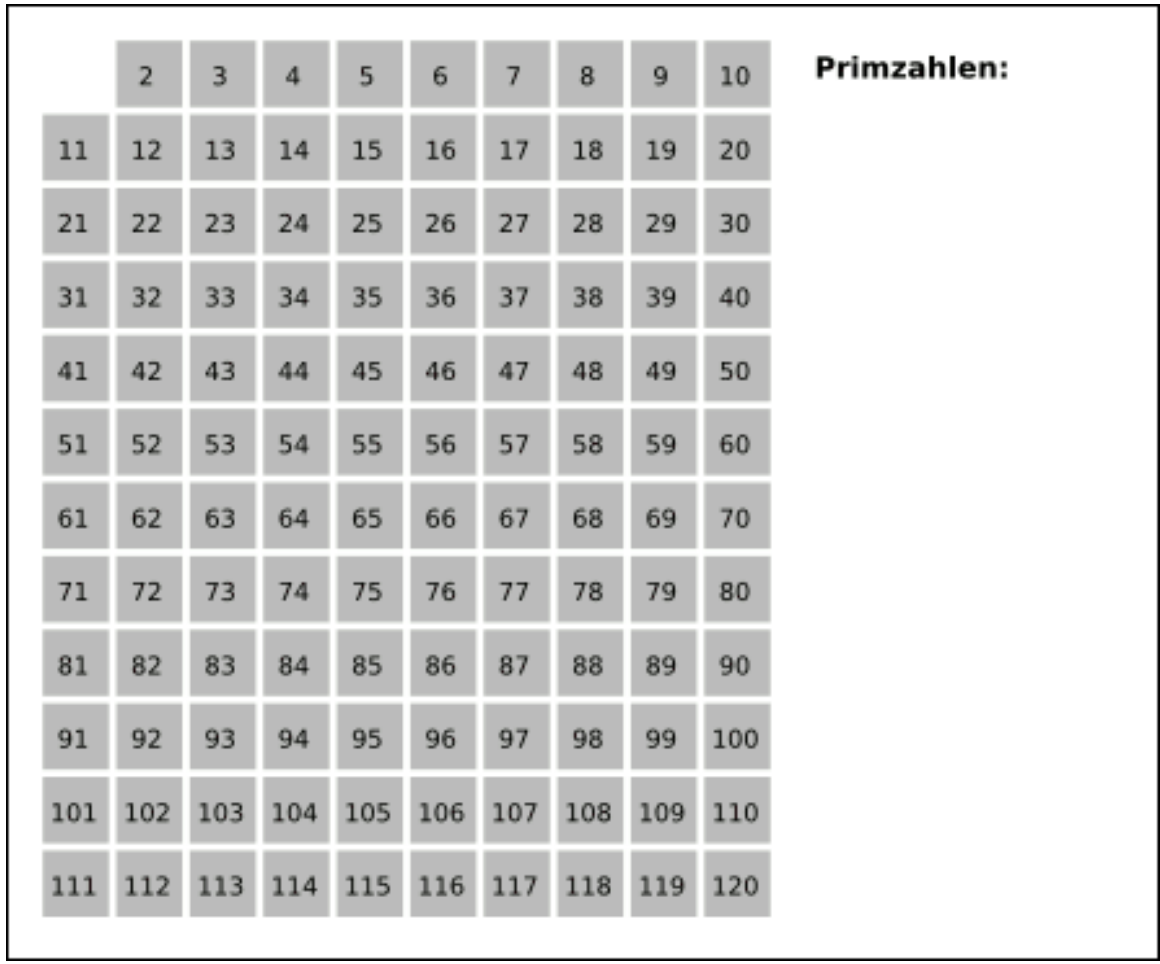
エラトステネスの篩というアルゴリズムを使い，
素数のリストを作ってから，数を数える．

エラトステネスの篩もユークリッドの互除法くらい
古いアルゴリズムとされている．

エラトステネスの篩

#1 2からスタート.
#2 2の倍数を全部削除.
#3 次の数字（3）に移る.
#4 3の倍数を全部削除.
#5 次の数字（5）に移る.
#6 5の倍数を全部削除.

以降，$\sqrt{n}$まで繰り返す.



https://ja.wikipedia.org/wiki/%E3%82%A8%E3%83%A9%E3%83%88%E3%82%B9%E3%83%86%E3%83%8D%E3%82%B9%E3%81%AE%E7%AF%A9

エラトステネスの篩の実装例

```
def prime_all(n):
    # is_prime[i]がTrue -> iが素数.
    is_prime = [True]*(n+1)

    is_prime[0] = is_prime[1] = False
    i = 2
```

エラトステネスの篩の実装例

```
def prime_all(n):
    ...
    while i*i <= n:
        [iが素数ならば]:
            [iの倍数でn以下の値は全て素数でないと記録]
        i += 1
```

エラトステネスの篩の実装例

```
def prime_all(n):
    ...
    while i*i <= n:
        ...


    # 0と1は取り除いて総数を返す.
    return len([i for i in range(2, n+1) if is_prime[i]])
```

エラトステネスの篩の計算量

2で篩に落とされるのはn/2個.
3で篩に落とされるのはn/3個.
5で篩に落とされるのはn/5個.
…

よって,

$$n\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\cdots+\frac{1}{\sqrt{n}}\right)$$

個振り落とす（Falseにする）ことになる.

エラトステネスの篩の計算量

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\cdots+\frac{1}{\sqrt{n}}\approx \log\log\sqrt{n}$$

ということが知られている（nが十分に大きければ）.
よって，振り落とす数の総和は$n\log\log\sqrt{n}$くらいになるので，計算量は$O(n\log\log n)$.

これは，実質$O(n)$となる.

素因数分解

エラトステネスの篩で予め$\sqrt{n}$までの素数を求めておき，
それらに対して割り算を行って求める．

これは，　$O(\sqrt{n}\log\log n)$.

これらの素数で割り切れたら残す．そうでなければ捨てる．

素因数分解

nまでの素数の数はnが大きければ$\frac{n}{\log n}$で近似できることが素数定理により知られている.

$\rightarrow\sqrt{n}$までの素数は$O\left(\frac{\sqrt{n}}{\log n}\right)$個ある.

よって全体の計算量は, $O(\sqrt{n}\log\log n+\frac{\sqrt{n}}{\log n})$で,
結局$O(\sqrt{n}\log\log n)$.

冪乗

「x^nを求めよ．」

まともに計算すると$O(n)$.
（オーバーフローが起きないと仮定）

繰り返し自乗法

$$x^n = (x^2)^{\frac{n}{2}}$$

と変形すれば，計算回数を半分にできる！

これを再帰で繰り返せば，$O(\log n)$で計算可能.

繰り返し自乗法

x**nでよくね？😅

繰り返し自乗法

x**nでよくね？😅

pythonだとそれでもたまたま扱えるだけで，言語によってはオーバーフローや型の変換等を考える必要が出てくる．

pythonでも値が大きくなりすぎると計算が大変になる．
（4回目の課題で体験した人もいらっしゃるはず）

剰余を求める

よって，32bit（10進数で10桁）で収まるようにするために，大きな素数の剰余を使い，それを計算に使う．

競技プログラミングなどでは「10^9+7で割った余りを求めよ．」という指示があることが多い．（我々の課題でもありましたね．）

この手の問題では計算過程中でも値が大きくなることがあるので，計算過程中でもmod 10^9+7を計算する必要が出てくる．

剰余で答えを出す場合

加算：

　　加算したあとでmod hを計算.

減算：

　　減算したあとでmod hを計算.　（ただし，言語によっては計算結果が負のときにはhを足す必要あり．）

乗算：

　　乗算したあとでmod hを計算.

剰余で答えを出す場合

(-20) % 7を実行すると，

Python：1

C++：-6

C++の場合，さらに7足すと，1となり一致する．

繰り返し自乗法（剰余で答えを出す場合）

```
def power(x, n):
    M = 10**9 + 7
    if n==0: return 1       # 0乗は1.

    tmp = power(x*x % M, n//2)     # 再帰で計算
    if n%2: tmp = tmp * x % M      # nが奇数の場合の処理

    return tmp
```

繰り返し自乗法の実行結果例

power(3, 3) -> 27

power(7, 30) -> 946,501,044

もし，剰余を取らないと，
22,539,340,290,692,258,087,863,249
とかいうものすごい値に．．．

剰余で答えを出す場合（再掲）

加算：

　　加算したあとでmod hを計算.

減算：

　　減算したあとでmod hを計算.　（ただし，言語によっては計算結果が負のときにはhを足す必要あり．）

乗算：

　　乗算したあとでmod hを計算.

剰余で答えを出す場合（再掲）

加算：

　　加算したあとでmod hを計算.

減算：

　　減算したあとでmod hを計算.（ただし，言語によっては計算結果が負のときにはhを足す必要あり．）

乗算：

　　乗算したあとでmod hを計算.

除算：

　　？

除算はちょっと厄介．．．

乗算の場合は，剰余を余計にとっても問題ない．

例）8*2 = 16 でmod 6をとる．

8*2 = 16　->　16 mod 6 = 4

8 mod 6 = 2　->　2*2 = 4　->　4 mod 6 = 4

除算はちょっと厄介．．．

除算はそうはいかない．．．

例）8/2 = 4 でmod 6をとる．

8/2 = 4　->　4 mod 6 = 4

8 mod 6 = 2　->　2/2 = 1　->　1 mod 6 = 1 ??

除算はちょっと厄介．．．

計算の最後だけで剰余を取るのであれば大丈夫.

だけど，そこに至るまでにすでに剰余を取ってしまっている場合には計算が狂ってしまう．．．

フェルマーの小定理

aが任意の自然数，mが素数でa, mが互いに素である時，
以下のことが成立する．

$$a^{m-1} \equiv 1 \bmod m$$

逆元

mが素数で，aがmでは割り切れない整数であるとき，
以下の式を満たすxがmに応じて一意に存在する．
このようなxを「$\mathrm{mod}\, m$におけるaの逆元（逆数のより一般的なもの）」と呼ぶ．

$$ax \equiv 1 \bmod m$$

つまり，

通常の世界：aに掛けると1になる数 -> $1/a$
$\mathrm{mod}\, m$ の世界：aに掛けると1になる数 -> x

この2つを使うと，

$a^{m-1} \equiv 1 \bmod m$は，　$a \times a^{m-2} \equiv 1 \bmod m$と見ることが出来る．つまり，　aの逆元は$\bmod m$の世界ではa^{m-2}になる．

$b \div a$は$b \times (1 \div a)$と変形できることから，$\bmod m$の世界では，aの逆元がわかれば割り算を計算できる！

つまり，　aで割ることはa^{m-2}をかけることに$\bmod m$の世界では等しい，ということになる．

順列の計算

「$_nP_k$の10^9+7で割った余りをもとめよ．」

$$_nP_k = \frac{n!}{(n-k)!}$$

だが，$n!$をまともに計算してしまうと大変な数字になるので，剰余で計算していく必要がある．では分母をどう処理するか？

順列の計算

$$\frac{n!}{(n-k)!} \bmod M = n!\,((n-k)!)^{M-2} \bmod M$$

と変形すれば計算できる.

よって，順列の値を剰余を取りながら計算し，上の計算を行えば良い.

順列の計算結果例（10**9 + 7の剰余）

permutation(4, 3) -> 24

permutation(10, 5) -> 30,240

permutation(20, 12) -> 831,129,627
（剰余を取らないと，　60,339,831,552,000）

まとめ

「難しい問題」

NP問題，NP完全，NP困難

整数関連

ユークリッドの互除法，素数判定，エラトステネスの篩，繰り返し自乗法，剰余の世界での四則演算

授業アンケートのご協力お願いします.

slackにも流してありますので，どうぞお願いいたします.
https://forms.gle/vepzjJTgtQZy9P388

コードチャレンジ：基本課題#8-a [1点]

$_nP_k$の10^9+7で割った余りを出力するコードを書いてください.

コードチャレンジ：基本課題#8-b [2点]

与えられた範囲（L，R）において，Nも(N+1)/2も素数となるような奇数の総数を求めてください.

考え方

素数の数え上げをしたい．→何を使う？

ただし，クエリが何個も飛んでくるので，毎回毎回数え上げをすると遅い.

→1からiまでで該当する奇数の総数を予め計算しておき，それを利用することを考える.

コードチャレンジ：Extra課題#8 [3点]

整数関連の話を踏まえた問題.