# ○max

この問題は、c++などのライブラリにあるmax関数などを使うとすぐに求めることができます。しかし、dpのコンテストなので、この問題を０次元dpとして考えます（参考文献[1]）。dpに保存するべき値は、数列の最大値なので

dp:=i番目までにおける最大値

となり、漸化式は次のようになります。

dp=max(dp,a[i])

この式を用いれば、N番目まで、すなわち、数列の最大値を求めることができます。ちなみに、計算量は0(N)です。

# ○tereport

この問題から、本格的にdpを用います。この問題を少し言い換えてみると次のようになります。

「0から、1と整数$A_1,A_2,A_3,...,A_M$を使って足し算を行い、整数Nにすることができる組み合わせは何通り(mod $10^9+7$)あるか」

今回のような場合では、いきなりNを求めるのではなく、N=1だったらどうなるか、N=2だったらどうなるか、N=3なら…といったように考えると良いでしょう。例として、N=3で、A={2}の場合を考えます。まず、初期条件を考えます。問題から、最初の値は0です。すなわち、整数0にするための方法は1通りしかありません。これが、初期条件となります。次に、整数1にするための方法は、0+1=1しか存在しないので、1通りとなります。整数2にするための方法は、0+1+1=2、もしくは、今回の例では、0+2=2とできるため、２通り考えられます。では、N=3ならどうなるでしょうか。全て列挙するならば、①0+1+1+1=3、②0+1+2=3、③0+2+1=3の3通りが考えられます。よく見てみると、式①および式③は、整数2にするための式に1を足したものと見なせます。また、式②の場合も、整数1にするための式に2を足したものと見なせます。このことから、整数3を求めるためには、整数1,2が求まればよいということがわかります。よって、これを一次元dpで考えると、

dp[i]:=整数iにするための組み合わせ(mod $10^9+7$)

とし、漸化式は次のようになる。

dp[i]+=dp[i-1]

dp[i]+=dp[i-A[j]](i-A[j]>=0)

この式を用いて、かつ計算毎に$10^9+7$のあまりを求めればこの問題を解くことができます。この計算量は、O(NM)なので間に合います。

# ○multiplication

これは、先ほどの問題(tereport)では、足し算をおこなったものを掛け算に変えてNにできるかどうか確かめるという問題です。ただし、先ほどと違う点は、NG数なるものが存在することです。このNG数にならないようにしながら、整数Nを作れるかどうか確かめ

るには、先ほどの一次元dpをどう変更すればよいでしょうか。実は、その通りにやればよいのです。どういうことかというと、

dp[i]:=整数iにすることができるかどうか

とし、漸化式を次のように立てます。

dp[i*a[j]]=dp[i](i≠$NG_k$ && i*a[j]<=N && dp[i]=true)

こうすることで、整数iがNG数であるときに遷移先の値がtrue(整数iにすることが可能)になることはなく、かつi番目の値がそもそも作れないときもdp[i*a[j]]がtrueになることはありません。また、一次元dpの初期化は、dp[a[j]]=true とすることで、一度も掛け算をしなかったときは、整数a[j]が作れるということになります。１つ注意することとしては、iがNG数かどうかの判定は、愚直にK回の探索では間に合わなくなるので、連想配列を用いるとよいでしょう。よって、計算量は、O(N×M×1)=O(NM)となります。

# ◯grid

この問題は、DFS、BFSなどの探索アルゴリズムで解けそうですが、全てのパターンを列挙しなければならないので間に合いません。よくみると、この問題では移動方法が、右もしくは下に移動する２パターンしかありません。つまり、座標(x,y)において、(x-1,y)から(x,y)に移動できるのなら、(x-1,y)に移動する方法を(x,y)に加算し、(x,y-1)から(x,y)に移動できるのなら、(x,y-1)に移動する方法を(x,y)に加算するというように考えればこの問題は解けます。よって、

dp[i][j]:=座標(j,i)に移動する方法(mod $10^9$+7)

とし、漸化式を次のように立てます。

dp[i][j]+=dp[i-1][j](i-1>=0 && c[i-1][j]≠”#”)

dp[i][j]+=dp[i][j-1](j-1>=0 && c[i][j-1]≠”#”)

二次元dpの初期化は、スタート地点の移動方法は、そこから動かないと考えればよいので当然1通りとなり、スタート地点の座標を(sx,sy)とすると、dp[sy][sx]=1、それ以外は、０で初期化すればよいです。よって、計算量は、O(HW)となります。

# ◯levenstein

この問題は、いわゆる編集距離(レーベンシュタイン距離)と言われるもので、dpの典型問題みたいなものの１つです。この問題は、二次元dpで解く必要があり、

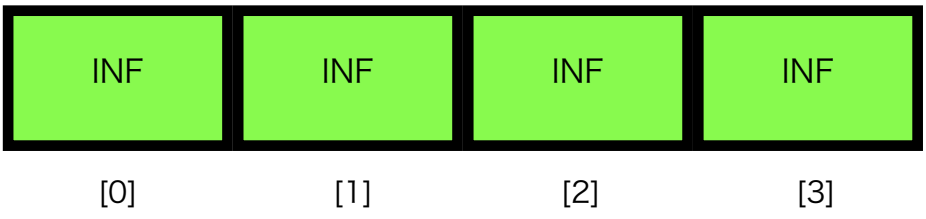
dp[i][j]:=文字列Tのi文字目、文字列Sのj文字目までにおける必要最小操作回数(すなわち、Tのi文字目までとSのj文字目までは同じ)

とします。次に漸化式ですが、この問題で行える操作を例を用いて考えてみます。例として、T=“abs”,S=“adgdc”とします。まず、T,Sの１文字目をみるとどちらも”a”なので操作を行う必要はありません。次に、２文字目をみるとTは”b”、Sは”d”となり置換する必要があり、操作を一回行います。次に３文字目をみると、Tは”s”、Sは”g”となり、これも置換する必要があるため、操作を一回行います。次に４文字目をみますが、Tはもともと3文字しかないので、文字を挿入する必要があります。よって、2回の操作を行い、合計で4回の操作が必要です。これらを踏まえて、各操作の漸化式は次のようになります。

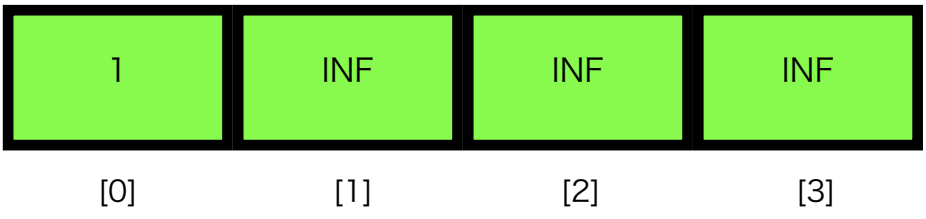
挿入:dp[i][j]=dp[i-1][j]+1
削除:dp[i][j]=dp[i][j-1]+1
置換:dp[i][j]=dp[i-1][j-1]+1
なぜ、こうなるかを説明します。まず、この２次元dpにおいて、Tのi文字目までとSのj文字目までは同じであるということを仮定していることを覚えておいてください。挿入をする必要があるということは、Sのj文字目までを構成するのにTのi文字目までては1文字足りないということになります。そのため、i文字目までとj文字目までを同じにするためには、Tのi-1文字目にSのj文字目を挿入する、すなわち、Tのi-1文字目、Sのj文字目において１回の操作を行うということになります。削除の場合は、挿入の逆と考えてよいので、Tのi文字目、Sのj-1文字目に１回の操作を行うと考えられます。置換をする必要があるということは、i文字目とj文字目が互いに違うということになります。つまり、Tのi-1文字目、Sのj-1文字目に一回の操作を行うということになります。これらのことから、漸化式は次のようになります。
dp[i+1][j+1]=min(dp[i+1][j],dp[i][j+1])+1
dp[i+1][j+1]=min(dp[i][j]+c,dp[i+1][j+1]) (ただし、S[i]=T[j]の時、c=0。S[i]≠T[j]の時、c=1)
よって、計算量は、O(|S||T|)となります。

# ○domino

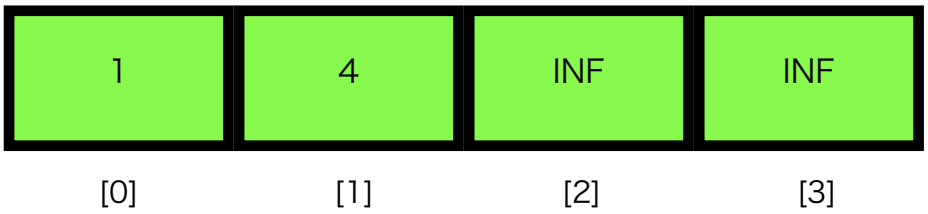
この問題も、LIS(最長増加部分列)というdpの典型問題みたいなものを少し応用させたものです。LISは、$a_1,a_2,a_3,..,a_N$の数列において、i<jの時、$a_i<a_j$を満たす部分列の最大の長さを求めるもので、今回は、それを逆転したものとして捉えればよいです。そのため、解法は２種類あります。１つ目は、数列Aを一度逆転させてLISを求める方法です。そして、２つ目は、i<jの時、ai>ajを満たす部分列の最大の長さを求める方法です。それでは、LISの解き方について説明します。まず、この一次元dpを
dp[i]:=長さi+1におけるLISの最小値
と定義します。
次に、更新について、例として、A={1,4,2,3}のLISを考えてみます。最初、dpの値は全てINF(非常に大きな値)で初期化します。



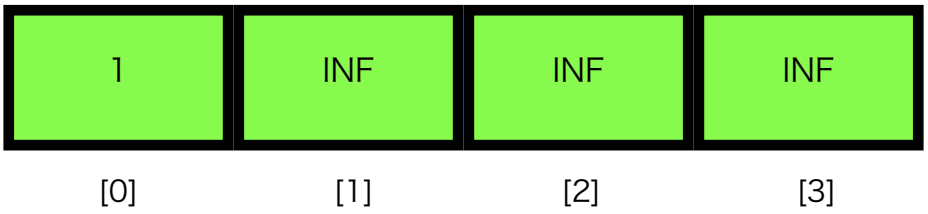

次に、dpには、何も値が入っていない状態なので、0番目に1を代入します。



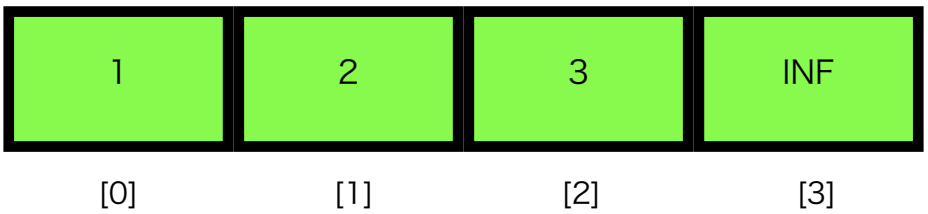
次に、Aの1番目の要素である4について考えます。今、dpには、1のみが入っており、これは、長さ1のLISにおける最小値を表しているため、この値と4を比較します。当然4の方が大きいので、0番目の値は更新せず、1番目に4を代入します。




次に、Aの2番目の要素である2について考えます。dpの1番目の要素と比較すると2の方が小さいため、さらに小さい値と比較するために0番目の要素とも比較します。0番目の要素は1なので、2の方が大きいとなり、更新するべきところは、1番目となります。よって、dpの1番目の値を2に更新します。




最後に、Aの3番目の要素である3について考えます。先ほどと同じように、後ろから比較していくと、1番目と比較した時に3の方が大きいとなります。よって、dpの2番目に更新するべきとなります。よって、dpの２番目に3を代入します。




これで、全ての操作が終了したため、AのLISの長さは3となります。これを実装する方法としては、dpの値は、単調増加列であるため２分探索で探索するのが最も有効です。この点については、サンプルコードを参考にしてください。あとは、先ほど述べた２つのうちどちらかの方法で実装することでこの問題は、およそO(NlogN)で実装することができます。

・参考文献

[1]https://qiita.com/bplain2/items/8e5bb79e4642368fe275
[2]https://cinnamo-coder.hatenablog.com/entry/2019/05/11/191255