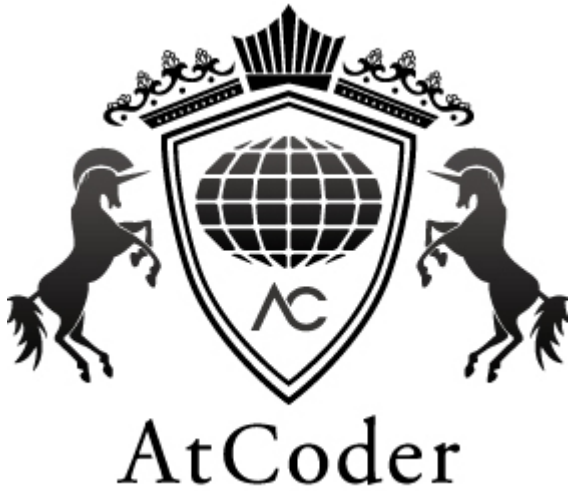
AtCoder Regular Contest 027
解説



A 問題－門限

問題概要

- 現在の時間 h と分 m が与えられる。
- 後何分で 18 時 0 分かを求める。
- $0 \leqq h \leqq 17$, $0 \leqq m \leqq 59$

※標準的な入出力については、ABC004 の解説等を参照してください。

解法その 1

- 18 時まであと何時間何分かを求めて、その後分に変換する。
- 現在時刻が h 時 m 分なら、
    - $m = 0$ ⇒ あと $18 - h$ 時間
    - $m \neq 0$ ⇒ あと $17 - h$ 時間と $60 - m$ 分
- 計算結果を a 時間 b 分とすると、$\mathrm{a} \times 60 + b$ 分が答え。

- 最初に、現在時刻を 0 時ちょうどからの経過分数に変換して、1080 から引いても良い。

解法その 2

- 現在時刻から、1 分後、2 分後、…と 1 分ずつ加算して、18 時ちょうどになるまでに合計何回足したかを数える。

- どちらの方針にするかは、実装時間やバグの入りやすさなどを考慮して選択 (個人差あり)。

B 問題 −
大事な数なのでZ 回書きまLた。

問題概要

- 0 から 9 までと、大文字アルファベットのみで構成された長さ N の文字列が 2 個与えられる。
- 大文字アルファベットを 0 から 9 までの整数に変換する処理を行った結果、両者が表す整数が一致した。
- 一致した整数として全部で何通り考えられるかを求めよ。
- $1 \leqq N \leqq 18$

部分点解法

- 答えとなる数字を全探索する。
- この場合、答えとして考えられないものは、
    - 同じアルファベットに複数の数字が割り当てられている。
    - 先頭に 0 が来ている。

となる場合なので、その場合を判定して、残ったものの総数が答えとなる。

満点解法

- 部分点解法では、計算量が $O(10^N)$ となるので、Nが大きい場合には時間が掛かり過ぎる。
- 一致しているという条件をうまく言い換えたい。
- ここでは、「グラフ」の概念を導入すると理解しやすくなる。
- それぞれの文字を頂点としたグラフを考える。
- 同じ数字でなければならない文字同士を辺で結ぶ。
- 同じ数字でなければならない文字同士というのは、同じ桁 (同じ位) にある文字同士のことである。(例えば、2 つの文字列がABCDとEFEHだった場合は、AE,BF,CE,DH 間に辺を結ぶ。)

満点解法

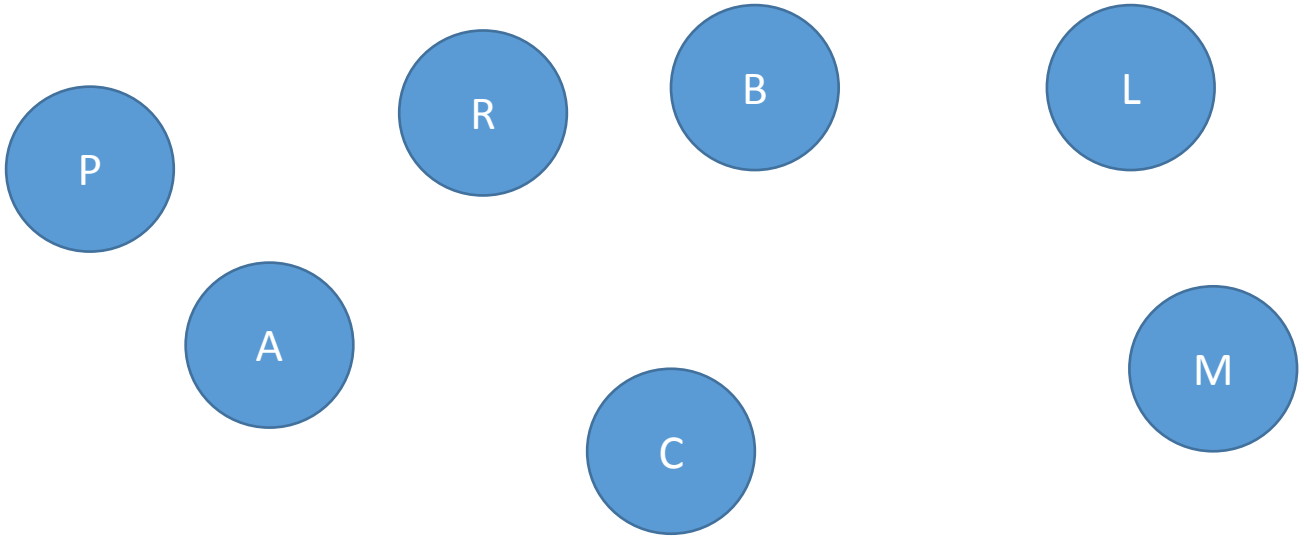
- 数字と文字が同じ桁にある場合は、その文字に、「その数字でなければならない」という条件を付加する。
- 最上位にある文字については、さらに「0以外」という条件も。
- なお、この条件は衝突しない(衝突したら答えがなくなるため)。

- その後、各連結成分(辺同士で行き来できる頂点たち)について、
    - 数字が割り当てられている⇒1 通り
    - 数字が割り当てられていない、かつ最上位⇒9 通り
    - 数字が割り当てられていない、かつ最上位でない⇒10 通り

をそれぞれ掛け合わせると答え。

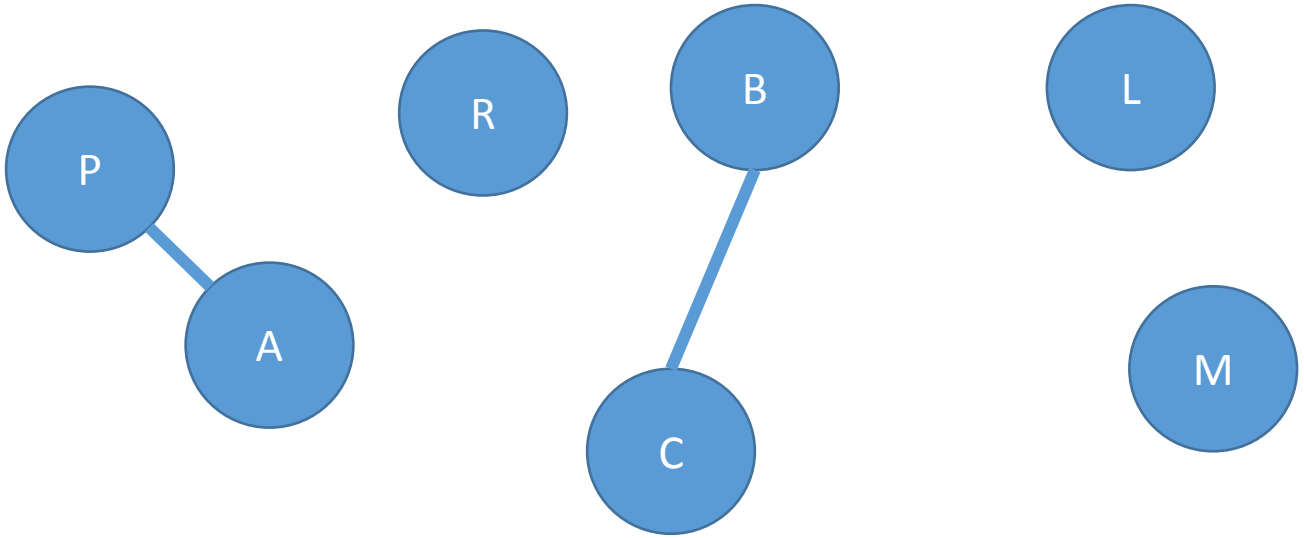
例(入力例 3 – PRBLMB と ARC027)

- 文字を頂点としたグラフを考える。



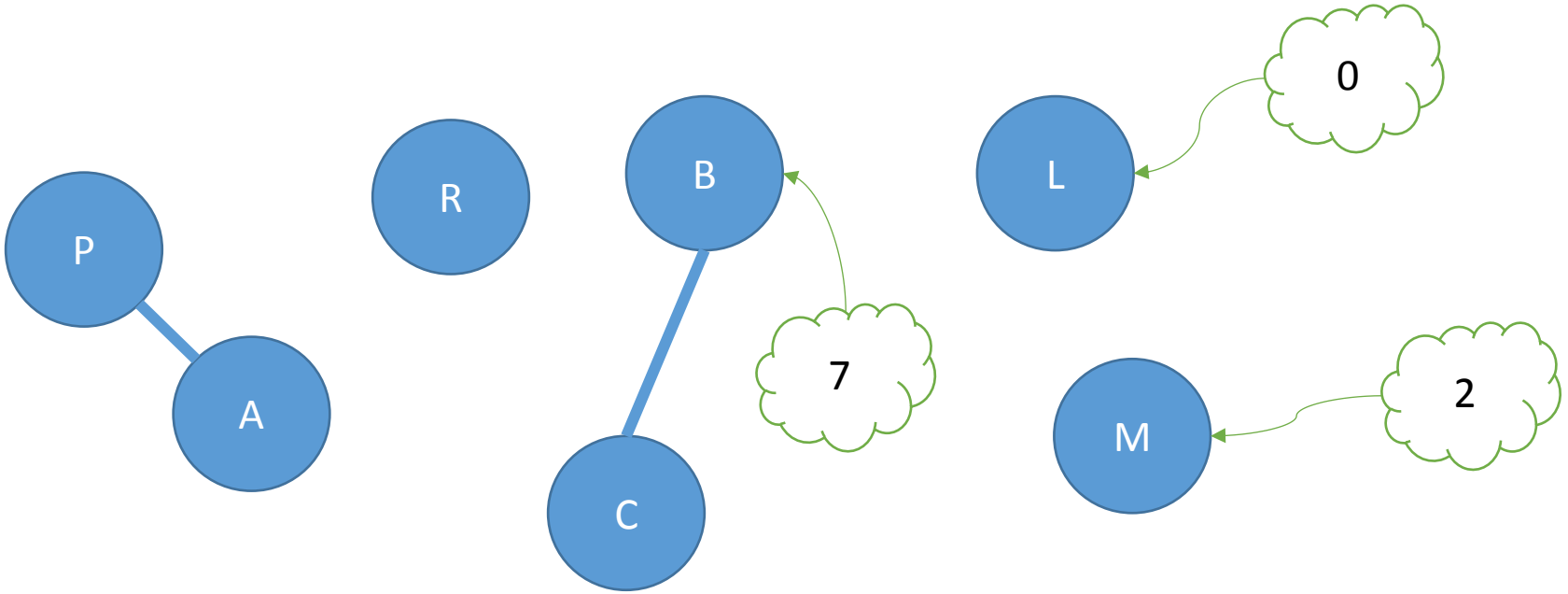
例(入力例 3 – PRBLMB と ARC027)

- 文字を頂点としたグラフを考える。
- 同じ数字になるもの同士を辺で結ぶ(PA,BC)。



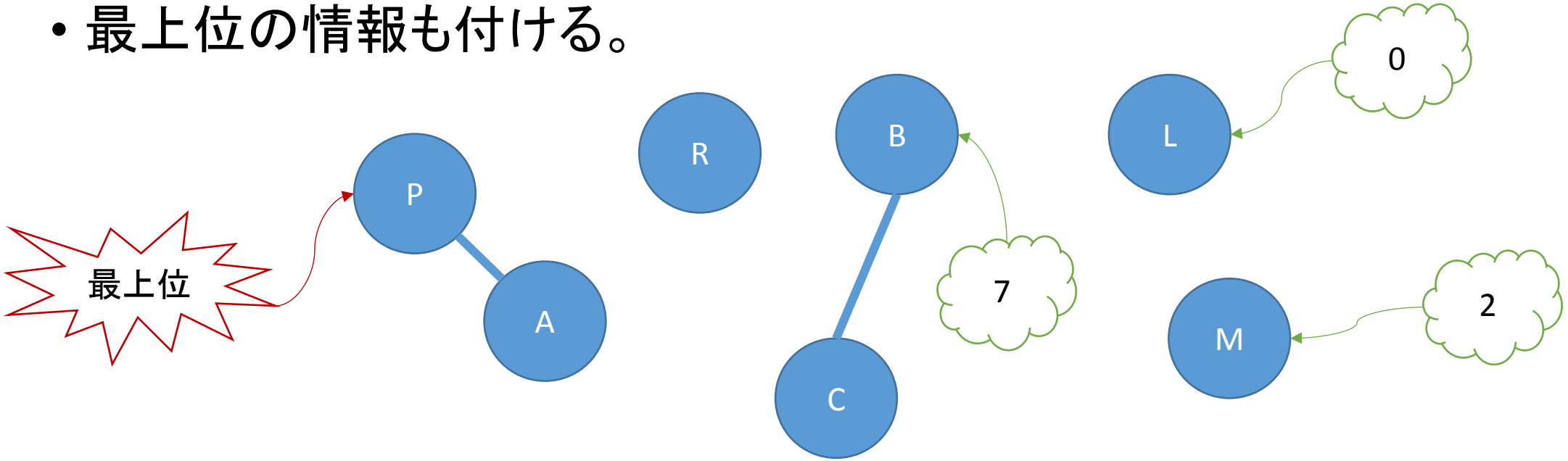
例(入力例 3 – PRBLMB と ARC027)

- 文字を頂点としたグラフを考える。
- 同じ数字になるもの同士を辺で結ぶ(PA,BC)。
- 文字と数字の割り当てを反映させる(L=0,M=2,B=7)。



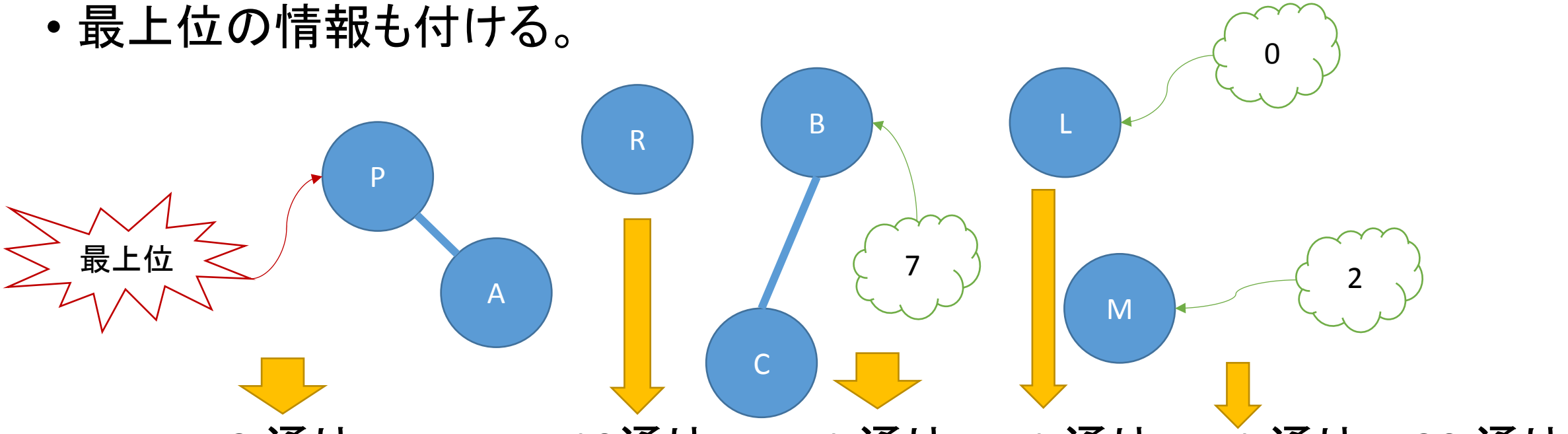
例(入力例 3 – PRBLMB と ARC027)

- 文字を頂点としたグラフを考える。
- 同じ数字になるもの同士を辺で結ぶ(PA,BC)。
- 文字と数字の割り当てを反映させる(L=0,M=2,B=7)。
- 最上位の情報も付ける。



例(入力例 3 – PRBLMB と ARC027)

- 文字を頂点としたグラフを考える。
- 同じ数字になるもの同士を辺で結ぶ(PA,BC)。
- 文字と数字の割り当てを反映させる(L=0,M=2,B=7)。
- 最上位の情報も付ける。



補足

- グラフの連結判定には、union-find を利用すると効率的 & 実装しやすい。
- サイズが小さいので探索をしても良いが、union-find を知っていると色々と便利なので、ぜひとも覚えておこう！
- 前回(ARC026)の解説にも載っているので、参考にしてください。
- http://www.slideshare.net/chokudai/arc026

C 問題 –
最高のトッピングにしような

問題概要

- スペシャルチケット X 枚と通常のチケット Y 枚を持っている。
- N 個のトッピングがあり、トッピング i は t_i 枚のチケット(うち 1 枚以上はスペシャルチケット) と交換でき、嬉しさは h_i である。
- 同じトッピングは複数回入手できない。
- 嬉しさの合計値として最大なものを求めよ。
- $1 \leqq X \leqq 300$, $0 \leqq Y \leqq 300$, $1 \leqq N \leqq 300$
- $1 \leqq t_i \leqq 600$, $1 \leqq h_i \leqq 5{,}000{,}000$

部分点解法

- この問題はナップサック問題の一種。
- 動的計画法を用いると効率的に解答することができる。
- dp[i][j][k] = (現在トッピング i まで処理しており、すでにスペシャルチケットを j 枚、通常のチケットを k 枚消費している場合での嬉しさの合計値の最大値) とすると、

  dp[i][j][k] = max{dp[i-1][j][k] ,

  dp[i-1][j-m][k- t_i +m] + h_i (1 ≦ m ≦ t_i)}

(細かい境界条件は割愛)

- O($XYN(X+Y)$) となる。

満点解法

- 先ほどの場合、交換可能なすべての組み合わせを試していたが、出来る限りスペシャルチケットの消費数を抑えられるものを選んだほうがいいのでは?
- 実際、消費枚数が最も少ない (m が最小) となるもののみを考慮しても最適解を得られる。
- 実はこの問題は、「チケットを $X+Y$ 枚持っており、トッピングは最大 X 個しか選べない場合の最大化」と同じ問題。
- dp[i][j][k] = (現在トッピング i まで処理しており、トッピングを j 個採用しており、チケットを k 枚消費している場合での嬉しさの合計値の最大値) とおく動的計画法なら O($XN(X+Y)$) となる。

D 問題 −
ぴょんぴょんトレーニング

問題概要

- N 個の石があり、石 i からは h_i 個先まで跳べる。
- 石 s_i から石 t_i までジャンプで移動する組み合わせが何個あるか、というクエリを D 個解け。
- 同じトッピングは複数回入手できない。
- 嬉しさの合計値として最大なものを求めよ。
- $1 \leqq N \leqq 300{,}000$, $1 \leqq D \leqq 5{,}000$, $1 \leqq h_i \leqq 10$
- $1 \leqq s_i < t_i \leqq N$

部分点解法

- この問題も動的計画法によって計算できる。
- dp[i]=(石 i までの行き方) とおくと、

$$\mathrm{dp[i]} = \sum_{\mathrm{j}=1}^{10} (\mathrm{dp[i-j]} * f(\mathrm{i-j, i}))$$

ここで、$f(x, y)$は$x + h_x \geqq y$なら 1 、そうでないなら 0 を取る関数とする。

- 手前 10 項 (dp[i-1],dp[i-2],…,dp[i-10])を覚えておけば良い。
- 毎回 O(N) で計算すれば O(ND) となる。

満点解法

- 毎回計算する際に、同じ部分を何回も使用していることに気がつく。
- 同じ部分を「平方分割」によって処理することを考える。
- 手前 10 項を覚えておけば次の項を計算することができることを利用する。
- func(s,t,dp[s-1],dp[s-2],…,dp[s-10]) = dp[t],dp[t-1], … , dp[t-9] のそれぞれの値を、dp[s-1], dp[s-2], … , dp[s-10] を用いて変換する関数。
- func の返り値は 10 個ある。

どゆこと?

- 例として dp[99] の値を知りたいときに、dp[20],dp[19],dp[18], … ,dp[11] の値さえわかっていれば、

  dp[99]=c(20,99,21) * dp[20] + c(19,99,21) * dp[19] + … + c(11,99,21) * dp[11]

  と表記できる。

ここで、c(x,y,z)=(x から y まで移動するものの中で、最初のジャンプで z 以上の石に移動するものの総数) となる。

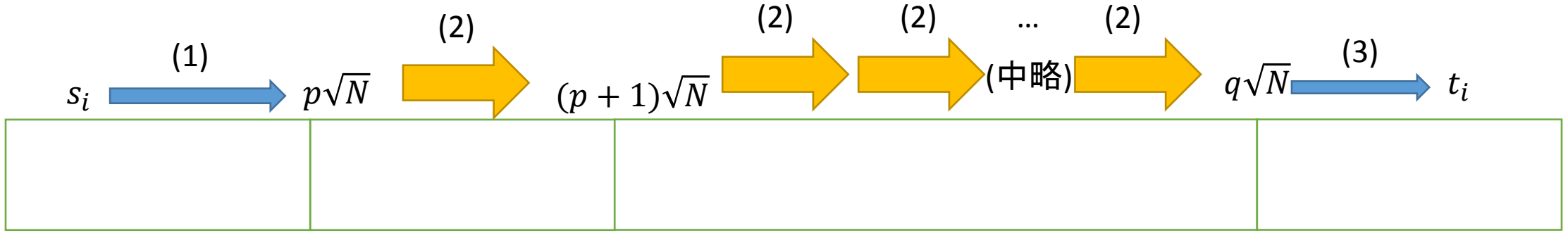
- dp[98] から dp[90] までについても同様に計算すれば func(21,99, ,dp[20],dp[19],...,dp[11]) の完成となる。

func の計算について

- c (s,t,u)は部分点解法と同じ方針で O(t-s) で計算できる。
- 「1 回目のジャンプで u 以上に移動する組み合わせ」としなければならないことに注意 (同じジャンプ重複して数えてしまう)。
- c (s,t,u) は、各 func に対して最大 100 種類計算すれば OK。
- func の計算は、1 つの func につき O($\sqrt{N}$) (100*$\sqrt{N}$) となる。

満点解法

- func($k\sqrt{N}, (k+1)\sqrt{N}$) ($0 \leqq k \leqq \sqrt{N}$) を計算しておき、各トレーニングについて、s_i から $\sqrt{N}$ の倍数になるまで部分点解法と同じDP(1)→func で $\sqrt{N}$ 個ずつ飛ばせるだけ飛ばす(2)→ t_iになるまで部分点解法と同じDP(3)
- func の計算は、1 つの func につき O($\sqrt{N}$) なので、全体で O(N) となる。
- この方針でfunc を求めた後は O($D\sqrt{N}$) となる。



補足

- func は 10×10 行列として表現できる。
- ($n \times n$行列)* ($n \times n$行列)*…* ($n \times n$行列)* (n 要素ベクトル) の計算において、* の個数を m とする。このとき、
    - * を左から処理→行列*行列=行列を繰り返し、O($n^3 m$)
    - * を右から処理→行列*ベクトル=ベクトルを繰り返し、O($n^2 m$)

となり、この問題でも速度に 10 倍もの差が出る。このため、ベクトルを利用した計算をしたほうが速くなる。func を利用した計算の際には注意。