数え上げテクニック集

DEGwer(Soh Kumabe)

2017/12/20

目次

1　はじめに

この記事は、Competitive Programming Advent Calender 2017 の 20 日目の記事として書かれました。

競技プログラミングにおいて、数え上げの問題が出題されることは少なくありません。しかし、数え上げは問題特有の考察が必要な分野だとして、汎用的なテクニックの解説があまり充実していない傾向にあります。

確かに問題特有の考察が必要なことは多いのですが、それでも「何を考えればいいか」ということに関しては、いくつかの定跡がある、と言うべき分野でもあります。

そこで本稿では、「解くときにどのようなことを考えればいいのか」ということを中心に、雑多なテクニックの解説も交えながら、数え上げの問題を解くコツについて解説していくことにします。

記事としての想定の難易度としては「プログラミングコンテストチャレンジブック」より少し難しい程度、対象読者層としては青上位〜赤下位程度を想定しています。トピックごとにほぼ独立して読める内容になっているので、好きなところから読んでいただいて構いません。AtCoder Grand Contest などの問題のネタバレを多分に含むので、ネタバレを好まない人は注意して読んでください。

2　状態をまとめる

何かを作っていく方法の個数を数える問題や、グリッドに色を塗っていく問題など、単純な全探索アルゴリズムで終端状態まで到達した場合の数がそのまま数えたい値となる場合は、DP を「全探索の高速化」として見ることができます。つまり、「全探索で訪れるどのような状態を同一視できるか」を見抜くのが鍵となります (逆に、一切の枝刈りのない探索アルゴリズムにおいて、複数の枝が同じ場合に到達することがあるときは、まだ条件の言い換えが不十分だ、と言うこともできます)。

状態をまとめる一般的な方法はおそらくありません。ですからこの項の結論としては、(身も蓋もないですが) 問題特有の考察をすすめるべき、ということになります。しかし、状態をまとめる際に気を付けるべきことはいくつかあります (特に DP を覚えたての人には、なんでもかんでも状態をまとめようとして区別すべき状態までまとめてしまう、という傾向が強くみられます)。なお、次の項で述べますが、場合によっては、「ものを決めて行く順番」を変えることで状態がまとめられるようになることがあります。

状態をまとめられる条件として、次のことを意識すると良いでしょう。

- 遷移先の状態が同じである
- 遷移につく係数が同じである
- 各状態について、そこにまとめられた状態のすべてが問題文の条件を満たすか、すべてが満たさないかのいずれかである

逆に、これらの 3 条件が満たされれば、状態をまとめることができます。

この問題を考えてみましょう。

問題

以下の 3 種類の操作を合計で N 回行ったところ、文字列 S が得られた。操作列としてありうるものは何通りあるか。$N \leq 5000$

- 文字列の末尾に 0 を追加する
- 文字列の末尾に 1 を追加する
- 文字列の末尾の文字を取り除く。もし文字列が空なら、何もしない

(出典: ARC 059 F)

単純な全探索を考えます。まず、操作を行った回数と現在の文字列がともに同じパターンは、明らかにまとめることができます。

現在の文字列の i 文字目までが S と等しく、$i+1$ 文字目が異なるとしましょう。このとき、$i+1$ 文字目は後々取り除かなければなりません。よって、$i+1$ 文字目以降の文字列がどうなっているかは関係なく、よってそれ以降もまとめることができます。

さらに、0 と 1 を区別する必要すらないことが分かります。これは、この問題では同じ文字数の文字列を作る場合の数はすべて等しくなることによります。よって、DP$[n][x]$: n 回の操作を行い、現在の文字列長が x である場合の数 として DP を行うことができます。

次は、この問題を考えてみましょう。

問題

N 頂点のグラフがあり、最初は辺がない。最初、人が頂点 1 にいて、以下を M 回繰り返す。

- 任意の 1 頂点を選び、今いる頂点から辺をはる。新しくはった辺に沿って移動する。

最終的にグラフ全体が強連結になるような移動方法は何通りあるか。$N, M \leq 300$
(出典: Code Festival 2016 Final F)

最終的に全体が強連結になることが条件なので、操作の途中での強連結成分に関する何らかの情報を持っておくと良い、と察しが付きます。グラフを強連結成分分解した結果が同じなら、それらは同じ状態とみなして良い、ということが分かります。

さて、作っている途中のグラフを強連結成分分解し、強連結成分を一点につぶすと、どのようなグラフが得られるでしょうか？　簡単な帰納法で、これは直線グラフに孤立点をいくつか加えたものになることが証明できます。また、人は常に最後の強連結成分にいることがわかります。しかし、まだ各強連結成分のサイズの組は指数個あるので、この問題を解けたことにはなりません。

少なくとも全体は連結でなければならないため、孤立点の個数は持っておく必要があるでしょう。孤立点を除いた全体が強連結になる瞬間は、高橋君が最初の強連結成分に戻ってきた瞬間です。このような (1 回の) 移動の方法は、最初の強連結成分のサイズ通りあります。遷移につく係数を計算する必要があることを考えれば、この値も持っておく必要がありそうです。

逆に、これまでの移動の回数に加えてこれら 2 つの情報だけを持っておけば十分であることも分かります。以上より、DP$[n][x][y]$: n 回移動し、孤立点が x 個あり、最初の強連結成分のサイズが y であるような場合の数 として、この問題を解くことができます。

次の問題のように、半ば強引なまとめ方をする場合もしばしばあります。上に挙げた 3 条件が満たされれば、「非直感的な」まとめ方も許されます。

問題

N 人の人に、1 位から N 位までの重複しない順位を 2 度つける。各 i に対し、2 度目の順位付けでは人 i が i 位だったとする。各 i について、人 i の順位が 1 度目の順位付けから 2 度目の順位付けまでの間に上昇したか、変わらなかったか、下降したかの情報が与えられるので、1 度目の順位付けにおける順位の列としてありうるものの個数を求めよ。$N \leq 200$
(出典: AOJ 2439)

順位が変わらなかった人については考える必要がないので、以下全員の順位が変わったとして考えます。

人 i の 1 度目の順位付けでの順位を A_i としましょう。与えられるものは、各 i に対する、i と A_i の大小関係です。

2 つの順位列を同時に、上から順に見ていきましょう。$A_i > i$ とします ($A_i < i$ の場合については、後で考えることにします)。人 i の 1 度目の順位付けでの順位は i より大きいので、順位列を上から見ていっている関係上、人 i の順位はまだ決めることができません。ひとまず、保留にしておきましょう。

では、1 度目の順位付けで i 位だった人は、どう決めればいいでしょうか？　これには、以下の 2 通りの場合があります。

- その人が、2 度目の順位付けで順位が上がった場合

- その人が、2 度目の順位付けで順位が下がった場合

前者の場合、i 位に置くことのできる人は、先ほど順位を決めるのを保留した人に他なりません。順位を上から見ている関係上、現在順位の決定が保留されている人はすべて対称なので、DP の状態として「保留にした人数」を持っておくことで、遷移の係数を決めることができそうです。

後者の場合、順位を上から見ているので、まだそこに誰かを置くことはできません。とりあえず、i 位は空位にしておくことにしましょう。

さて、$A_i < i$ の場合を考えることにしましょう。人 i を置くことのできる順位は、これまでに空位にした順位に他なりません。よって、DP の状態として、「空位にした個数」を持っておくことで、やはり遷移の係数を決めることができそうです。

以上より、DP$[n][x][y]$: 上から n 位まで見て、x 人の順位決めを保留にし、y 個の順位を空位にしている場合の数 という DP でこの問題を解くことができます。さらに、保留にしている人数と空位にした個数は等しいことが簡単にわかるので、$O(N^2)$ 時間でこの問題を解くことができました。

3　探索順の変更

前項でも述べましたが、DP の状態をまとめやすくするために、ものを決めていく順序を変更するのがしばしば強力です。また、条件を満たす部分集合を数える場合など、決める順番がそもそも定まっていない (自由である) 場合には、その決定順を「恣意的に」決めてやることが重要になります。

ここでは、典型的な決定順を 3 つ紹介します。

3.1　大きい順に並べる

「大きい順」は指定する順序として一番単純ですが、多くの場合に強力です。

問題

N 個の荷物があり、i 個目の重さは w_i である。重量の合計が W 以下であるような荷物の部分集合であって、極大なものは何通りあるか。$N \leq 200, W \leq 10000$
(出典: AOJ 2333)

この問題を考えてみましょう。ナップサック問題として見ると、極大性の条件が厄介です。

「極大」という条件は、「選んでいないもののうち一番軽いものも、追加で選ぶことはできない」という条件に言い換えられます。選んでいない最も軽いものが何か、で場合分けをすることにしましょう。この最小のものを x とします。

荷物を大きい順に並べて、通常のナップサック問題の DP をします。すると、条件は x の前までの荷物のみを使って特定の範囲内の合計重量を達成する、と言い換えられ、この場合の数は DP 配列の情報から簡単に求めることができます。

3.2　順列は挿入 DP

「与えられた条件を満たす順列はいくつあるか」という形式の問題は多くあります。このような場合、DP などで順列を前から決めて行くと、「どの要素を既に使ったか」という情報を持つ必要が出てきてしまいます。この状態数は $2^{\text{要素数}}$ 通りあるので、あまり得策とは言えません。

このようなときに挿入 DP が便利です。与えられた条件において、要素ごとの大小関係のみが重要であるとしましょう。このとき、並べたいものをソートして、(前から順番に順列の要素を決めていくのではなく) 大きい順 (もしくは小さい順) に数を好きな位置に挿入していく、という DP が有用です。

さて、なぜこのような DP が上手くいく場合が多いのでしょうか？　これは、大きい順に値を埋めていけば、「それまでに埋めたものは全て今埋めるものより大きく、これから埋めるものは全て今埋めるものより小さい」という性質が成り立つためです。特に、「今埋めるものより大きいもの」同士の並び順が、今何を埋めるかに特に関係しない場合、この順序変更は非常に強力です。

以下の問題を考えてみましょう。

問題

高さ 1 から N までのビルを一つずつ一列に並べる。左から見ると K 個、右から見ると L 個のビルが見えるような並べ方は何通りあるか。ただし、左/右からあるビルが見えるとは、そのビルより左/右にそれより高いビルがないことを指す。$N \leq 300$

左から順にビルを並べていくのは、上述の理由で得策ではありません。その代わりに、ビルを高い順に配置していくことを考えましょう。

i 番目のビルを置く位置は、一番左、左から 1 番目と 2 番目のビルの間、...、一番右の $i+1$ 通りあります。このうち、一番左にビルを置いた場合は左から見えるビルが 1 つ、一番右にビルを置いた場合は右から見えるビルが 1 つ、それぞれ増加し、それ以外の場合左右から見えるビルの個数は変化しません。以上の考察より、DP$[i][x][y]$: 高いほうから i 個のビルを置いて、左から x 個、右から y 個のビルが見えるような場合の数 とすれば、この DP は $O(N^3)$ 時間で計算可能です。

3.3 区間は終点でソート

区間がいくつか与えられ、そのうちのいくつかを選んで特定の被覆条件を満たすようにする方法は何通りか、という問題はしばしば出題されます。このような問題では、与えられた区間たちを終点座標の昇順に見ていくのが非常に強力です。これには、区間スケジューリング問題が貪欲法で解けることと同様の性質を使っています。

次の問題を考えてみましょう。

問題

$[0, X]$ 内の区間が N 個与えられる。これらの区間たちのうちいくつかを選んで、$[0, X]$ を被覆する方法は何通りあるか。$N \leq 10^5$

区間たちを終点でソートします。すると、DP$[n][x]$: 区間を n 個見て、被覆されていない最初の座標が x であるような場合の数 という DP ができます。区間を終点でソートしていることが、DP のキーがこれらの情報だけで十分である理由です (ソートしない場合、被覆されていない座標をすべて持っておく必要が出てきます)。

また、この DP はデータ構造を用いてさらなる高速化が可能です (この問題に限らず、区間をいくつか選ぶタイプの問題はデータ構造を用いた高速化ができる傾向にあります)。

4　条件の言い換え

問題文で与えられている操作や条件が、とてもそのままの利用には堪えない場合があります。このようなときは、与えられた条件を「数えやすい」条件に適切に言い換えてやる必要があります。

さて、状態のまとめ方と同じく、やはりここでは ad-hoc な考察が必要となります。時には理詰めで言い換えを導き出すことが厳しい場合もあるでしょう。しかしやはり、「最初に思いついた適当な条件を正しいと信じ込む」というのは避けるべきです。それらしい条件を思いついたら証明を試みるか、最低限反例が容易には作れないことを確認してみるのが良いと思います (「○○でなかったらとても解ける問題には見えない」という判断も、時には役に立ちます)。

条件の列挙方法としては、「必要条件を列挙したらそれが十分条件だった」というパターンが多いです。必要条件が思いついたら、それが十分条件になっているかどうか検証してみるのが良いでしょう。

この問題を考えてみましょう。

問題

A 以上 B 以下の整数たちから 1 つ以上選び、それらすべての bitwise or を取る。こうしてできる整数は何通りあるか。$A \leq B \leq 10^{18}$

(出典: AGC 015 D)

A 以上 B 以下の整数の部分集合は 2^{B-A+1} 通りありますが、bitwise or としてありうる値は高々 $O(B)$ 個です。このように、操作の種類数に比べてできるものの種類数が極端に小さいときは、条件の言い換えが有用です。適切に条件を言い換えましょう。

A と B の 2 進表記を上位ビットから順に見ていきましょう。これが一致している間は、bitwise or によってその部分は不変です。よって、($A = B$ の場合を除けば) A の最上位ビットが 0 で、B の最上位ビットが 1 であるとして一般性を失いません。$A < 2^t \leq B$ なる t をとります。

どのような整数が作れるでしょうか？　最上位ビットが 0 の整数については簡単で、整数が A 以上 $2^t - 1$ 以下のとき、またその時に限り、作ることができます。

最上位ビットが 1 の整数をつくるためには、2^t 以上の要素を最低 1 つは使うことになります。もし最上位ビットが 0 の整数も使うなら、$2^t + A$ 以上 $2^{t+1} - 1$ 以下の整数すべて、またそれらのみを作ることができます。

あとは、最上位ビットが 1 の整数のみで作れる整数の条件を挙げればよいです。$2^t + 2^r \leq B < 2^t + 2^{r+1}$ となる r をとれば、2^t 以上 $2^t + 2^{r+1} - 1$ 以下の整数すべてを、またそれらのみを作ることができることが分かります。以上の考察より、作れる整数の条件が 3 つの区間の union として書けたので、この問題を解くことができました。

この問題を考えてみましょう。

問題

0 が N 個と 1 が M 個書かれている。このうち K 個を選んで消し、その平均を新たに書き加えるという操作を繰り返す。
$N+M-1$ が $K-1$ の倍数のとき最後に有理数が 1 つ残る。最後に残る有理数としてありうるものは何通りあるか。$N, M, K \leq 2000$
(出典: AGC 009 E)

作る有理数を K 進法で表記することを考えます。

各桁の和を考えてみましょう。1 を M 個と 0 を N 個使って作った数の各桁の和は M 以下であることが M に関する帰納法で分かります。また、0,1 を反転させて考えれば、作ることのできる有理数を 1 から引いた値の各桁の和は N 以下であることも同様に分かります。

さらに、平均を取る操作を和を取る操作と K で割る操作に分解して見ることにすれば、黒板に書かれている整数の各桁の和の合計値は操作によって $K-1$ の倍数だけ減少することが分かるので、作ることのできる整数の各桁の和は $mod(K-1)$ で M に等しいです。

逆に、これらの 3 条件を満たす有理数は全て作ることができることが証明できます。実際、0 だけ、1 だけで平均を取る操作を繰り返せば各桁の和がちょうど M である場合に帰着でき、そのときの構成は (K 進表記を考えれば) 容易です。

以上より、作ることのできる有理数の条件が 3 つの条件を使って言い換えられました。これらは「数えやすい」条件です。実際、DP$[n][k]$: K 進 n 桁まで見たときの各桁の和が k であるような場合の数 とすれば、必要なすべての情報を計算可能です。

5　greedy からの帰着

「○○の操作で作られる可能性のある列の個数を求めよ」という問題は多くあります。この手の問題の難しいところは、複数の操作列が同じものを生成することがあることです。(同じものを生成しないのであれば、「状態をまとめる」の項で述べたように自然に DP をすると簡単な場合が多いです)

しかし逆に言えば、できた列から操作列を一意に定めるような方法があれば、(前の項で述べたような、できた列の特徴づけを経由せずとも) そのような操作列のほうを数えても良い、ということになります。そして、この対応付けは、しばしば greedy によるものになります。

「その列が作れる列であるかどうか」という判定問題を考えてみましょう。もしこの問題を $O(1)$ 個のパラメータのみを保持し、列を前から見ていくことで解くことができるなら、そのパラメータの値をキーにして DP を行うことで、判定問題の解法を機械的に数え上げ問題の解法に変換することができます。

次の問題を考えてみましょう。

問題

1 以上 N 以下の整数からなる順列であって、単調増加な 2 つの列のマージとして書けるものはいくつあるか。ただし、列 X が 2 つの列 A, B のマージであるとは、 X の要素すべてに A, B のラベルをつけ、A のラベルのついた要素だけを順番を保って取り出すと列 A に一致し、B のラベルのついた要素だけを順番を保って取り出すと列 B に一致するようにできることを指す。$N \leq 2000$

条件を満たす順列を 2 つの列のマージとして書く方法は、一般には 1 通りではありません。ここでは、条件を満たす順列に対し、それを 2 つの列のマージとして書く方法をちょうど 1 通り定めることを考えてみましょう。そのために、与えられた順列が 2 つの列のマージとして書けるか、という判定問題を考えます。

前から k 要素を見たとします。このときの列を 2 つの単調増加な列に分解する方法を一意に定めます。2 つの列を A, B とし、A の最後の要素が前 k 要素の最大値となるようにすれば、(今後のことを考えれば) B の最後の要素が最小となるような分解を取っておくのがよいことが分かります。つまりは、各要素を見るとき、

- その要素が A の最後の要素より大きければ、それを A の末尾に追加
- そうでなく、その要素が B の最後の要素より大きければ、それを B の末尾に追加
- そうでなければ、分割は不可能なので終了する

とすれば、判定問題版を解くことができます。

これを機械的に、数え上げの解法に変換しましょう。列を前から順に作っていくことを考えます。

新たな要素を列の末尾に加える際、掛ける係数を決めるものは何でしょうか？　これは、列 B の末尾の要素が、先頭 k 個の要素の中で何番目か、という情報です。この情報が、判定問題の解法における「greedy のパラメータ」になります。よって、DP$[k][x]$: 前 k 要素を決めて、 B の列の末尾の要素が前 k 要素の中で大きいほうから x 番目であるような、条件を満たす列の個数 とすれば、この問題を解くことができます。

次の問題を考えてみましょう。

問題

N 頂点の有向グラフがあり、最初辺はない。まだ辺がない 2 点の順序対の間に辺をはる操作を $N(N-1)$ 回行い、その都度強連結成分の個数を書いていく。最後にできる列は何通りあるか。$N \leq 50$
(出典: Open Cup GP of Romania I)

強連結成分の個数の列が与えられたとき、その列を生成するような辺のはり方が存在するか？　という問題を考えてみましょう。強連結成分の個数は各ステップで減るか、変わらないかのいずれかであり、減る場合は A から B へのパスが存在するような辺 (B, A) を、減らない場合はそのようなパスが存在しないような辺 (B, A) を、それぞれはることになります。

さて、証明は省きますが、この判定は次のような greedy で行うことができます。この greedy の正当性は、操作途中で、「孤立点がいくつかと、まっすぐにいくつか連なった強連結成分たち (すべての i について、i 番目の強連結成分から $i+1$ 番目の強連結成分に辺が伸びているようなもの)」というグラフのみを考えればよい、という結果によります。

- 強連結成分の個数が $k(>0)$ だけ減る場合、連なった強連結成分の $k+1$ 個目の頂点から 1 個目の頂点に辺をはる。それが不可能なら、この列は作れない。
- 強連結成分の個数が減らず、まだ孤立点がある場合、連なった最後の強連結成分の頂点から、孤立点に辺をはる。
- そうでなく、まだ同じ強連結成分の点同士を結ぶ辺であってはられていないものが存在するなら、その頂点間に辺をはる。
- そうでない場合、この列は作れない。

この greedy を機械的に数え上げの解法へと変換しましょう。上記の操作によって、連なった強連結成分のうち最初のもの以外のサイズは、常に 1 で不変です。最初の強連結成分のサイズは孤立点の個数と強連結成分の個数から計算できるので、DP$[n][a][b]$: n 本辺をはり、連なった強連結成分が a 個、孤立点が b 個あるときの列の個数 として、$O(N^5)$ 時間でこの問題を解くことができます。

6　場合分けのテクニック

数え上げにおいて、何かのパラメータの値で場合分けをしたくなることがよくあります。このとき、全ての場合が disjoint になるように注意する必要があります。(disjoint でないと、同じ場合を 2 度以上数えてしまうことになります。複数回数えてしまったものを後々補正するのは、包除原理が使えるような特別な場合を除き、簡単なことではありません。)

つまり、パラメータとしては、「数えたいものから一意的に定まる量」を選ぶ必要があります。(余談になりますが、最適化タイプの問題の場合、重複があっても構わないという性質がしばしば重要になります。これが数え上げと最適化の一つの大きな違いになります)。ここでは、次のような問題を例にとって考えてみることにしましょう。

問題

N 個の相異なる整数 $x_1, ..., x_N$ が与えられる。A さんと B さんは、この中からそれぞれ整数をいくつか選ぶ。ただし、A さんは、B さんの選んだどの整数よりも小さい整数のみを選ぶことができる。A さんの選んだ整数の合計と B さんの選んだ整数の合計が等しくなるような選び方は何通りあるか。
$N \leq 100, |x_i| \leq 100$

A さんと B さんの選ぶ整数の、大きさの意味での「境目」を固定すれば、通常のナップサック問題の DP を用いて解くことができそうです。つまり、整数 X を固定し、A さんは X 以下の、B さんは X より大きい整数だけを選ぶことができる、という条件をつければ、これは数え上げの有名問題に帰着されます。(前後から DP をすることを考えれば、一度の DP 列の計算で、全ての X に対して必要な情報を計算することができます。)

しかし、ここで注意すべきことがあります。A さんが $1, 2, 3, 4$ を選び、B さんが 10 を選ぶような場合、上の X としては $4, 5, 6, 7, 8, 9$ の 6 通りが考えられます。すなわち、同じものを 6 回も数えてしまうことになります。

これを回避するためには、「A さんが必ず X を使う」という条件を追加すればよいです。この条件は、DP の状態に「今の整数を使ったかどうか」というフラグを加えてやることで追加可能です。

次は、この問題を考えてみましょう。

問題

赤と青の積み木が合計で N 個箱に入っている。以下の一連の操作を M 回繰り返してできる列としてありうるものは何通りあるか。

- 箱から積み木を取り出し、現在の列の末尾に並べる
- 赤と青の積み木を 1 つずつ箱に入れる
- 箱から積み木を取り出し、現在の列の末尾に並べる

(出典: AGC 013 D)

DP[x][y]: x ターン経過し、赤玉が現在 y 個残っているときの列の個数 という DP をしたくなりますが、これでは同じ列を 2 度以上数えてしまいます。できた列から、一意的に定まるパラメータは何でしょうか？

袋の中の赤の玉が最も少なくなったタイミングを、赤玉が 0 個になったタイミングとするように、最初の玉

の個数を調整することを考えます。このとき、できた列から、最初の赤玉の個数が一意に定まるので、重複なく数え上げることができます。

最初の赤玉の個数が k 個の場合を考えてみましょう。このとき、DP[n][x][f(bool 値)]: f が true なら、操作を n 回行い、袋の中の赤玉の個数が x 個で、一度でも袋の中の赤玉を 0 個にしたことがある場合の数、f が false なら同様の条件で赤玉を 0 個にしたことがない場合の数 とすれば、$O(N^2)$ 時間の DP を行うことができます (f を持たないと、異なる k に対して同じ場合を数えてしまいます)。

さて、この DP の遷移は最初の赤玉の個数によらないので、全ての k についてまとめて DP を行うことができます。時間計算量は $O(N^2)$ です。

7　線形和への分解

数え上げの問題では、「全ての○○に対して△△の値を計算し、その合計値を求めよ」という問われ方がなされることがあります。「△△の期待値を求めよ」というのも、その一種です。

ここでは、「求めたいものを別の和の形式で書く」、というテクニックがしばしば有用となります。具体例として、次のような問題を考えてみることにしましょう。

問題

N 個の整数 $x_1, ..., x_N$ が与えられる。この中から K 個選ぶ方法すべてに対する、選んだ数の中央値の合計を求めよ。$N \leq 10^5, K$ は偶数 (ただし、偶数個の数の中央値とは、その大きさの意味での中央 2 つの平均のことを指す)

$\binom{N}{K}$ 通りをすべて試していては、当然間に合いません。さらに、中央値はある 2 整数の平均なので、中央値を全通り試しても、やはり間に合いません。

中央値の式を「分解」することを考えてみましょう。$a_1, ..., a_{K/2}, a_{K/2+1}, ..., a_K$ の中央値とは、$a_{K/2} + a_{K/2+1}$ を 2 で割った値です。この $a_{K/2}$ と $a_{K/2+1}$ を別々に足してやることを考えます。

$a_{K/2} = X$ となるような整数の選び方は、簡単な combination の式を用いて数えることができます。$a_{K/2+1} = X$ なる選び方についても同様です。両者を適切な係数で足し合わせれば、求めるべき合計値を得ることができます。

さて、ビット演算が題材の場合、桁ごとに見る手法が非常に有用です。次の問題を考えてみましょう。

問題

$0 \leq x \leq N, 0 \leq y \leq M$ を満たすすべての (x, y) の組について、x xor y の総和を求めよ。ただし、x xor y で x, y の bitwise xor を表す。$N, M \leq 10^9$

N, M が大きいため、xor の値を全列挙することも許されません。桁ごとに分解して見ていくことを考えましょう。

x xor y を 2 進表記したとき、2^k の位が 1 になるのは何通りでしょうか？　$f(X, k)$ で 0 以上 X 以下の整数のうち 2^k の位が 1 であるようなものの個数を表すことにすれば、これは $f(N, k)(M + 1 - f(M, k)) + (N + 1 - f(N, k))f(M, k)$ 通りです。各 k に対し、$f(X, k)$ の値は簡単に求められるので、求める総和も簡単に求められます。

次は、この問題を考えてみましょう。

問題

N 頂点の木が与えられる。K 点からなる頂点部分集合 $\binom{N}{K}$ 通りに対し、その頂点集合を含む最小の部分木のサイズを求めてすべて足し合わせた値を、$K = 1, 2, ..., N$ についてそれぞれ求めよ。$N \leq 2 \times 10^5$ (出典: AGC 005 F)

ここでは、K を固定した場合の問題を解くことにします (AGC の問題文での「easy problem」です。満点解法については、「高速化のテクニック」の項で述べます)。

$\binom{N}{K}$ 通り試していては間に合いません。選んだ K 点の集合を S とします。ある頂点 v が S を含む最小の部分木に含まれないとき、S の満たすべき条件は、v を取り除いたときの連結成分のうちの 1 つに S の頂点

がすべて含まれていることです。このような場合の数は combination の計算で求めることができるので、K を固定した場合のこの問題を解くことができました。

この解法では、「$\binom{N}{K}$ 通りに対する和」を「各頂点が含まれる場合の数の和」に言い換えることが重要となりました。このような分解が有用なことは多いです、

さて、分解の際に、「足すべきものが線形に分解できる」ということが重要となります。もし求めるべきものが何かの 2 乗和なら、このような単純な分解は上手くいきません。しかしこのような場合も、問題によっては $(x_1 + ... + x_N)^2 = (x_1^2 + ... + x_N^2) + 2(x_1x_2 + x_1x_3 + ... + x_{N-1}x_N)$ と分解し、2 項の和を別々に求めてやることで解けることがあります。

8　部分群のテクニック

「○○の操作を用いることでできる状態は何通りあるか」という形式の問題は多くあります。もしこの操作が可逆かつ全域的 (操作ができるかどうかが、現在の状態に依存しない) なら、次の定理を用いたテクニックが有用です。

- Lagrange の定理: G を有限群、H を G の部分群とする。このとき $|H|$ は $|G|$ の約数である。

ルービックキューブを解くことを考えてみましょう。ルービックキューブをバラバラにして、そのあと組みなおして作れるような状態全体の集合を G、完成状態から正規の操作を繰り返してできる状態の集合を H とします。H は G の部分集合ですが、このとき $|H|$ は $|G|$ の約数となります。

このような問題を解くとき、はじめに小さい場合で実験をしてみるのが有用です。$|G|$ は得てして簡単に求まり、また小さい場合には $|H|$ を全探索などで求めることができます。$|G|/|H|$ の値を k とすれば、作れる状態を特徴づける量が、ちょうど掛け合わせて k 個分あるという予測を立てることができます。この予測は、実際に操作による不変量を見つけるのに大いに役に立ちます。k が偶数なら、総和や置換など、何らかの偶奇性が効いてくると考えることができるでしょう。

例えばこの問題を、このテクニックを用いることで解くことができます。

問題

$H \times W$ のグリッドに文字が書かれている。以下の操作を繰り返してできる盤面は何通りあるか。

- ある行を選び、左右反転する
- ある列を選び、上下反転する

$H, W \leq 200$
(出典: Code Festival 2016 Final I)

反転操作が可逆かつ全域的なので、作れる盤面全体はありうる盤面全体の部分群として見ることができます。マス (i, j) に書かれた文字はマス $(H+1-i, j), (i, W+1-j), (H+1-i, W+1-j)$ にしか移らないことなどを考慮しながら全探索で実験を行えば、性質が自然に見えてきます。

また、形こそ数え上げではないですが、同様のテクニックでこの問題を解くことも可能です。

問題

$3 \times N$ のグリッドがあり、i 行 j 列には整数 $3(j-1)+i$ が書かれている (すなわち、左から順に $(1,2,3), (4,5,6), ..., (3N-2, 3N-1, 3N)$ が上から書かれている)。
3×3 の部分グリッドを選んで 180 度回転させる操作を繰り返し、与えられたグリッドを作ることができるか判定せよ。
(出典: AGC 006 E)

この問題においても、もとの状態から作れる盤面の個数を N の小さい範囲で実験によって数えると、「全体」の 4 つに 1 つの盤面のみが作れることがわかります。実際、偶奇の不変量が 2 種類見つかるので、この問題を解くことができます。

9　再帰的な定義の利用

フラクタルなどの再帰的に定義されたものの上で何かを数え上げるタイプの問題が稀に出題されることがあります。そして再帰的な定義は DP との相性が良いため、「そのまま」DP をすることができます。

このとき、「どのような状態が必要か」を見極める必要があります。レベル k のフラクタルが、レベル $k-1$ のフラクタルをいくつか並べるなどの適当な操作で作れるもの、と定義されているとします。このとき、レベル k のフラクタルでの求めたい値を求めるためには、レベル $k-1$ のフラクタルにおいてどのような値が求まっているべきか、ということを考える必要があります。さらに、その「追加で求めておくべき値」のほうも、別の方法で求められるものを選ぶ必要があります。

次の問題を考えてみましょう。

問題

レベル k のシェルピンスキーのガスケット (以下ガスケットと呼ぶ) を、以下のように再帰的に定義する。

- レベル 0 のガスケットは、全体が黒の一辺 1 の正三角形
- レベル $k+1$ のガスケットは、レベル k のガスケットの黒の (一辺 2^{-k} の) 正三角形すべてに対し、各辺の中点を結ぶことで正三角形を 4 つに分け、真ん中を除く 3 つを黒で、真ん中を白で塗ったもの

レベル L のガスケットの黒い部分と白い部分の境界を辿ってできる閉路のうち、同じ点を二度通らないもの (単純閉路と呼ぶ) は何通りあるか。$L \leq 10^5$

(出典: ARC 037 D)

レベル k のガスケットは、レベル $k-1$ のガスケット 3 つを並べたものと見ることができます。レベル k のガスケットに存在する単純閉路の個数を A_k とします。レベル $k-1$ のガスケットにおけるどのような値が求まっていれば、A_k の値を求めることができるでしょうか？

レベル k のガスケットをレベル $k-1$ のガスケットを並べたものとして考えたときに、単純閉路はどのような現れ方をするか、ということを考えます。レベル k のガスケットの単純閉路は、次の 2 通りに分けられます。

- ガスケットを構成する 3 つのレベル $k-1$ のガスケットのうちのどれかに、完全に含まれるようなもの
- ガスケットを構成する 3 つのレベル $k-1$ のガスケットすべてにわたるもの

前者の個数は $3A_{k-1}$ に等しいので、後者を考えましょう。レベル $k-1$ のガスケットにおいて、レベル k のガスケットの単純閉路との共通部分は単純パスとして現れます。すなわち、レベル k のガスケットの (全体を大きな三角形として見たときの) 2 頂点を結ぶような単純なパスの個数を B_k とすれば、求める値は B_{k-1}^3 となります。

ひとます、A_k を A_{k-1} と B_{k-1} を用いて表せました。しかし、まだ問題が残っています。今度は B_k を求める方法を作らなければなりません。

レベル k のガスケットの 2 頂点を結ぶような単純なパスには、以下の 2 通りがあります。

- ガスケットを構成する 3 つのレベル $k-1$ のガスケットのうち、2 つにわたるもの

- ガスケットを構成する 3 つのレベル $k-1$ のガスケットのうち、3 つにわたるもの

前者は B_{k-1}^2 通りあります。後者の場合の数は、B_{k-1}^3 通りから、同じ場所を 2 回通るような場合の数を引いた値です。これは、レベル k のガスケットにおける 2 頂点を結ぶパスであって、残りの 1 頂点を通るものの個数を C_k とおけば、$B_{k-1}^3 - B_{k-1}C_{k-1}^2$ 通りとなります。

今度は、C_k を求める方法を考える必要があります。B_k の場合とほぼ同様の議論により、C_k は $B_{k-1}^2 C_{k-1} - C_{k-1}^3$ に等しいことが分かります。$A_{k-1}, B_{k-1}, C_{k-1}$ を用いれば、A_k, B_k, C_k を求めることができることが分かり、「状態の遷移式が自己完結」したので、A_k, B_k, C_k の値を k を増やしながら順に求めていくことができるようになり、この問題を解くことができました。

10　桁 DP について

桁 DP を用いる問題の解法は、本質的には 1 種類しかありません。すなわち、一度やり方を覚えてしまえば、この分野の問題のほとんどがほぼ同じ方法で解けることになります。

桁 DP を用いる問題の問われ方を、まず確認しましょう。大抵の問題は、「○○を満たす正整数のうち、X 以下のものは何通りあるか」という形をしています (「X 以上 Y 以下のものは何通りあるか」という問われ方をすることもありますが、この場合は Y 以下の場合の数と $X-1$ 以下の場合の数を個別に求めて引き算をしてやることで対処可能です)。

次の問題を考えてみましょう。

問題

A 以上 B 以下の整数のうち、M の倍数であり、10 進表記を上の桁から順に見ていったとき、前の桁から増加する項と減少する項が交互に現れるようなものは何通りあるか。$A \leq B \leq 10^500, M \leq 500$
(出典: AOJ 0570)

上述の通り、問題の前半を「X 以下の整数のうち」と読み替えた問題を解くことにしましょう。上の桁から順に決めていくことで、DP$[n][x][r][f][g]$ を、以下の条件を満たすような場合の数とした DP ができます。

- (X の一番上の桁からはじめて) 上から n 桁見た
- 決めた中で一番下の位が x
- 現在の整数を $mod\ M$ で見ると r
- f が 0 のとき決めた中で一番下の位は二番目の位より大きい、1 のとき小さい、2 のときまだ 1 桁しか決めていない
- g が 0 のとき現在の整数が X の上 n 桁より小さい、1 のとき等しい

この DP は $O(M \times (X$ の桁数$))$ 時間で計算可能です。

さて、桁 DP に特有なのは、この最後のフラグ g でしょう。このフラグを持たせることで、作った整数が X 以下かどうかを判定することができます。また、今回は桁を上から埋めていきましたが、下の桁から埋めていくことも可能です。両者の違いとして、

- 下から決めると上記の g の値が (大きい、等しい、小さい、の) 3 通りになるが、上から決めると 2 通りで済む
- leading-zero があるとき、上から決めると桁ごとにその桁で始まる整数に関する追加の更新を行わなければならないが、下から決めるとそれが不要
- 桁を埋める操作が、上から決めると 10 倍して何かを足す操作になるが、下から決めると 10 の何乗かをいくつか足す操作になる

などが挙げられます。場合に応じて実装が楽になるほうを選ぶとよいでしょう。

さて、次のような問題も、桁 DP を用いて解くことができます。

問題

硬貨が N 種類あり、i 種類目の硬貨の額面は A_i 円である。各硬貨が無限にあるとき、X 円ちょうどの支払い方は何通りあるか。ただし、2 つの支払い方が異なるとは、ある硬貨を出す枚数が異なることを指す。$N \leq 20, A_i \leq 30, X \leq 10^{18}$

i 種類目の硬貨を使う枚数を 2 進表記し、すべての i について同時に、下の桁から決めていくことを考えましょう。

2^n の桁を決めるとき、各硬貨は 2^n 枚追加で使うか、まったく追加で使わないかのいずれかです。すべての x に対し、合計で $x \times 2^n$ 円を使うような場合の数は全探索または部分和問題の DP で簡単に前計算しておくことが可能です。

さて、合計をちょうど X 円にするためにここで満たしておくべきことは、X の 2^n の位と合計額の 2^n の位を合わせておくことだけです。上記の方法で硬貨の使い方を決めれば、現在の合計額には繰り上がりが発生するでしょう。この繰り上がりの大きさ (を 2^n で割ったもの) を DP の状態として持つことにすれば、DP$[n][c]$: 下から n 桁を決め、繰り上がりが c であるような場合の数 として DP を行うことができます。繰り上がりの大きさとしては、無限級数の和を考えれば、高々 $2\sum A_i$ 程度までを考えればいいことがわかるので、$O((\sum A_i)^2 logX)$ 時間でこの問題を解くことができました。

さて、桁 DP は得てして高次元の DP になるので、慣れないうちは実装に戸惑うことが多いでしょう。「今決めている桁は何か」ということを強く意識しながら実装すると、混乱を幾分和らげることができます。

11　高速化のテクニック

数え上げの問題の高速化に使えるテクニックは、そう多くありません。それは、数え上げの「全ての場合を網羅しなければならない」という性質によります。

最適化の問題では、「○○の性質を満たさないところに最適解はないため、○○を満たすところのみ考えればよい」という手法が使えます。Convex Hull Trick などの高速化手法も、突き詰めればこの性質に依存している、と言うこともできます。

一方、数え上げの場合、いかに「回りくどい」場合でも、条件を満たしていればそれは数えなければなりません。ですから、状態数や遷移の削減が困難になります。

以下に挙げる高速化の手法は、数え上げ以外にも使える手法が多いです。本稿は数え上げのテクニック集なので、個別のテクニックに関して (数え上げ以外の面から) 詳しく解説することはしません。以下にテクニックを挙げます。

11.1　累積和の利用

DP の漸化式を眺めると、累積和が使える形であることが多くあります。やることは普段の累積和と同じなので特に目新しいことはないですが、DP の中で累積和を使うとしばしば添え字の管理が難しくなります。適切に番兵を入れるなどして、すっきりと書けるように心がけましょう。

11.2　データ構造の利用

適当な mod で割ったあまりを求めることの多い数え上げにおいて、区間の最小値などといった値は特に意味を持ちません。ですから、ここでのデータ構造はほとんどの場合「区間の和を求める」という操作のために使われます。

BIT を用いれば十分な場合が多いですが、遅延更新の必要な場合や、「この区間の値をすべて 2 倍する」のようなクエリを処理する必要のある場合も少数ながらあります。いずれにせよ、どのようなクエリに答える必要があるのかは DP の遷移式をきちんと書かないと分からないので、しっかりと遷移を詰めてからデータ構造を作り始めるようにしましょう。

11.3　配列の使いまわし

DP$[x][y]$: 列の x 番目まで見たときに、○○を満たす最後のものが y 番目である場合の数 のような形の DP は、同じ配列を使いまわすことによって高速化できる場合があります。特に、区間の集合からいくつかを選ぶ問題などは、この方法で高速化を行うことができる場合が多いです。

この問題を考えてみましょう。

問題

整数 N 個からなる集合が与えられる。以下の条件を満たすように、この集合を 2 つの集合 X, Y に分割する方法は何通りあるか。

- X のどの 2 要素の差の絶対値も A 以下である
- Y のどの 2 要素の差の絶対値も B 以下である

(出典: AGC 009 C)

$A \leq B$ として一般性を失いません。DP$[x][y]$: 小さいほうから x 個の数を振り分けたときに、最後に Y に振り分けた数が小さいほうから y 番目である場合の数 という DP ができます。

さて、DP$[x-1][y]$ と DP$[x][y]$ の関係を見ていきましょう。$x \neq y$ に対し、DP$[x][y]$ は DP$[x-1][y]$ と等しいか、または 0 であることがわかります (最後に Y に振り分けた値が更新されないなら、その値は X に振り分けるしかないため)。つまり、$x-1$ 番目の DP 配列から x 番目の DP 配列を計算するときにおこる更新は、

- DP$[x][x]$ が更新される
- DP$[x-1][y]$ が 0 でなかったところが DP$[x][y]$ では 0 になる

しかなく、これは全体で $O(N)$ 回です。尺取り法のように 0 でない値の区間を管理しながら同じ配列を使いまわして DP を行うことで、時間計算量 $O(N)$ を達成できます。

11.4 高速フーリエ変換

計算したいものを見ると、畳み込みの形になっていることがたまにあります。combination の計算式を展開すると現れる場合や、DP の遷移式がこの形になっている場合があります。ここでは、前者の場合を扱うことにします。

列 A と列 B に対し、$C_i = \sum_{0 \leq j \leq i} f(j, i-j) A_j B_{i-j}$ で定まる列 C を求めたいとしましょう。ただし、f は適当な関数とし、添え字の範囲外の A, B の値は 0 であるとします。

もし f が全域で 1 なら、これは FFT そのものです。そうでない場合も、f の表示が $f(x,y) = p(x)q(y)r(x+y)$ と分解できる場合には、i 番目の要素を $A_i p(i)$ として作った列 A' と i 番目の要素を $B_i q(i)$ として作った列 B' の畳み込み C' を計算した後、$C_i = \frac{C'_i}{r(i)}$ として列 C を計算できます。

「線形和への分解」の章で扱ったこの問題の続きを考えてみましょう。

問題 (再掲)

N 頂点の木が与えられる。K 点からなる頂点部分集合 $\binom{N}{K}$ 通りに対し、その頂点集合を含む最小の部分木のサイズを求めてすべて足し合わせた値を、$K = 1, 2, ..., N$ についてそれぞれ求めよ。$N \leq 2 \times 10^5$
(出典: AGC 005 F)

考察を進めると、結局求めたいものは、各 $1, ..., N$ に対して、$\sum a_i \binom{i}{K} = \sum a_i \frac{i!}{(i-K)!K!}$ であることがわかります。これを combination の係数のついた上記の形式の畳み込みと見ることを考えましょう。

上記の A_i, B_i を $a_i, 1$ とし、$p(x), q(y), r(z)$ をそれぞれ $a_x x!, \frac{1}{(N-y)!}, \frac{1}{(z-N)!}$ として上記の畳み込みを行えば、求める N 個の値は $C_{N+1}, ..., C_{2N}$ となるため、$O(NlogN)$ 時間でこの問題を解くことができます。

さて、理論的にはこれでこの問題が解けたことになりますが、FFT の誤差の問題が残っています。一般に、数え上げの問題で出てくる 9 桁程度の整数の畳み込みの結果として出てくる 14 桁程度の整数において、FFT の計算誤差は 1 を超えるので、特に気を遣わずに FFT をすると答えがずれることになります。

近年では、FFT が想定解の数え上げの問題では、*mod* が小さいか、あるいは 998244353 などの (10^9+7 でない) 特殊な *mod* が使われることが多いです。それぞれ、小さい数の演算では FFT の誤差が十分小さくなることを利用して単純な FFT で解を求めることを許す、位数が 2 べきの元が存在するような *mod* を選ぶことで NTT(Number Theoretic Transform) による計算を可能にする、という役割があります。

11.5 高速ゼータ変換

集合 X のすべての部分集合 A に対して、値 x_A が与えられるとします。このとき、全ての $A \subset X$ に対して、$\sum_{Y \subset A} x_Y$ の値を高速に求める手法が知られています。これを高速ゼータ変換と呼びます。

愚直な実装では $O(4^{|X|})$ や $O(3^{|X|})$ 時間かかってしまいますが、このアルゴリズムは $O(2^{|X|}|X|)$ 時間で上のすべての値を計算します。

発想はいたってシンプルです。DP[n][p]: p の表す集合に包含される集合 Y であって、後ろの $|X|-n$ 要素を含むかどうかは p の表す集合と一致するものについての x_Y の和 として、n の小さいほうから順に計算していけばよいです。

上記の値を求めたくなることはしばしばあるので、「このような値を求めることは可能である」ということを覚えておくとよいでしょう。実装も非常に単純です。また、今回は部分集合の値の総和を求める手法を紹介しましたが、全く同様のアルゴリズムを用いることで、部分集合の値の最小値など結合律と交換律を満たす任意の値を計算可能です。

11.6 And と Add の畳み込み

高速フーリエ変換は、C_{i+j} に A_iB_j の和を足しこむアルゴリズムです。では、足しこむ先が C_{i+j} ではなく、$C_{i\&j}$ の場合はどうなるでしょうか？　(ただし、$i\&j$ で i,j の bitwise and を表します)

実は、高速フーリエ変換と類似のアルゴリズムを用いて、この畳み込みを高速に実装することが可能です。長さ 2^N の列どうしの畳み込みを、長さ 2^{N-1} の列どうしの畳み込み 2 回に帰着する方法を考えてみましょう。

畳みこみたい 2 つの列をそれぞれ AB, CD としましょう。ただし、A, B, C, D は長さ 2^{N-1} の列で、AB や CD は列の連結を表します。

列どうしの和を、列をベクトルとして見たときの和としましょう。このとき、$A+B$ と $C+D$ の畳み込みでできる列 X と、B と D の畳み込みでできる列 Y が求まれば、求めたい畳み込みは $(X-Y)Y$ と書けることが分かります (B と D が、最上位ビットが 1 の要素についての列を表すことに注意しましょう)。X や Y は再帰的に求めることができるので、長さ 2^N の列どうしの畳み込みは、$O(2^N N)$ 時間でできることが分かりました。

ビット列に対して値が求まっている DP の遷移を高速化したい場合、この形になっているかどうかを考えてみると良いでしょう。また、この畳み込みを二分累乗の 1 ステップとして累乗を行う問題も出題されたことがあります。

11.7　簡単な枝刈り

配る DP を考えます。このとき、DP 配列の見ている場所の値が 0 なら遷移をしない、という単純な枝刈りが、しばしば (特に高次元の DP における) 定数倍高速化において威力を発揮します。

問題によっては、あるパラメータの値によって他のパラメータの値が大きく制約を受けることがあります。制約条件を求め、ループを回す範囲を手動で決定しても良いのですが、単に「0 通りなら遷移しない」という一文を書くだけで、詳細を詰めることなしに定数倍高速化をすることができます。

12　行列を用いたテクニック

数え上げの問題では、行列を用いたテクニックが使われることがあります。ただし、線形代数の高度な知識を要求されることはめったになく、ほとんどの場合は行列の掛け算の方法と行列式の求め方のみ分かっていれば十分です。本項では、このテクニックについて解説していきます。

12.1　二分累乗

パラメータが 2 つあり、片方が 10^9 などの大きな値、もう片方が 100 などの小さな値で、大きな値の方の「各段階」の対称性が高い場合、行列累乗を疑ってみるのが良いでしょう。DP$[n][x]$ が DP$[n-1][y]$ たちのみの線形結合として書け、x の取りうる値が比較的少ない場合に、良く用いられるテクニックです。また、行列累乗に代表されるような、二分累乗のアルゴリズムを応用したテクニックもいくつか解説します。

12.1.1　行列の導出

行列累乗が想定解の簡単な問題は、普通の DP の遷移を陽に行列乗算として表すだけで解くことができます。しかし、その行列の導出が難しい場合もあります。

次の問題を考えてみましょう。

問題

長さ N の区間を、左端からの長さが整数である位置で区切り、任意の個数の区間に分割する。区間への分割のスコアとは、分割でできた各区間の長さの 2 乗の積である。
区切りの位置として使えない座標が M 個指定されるので、すべての区間の分割の方法に対する、スコアの合計を求めよ。$N \leq 10^9, M \leq 10^5$
(出典: AGC 013 E)

まずは、$M = 0$ の場合を考えてみましょう。DP$[x]$: $N = x$ のときの答え とすれば、(DP[0]$= 1$ として) DP$[x] = \sum_{y=1}^{x} y^2$DP$[x-y]$ となります。

N が大きいので、このままでは間に合いません。この漸化式を、行列累乗の形に変換することを考えます。

行列累乗では、DP$[x][y]$ の値を DP$[x-1][z]$ の値たちの線形結合で表したいので、この DP の遷移式が直前の項のみに依存するようにすることを目的にして式変形を行っていきます。「線形和への分解」の項の最後に述べたのと同様、コスト関数が 2 次のときはそのコストを $0, 1, 2$ 次の項に分ける、という変形が有用なことが多いです。また、今回に限った話ではないですが、闇雲に式を変形するのではなく、行列累乗の形に書けるように目的をもって式変形をしていくことが重要になります。

まず、

$$
\begin{aligned}
\mathrm{DP}[x] &= \sum_{y=1}^{x} y^2 \mathrm{DP}[x-y] \\
&= \mathrm{DP}[x-1] + \sum_{y=1}^{x-1} (y+1)^2 \mathrm{DP}[x-1-y] \\
&= \mathrm{DP}[x-1] + \sum_{y=1}^{x-1} y^2 \mathrm{DP}[x-1-y] + \sum_{y=1}^{x-1} (2y+1) \mathrm{DP}[x-1-y] \\
&= 2\mathrm{DP}[x-1] + 2\sum_{y=1}^{x-1} y \mathrm{DP}[x-1-y] + \sum_{y=1}^{x-1} \mathrm{DP}[x-1-y]
\end{aligned}
$$

と変形すれば、総和の中の y の次数をひとつ下げることができました。さらに次数を下げていきます。$\mathrm{DP}'[x]=\sum_{y=1}^{x} y\mathrm{DP}[x-y]$ とおき、さらに $\mathrm{DP}''[x]=\sum_{y=1}^{x} \mathrm{DP}[x-y]$ とおくと、

$$
\begin{aligned}
\mathrm{DP}'[x] &= \sum_{y=1}^{x} y \mathrm{DP}[x-y] \\
&= \mathrm{DP}[x-1] + \sum_{y=1}^{x-1} (y+1) \mathrm{DP}[x-1-y] \\
&= \mathrm{DP}[x-1] + \sum_{y=1}^{x-1} y \mathrm{DP}[x-1-y] + \sum_{y=1}^{x-1} \mathrm{DP}[x-1-y] \\
&= \mathrm{DP}[x-1] + \mathrm{DP}'[x-1] + \mathrm{DP}''[x-1]
\end{aligned}
$$

これで 1 次の項も消えました。最後に、0 次の項を消して完結させます。

$$
\begin{aligned}
\mathrm{DP}''[x] &= \sum_{y=1}^{x} \mathrm{DP}[x-y] \\
&= \mathrm{DP}[x-1] + \sum_{y=1}^{x-1} \mathrm{DP}[x-1-y] \\
&= \mathrm{DP}[x-1] + \mathrm{DP}''[x-1]
\end{aligned}
$$

となります。以上より、漸化式

$$
\begin{pmatrix} \mathrm{DP}[x] \\ \mathrm{DP}'[x] \\ \mathrm{DP}''[x] \end{pmatrix} = \begin{pmatrix} 2 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{DP}[x-1] \\ \mathrm{DP}'[x-1] \\ \mathrm{DP}''[x-1] \end{pmatrix}
$$

を得るため、行列累乗で解を求めることが可能です。

$M \neq 0$ の場合も、与えられた「使えない座標」で区間を切り分け、各部分で行列累乗を行えばよいです。

12.1.2 コンパニオン行列の累乗

遷移が $A_n = \sum_{i=1}^{k-1} c_i A_{n-i}$ という線形漸化式で表される場合 (行列がコンパニオン行列の場合)、行列累乗の計算量を $O(k^3 logn)$ から $O(k^2 logn)$ に落とすことができます (FFT を用いれば、さらに $O(klogklogn)$

に改善されます)。詳細はここでは述べませんが、「きたまさ法」などと呼ばれる場合が多いようです。

さて、累乗をして得られた行列の 1 要素のみ、またはその要素の特定の線形結合の値ひとつのみを知りたいとしましょう。実はこのような場合、どのような行列も適切な変換によって、等しい漸化式を定める等しいサイズのコンパニオン行列に変換することができます。すなわち、k 次の行列の累乗の特定の要素やその線形結合は、ある線形漸化式に従うことになります。

k 項間漸化式を前から計算した値の最初から $2k$ 項以上が求まっている場合、$O(k^2)$ 時間で動作する Berlekamp-Massey のアルゴリズムを用いて漸化式を復元することができます。すなわち、行列を陽に書き下さなくても、愚直な DP で最初の $2k$ 項を求めてやれば、行列累乗と同じことができる、ということになります。複雑な問題では、そもそも k の値が簡単には分からないこともありますが、この場合も十分な数の項を計算しておくことで、やはり漸化式の復元が可能です。さらに上記の手法を用いれば、計算量も $O(k^3logn)$ から $O(k^3 + k^2logn)$ に改善されます (この log 倍の改善を要求する問題を筆者は見たことはありません)。

12.1.3 多項式の累乗

高速化のテクニックの章の FFT の項で、DP の遷移式が FFT を用いて書ける場合がある、と書きました。また、前項でも FFT による高速化に言及しました。本項では、そのさらなる高速化手法について述べます。

さて、DP の遷移式が多項式を掛ける操作と見做せる場合、すなわち適当な数列 A があり、$\mathrm{DP}[n][k]=\sum_{i=0}^{k} A_{k-i}\mathrm{DP}[n-1][i]$ とあらわせる場合に、FFT を用いることができます。

この遷移を繰り返すことは、多項式 $\sum A_i x^i$ の累乗を求めることに他なりません。多項式同士の積は FFT を用いて高速に行えるため、二分累乗のアルゴリズムを用いてこの問題を高速に解くことができます。

さて、計算量はどうなるでしょうか。A をフーリエ変換してできた列の各項を累乗し、それを逆フーリエ変換で元に戻す、というアルゴリズムを思いつくかもしれませんが、そのアルゴリズムは不適切です。これは、DP の遷移の各々のタイミングで、DP の k の値、すなわち多項式の次数の大きすぎる部分を「切り捨てる」操作が必要なことによります (FFT において、添え字は mod 列の長さ で見られることに注意してください。すなわち、溢れた分が循環して足されてしまうことになります)。

多項式を 2 乗する (もしくは掛ける) たびに高次の部分を「切り捨てる」ことでこの問題は回避可能で、$O(logn)$ 回の FFT の計算で累乗を求めることができます。

12.2 行列式のテクニック

ごくまれに、行列式を用いると解ける数え上げの問題が存在します。これらの問題はほとんどの場合、適当な行列の行列式の値が何らかの場合の数に一致する、という形の定理を背景に持ちます。

このような定理としては、行列木定理と LGV 公式の 2 つを覚えておけば十分でしょう。あからさまな形式で出題されることが多く、これといった「考え方」はありません。「全域木の個数」「非交叉経路の個数」の数え上げは、特別な定理のおかげで何故か高速にできる、程度の認識を持っておけば十分でしょう。

定理の主張や証明については、筆者が JOI で講義したときの資料がここ (https://www.ioi-jp.org/camp/2017/2017-sp_camp-kumabe2.pdf) にありますので、参照してください。

13　小さい確率を無視する

確率・期待値系の問題の問われ方には 2 種類の形式があります。片方は確率や期待値を実数として出力させるもの、もう片方は確率や期待値を有理数で表し、適当な mod で出力させるものです。

些細な違いだと思われるかもしれませんが、前者の場合にのみ適用できる、半ば「ずるい」テクニックが存在します。それが、「確率が十分低い場合を無視する」という手法です。

次の問題を考えてみましょう。

問題

以下を N 回繰り返したときの、得点の期待値を求めよ。$N \leq 10^5$

- これまでに k 回連続で得点を得ているとき、確率 2^{-k} で 1 点を得て、確率 $1-2^{-k}$ で得点を得ない。

(出典: AOJ 1056)

DP$[n][x]$: n 回の操作を行い、x 回連続で得点を得ているときの得点の期待値 という DP がすぐに思いつきます。しかし入力が大きいため、これでは間に合いません。

x として考えるべき値はどれほどでしょうか。厳密には n 以下の全ての x を考える必要がありますが、許容誤差以内の解を得るためであれば、あまり大きい x は考える必要がないことがわかります。これは、x 回連続で得点を得る確率が $1/2^x$ ととても小さく、さらにそれによって得られる得点も各操作によって 1 点と、確率の割に十分小さいことによります。

以上より、x の値は小さいところのみ考えればよい (閾値の値は、問題ごとにきちんと解析してやる必要があります) ので、この問題を十分高速に解くことができます。

さて、確率が小さくてもその時の利得が大きい場合、このテクニックを用いることはできません。これは、低確率の事象が期待値には大きな影響を与える可能性があることによります。またこのような場合、特に期待値が大きく log を用いた計算をしなければならない場合や包除原理を用いる場合などは、理論的には正しい解法も計算誤差を回避できないことがあるので、誤差の出にくい手法を考える必要があります (極端な問題では、long double を用いてなお 2 倍以上の誤差が出ることすらあります)。

14　二項係数のテクニック

組合せ的な問題では、二項係数 $\binom{N}{K}$ や、その和がしばしば登場します。本項では、二項係数に関するテクニックについて述べることにします。

14.1　頻出公式集

二項係数の和や積は、より綺麗な少数の二項係数の加減乗除で表せることが往々にしてあります。

以下に代表的な変形を示します。パスカルの三角形や格子上の経路数を考えることで導かれるものが多いです。

- $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
- $\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2i} = 2^{n-1}$
- $\sum_{i=0}^{k} \binom{n+i}{i} = \binom{n+i+1}{i}$
- $\sum_{i=0}^{k} \sum_{j=0}^{l} \binom{i+j}{i} = \binom{k+l+2}{k+1}$
- $\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$
- $\sum_{i=0}^{k} \binom{n+i}{i}\binom{m-i}{k-i} = \binom{n+m+1}{k}$

5 番目の式の導出を考えてみましょう。格子上を x 座標か y 座標の増加する方向に 1 だけ進むことを繰り返して、$(0,0)$ から $(n+m-k,k)$ まで進む場合の数を考えましょう。右辺はそのままこの場合の数です。

左辺は、$(0,0)$ から $(n-i,i)$ まで進む場合の数と、$(n-i,i)$ から $(n+m-k,k)$ まで進む場合の数の積を、全ての i にわたって足し合わせた値となります。$(0,0)$ から $(n+m-k,k)$ まで進む経路は、$x+y=n$ なる点を必ずちょうど 1 回通るので、両者は一致することが分かります。

他の公式についても、パスカルの三角形や格子上の経路数を考えることで、導くことができます。証明は割愛しますが、どれも直感的な導出が可能なので、一度導いてみるのも良いでしょう。

14.2　経路数への帰着

さて、前項で述べた変形は、導出にパスカルの三角形や格子状の経路数を用いると簡単になりました。一般的に、二項係数の和や積を、定義式を数式として見て変形していくのは得策とは言えません (次項と次々項で述べるような変形を、定義式から直接導出しようとしてみるとわかると思います。特に次々項で述べるカタラン数については、シンプルな形の式ですが組合せ的でない導出の中で最も簡単なものも母関数に関する知識を必要とします)。

複雑な式変形を避けるためにも、そのままでは計算量の大きい二項係数を見たときには、経路数へ帰着することを考えると良いでしょう。なお、次項では、この特別な場合を扱います。

次の問題を考えてみましょう。

問題

N 個の袋があり、i 番目の袋には文字 A が A_i 個、文字 B が B_i 個入っている。袋を 2 つ選んでその中の文字をすべて取り出し、好きな順番に並べて列を作る。できた列と袋 2 つの組としてありうるものは何通りあるか。$N \leq 2 \times 10^5, A_i, B_i \leq 2000$
(出典: AGC 001 E)

求める場合の数は、

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \binom{A_i + A_j + B_i + B_j}{A_i + A_j} = \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \binom{A_i + A_j + B_i + B_j}{A_i + A_j} - \sum_{i=1}^{n} \binom{2A_i + 2B_i}{2A_i} \right)$$

です。第 2 項は簡単に求められるので、第 1 項を求めることを考えます。

$\binom{A_i+A_j+B_i+B_j}{A_i+A_j}$ は、格子上を $(-A_i, -B_i)$ から (A_j, B_j) まで移動する場合の数に他なりません。よって、以下のアルゴリズムで第 1 項を求めることができます。

- 各 i に対し、$(-A_i, -B_i)$ に値 1 を書き込む
- 通常の経路数の DP を行う。この操作で、各 (x, y) に対し、$(-A_i, -B_i)$ から (x, y) への経路数の i に関する和が求まる。
- 各 i に対し、上記の (A_i, B_i) に関する値を足し合わせた値が、第 1 項である。

計算量は $O(|\text{座標}|^2)$ となります。

14.3 45 度回転

グリッド上を上下左右に点が移動するというのは、競技プログラミングの問題ではよくある設定です。そのような場合に、盤面を 45 度回転することで xy 座標を独立に扱えるようになる場合があります。

次の問題を考えてみましょう。

問題

(x_i, y_i) の組が N 個与えられる。N 個すべてに対する、以下の値の積を求めよ。

- 上下左右のいずれかに 1 だけ動くことをちょうど T 回繰り返し、(x_i, y_i) から $(0, 0)$ へ格子上を移動する場合の数

$N, T \leq 10^5, |\text{座標}| \leq 10^6$
(出典: ARC 012 D)

(x, y) から $(0, 0)$ に、ちょうど T 回の移動で移動する場合の数を高速に求める問題です。このまま求めるべき値を二項係数の和で書いても、その式を変形することで計算のしやすい形にすることは (不可能ではないにせよ) 難しいです。

このような時に、45 度回転を考えると便利です。45 度の回転によって、1 ターンで行える操作は、x 方向に 1 または -1 動いた上で、y 方向にも 1 または -1 動くという操作になります。これは x, y について独立なので、二項係数の簡単な式で求めるべき場合の数を求めることができます。

14.4　カタラン数

答えがカタラン数になる、あるいはカタラン数を使った式で表せる問題はしばしばあります。一番有名なものは、以下の場合の数たちでしょう。

- x 座標か y 座標の増加する方向に 1 だけ進む操作を繰り返し、$x < y$ なる領域を通らずに $(0,0)$ から (N,N) まで移動する場合の数
- 長さ $2N$ の、バランスされた括弧列の個数
- 凸 N 角形の三角形分割の個数
- N 個の葉を持つ二分木の個数

これらは全て、カタラン数を用いて表されます。一番上の場合の数をもとに、カタラン数の一般項を導いてみましょう。

$(0,0)$ から (N,N) まで (任意の場所を通って) 移動する場合の数は、$\binom{2N}{N}$ 通りです。このうち $x < y$ なる領域を通るようなものについて、初めて $x < y$ となって以降の経路を直線 $x+1 = y$ を対称に折り返すと、$(0,0)$ から $(N-1,N+1)$ へ向かう経路となります。また逆に、$(0,0)$ から $(N-1,N+1)$ へ向かう経路について、初めて $x < y$ となって以降の経路を直線 $x+1 = y$ を対称に折り返すと、$(0,0)$ から (N,N) へ向かう経路となります。すなわち、$(0,0)$ から $(N-1,N+1)$ へ向かう経路と、$(0,0)$ から (N,N) へ向かう経路であって $x < y$ なる領域を通るものは、一対一に対応します。よって、求める場合の数は $\binom{2N}{N} - \binom{2N}{N-1} = \frac{2N!((N+1)-N)}{N!(N+1)!} = \frac{1}{N+1}\binom{2N}{N}$ となります。

この表示は、たまに用いられることがあります。覚えるか、導出できるようになっていると良いでしょう。また、上記の形の数え上げがカタラン数を用いてできることを知っておくとよいでしょう。

15　包除原理

「○○を満たすような△△の個数を求めよ」という形式の問題で○○の条件を独立した場合に分けられそうにないときや、「○○にわたる△△の総和を求めよ」という形式の問題で△△を何らかの和に分解できそうにないときは、包除原理が想定解の場合があります。

数え上げの問題において、包除原理は以下の形でよく使われます。

- N 個の性質 $P_1, ..., P_N$ が与えられる。この性質たちの全ての部分集合 A に対し、それらを全て満たすような「もの」の個数 x_A が求められるとする。このとき、N 個の性質のうち 1 つも満たさないような「もの」の個数は、$\sum_{A \subset \{P_1,...,P_N\}} (-1)^{|A|} x_A$ で求められる。

実際には、全ての部分集合に対する x_A の値を具体的に列挙すると間に合わない場合が多いため、対称性を用いて考える場合を減らすか、DP などで上記の和の値を直接求めにいくことになります。

15.1　対称性を用いる場合

N 個の性質からどの K 個を選んでもそれらをすべて満たすような場合の数が等しい場合、その値を各 K について求めれば、$O(N)$ 時間で包除原理の式を計算することができます。

次の問題を考えてみましょう。

問題

$1, 2, ..., N$ の順列 a であって、全ての i に対し、$|a_i - i| \neq K$ を満たすものはいくつあるか。
(出典: AGC 005 D)

この条件の言い換えや、独立な場合への分解はできそうにないので、包除原理を用いることを考えてみましょう。$\{1, 2, ..., N\}$ のすべての部分集合 S に対し、S の元 s では $|a_s - s| = K$ となるような列の個数を数えられれば、包除原理を用いてこの問題を解くことが可能です。

S を指定し、S のすべての元 s に対して $|a_s - s| = K$ となるように値を割り振ったとします。このとき、a_s たちがすべて異なれば $(N - |S|)!$ 通り、そうでなければ 0 通りの順列を作ることができます。

すなわち、各 x に対し、$|S| = x$ なる S と、S のすべての元 s への、$|a_s - s| = K$ かつ a_s たちがすべて相異なるような a_s の値の割り当て方の組の個数が求められれば良いことになります。詳細は本項の内容から外れるので記載しませんが、その値は DP を用いて計算可能です。

15.2　DP を用いる場合

次の問題を考えてみましょう。

問題

1 ターンで x 座標または y 座標のどちらかがちょうど 1 増加する方向に進める人が、$(0, 0)$ から (X, Y) まで移動する方法は何通りあるか。ただし、通ってはいけない地点が N 個指定される。
$X, Y \leq 10^5, N \leq 2000$

盤面が広大なので、$O(XY)$ の DP を行うことはできません。通ってはいけない地点を禁止地点と呼ぶこと

にしましょう。禁止地点を使わない場合の数を数えるために、禁止地点の各部分集合に対し、それらの地点をすべて通るような経路の個数を求め、包除原理を適用しましょう。

まず、禁止地点たちを x 座標の昇順にソートします。包除原理を用いるときは、通る禁止地点を偶数個指定したときの経路の個数を 1 倍、奇数個指定したときの経路の個数を -1 倍して足し合わせます。これまでに通った「指定した禁止地点の個数」が同じ場合をまとめて考えることで、DP$[n][k]$: n 番目の地点に到達したとき k 個の指定した禁止地点をすでに通っている場合の数の、全ての k 点の指定のしかたに関する和 とすれば、$O(N^3)$ 時間でこの DP を計算することができます。

さて、今、k をそのまま持つ必要はあるでしょうか？　包除原理の定義式に立ち返れば、k が偶数の場合の DP$[n][k]$ の和と、k が奇数の場合の DP$[n][k]$ の和の差のみが重要であることが分かります。以上より、DP'$[n]$: DP$[n][k]$ の、k が偶数のものの和から奇数のものの和を引いたもの とすれば、$O(N^2)$ 時間でこの問題を解くことができます。

15.3　約数系包除

数えたいものが、「あるパラメータが k の倍数である」「あるパラメータが k の約数である」のような形の条件のもとで (除去できる重複は許して) 数え上げられる場合、包除原理は有用です。ここでは、前者と後者の問題を 1 問ずつ考えていくことにしましょう。

まずは前者の場合を考えます。この問題を考えてみましょう。

問題

整数 N 個からなる集合が与えられる。これらの非空な部分集合としてありうるもの 2^N-1 通りすべてに対してその中の要素の最大公約数を求め、足し合わせた値を求めよ。N, 与えられる整数 $\leq 10^5$

各整数 x に対し、最大公約数が x の倍数となるような集合を数え上げるのは簡単です。しかし、このまま足し合わせると、例えば最大公約数が 6 の場合は $x=1,2,3,6$ で重複して数えられてしまうことになります。

ここで包除原理を用います。このとき、このタイプの問題では、x を大きい順に見ていき、順次「最大公約数がちょうど x であるような場合の数」を求めていくのが簡単でしょう。x について見るとき、x の ($2x$ 以上の) 倍数についてはその「ちょうどの値」が計算されているので、単純な引き算で x についての「ちょうどの値」を求めることができます。

計算量は、調和級数の和を考えると、与えられる整数の最大値を X として、$O(XlogX)$ であることが分かります。

次に後者の場合を考えてみましょう。今度は、この問題を考えます。

問題

1 以上 K 以下の整数からなる長さ N の数列であって、以下の方法でつくることのできるものは何通りあるか。

- 長さ N の回文を用意する。その後、先頭の要素を末尾に持ってくる操作を好きな回数繰り返す。

$N, K \leq 10^9$
(出典: ARC 064 F)

作ることができる数列 A に対し、末尾の要素を先頭に持ってくる操作を繰り返して初めて回文になったと

きの列を $f(A)$ としましょう。$f(A)$ の周期 t が偶数なら $\frac{t}{2}$ 個、奇数なら t 個の列が同じ $f(A)$ に対応していることが、「この項とこの項が同じ」という関係を数列に対して列挙していくことで分かります。

以上より、各 t に対し、周期が t の回文の個数を求めればよいことが分かります。t としてありうるものは N の約数だけなので、それらについて考えることにしましょう。

周期が t の約数であるような回文は、簡単に数えることができます。先程は x を大きい順に見ていきましたが、今度は t を小さい順に見ていくのが簡単です。すなわち、「周期がちょうど t であるもの」を t が小さい順に更新していけば、その「ちょうどの値」が単純な引き算のみで求められることになります。

N の約数の個数は十分小さいので、約数の個数の 2 乗の計算時間をかけても十分間に合います。

16 「解けない問題」を見極める

問題に関する考察を進めていくと、より一般的な数え上げの問題に帰着されることがあります。このとき、数えられない (と信じられている) 問題に帰着してしまった場合、その方針では解けない、ということを知ることができ、不毛な考察に時間を使わないようにできます。そのため、どのような数え上げの問題は「解けない」か、ということを知っておくことが重要です。

以下に挙げるのは、#P-complete と呼ばれるクラスに属する問題です。P != NP のもとでは、以下の (判定問題は多項式時間で解ける) 問題には多項式時間解法が存在しないということが知られています。

- 2-SAT を充足する真偽値割り当ての数え上げ
- 二部マッチング、一般マッチングの数え上げ
- 行列の permanent の計算
- グラフのトポロジカルソートの数え上げ
- Euler 閉路の数え上げ

「行列式のテクニック」の項で述べた全域木、非交叉経路の数え上げに関しては、述べた通り多項式時間のアルゴリズムが知られています。また、平面的グラフの完全マッチングの数え上げに関しては、多項式時間で可能です (競技プログラミングの問題としての出題は見たことがありません)。しかし、これらの特殊な場合を除いては、簡単なアルゴリズムの思いつかない「古典的な」数え上げの問題には、多項式時間の解法は (P != NP なら) 存在しない、と思っておくのが良いでしょう。

17　おわりに

これまで、数え上げの問題の考え方やテクニックを挙げてきました。確かに、これらのテクニックで解けない問題も存在するでしょう。また、特に前半に述べたテクニックは、「何を考えるべきか」がわかっていても、その考察が依然難しいことは多いです。

しかしそれでも、「何を考えればいいか」を知るだけで、数え上げの問題は一気に解けるようになります。筆者自身、二年ほど前までは数え上げは苦手だったのですが、いったんコツをつかんでからは一番の得意分野になりました。また、数え上げの問題はえてして実装が軽いものが多いため、考察がはやく済むことは (特に AtCoder などの) プログラミングコンテストにおいて非常に大きなアドバンテージとなります。

本稿を読まれた方が、ここから核となる「考え方」を身につけ、そして数え上げが得意分野になったならば、筆者としてこれ以上に嬉しいことはありません。

それでは、皆様のコンテストにおける活躍を願って、本稿を締めさせていただきます。論理上の誤りや誤字脱字、また改善案などありましたら、DEGwer までお気軽にお知らせください。