蟻本

(2019/07/08(月)) $\boxed{\text{DFS}}$

[深さ優先探索による部分和問題]

[a_1, a_2, ⋯, a_n] の部分和で k にできるか? の判定

func dfs (i, sum int) bool というシグニチャの
再帰関数で 2^n の探索が可能

※勿論 bit 全探索でもいいが、忘れているので。

[Lake Counting]

グリッド上で、8近傍でつながれた水溜まりの数を
数える問題。

ポイントは、

1. 'W' を踏んだ時にカウントする
2. 一度数えた 'W' はダブルカウントしないように '.' で埋める
3. 不要な探索はしない

といったところ。

BFS

- 初期状態に近い順に探索をするので、簡単に最短経路や最短手数などを求めることが出来る。
  - 途中で処理を打ち切らず、キューが空になるまで続けることで、全ての点まで最短距離を計算することが出来る。

貪欲法 ：「その場での最善」を選択し続ける

[硬貨の問題]

1円玉，5円玉，10円玉，50円玉，100円玉，500円玉

を使ってできるだけ少ない枚数で精算する方法は？

→ 代表的な貪欲法。

(ナップザックDPでも解ける。)

[区間スケジューリング問題]

$[s_i, t_i]$ な N 個の仕事。できるだけ多くの仕事をこなすには？

＠選べる仕事の中で，「終了時間が最も早いもの」を選び続ける。

(直感的に，終了時刻が早いと，その後の選べる仕事が多いから，

という方針がよい。これにより，他の選び方よりも有利に進められる)

[辞書順最小の問題]：貪欲法がしばしば有効！

空の文字列Tに対して，Sの先頭か末尾を繰り返し連結する。

辞書順最小のTはどうやって作れるか？

Sの先頭と末尾の文字がequalの時が難しい。その先の文字までケアする必要があるので，残ったSとその反転S'を比較し，

先頭・末尾のいずれを選ぶかを決める！

※考え方は勿論，プログラムもスニペットで，一度書いてみること!!

[Saruman's Army]

N個の数直線上の点列について、各点に印をつけるとき、半径R以内に印のついた点が必要。最小で何個の点に印が必要か?

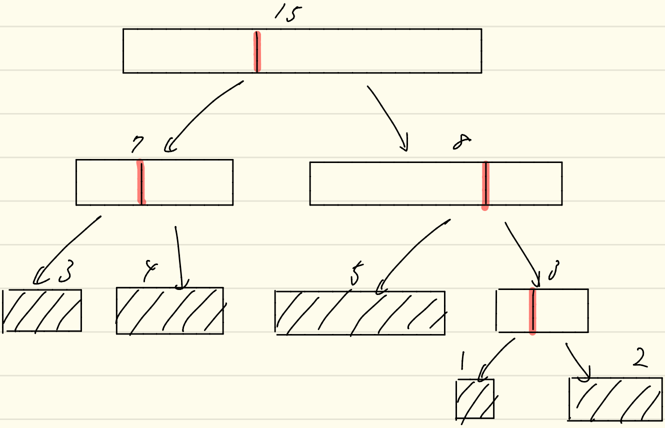
左から貪欲に考える。一番左の点から、距離R以内で最も遠い点を選ぶのが賢い。
(限界ギリギリをせめるイメージ)
すでにカバーされた点は忘れて、左に残った最も左の点について、更に再帰的に考える。

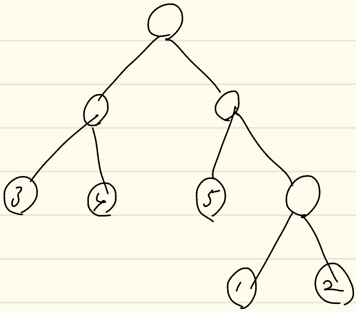
[Fence Repair]

板を切るコストを最小化するには?

(「ハフマン符号」や以前解いたEPPCの問題との関係を考えたい。)

[Fence Repair]（続き：図と考え方が重要!）





ハフマン符号：文字列をはじめとする、データの可逆圧縮などに使用される。
方針としては、「よく出現する文字には短いビット列を、
あまり出現しない文字には長いビット列を与える」というもの。
これにより、データ量が全体で削減される。

☆この問題における、各板→文字、各板の長さ→頻度
として二分木を作る。左に進むと0、右に進むと1を
末尾に追加することで、各文字(板)に対する符号が得られる。

(2019/07/14(日)) 易しめの DPラッシュ

[個数制限なしナップサック問題]

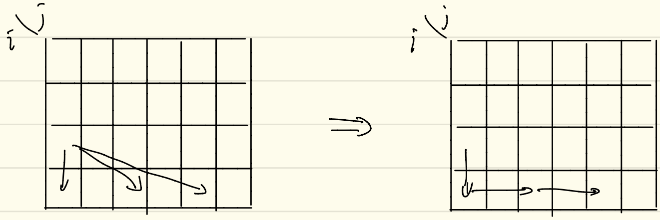
ナップサック問題の、ある品物について何個でも選べるバージョン。

$dp[i+1][j]$:= i番目までの品物から、重さの総和がj以下となるように選んだときの、価値の総和の最大値。

普通に考えると、

$$dp[0][j] = 0$$
$$dp[i+1][j] = \max\{dp[i][j-k\cdot w[i]] + k\cdot v[i] \mid k \geq 0\}$$

となり、DPテーブルの更新で、kに関するループも増えてしまい、$O(nW^2)$となってしまう。



解決策は、DPテーブルの遷移のさせ方の変化で上のようなイメージ。
左上から右へ、とテーブルを埋めていくと、すでに必要なパーツは求まっていることが分かる。(必要な計算が右へ行くほど同じ行で累積されるイメージ)

※ 01ナップサックや、今回の個数制限なしナップサックは、1次元の配列のみでも解くことが出来る。
(難問では必須となるケースもあるらしい…)

※ iと$i+1$の間でしか遷移のないDPは、2つの一次元配列だけで済ませられる(?)

(2019/07/15(月))

(個数制限なしナップサック問題の式変形部分)

コードで書いてみたら、この部分も自分で出来るようにならないといけない、と感じたため。

定義: $dp[i+1][j] = \max\left(dp[i][j-k\cdot w[i]] + k\cdot v[i] \mid k \geq 0 \right) \quad \cdots \quad (*)$

↑
暗黙的に、$k\cdot w[i]$ が j を越えないが条件となるよう注意!
(上限が各遷移で自然に存在する!)

(変形部分)

$k=0$ のケースで抜き出したもの

$$(*) = \max\left\{ \underline{dp[i][j]},\ \max\left\{ dp[i][j-k\cdot w[i]] + k\cdot v[i] \mid \underline{k \geq 1} \right\} \right\}$$

$$= \max\left\{ dp[i][j],\ \underbrace{\max\left\{ dp[i][\underline{(j-w[i])} - k\cdot w[i]] + k\cdot v[i] \mid \underline{k \geq 0} \right\}}_{dp[i+1][j-w[i]]\ (!)} + v[i] \right\}$$

$$= \max\left\{ dp[i][j],\ dp[i+1][j-w[i]] + v[i] \right\}$$

[01ナップサック問題その2]

01ナップサックにおいて、以下の制約がある場合。

$1 \leqq n \leqq 100$

$1 \leqq w_i \leqq 10^7$

$1 \leqq v_i \leqq 100$

$1 \leqq W \leqq 10^9$ $(\underset{(w_i)}{10^7} \times \underset{(n)}{100})$

dpテーブルの定義を変える。

$dp[i+1][j]$:= i番目までの品物から価値の総和がjとなるように選んだときの、重さの総和の最小値。

(価値の総和!)

初期化で $dp[0][0] = 0$
$dp[0][j] = INF$
とする以外、遷移は素直に行えばよい。

最終的な答えは $dp[n][j] \leqq W$ を満たす最大の j、になる。

制約によってアルゴリズム(DPテーブルの定義)を変えるパターン。

☆[個数制限付き部分和問題]

n種類の数 a_i がそれぞれ m_i 個ずつあります。これらの中からいくつか選び、その総和でちょうど K とすることが出来るか判定しなさい。

$1 \leqq n \leqq 100$, $1 \leqq a_i, m_i \leqq 100000$, $1 \leqq K \leqq 100000$

※蟻本はじめて8.9ヶ月前頃に読んだ時は難しすぎてわからなかったのを覚えているが、今読んでもやはりかなり難しい。
典型パターンとして覚えておきたい。(類題はまだ遭遇したことがない。)

(素直なやり方)

$dp[i+1][j]$:= i番目までで j が作れるか? (bool値)

i番目までで j を作るためには、i−1 番目までで
$j, j-a_i, j-2 \cdot a_i, \ldots, j-m_i \cdot a_i$ のいずれか1つは
true である必要がある。

よって遷移は、

$dp[i+1][j]$ = ($0 \leqq k \leqq m_i$ かつ $j \geqq k \cdot a_i$ で $dp[i][j-k \cdot a_i]$ == true ならば true)

文章だとややこしいが、プログラムだと for ループで論理和をとる形になるので、案外書きやすい。

三重ループとなってしまい、TLEする…

（正解）

> 一般に，bool値を求めるDPをすることは無駄であることが多く，
> 同じ計算量でもっと多くのことを知ることが出来ます。
> 今の場合，作れるかどうかだけでなく，作れる場合には，
> 「残りどれだけ a_i が余っているか？」を持たせることによって，
> 計算量を落とすことが可能です。

$dp[i+1][j] :=$ i番目までで j を作る際に余る最大の i 番目の個数（作れない場合は -1）

$$dp[i+1][j] = \begin{cases} ① \quad m_i & (dp[i][j] \geqq 0) \\ ② \quad -1 & (a_i > j \text{ または } dp[i+1][j-a_i] \leqq 0) \\ ③ \quad dp[i+1][j-a_i] - 1 & (\text{それ以外}) \end{cases}$$

① $i-1$ 番目までで j が作れているのだから，当然，i番目の数は1つも使わずに，すでに j が出来ている。

② まず，i番目の数値 a_i が j を上回っていれば，どうあがいても j を作れるわけがない（①をパスした後のチェックなので，当然 $i-1$ 番目までで j は作れていない）。
また，$a_i \leqq j$ であっても $dp[i+1][j-a_i] \leqq 0$ である，すなわち，
i番目までで $j-a_i$ が作れない，もしくは，作れたとしても a_i がもう余っていない，
このような場合は j を作れないので $dp[i+1][j]$ は 0 となる。

③ $j-a_i$ が，a_i を1つ以上余らせて作ることが出来ているのであれば，
a_i を1つ消費することで，j を作ることが出来る。

※ 2次元配列の初期化だと，$dp[0][0]=0$，$dp[0][j]=-1$ とするはず。
※ なんだかんだ，遷移は個数制限ナップサックと似ている気がする。
ある種別に対して複数個選べるパターンには，左から真横（右）への
遷移をさせる，このようなパターンが応用できるのだろうか？

[最長増加部分列問題 (Longest Increasing Subsequence : LIS)]

長さ n の数列 $a_0, a_1, \ldots, a_{n-1}$ があります。この数列の増加部分列のうち、最長のものの長さを求めなさい。

ただし、増加部分列とは、すべての $i<j$ で $a_i < a_j$ を満たす部分列のことをいいます。

$$\begin{cases} 1 \leqq n \leqq 1000 \\ 0 \leqq a_i \leqq 10^6 \end{cases}$$

$O(n^2)$ 解法と $O(n \log n)$ 解法があり、前者はラクだが後者は難しい。

$dp[i]$:= 最後が a_i であるような最長の増加部分列の長さ

これは、

$$dp[i] = \max(1, \ dp[j]+1 \mid j<i \text{ かつ } a_j < a_i)$$

で求まる（割と自然な考え方でいける）。

※ DPテーブルは、確実に前の方から埋まっていく！

これは $O(n^2)$ となる $\left(\frac{1}{2}n(n+1) \text{ 回}\right)$

($O(n\log n)$ 解法)
DPテーブルの定義を次のように変える。

$dp[i]$:= 長さが $i+1$ であるような増加部分列における、
最終要素の最小値 (存在しない場合は INF)

※アイデアは、「同じ長さの増加部分列ならば、最終要素が小さいほうがその後有利だから」
というもの。

1. $dp[0]$ 〜 $dp[n-1]$ を全て INF で初期化
2. 与えられた数列を前の方から見ていく。
   各 a_j に対して、$i=0$ または $dp[i-1] < a_j$ ならば、
   $dp[i] = \min(dp[i], a_j)$ と更新される。
3. $dp[i] <$ INF となるような最大の i について、$i+1$ が答え。

これを普通にやると $O(n^2)$ だが、
① dp の配列は INF を除いて単調増加になっている
② 各 a_j に対して更新はたかだか1回しか起こらない
という観察から、各要素 a_j についての二分探索によって、
$O(n\log n)$ に出来てしまう。

※ダイレクトに dp 配列が、各要素のチェックによって前から埋まっていく形だが、
正当性の理解が難しい…。
※ $O(n^2)$ の遷移についても、例えば $n=5$ のとき

$$[\infty, \infty, \infty, \infty, \infty] \to [a_0, \infty, \infty, \infty, \infty] \to [a_0, a_1, \infty, \infty, \infty]$$

次の a_1 が埋まる候補
ただし、DP定義から埋まるとしてもどちらか片方のみ

次に a_2 が埋まる候補
ただし、埋まるとしてもどれか1つだけ

という風に図で考えると分かりやすいかもしれない。
(とにかく、与えられた数列を前から見ていく、という前提が重要!!)

[分割数] (蟻本のDPではじめての数え上げ!)

n個の「互いに区別できない品物」を、m個以下に分割する方法の総数を求め、Mで割った余りを答えなさい。

$1 \leq m \leq n \leq 1000$, $2 \leq M \leq 10^4$

※このような分割をnのm分割と言う。
※m=nの時、特に「nの分割数」と言う。

(Wikipediaで調べた)
「nを、その順番の違いを除いて、自然数の和として表す方法」の総数
という旨の記述があった。この方が個人的にはわかりやすい。

> このように、場合の数の問題では、いかに重複なく数え上げるかの工夫が
> 重要となります。

※誤解法も含めて全てが難しいので、丁寧に1つ1つを理解していく。

$dp[i][j] :=$ jのi分割の総数 (※i, jの表すものに注意!)

漸化式(*): $dp[i][j] = \sum_{k=0}^{j} dp[i-1][j-k]$

(*)は、① まずk個とり出す(隔離する)
② j-k個を(i-1)分割する

として、②のパターンに、①のk個をジョイントするようなイメージだが、
これは重複して数えてしまっている!
(例えば、n=10としたとき、k=2で(2, 8)というのが数えられるが、
k=8でもまた(8, 2)というものが別物としてカウントされてしまう。)

（正解の漸化式）

n の m 分割 a_i を考える（$\sum_{i=1}^{m} a_i = n$）

① すべての i で $a_i > 0$ ならば、$\{a_i - 1\}$ は $(n-m)$ の m 分割となる。
↑ 全要素 -1 すれば全体で m 減る

② 次、$a_i = 0$ となる i が存在したら、これは n の $(m-1)$ 分割となる。
↑
②の方は？？？という感じだが、例えば5を3分割するものの中で
(0, 0, 5) というものがあるとき、これ1つの0をとり除いて
(0, 5) という2((3−1))分割のパターンの1つと見ることが出来る。

①と②をまとめて遷移は、

$$dp[i][j] = \underbrace{dp[i][j-i]}_{①} + \underbrace{dp[i-1][j]}_{②}$$

となる。

※②は、「n の m 分割の総数は、n の $(m-1)$ 分割のパターンを全て含む」
というシンプルな捉え方もできそう。
それでその上で、①が排反なパターンとして純粋に加算される？

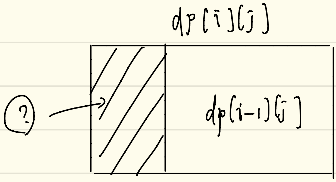
※これも個数制限なしナップサックや、個数制限付き部分和問題
のように、真横(右)への遷移を考える問題といえそう。

※初期化は $dp[0][0] = 1$ のみ、0の0分割は1だが、
$n \geq 1$ の n について0分割は0とする！
（DPの初期化の中でもかなり直感的でない気がするので、
知識として覚えておいてもいいかもしれない）

(2019/07/15(月)) [分割数DPについてもう少し]

書籍での説明はトップダウンというか、全て理解している人の解説、という感じで、もう少し自然な考え方もできるように思う。

dp[i][j]: j の i 分割は、j の (i−1) 分割を完全に含む
(dp[i-1][j])



図にすると上のような感じ。(i−1)個以下にjを分割するパターンはdp[i][j]に含まれるが、これはdp[i-1][j]に他ならないので、dp[i-1][j]を加算するだけで、もれなく・重複なく数え上げることが出来る。

斜線部分は何かというと「jをピッタリi分割する(0を含まない)パターン数」と言える。恐らくこれを賢く求める部分が、この問題で一番難しいところ。

これは解説通り、「j−iをi分割するパターン数」がすでに求まっていることに注意し、「このパターン全てについて、各要素(全体でi個となるように補間した0も含む)に対して+1することで、jのピッタリi分割が出来上がる」ということから、dp[i][j−i]を加算すればよい、ということになる。

※真横の遷移が必要なDPであまり解いたことがないが、(i−1)以下のDPテーブルだけでなく、i行におけるj−1以下の部分問題もすでに解けていること(!)、を強く意識した方がいいかもしれない。

[(個数制限付き(?))重複組合せ]

n種類の品物があり、i番目の品物はai個ある。
同じ種類の品物同士は区別出来ない。
これらの品物の中からm個選ぶ組合せの総数を求め、Mで割った余りを答えなさい。

$1 \leqq n \leqq 1000$, $1 \leqq m \leqq 1000$, $1 \leqq a_i \leqq 1000$,
$2 \leqq M \leqq 10000$

☆重複なく数え上げるためには、同じ種類の品物を一度に処理すればよさそうです。

$dp[i+1][j]$:= i番目までの品物から、j個選ぶ組み合わせの総数

i番目までの品物からj個選ぶためには、
i−1番目までの品物からj−k個選んで、i番目の品物をk個加えればよいので、

$$dp[i+1][j] = \sum_{k=0}^{\min(j,\, a[i])} dp[i][j-k]$$

※ここもややトップダウン的で、自分の解釈としては、
「i番目の品物を0個から取れる最大の数まで取る、というパターンは全て互いに排反なので、個別に求めて全て加算する」
ぐらいの方が納得しやすい。

ただし、このままだと3重ループになってしまう…(次ページへ)

(続き)

$$\sum_{k=0}^{\min(j,a[i])} dp[i][j-k] = \underbrace{\sum_{k=0}^{\min(j-1,a[i])} \left(dp[i][j-1-k]\right)}_{dp[i+1][j-1]} + dp[i][j] - dp[i][j-1-a_i]$$

これをつくりたいための式変形

これが足りてないのは、左辺のk=0を考えれば自明

この部分最初はよく分からなかったが、$dp[i+1][j-1]$の部分で$dp[i][j-1-a_i]$が加算されるケースがありうるので、その場合は引く必要がある、ということ(左辺では登場し得ないので!)

よって整理すると、 $dp[i+1][j] = dp[i+1][j-1] + dp[i][j] - dp[i][j-1-a_i]$

となる(添え字で引き算が絡む部分は0未満の時に注意!!)

※これも整理すると真横の遷移で効率化される。
個数制限系では鉄板の遷移形態である可能性が高そう。

※初期化も結構トリッキー

```
// 1つも選ばない方法は常に1通り
for i := 0 ; i <= n ; i++ {
    dp[i][0] = 1
}
```

[2-4] データを工夫して記憶する"データ構造"

[プライオリティキューとヒープ]
- 数の追加
- 最小の数値を取り出す(削除する)

ヒープ ⇔ 二分木 のポイント
- 必ず子の数値は親の数値よりも大きい
- 木は上から下へ、左から右へ順にノードがつまっている

☆ 2分木ですき間なく詰められるため、追加も取得も $O(\log n)$ で可能!

※ ガソリンスタンドの問題は、読み換えがとても面白いので
一度解いてみること!

[二分探索木]
次の操作が効率的に行えるデータ構造
- 数値を追加する
- ある数値が含まれているかを調べる
- ある数値を削除する

※ 恐らく、最大・最小も効率的に調べられるはず。

☆ すべてのノードについて、
① 左の子以下の数は自分の数よりすべて小さく
② 右の子以下の数はすべて大きくなるように管理する

☆ いずれの操作も木の深さが関係するので、$O(\log n)$ で可能!

※ set や map は二分探索木がベース
※ 純粋な二分探索木は木に「偏り」が出来てしまうと効率が悪くなるため、実際は
「回転処理」を行う「平衡二分木」が必要!

[食物連鎖 (Union-Find木の例題)]

N匹の動物、A→B→C→A の食物連鎖、

K個の情報(クエリ)が与えられ、クエリは以下の2タイプのいずれか。

タイプ1、xとyは同じ種類である

タイプ2、xはyを食べる

矛盾しているクエリの個数を求めよ。 ($1 \leqq N \leqq 5 \cdot 10^4$ 、 $0 \leqq K \leqq 10^5$)

※前から処理して、「以前の情報と矛盾するクエリ」で判断すればよい(はず)。

(手法)

各動物iについて、3つの要素 i-A、i-B、i-C を作り、

$3 \cdot N$ 個の要素で Union-Find を作る。

(コード例では、[i]でi-A、[i+N]でi-B、[i+2N]でi-C を表している)

この Union-Find 木は何を表すか ① 以下の「事象」を表すとする。

- i-X は、「動物iが種類Xである場合」を表す
- 各(同一)グループは、それらの場合がすべて同時に起こることがわかっていることを表す。

② UFの操作が以下のようになる

① タイプ1:xとyが同じ種類 → x-A ∪ y-A、x-B ∪ y-B、x-C ∪ y-C

② タイプ2:xはyを食べる → x-A ∪ y-B、x-B ∪ y-C、x-C ∪ y-A

③ このUFを使った矛盾のチェック例:

タイプ1の場合は、x-Aと、y-Bかy-Cが同じグループなら矛盾!

タイプ2の場合は、x-Aと、y-Aかy-Cが同じグループなら矛盾!

(次ページへ)

※ タイプ1で $x-B$ と $y-C$ or $y-A$ や $x-C$ と $y-A$ or $y-B$ をチェックしなくてよいのはなぜか？

…よくわからないが、恐らくそれぞれが同値なのだろう…。

※ この問題が初見で解ける人天才だと思う。

※ 与えられた情報から複数の場合が考えられる、暫定の状況からの矛盾で判定するなら Union-Find が使える可能性がある、というのは重要な知見！

[2-5] あれもこれも実は"グラフ"

[無向グラフの用語]

隣接 : 2つの頂点間に辺があるとき、その2つの頂点を隣接している、という。

パス : 隣接している頂点の列

閉路 : 始点と終点が同じようなパス

連結グラフ : 任意の二頂点間にパスが存在するようなグラフ

頂点の次数 : 頂点につながっている辺の数

木 : 連結グラフで、閉路を持たないようなもの

森 : 連結でないグラフで、閉路を持たないようなもの

※ 木は辺の数がちょうど(頂点数−1)

※ 逆に、辺の数が(頂点数−1)に等しい連結なグラフは木になる

@ 根付き木について :

- 根を一番上に、根から遠い頂点ほど下になるように木を配置することが出来る。
- 辺に向きを与えたと考えることもできる

(木を根付き木と見ることで考えやすくなる場合もある！)

— 家系図のように親子関係を見立てることもできる

[有向グラフの用語]

有向グラフの頂点 v に対して、v から出ていく辺の集合を $\delta_+(v)$

入ってくる辺の集合を $\delta_-(v)$ と表記することにする。

v の出次数 : $|\delta_+(v)|$

v の入次数 : $|\delta_-(v)|$

DAG : 有向グラフで閉路を持たないグラフ (Directed Acyclic Graph)

※ 例えば、整数を頂点とし、「n が m を割り切るときに n から m に辺を引いたグラフ」を考えると、これは DAG になる。

(n と m の大小関係を考えると閉路は出来なさそう。)

(次ページへ)

(続き) トポロジカル順序：DAGの各頂点への番号付けで，頂点 v_i から v_j に向かって
辺があるとき，$i < j$ が成り立つような番号付けのこと！
※ DAGの問題をDPで解くための準備方法！
トポロジカルソート：トポロジカル順序を求めること。

[グラフの表現]

[隣接行列 (有益そうなものだけ)]

- $i \to j$ の辺がない場合は $g[i][j] = INF$ とする (よくミスしてしまうので覚えておく)
  - 0とすると コスト0と区別がつかなくなる。
- 多重辺や自己ループも，重みつきでないなら $g[i][j]$ に辺の個数を入れればよい。
  - 重み付きの場合は無理。

[二部グラフ判定]

隣接している頂点同士を違う色にするとき，与えられた無向グラフを2色以内ですべての頂点を塗れるか？
($1 \leq n \leq 1000$)

☆ グラフの彩色問題：隣接頂点同士を違う色になるようにする問題。
☆ グラフの彩色数：彩色問題で必要な色の数の最小値。
☆ 二部グラフ：彩色数が2であるグラフ。

(解法)

1つの色を決めれば，連鎖的にすべて決まるので，DFSで判定できる！

☆ グラフのDFSでできること：① 連結判定 ② 木判定 ③ トポロジカルソート
☆ DFSの計算量は，すべての頂点と辺を一度ずつ見るため，$O(|V| + |E|)$！

[ベルマンフォード法] ※要点だけ思い出しておく

始点 s から頂点 i への最短経路を $d[i]$ とすると、

$$d[i] = \min\{d[j] + (j \text{から} i \text{への辺のコスト}) \mid e=(j,i)\in E\}$$

計算量は $O(|V|\times|E|)$ ①

※ベルマンフォード法はアルゴリズム図鑑がとても分かりやすいので、
くわしい内容をまとめない
※橋本は計算量の部分だけ学んでおく。
(コードも分かりやすいので参考にしたい!)

[ワーシャルフロイド法]

DPだと言われると納得しやすかったので、書いて理解を深める。

3次元のDPテーブルを考える。

dp[k+1][i][j] := 頂点0〜kとi, jのみを使う場合の、iからjへの最短路

(dp[0][i][j] = e[i][j] or INF で初期化)

↑ i, jは0〜k以外でも勿論よい！

☆ 頂点0〜kのみを使うとき、iからjへの最短路は

① 頂点kをちょうど一度通る場合
② 頂点kをまったく通らない場合
) の2つに(排反に)分かれる。

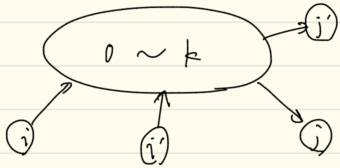
↑ 最重要!!

①のときは dp[k][i][j] = dp[k-1][i][j] (k-1までの最短路が採用される)

②のときは dp[k][i][j] = dp[k-1][i][k] + dp[k-1][k][j]

(i→kの最短路とk→jの最短路にkが分解する)

☆ dp[k+1][i][j]を求めるにあたって、任意のi⟷jにおいて、頂点0〜kを介する場合の最短路は分かっている。



☆ しばしば「kを一周するだけで収束するのか?」十分なのか?」という疑問がわくが、

上の赤枠・赤字部分をふまえ、DPであることを考えると、

疑問は解消されると思う。

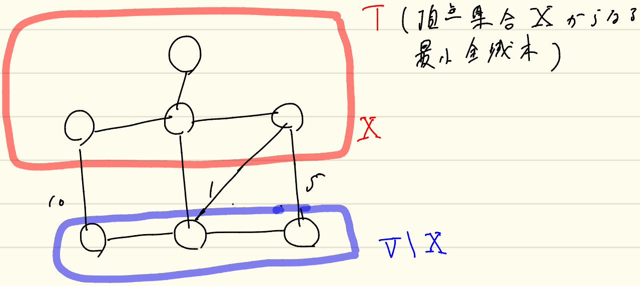
☆ 配列の再利用の部分は相変わらずよく分からない(難しい…)

☆ 経路復元する場合は再利用してはいけない?(どのみち時間が $O(|V|^3)$ なら空間も $O(|V|^3)$ でいい気がする…??)

(2019/07/17(水))

[プリム法]

要点：ダイクストラもDPっぽいので、似ているプリム法もDPっぽい。



☆ Tが含まない点へ伸びる辺の内、最小コストの辺を追加していけばよい。

[クラスカル法]

閉路が出来ないように、コストの小さい順に辺を見ていく。

※ プリム法は(ダイクストラの様に)頂点中心、
クラスカル法は辺中心、と覚えるといいかもしれない。

[Roadblocks]

二番目の最短路を求める問題。

☆ 各頂点について、最小の距離と、二番目に小さい距離を記憶しておくことで解ける！

(理由) ある頂点vへの二番目の最短路は、

① 他の頂点uへの最短路に u → v をつけたもの

② uへの二番目の最短路 u → v をつけたもの

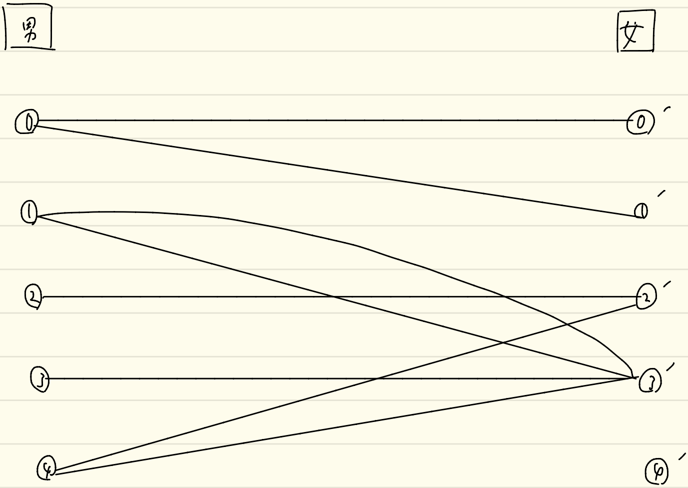
のいずれかだから。

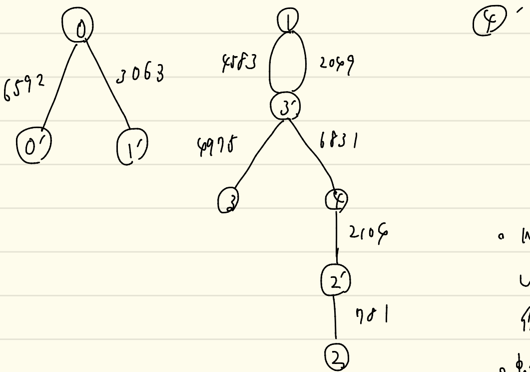
※ ちょっと難しいが、「とりあえず候補としてはこれ以上挙げる必要はないから、とりあえずキューにつっこんで、時期が来たら確定させればよい」ぐらいの考えはよさそうか②

(Various Sushi や、いつだかの双子のABC-D問題が想起される)

[Conscription] ※「徴兵」という意味らしい。

N人の女とM人の男全員を徴兵したい。
女⇔男では親密度が設定されており、すでに片方が兵にいる場合、
10000よりも安く徴兵できる。(min(10000−(すでに兵にいる方での親密度)))
適切な順序で徴兵するときの最小コストは?





○関係を利用して安く徴兵した場合を「辺を張った」と見る
(そのために、10000(M+N)からコストでスタートし、いくら引けるかで考える)

○ループが出来る場合は誰かを2回徴兵したことになってしまうため、閉路を作ってはいけない!

○蟻本にある「有向森へ変える」の部分は良く分からない…
(単に使用した関係を向かい付くするため?)

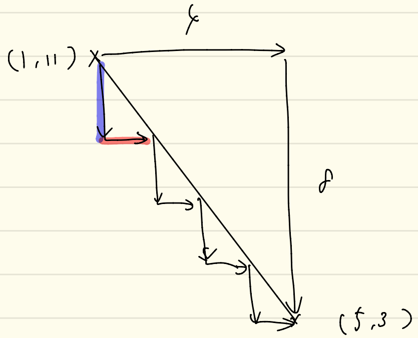
[Layout] もう問題文から難しい

① ⟷ ③ 10以内 , ② ⟷ ④ 20以内 , ② ⟷ ③ 3以上

番号順に, 一列に並ぶ (従い制約)

(2019/07/22 (月)) 2-6：数学的な問題を解くコツ

[線分上の格子点の個数]



傾きは $\frac{\Delta y}{\Delta x} = \frac{8}{4}$

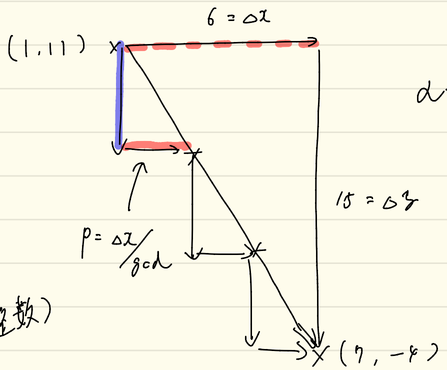
左図の青の長さは $8 / \gcd(4, 8)$

赤の長さは $4 / \gcd(4, 8)$

となるため、$\gcd(|x_1 - x_2|, |y_1 - y_2|)$

で求めれば一よい！

（答えは $(\gcd(|x_1 - x_2|, |y_1 - y_2|) - 1)$ ）



$\alpha = \frac{5}{2}$

$\gcd(6, 15) = 3$, $3 - 1 = 2$ が答え

(nαは整数)

$0 < n\alpha < \Delta y$, $n \geqq 1$ を満たす、nの数が知りたいもの。

αの既約分数式 $\frac{q}{p}$ に関して、pに興味がある。

$\frac{\Delta y}{\Delta x} = \frac{q \cdot \gcd}{p \cdot \gcd}$ という形になるため、$\Delta x / \gcd = p$ となる。

上の図より、Δxとpで割れば欲しいものが手に入るので、

$\Delta x / (\Delta x / \gcd) = \gcd$

これは左下の端点を含んでいるので $(\gcd - 1)$ が答え！

※ $\gcd(0, 0)$ の時は答えを0とする必要がある（例外ケース）

[ユークリッドの互除法] ※基本流の考えでしっかりと身につける!

自然数 a, b に対して、最大公約数を求める関数を gcd(a, b) とする。
a を b で割った商と余りを p, q とします。

$a = bp + q$ より、gcd(b, q) は a と b を割り切るので、

☆(1) gcd(a, b) も割り切ります。

逆に、$q = a - bp$ より同様にして、gcd(a, b) は gcd(b, q) を割り切ります。

☆(2) よって、gcd(a, b) = gcd(b, a % b) となる。

gcd の第二引数は必ず減少していくので、これを繰り返していくことで、いつかは gcd(a, b) = gcd(c, 0) となります。

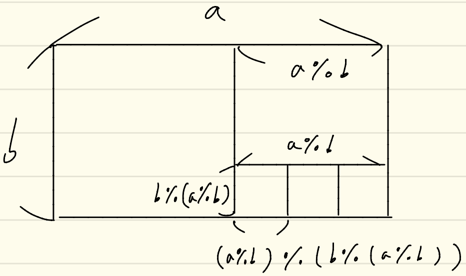
☆(3) 0 と c の最大公約数は c なので、gcd(c, 0) = c となり、

これで、gcd(a, b) が求まりました。

※ ☆(3) は知識。

※ ☆(2) は i) a が b を割り切り、かつ、ii) b が a を割り切る
つまりは a=b というもの(多分)。
gcd(a, b) = gcd(b, q) = gcd(b, a % b) という、
推移律っぽい形をしっかり頭に根付かせたい!

※ ☆(1) が一番難しい気がする。
gcd(b, q) は gcd(a, b) を割り切るとなぜ言えるのか?
gcd(b, q) は a と b を割り切る
gcd(a, b) は a と b を勿論割り切るし、定義から、そのような数の最大値である。
gcd(a, b) は、因数として当然 gcd(b, q) を含んでいる
(含んでいなかったらGCDの定義に反する)ため、
☆(1) が主張できる!



☆最も小さい正方形で
長方形をすき間なく埋めつくす
ことが出来る。

※標本の再帰によるコードはとても簡潔なため、ライブラリとして扱えること!!

[計算量]

$a > b$ と考える（$b > a$ でも $gcd(b, a\%b) = gcd(b, a)$ となり、
次に $gcd(a, b\%a)$ のように、第二引数の方が小さい状態になるので、
計算量を見積もる上では無視してよい！）。

① $b > \frac{a}{2}$ のときは $a\%b = a - b < \frac{a}{2}$ となる

② $b < \frac{a}{2}$ のときは $\underbrace{a\%b < b}_{\text{余りなので当然！}} < \frac{a}{2}$ となる

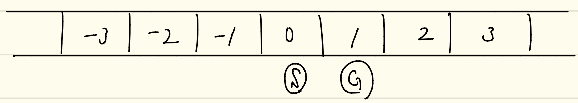
①、②より、「2回再帰すれば第一引数は半分以下になる」ということが分かる。

したがって計算量は $O(\log(\max(a, b)))$ に抑えられる！

[拡張ユークリッドの互除法]

$ax + by = 1$ となる整数を求めよ、という問題。

☆この式から明らかなように、$gcd(a,b) \neq 1$ の場合は解は存在しません。
→ 自明か?と思ったけど、例えば $a=2$, $b=6$ とかのケースで考えると、どう頑張っても左辺を1には出来ないことが分かる。



元のすごろく問題を考えればよりイメージがしやすくなる。

よって $gcd(a,b)=1$ のケースで考える(つまり、aとbが互いに素)。
この時点で、解くべき方程式が $ax+by = gcd(a,b)$ となる!

☆この方程式の解を求める関数を int extgcd(a, b int, x, y &int) とし、返り値を $gcd(a,b)$ とする。

$ax + by = gcd(a,b)$, $gcd(a,b) = gcd(b, a\%b)$ なので、
$bx' + (a\%b)y' = gcd(a,b)$
の整数解 x', y' が求まっているとする。ここで、$a\%b = a - \lfloor \frac{a}{b} \rfloor \times b$ なので、
これを代入すると、

$$bx' + (a - \lfloor \tfrac{a}{b} \rfloor b)y' = gcd(a,b)$$
$$ay' + b\underline{\underline{(x' - \lfloor \tfrac{a}{b} \rfloor y')}} = gcd(a,b) \text{ となる。}$$

当然、整数

aとbは互いに素なので $gcd(a,b) = gcd(1,0)$

以上のことから、これを繰り返すと
$a \times 1 + b \times 0 = gcd(a,b)$ $(= gcd(1,0))$ のようになる(?)

(extgcd(a, b, 1, 0) と入力すればよい?)
$ x \quad y$ ← 何でもOK!!

(2019/07/23 (火))　ユークリッドの互除法の補講（けんちょう氏の記事から）

◎ GCDの重要っぽい性質

① $\gcd(a, b) = \gcd(a - qb, b)$

※ 右辺は $\gcd(b, a\%b) = \gcd(a\%b, b)$

$= \gcd(a - qb, b)$ から。

② $\gcd(a, b) = \gcd(a - b, b)$

※ ①で q は a を b で割った時の商でなくてもよい。

③ $a \equiv c \pmod{b}$ ならば $\gcd(a, b) = \gcd(c, b)$

※ ②の一般化。

[拡張ユークリッドの互除法]

一次不定方程式 $ax + by = c$ の整数解 (x, y) について考える。

整数解をもつ必要十分条件：c が $\gcd(a, b)$ で割り切れること。

[具体例]：$111x + 30y = 12$

$\gcd(111, 30) = 3$ なので、$111x + 30y = 3$ を満たす x, y を考える。

$$111 \div 30 = 3 \cdots 21 \iff 111 = 3 \cdot 30 + 21 \iff 21 = 111 - 3 \cdot 30$$
$$30 \div 21 = 1 \cdots 9 \iff 30 = 1 \cdot 21 + 9 \iff 9 = 30 - 1 \cdot 21$$
$$21 \div 9 = 2 \cdots 3 \iff 21 = 2 \cdot 9 + 3 \iff 3 = 21 - 2 \cdot 9$$
$$9 \div 3 = 3 \cdots 0$$

これを「逆にたどる」。

$$\begin{aligned}
&3 \\
&= 21 - 2 \cdot \underline{9} \\
&= 21 - 2 \cdot \underline{(30 - 1 \cdot 21)} \\
&= (-2) \cdot 30 + 3 \cdot \underline{21} \\
&= (-2) \cdot 30 + 3 \cdot \underline{(111 - 3 \cdot 30)} \\
&= 3 \cdot 111 + (-11) \cdot 30
\end{aligned}$$

よって、$(x, y) = (3, -11)$ が $111x + 30y = 3$ を満たすので、

$111 \cdot (4x) + 30 \cdot (4y) = 4 \cdot 3 = 12$ から、

$(x, y) = (12, -44)$ が元の方程式を満たす！

すべての解を求めるには、解を代入したものを引いて

$111(x - 12) + 30(y + 44) = 0$　両辺を $\gcd(111, 30) = 3$ で割って、

$37(x - 12) = -10(y + 44)$、37 と 10 は互いに素なので、

$x - 12$ は 10 の倍数！よって t を任意の整数として、

$x - 12 = 10t$、これを代入すると $37 \cdot \cancel{10}t = -\cancel{10}(y + 44)$

$\therefore y + 44 = -37t$　よって、$(x, y) = (10t + 12, -37t - 44)$ □

[拡張ユークリッドの互除法のアルゴリズム的記述]

☆数学的な証明も面白いが，プログラマとして，こちらの理解を目指す。

(☆) $ax + by = \gcd(a, b)$ を満たす (x, y) を求めるアルゴリズムの導出。

このアルゴリズムのことを，特に拡張ユークリッドの互除法と呼ぶ。

$d = \gcd(a, b)$ とおく。

a を b で割って $a = bq + r$ とする。これを元の方程式に代入すると，

$(bq + r)x + by = d \Leftrightarrow b(qx + y) + rx = d$

となります。これによって，(a, b) に関する問題が，

それより数値の小さな (b, r) に関する問題に帰着できました。

※ $r = (a \% b)$ なので $(b, a \% b)$ とするとユークリッドの互除法と同じ形!!

これを再帰的に解くのが拡張ユークリッドの互除法。

具体的には (b, r) に関する小問題を解いて

$bs + rt = d$ を満たす解 (s, t) が再帰的に得られたとすると，

$qx + y = s$, $x = t \Leftrightarrow x = t$, $y = s - qt$

という風に，元の問題の解を構成できます。

最終的にこの問題は，$dx + 0y = d$ まで「小さく」なる(互除法より)。

この解は明らかに $(x, y) = (1, 0)$ となる(＊)。

よってここから再帰的にドンドン上へさかのぼることで元の方程式の解を構成できる。

※(＊)の部分は自明なのか? y の係数が 0 であるため，y は自由に設定出来てしまいそうに見える。

(難しいので，ここまでにとどめておく…)

※プログラムは再帰の終了条件が先に来ており，かつ早期 return によって非常に読みやすいため，参考にしたい!

[計算量]

再帰の深さがユークリッドの互除法の繰り返し回数と同じなので、

計算量も同じ!!

[$ax + by = \gcd(a, b)$ の解の大きさ]

$ab \neq 0$ のときは $|x| \leq b$, $|y| \leq a$ となる。

(帰納的に示すことが出来る。)

素数関連

[区間内の素数の個数]

与えられた整数 a, b に対して、区間 [a, b) に素数はいくつある?

(制約)

$a < b \leq 10^{12}$, $b - a \leq 10^6$

※ 1つ目の制約のため、競プロの基礎であるような引き算による方法はNG。

☆ b未満の素数である整数の最小の素因数はたかだか $\sqrt{b}$ である。
(以外と自まというか、言われないと想起しづらい、と思う。)

☆ [2, $\sqrt{b}$) の表と [a, b) の表を別々に作っておいて、
[2, $\sqrt{b}$) の表から素数が得られるたびに、
その素数の倍数を [a, b) から除けば、[a, b) にある素数が
列挙できる!

☆ $\lfloor (a+i-1)/i \rfloor^{(*)}$ とは何か?
[a, b) の中の i の倍数を取り除きたい
今、[0] ~ [b-a] という添え字が [a] ~ [b] に対応していることに注意!
0 ~ b-a
a ~ b
[a, b) の最初の i の倍数を ni とすると n は $\frac{a}{i}$ の切り上げの値。
これは $(a+(i-1))/i$ となる!
※ 切り上げの公式は覚えておこう!!

☆ $\left\lfloor \dfrac{(a+(i-1))}{i} \right\rfloor = \left\lceil \dfrac{a}{i} \right\rceil$ を示す。　($i > 0$ と仮定する)

(i) $a = n \cdot i$ のとき
($a \% i = 0$ のとき)

$$P = \left\lfloor \frac{ni+i-1}{i} \right\rfloor = \left\lfloor (n+1) - \frac{1}{i} \right\rfloor = \lfloor n+\alpha \rfloor = n$$

$(0 \leqq \alpha < 1)$
(等号は $i=1$ のとき)

$$Q = \left\lceil \frac{ni}{i} \right\rceil = \lceil n \rceil = n$$

よって $P = Q$

(ii) $a \% i = m$ $(m = 1, 2, \cdots, i-1)$ のとき　($i \geqq 2$ と仮定する)

$a = n'i + m$ とおく

$$P = \left\lfloor \frac{(n'i+m)+(i-1)}{i} \right\rfloor = \left\lfloor n'+1+\frac{m-1}{i} \right\rfloor$$

$0 \leqq m-1 \leqq i-2$ なので

$$= \lfloor n'+1+\alpha \rfloor = n'+1$$

$0 \leqq \alpha < 1$

$$Q = \left\lceil \frac{n'i+m}{i} \right\rceil = \left\lceil n' + \frac{m}{i} \right\rceil$$

$1 \leqq m \leqq i-1$ より

$0 < \frac{1}{i} \leqq \frac{m}{i} \leqq 1 - \frac{1}{i} < 1$　$\therefore$ $0 < \frac{m}{i} < 1$

よって $Q = n'+1$

よって $P = Q$　□

(汚い証明だけど明らかにおかしいことはしてないはず,,,。)

(2019/07/24(木))

[余りの計算]

☆ a が負の場合、$a \% m$ も負になるので、
$(a \% m) + m$ と置き換えて $0 \sim (m-1)$ の範囲に
収まるようにする。

※実際の競技では、$a \% m$ が負かどうかの場合によらず、
$(a+m) \% m$ とした方が安全か?

[Carmichael Numbers]: べき乗を高速に計算する(繰り返し2乗法)
任意の $1 < x < n$ に対して、
$x^n \equiv x \pmod n$ が成り立つような素数でない整数 n
を Carmichael Number といいます。与えられた n に対して、それが〜か
判定しなさい。($2 < n < 65000$)

※後日解いてみる!!

3-1：値の検索だけじゃない！"二分探索"

[平均最大化]

重さと価値がそれぞれ w_i, v_i であるような n 個の品物があります。この中からちょうど k 個選んだときの「単位重さあたりの価値」の最大値を求めなさい。
(※制約は、全組合せで調べるのは無理な程度。)

条件 $C(x)$：単位重さあたりの価値が x 以上となるように選ぶことができる。

小 | ok | ng | 大 な感じで単調性を有していることは分かる。

$C(x)$ の判定がそれほど簡単ではない！

$$\frac{\sum_{i\in S} v_i}{\sum_{i\in S} w_i} \geq x$$

という条件の判定がしたい

変形すると $\sum_{i\in S}(v_i - x\cdot w_i) \geq 0$ を判定すればよい
(⇔ これを満たす S があれば OK!!)

これならば $(v_i - xw_i)$ の値で降順ソートして貪欲に選ぶことができる!!

よって、

☆ $C(x)$ = ($(v_i - xw_i)$ の大きい方から k 個の和が 0 以上)

※「平均最大化」という難しい問題が、「平均がある given な x 以上かどうか？」という、本質的に易しい2値判定問題になった、ということだと思う。

2019/07/26(金)　3-2：厳選！頻出テクニック(1)

[反転(フリップ)系のやつ]　[Face The Right Way]

重要な観察：

① 区間を反転させる順序を入れ換えても、結果に影響しない

② 同じ区間を2回以上反転させる必要はない

$f[i]$:= 区間 $[i, i+k-1]$ を反転させたならば1、そうでないなら0

牛 i に着目している段階において、その向きは $\sum_{j=i-k+1}^{i-1} f(j)$ が

① 奇数である場合は最初の向きと逆

② 偶数なら最初の向きのまま

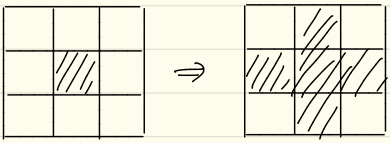
※これは i 番目の牛のフリップ回数を表す!!

$$\sum_{j=(i+1)-k+1}^{i} f[j] = \left(\underbrace{\sum_{j=i-k+1}^{i-1} f[j]}_{\boxed{f[i-1]}}\right) + f[i] - f[i-k+1]$$

なので(後ろの結果を使って)毎回定数時間で更新可能!

※観察①、②ともに少し「本当かな?」と思ってしまいそうだが、
この数式を見た通り、「何回そのマスがflip対象となったか?」
を考えると、正しいことが自然と分かる!

[Fliptile]



$1 \leqq M, N \leqq 15$

ひっくり返したい中央を選ぶと、その縦横も含めた5マスがひっくり返ってしまう。
M×Nマスを全て白色にしたい、という問題。

今回も ① 同じマスに最大1回
② 反転させるマスの集合が同じならば順序は関係ない
が成り立つ。よって単純な全探索だと 2^{NM} 通り試せばよい。(10^9ぐらい)

☆ 一番上の一行のひっくり返し方を決めてしまったとする。
→この場合、(1, 1)をひっくり返すには、新たに(2, 1)をひっくり返すしかない。つまり、(2, 1)をひっくり返すべきかどうかが決まる!
(以降の行についても同様なことが言える。)

☆ つまり要約すると、一番上の一行のひっくり返し方を決めてしまうと、そのような解が存在するかと、存在する場合の最小手数が簡単に決まる。

☆ 一番上の行のひっくり返し方を全通り試してしまえばよい!

☆ 一番上のひっくり返し方(=中心の選び方)は 2^N 通りなので、よって全体で $O(MN \cdot 2^N)$ となる!!

※ 絞り込んで全探索という流れがとても大事な問題!

2019/07/27(土)

[半分全列挙]

upperbound, lowerbound について もう少しなじんでおいた方がいいと思ったので、少し振り返り復習。

	0	1	2	3	4	5	6
A =	1	2	3	3	3	4	5
			ℓ			r	

3の数が知りたい！

r − ℓ で求まる！

ℓ で求めるめぐる式は

False	True

u で求めるめぐる式は

False	True

のように同じ。ただし条件が違う。

lowerBound は「key 以上」, upperBound は「key より大きい」

右側について開区間で出してくれるようなイメージ!!

(2019/07/31(水))

[巨大ナップサック]

重さと価値がそれぞれ w_i, v_i であるような n 個の品物があります。これらの品物を重さの総和が W を越えないように選んだときの価値の総和の最大値を求めなさい。

$1 \leq n \leq 40$ ①

$1 \leq w_i, v_i \leq 10^{15}$

$1 \leq W \leq 10^{15}$

※ DPの解法は O(nW) などであり、今回はアウト。

☆ n の小ささを利用した別の解法を考える!

☆ 2^{40} はダメだが、2^{20} ならばセーフ、という意識を持ちたい。

半分に分けた片方から、重さと価値の総和がそれぞれ w1, v1 であるような選び方を考える

→ もう片方から、重さが $w2 \leq W - w1$ の条件の下で v2 が最大となるように選べばよい!!

② 全列挙により得られた重さと価値の総和 (w2, v2) のペアの集合から、$\max\{v2 \mid w2 \leq W'\}$ を効率的に計算する方法は②

→ $w2[i] \leq w2[j]$ かつ $v2[i] \geq v2[j]$ であるような j は取り除いて良い!(i番目の組み合わせを選ぶべきだから)

これ自体は (w2, v2) で辞書順ソートすればよい!

※書籍では「(ソートさえしておけば)簡単に行うことができる」とあるが、コードで見るとそんなに簡単でもないように見えた。

(例え)

×

(weight) : ⓪ ≦ ① ≦ ② ≦ ③ ≦ ④ ≦ ⑤

(value) : ⓪ ≦ ① ≧ ② ≦ ③ ≧ ④ ≦ ⑤

(i) ① と ② で比較して、② を除く

(ii) ① と ③ で比較して、③ を除かない。

その場合、比較(元)対象を ① から ③ に移す

⋮

※ 多分、配列の再利用をしているが、ぱっと見て難しいコードになっている…。

(最後の2分探索部分)

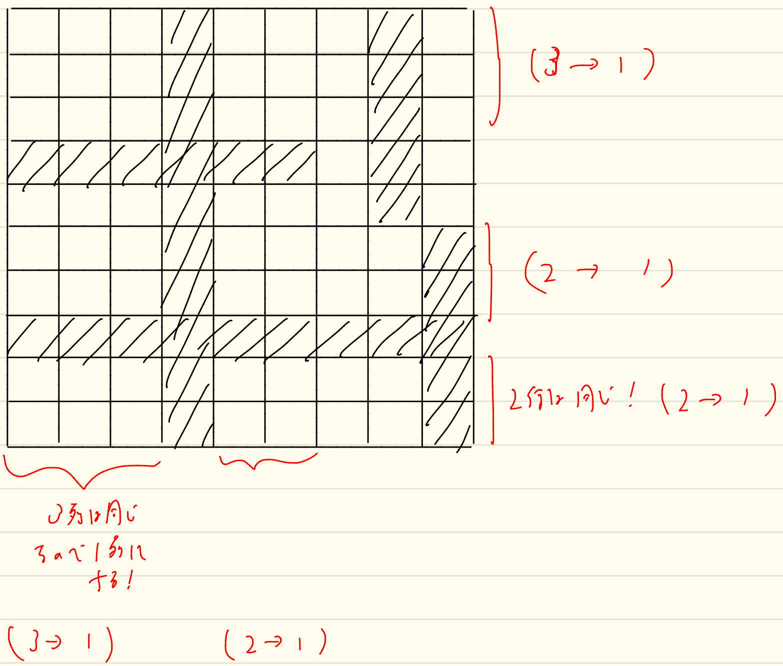
w2[i] ≦ W' となるような最大の i

ok | true | false | ng

[座標圧縮]：領域の個数

w × h の格子上に n本の垂直または水平な幅 1 の直線が引かれている。
格子は線によって何個の領域に分断されるか？

$1 \leqq w, h \leqq 10^6$, $1 \leqq n \leqq 500$

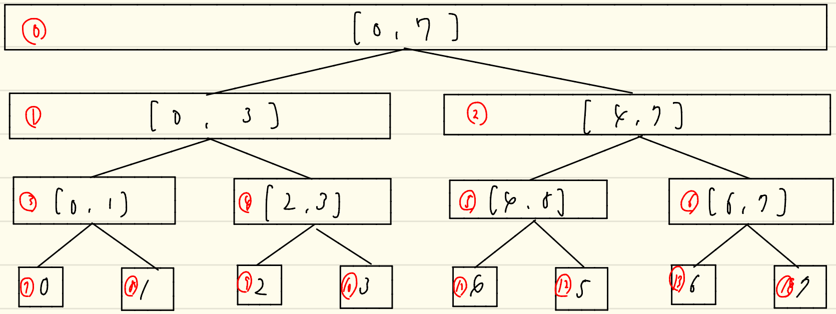


※どうやってプログラムにする？

> 配列には「線の端点のある行・列およびその前後の行・列」があれば十分。
> その大きさはたかだか 6n × 6n となります。
> これであれば配列で持つことが出来、この上で探索を行えば、領域の個数
> が分かります。

(2019/08/04 (月))

[セグメント木]



☆要素数 n のセグメント木の初期化の(時間)計算量および空間計算量は $O(n)$ です

これは節点の数が $n + \frac{n}{2} + \frac{n}{4} + \dots = 2n$ であるから!

$$\Downarrow$$

$$n + \frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{n} = 2n$$ ($2n-1$ が正確?)

最下層　　　最上層

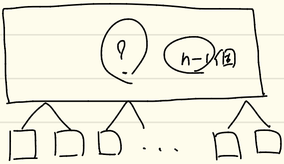
[配列による実装]

dat[2*MAX_N−1]: 2n−1 今あればよいので納得

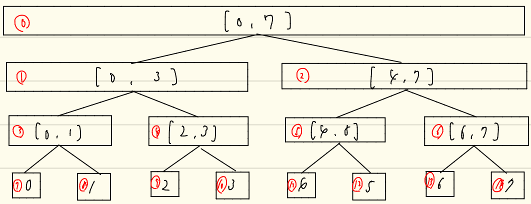
木の登り: k番目の要素をaに変更したい
(0-based)

k += (n−1): 木の葉でk番目のものを指すようにする

n−1個 → n−1個のノードが存在するので += (n−1)で葉ノードにシフトできる!



$k = \frac{k-1}{2}$: 一段ずつ登る



このkはセグ木の配列indexである。
セグ木の配列インデックスを−1して2で割った商が、自身の親のセグ木上の配列インデックスであることが分かる。

dat[k] = Min(dat[k*2+1], dat[k*2+2]): 自身の2つの子を見る

ここから、自身の2つの子のセグ木上の配列インデックスを得るには、自身を2倍したものに+1, +2それぞれ加算すればよいことがわかる。

☆ updateについては、上の考察で納得が出来る。

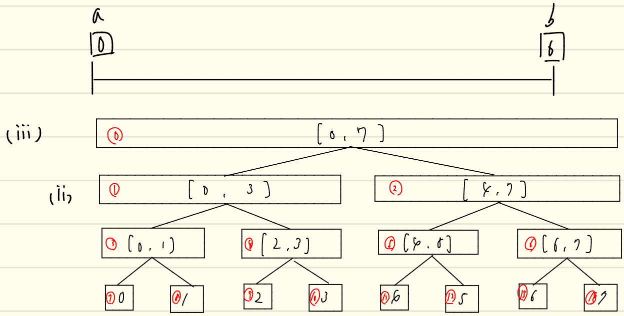
続いて query 部分、コードの前にまずは原理を整理する。

ある区間の最小値は再帰的に求める（根から下っていく！）

(i) 与えられた区間とその節点の区間がまったく交差していなければ、最小値に影響しない値を返す（e.g. INT_MAX）

(ii) 与えられた区間が完全にその節点の区間を含むような節点であれば、その節点のもつ値を返す。

(iii) そうでなければ、2つの子ノードについて再帰的に計算し、その2つの値の最小値を返す。



(i) 全く交差しない（無関係）

(ii) クエリ区間が完全に包含

(iii) クエリ区間と（注目）節点区間が部分的に交差

これらは再帰呼び出しを行わない！

(iii) は注目節点の2つの子ノードへ伝播する必要がある！

◦ クエリ区間である a, b は不動、k（だけでなく k に相当する l, r）も計算する必要がある！

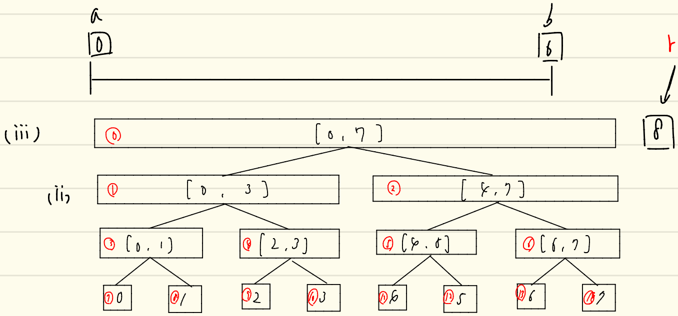
$k \to 2k+1$, $l \to l$, $r \to \frac{l+r}{2}$

$k \to 2k+2$, $l \to \frac{l+r}{2}$, $r \to r$

（上は左の子、下は右の子）

[query のコードが少しおかしい?]

(i), (iii) の条件分岐が少しおかしい気がするが… ?



rの初期値はノード数に相当する値が指定されることに注意!!

[a, b) のような半開区間のクエリであることをおさえる。

× (r ≦ a || b ≦ l) → ○ (r < a || b ≦ l)

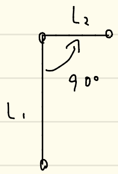
× (a ≦ l && r ≦ b) → ○ (a ≦ l && r < b) な気がする…。

[l, r) も半開区間だったのでずれるはず…

[a, b) と [l, r) の交差を考える必要がある

これならば確かに蟻本の実装で問題なさそう…。

[Crane]



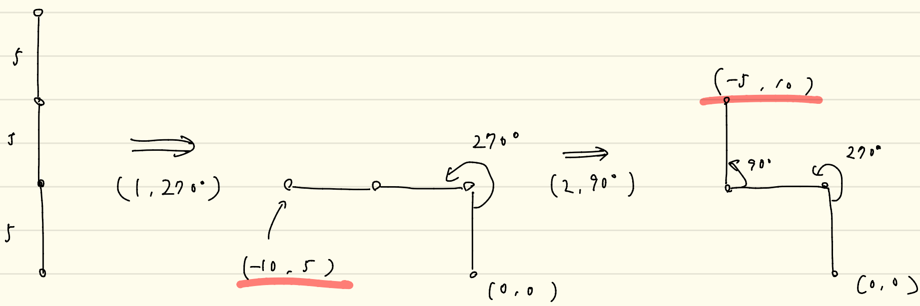
(1, 90) という命令の後

i から i+1 へ「反時計周りに」90°

命令は、各線分間（1つのクエリにつき1つ）を何度にするか、というもので、各命令後のクレーンの先端の座標を出力する必要がある。

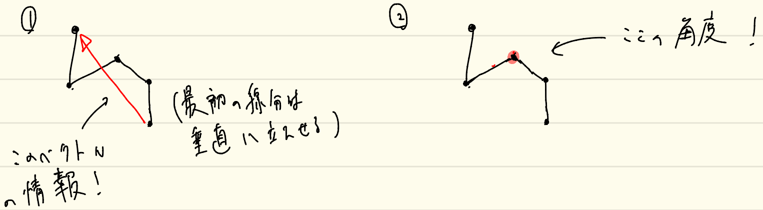
☆各ノードで連続する線分の集合に対応させ、次の情報を持たせる。

☆2つ目のサンプルで図示してみる。



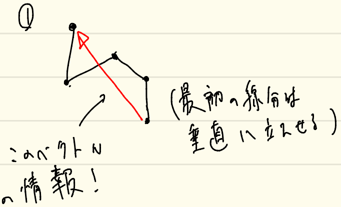
赤下線部の float ペアを出力すべきもの！

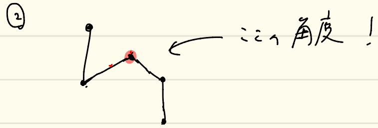
セグメント木：各ノード＝連続する線分の集合



(Crane 用に空けておく)

セグメント木： 各ノード = 連続する線分の集合

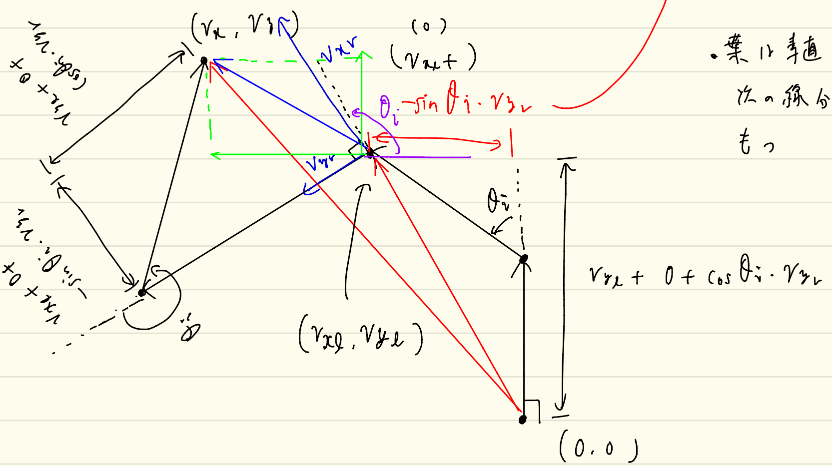


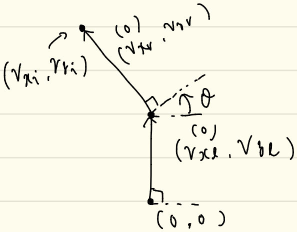


$$vx_i = vx_l + (\cos\theta_i \cdot vx_r - \sin\theta_i \cdot vy_r)$$
$$vy_i = vy_l + (\sin\theta_i \cdot vx_r + \cos\theta_i \cdot vy_r)$$

(vx_r は 0)

・葉は垂直に自分から見た次の線分へのベクトルをもつ





基準地面からの傾きが θ_i ?

回転の1次変換で思い出す!!

$$\begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \begin{pmatrix} vx_r \\ vy_r \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

☆左子ノードを地面の基準として右子ノードの「回転角度」を取得、
それを用いた回転の1次変換で右子ノードのベクトルに作用させて
左子ノードのベクトルに加算…が正解っぽい。

(2019/08/05 (月))

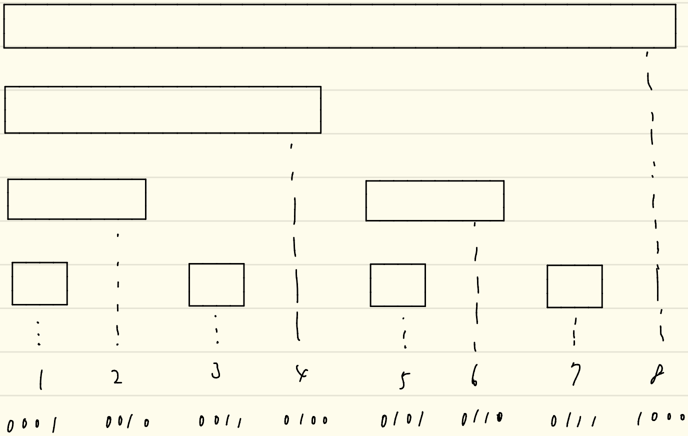
[Binary Indexed Tree]

列 $a_1, a_2, \dots, a_n$ が与えられて、

・$a_1 + \dots + a_i$ を計算 ← これにより、任意区間の累積和が

・a_i += x　　　　高速に計算できる

☆ BITの仕組み



☆ idxが対応する区間の長さは、末尾の0の数をn個とすると 2^n となる！

⇒ 各操作がシンプルなビット演算で書けるようになる!!

☆ BITでの和の計算

$i \to 0$ になるように、iから最後の1のビットを減算しながら、場所iの値を加算していく。

i=5 (0101) の場合、

① A[5]を加算　② A[4] (0100) を加算　③ A[0]=0を加算 ?

☆ BITでの値の加算

iの値にxを加えるためには、iからはじめて、iに最後の1のビットを加算しながら場所iの値にxを加えていく(伝播)

i=5の場合、

① A[5]に加算 ② A[6] (0110) に加算 ③ A[8] (1000) に加算

0101 +0001 +0010

☆ 計算量

(結局木の高さなので) 共に O(log N)

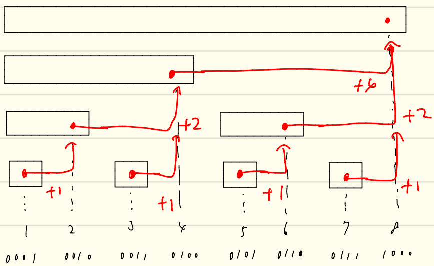
これは覚えておってよし!!

☆ iの最後の1のビットは、"i & (-i)" と書ける! (なぜ?)

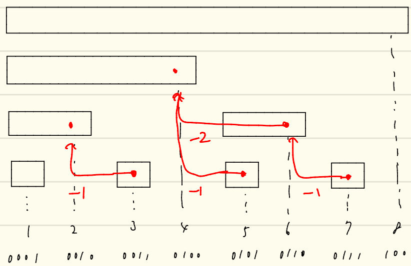
(最も下位の)

⇒ これは「区間の長さ」を意味している (e.g. '0100' は区間の長さ4)

⇒ 区間の長さを自身に足し(引き)、自身がジャンプすることで、目的のノード番号が得られる(以下の図を参照すべし!)



[値の更新]



[和の計算]

(2019/08/11(月))

[バブルソートの交換回数]

求めるスワップ回数は、$i<j$, $a_i > a_j$ となるような
(i, j) の組の個数。

※このような組の数を**反転数**という。

Q 各 j に対して、$i<j$, $a_i > a_j$ となるような「i の個数」
を効率的に求める、というアプローチをとる。

↓

☆ 値の範囲が $1 \sim n$ の BIT を用いて、$j = 0, 1, 2, \cdots, n-1$
の順に以下を行う。

- ans += j - sum(A[j]);
- add(A[j], 1);

$\}$ よく分からん…. ?

sum(A[j]) : $i<j$, $a_i \leqq a_j$ なる i の個数 (らしい…)

(※加算の操作を逐次行うことが前提(のはず…))

$A = [3, 1, 4, 2]$, ~~sum(A[0]) = sum(3) = 8~~

Q 初期値が全て 0 の BIT から始めるのかもしれない!

bit = [0, 0, 0, 0, 0]

① ans += 0 - sum(A[0]) = 0 - sum(3) = 0

bit = [0, 0, 0, 1, 1]

② ans += 1 - sum(A[1]) = 1 - sum(1) = 1

bit = [0, 1, 1, 2, 2]

③ ans += 2 - sum(4) = 2 - 2 = 0

bit = [0, 1, 1, 2, 3]

④ ans += 3 - sum(2) = 3 - 1 = 2

bit = [0, 1, 2, 3, 4]

(まだすっきりできないので、もう少しかみくだきたい)

$bit = [0, 0, 0, 0, 0]$

① ans += $0 - sum(A[0]) = 0 - sum(3) = 0$

$bit = [0, 0, 0, 1, 1]$

② ans += $1 - sum(A[1]) = 1 - sum(1) = 1$

$bit = [0, 1, 1, 2, 2]$

③ ans += $2 - sum(4) = 2 - 2 = 0$

$bit = [0, 1, 1, 2, 3]$

④ ans += $3 - sum(2) = 3 - 1 = 2$

$bit = [0, 1, 2, 3, 4]$

この動きをもう少し分かりやすく解釈してみる

3が登場 → bit[3], [4] が +1

1が登場 → bit[1], [2], [3], [4] が +1

⋮

bit[i]：これまで見てきた中で(=j番目の直前までで)、
i以下の数の登場回数

※↑を、「sum(A[j]) は $i<j$, $a_i \leqq a_j$ なる i の個数」
と併せると、理解できると思う。

これを j から引けば、目的の反転数が得られる、
というのは納得！

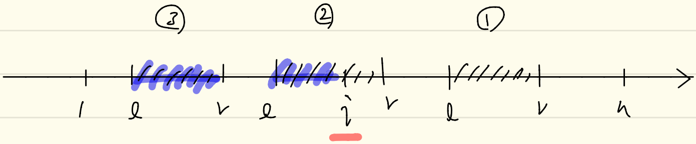
(例えば、j=2まで見て、前2つの要素のうち、
自分以下の数値の数が0個だったら、2−0個
全てが自分より大きいはず。)

※今回は、1〜nのn個の全て異なる数だったが、
N個の数が整数で、上限が10^7程度なら、
同じ原理で解けるはず！

(同じ数値が含まれていてもいい)

[A Simple Problem with Integers] (遅延セグ木の関連問題②)
(BITによる解法)

$s(i) = x$ を加える前の $\sum_{j=1}^{i} a_j$, $s'(i) = x$ を加えた後の $\sum_{j=1}^{i} a_j$



① $i < l \longrightarrow s'(i) = s(i)$, 影響なし

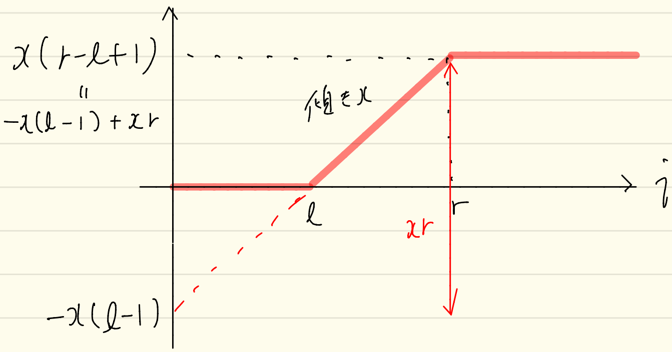
② $l \leq i \leq r \longrightarrow s'(i) = s(i) + \underline{x \times (i - l + 1)}$

$= s(i) + x \cdot i - x(l-1)$

③ $r < i \longrightarrow s'(i) = s(i) + \underline{x(r - l + 1)}$

$= s(i) - x(l-1) + xr$

累積和の増加分



bit0 の l に $-x(l-1)$ をプラス

bit0 の r+1 に xr をプラス

bit1 の l に x をプラス

bit1 の r+1 に $-x$ をプラス

$$\sum_{j=1}^{i} a_j = \text{sum}(\text{bit1}, i) \times i + \text{sum}(\text{bit0}, i)$$

↑

求めたいもの！

(続く)

(続き)

コードを読んだ上で考えなおしてみる...

- bit0とbit1について、A[i]にもとづいた初期化を行うのは、bit0の方のみ!!!
  - 最重要な大前提!!!!
- いもす法っぽい空気もなんとなく感じられる... ?
  - bit1の方がいもすっぽい役割を担っているように見える。

$$\sum_{j=1}^{i} a_j = \underline{sum(bit1, i) \times i} + sum(bit0, i)$$

求めたいもの!

いもす法っぽいBITの和にiをかけて加算する

- とはいえ、いもす法のイメージで理解に役立てるのは難しそう... ?
  - ちゃんと数式を追うのが近道かもしれない...

[バケット法と平方分割]

バケット法：列や平面をバケットなる単位に分割して、
(bucket method) バケットごとにデータを管理することにより、
効率的に計算や操作を行う手法。

平方分割：n個の要素からなる列を、$\sqrt{n}$程度ごとのバケットにまとめて管理する手法の俗称。
(sqrt decomposition)
(※バケット法の特殊形)

☆セグ木と同様に、どのようにデータを持たせるかによって、
さまざまな機能を実現できる!

☆RMQについてはかなり愚直に考えればよいことが分かる
(計算量も、更新・クエリ共に $O(\sqrt{n})$ というのも感じよい。)

☆利点
- 実装がセグ木よりもシンプルなので、余裕があれば、平方分割で選ぶのもよい。
- セグ木では効率的に実現できないような機能でも、平方分割では実現できる場合がある。

[K-th Number]

☆ 数xがk番目ということは、
- その区間にx以下の数はk個以上存在
- その区間にx未満の数はk個未満存在

ということに他ならない（同値？）

区間：①　②　③　④　⑤　⑥　⑦　⑧

$\cdots \quad x' < x = x = x < x'' \cdots$

x以下：5個
x未満：2個
} xは k=3, 4, 5 番目

☆ （x未満）< k ≦（x以下）

l　　　r

※ 当然 $r \geqq l+1$

☆ 区間に存在するx以下の数の個数を効率的に数えることができれば、入れって二分探索を行って、k番目の数を求めることが出来る。

↖ 平方分割でもセグ木でも、どちらの解法でも、このベースは同じ！

☆ 平方分割による解法：
気持ちはRMQと似ている。クエリがバケットを全て含む場合は、ソート済みのバケットの中で二分探索、はみ出した部分は愚直に見る。

☆ セグ木による解法：
ノードに「ソート済み列」を持たせるとが面白いが、やはりベースは同じ（再帰的に二分探索）

※ $\log N$の見積もりをカンタンにできるようにしておきたい。

※ AOJにverify問はないだろうか？

[平方分割解法をより詳しく]

$$\underbrace{⓪\quad ①\quad ②}_{\text{ソート}}\quad \underbrace{③\quad ④\quad ⑤}_{\text{ソート}}\quad \underbrace{⑥\quad ⑦\quad ⑧}_{\text{ソート}}$$

クエリ (l, r, k) = $[l, r]$ でソートした時に(小さい方から) k 番目の数が知りたい!

$b = \sqrt{n}$ とする。

- 配列全体をソートしておく
- 配列全体のインデックスについて、二分探索する
  - $lb := -1$, $ub := n-1$, $md := \frac{lb+ub}{2}$ $\Rightarrow$ $x :=$ nums[md]

sorted nums : (小) | false | true | (大)

condition は、「自身より小さい数が k 個以上存在するかどうか?」

これを満たすギリギリの x が欲しい! ← これは大事な典型思考かもしれない

区間 $[l, r]$ の中の

$[l, r]$ の中にない数値も x として選ばれるが、何故これが答えとして選ばれないのか? をはっきりさせる必要がある…。

(i) x が区間外だが、正解の数値の場合(=$[l, r]$ にも存在する場合)
 → x が答えとして判定されて、何も問題がない

(ii) x が区間外かつ、正解の数値より大きい場合
 → より小さい、かつ条件を満たす正解の数値を求めて、二分探索が続く

(iii) x が区間外かつ、正解の数値より小さい場合
 → 正解の数値は条件を満たすギリギリの値(=最小の値)なので、
 この x は条件を満たさない! よってより大きい方向に x を動かして、
 二分探索が続く

↑の (i), (ii), (iii) を合わせると、ちゃんと正しく動作することが分かる。

[3-4] 動的計画法を極める!

[巡回セールスマン問題]

※ 再帰バージョンは解いたことがないので、一度写経してみること!

※ ループも何かヒントが得られるかもなので、一度写経しておく!

[Traveling by Stagecoach]

- 乗車券を集合で管理すると、遷移図をかいたグラフ上でダイクストラ法を用いれば、最短路が求まる。
- 集合は真に小さくなる(=戻れない)ため、DAGとなる。

DAGの最短路はわざわざダイクストラ法を用いなくてもDPでカンタンに計算できる(!)

↑

これちょっとまだよく分からない…

[ドミノ敷き詰め]

これについては、愚直解バージョンとbitDPバージョンの両方を写経してみて、都度ポイントを抑えていくつもりで取り組んだ方がよい。

(図は分かりやすいと思うので、コードとあわせて理解を深める。)

※ このようなドミノしきつめ問題は、グラフ理論における完全マッチングと表現されるらしい。

[行列累乗]

[フィボナッチ数列]

フィボナッチ数列のn項目の値を10^9で割った余りは？

$(0 \leqq n \leqq \underline{10^{16}})$

!

フィボナッチ数列(定義)：

$$F_0 = 0$$

$$F_1 = 1$$

$$F_{n+2} = F_{n+1} + F_n$$

$$\Rightarrow \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

$$\cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

$$= A^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

添え字に注意!!

$$\therefore \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

よってA^nを求めることで、F_nを計算できる。

A^nの計算には繰り返し二乗法を用いることができる！

よって$O(\log n)$でn項目が求まる。(let's 実装！)

[Blocks]

N個のブロックが横一列に並ぶ。赤・青・緑・黄の4色のペンキがあり、これらで各ブロックを塗っていく。
赤と緑のブロックの個数が共に偶数個となるような塗り方の総数を、10007で割った余りを出力せよ。
$(1 \leqq n \leqq 10^9)$

こういう決め方って一つの典型思考っぽい。

☆左から順に塗っていく方針をとる!
i個目までについて「塗り方の総数」を考えると…
a_i：赤・緑ともに偶数個となるようなもの
b_i：赤・緑の片方のみが奇数個 〃
c_i：赤・緑ともに奇数個 〃
このように定義すると、以下の3つの漸化式が導かれる!

$$
\left.\begin{aligned}
a_{i+1} &= 2a_i + b_i \\
b_{i+1} &= 2a_i + 2b_i + 2c_i \\
c_{i+1} &= 2b_i + c_i
\end{aligned}\right\}
$$

i+1の色の選び方次第で納得できる。

$$
\begin{pmatrix} a_{i+1} \\ b_{i+1} \\ c_{i+1} \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 2 & 2 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \\ c_i \end{pmatrix}
$$

$$
\begin{pmatrix} a_i \\ b_i \\ c_i \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 2 & 2 \\ 0 & 1 & 2 \end{pmatrix}^i \begin{pmatrix} a_0 \\ b_0 \\ c_0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 2 & 2 \\ 0 & 1 & 2 \end{pmatrix}^i \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}
$$

※この初期値を見るとDPっぽさを深く感じる。

[グラフの長さkのパスの総数]

頂点数nで、すべての辺の長さが1であるような「有向グラフ」の隣接行列が与えられる。このグラフに含まれる長さkのパスの総数を10007で割った余りを求めよ。ただし、同じ辺を複数回通るようなパスも含む。

$(1 \leqq n \leqq 100,\ 1 \leqq k \leqq 10^9)$

☆ uからvへの長さkのパスの総数を $G_k[u][v]$ とする。

- $k=1$ の場合は辺の本数と等しくなるため、G_1 はグラフの隣接行列と等しい。
- G_{k_1} と G_{k_2} が計算できたとする。すると、

$$G_{k_1+k_2}[u][v] = \sum_{w=1}^{n} G_{k_1}[u][w] \times G_{k_2}[w][v]$$

となるため、

$$G_{k_1+k_2} = G_{k_1} G_{k_2}$$

と、行列積で表すことができる。

行列の積の定義そのものなので！

したがって、

$$G_k = G_1^k$$

と隣接行列の累乗で表すことができ、これは $O(n^3 \log k)$ で計算できる。

※コードが多いので、とりあえずサンプルの G_k を出力して結果を確かめてみることにする。

[Matrix Power Series]

$n \times n$ 行列 A と，正の整数 k, M が与えられる。
行列の累乗和 $S = A + A^2 + \cdots + A^k$

を求め，行列 S の各要素を M で割った余りを求めよ。

$1 \leqq n \leqq 30$
$1 \leqq k \leqq 10^9$
$1 \leqq M \leqq 10^4$

※ $n \times n$ 行列の k 乗は繰り返し二乗法により $O(n^3 \log k)$ で計算できるが，今回の場合，各項求めて足す方法は $O(n^3 k)$ かかってしまう。

$O(n^3 k \log k)$ っぽいがどうもちがうらしい…。

☆ 累乗和の場合，以下の行列を考えるとうまくいく！

$$\left(\begin{array}{c|c} A & 0 \\ \hline I & I \end{array}\right)$$

（ I は $n \times n$ の単位行列，よってこの行列は $2n \times 2n$ ）

このとき，$S_k = I + A + A^2 + \cdots + A^{k-1}$ とすると，

$$\begin{pmatrix} A^k \\ \hline S_k \end{pmatrix} = \left(\begin{array}{c|c} A & 0 \\ \hline I & I \end{array}\right)\begin{pmatrix} A^{k-1} \\ \hline S_{k-1} \end{pmatrix} = \left(\begin{array}{c|c} A & 0 \\ \hline I & I \end{array}\right)^k \begin{pmatrix} I \\ \hline 0 \end{pmatrix}$$

となる。

☆ この行列の k 乗を求めることで A の累乗和の計算が可能となり，$O(n^3 \log k)$ となる。

※実際は $(k+1)$ 乗が必要な気がするが…？
（コードと照らし合わせて考えてみる！）

補 行列累乗の一般化

一般のm項間漸化式の場合，漸化式を

$$a_{n+m} = \sum_{i=0}^{m-1} b_i a_{n+i}$$

とすると，行列を用いて

$$\begin{bmatrix} a_{n+m} \\ a_{n+m-1} \\ \vdots \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} b_{m-1} & \cdots & b_1 & b_0 \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{n+m-1} \\ a_{n+m-2} \\ \vdots \\ a_n \end{bmatrix}$$

と表すことができ，この係数行列をn乗することで，$O(m^2 \log n)$で第n項目の計算が可能となる!!

ただし，漸化式に定数項がある場合は少し複雑で，

$$\begin{bmatrix} a_{n+m} \\ a_{n+m-1} \\ \vdots \\ a_{n+1} \\ 1 \end{bmatrix} = \begin{bmatrix} b_{m-1} & \cdots & b_1 & b_0 & c \\ 1 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n+m-1} \\ a_{n+m-2} \\ \vdots \\ a_n \\ 1 \end{bmatrix}$$

となる。

※配列再利用とセグ木の併せ技もあってかなりキツイ。復習しにいつもとまとめよう…

[データ構造を用いて高速化]

[Minimizing Maximizer]

※何を出力することが求められているのか分かりにくいが、
答えるべきは sorter の部分列の長さの最小値。

☆ $i=1$ に最大値が存在する場合に、ちゃんと
maximizer がその値を出力してくれればよい。

☆ $dp[i][j]$:= i 番目の Sorter までで、最大値が
j 番目に移動するような最小の部分列の長さ
(存在しない場合は INF)

$dp[0][1] = 0$: 0 個の Sorter (= Sorter がなくても) でも、
最大値は 1 番目にいられる (= 初期位置) ので。

$dp[0][j] = \mathrm{INF}$ $(j>1)$: ↑の通り、Sorter なしでは最大値は 1 番目から動けないので、INF。

$$☆\ dp[i+1][j] = \begin{cases} dp[i][j] & (t_i \neq j) \\ \min\left(dp[i][j],\ \min\{dp[i][j'] \mid s_i \leq j' \leq t_i\} + 1\right) & (t_i = j) \end{cases}$$

3-5：水を流して問題を解く"ネットワークフロー"

[最大流]

[最大通信量]

最大流：最大のデータ通信量を達成する f のこと

最大流問題：最大流を求める問題

c：辺の容量

f：辺の流量

s：始点，source

t：終点，sink