第一回日本最強プログラマー学生選手権 予選 解説

sheyasutaka, drafear, gazelle

2019 年 8 月 24 日

For International Readers: English editorial starts on page 8.

A: Takahashi Calendar

各 $m = 1, 2, ..., M$, $d = 1, 2, ..., D$ に対して実際に m 月 d 日が積の日か調べ、そうであれば見つけた数に 1 加算するようにしていけば、答えを求めることができます。条件を調べる際には、d_1, d_{10} を d から計算する必要があります。d_1 は d を 10 で割った余り、d_{10} は d を 10 で割った商として計算できます。これを C++ 言語で実装した例を示します。

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  int M, D; cin >> M >> D;
  int ans = 0;
  for (int m = 1; m <= M; ++m) {
    for (int d = 10; d <= D; ++d) {
      int d_1 = d % 10, d_10 = d / 10;
      if (d_1 >= 2 && d_10 >= 2 && d_1 * d_10 == m) {
        ++ans;
      }
    }
  }
  cout << ans << endl;
}
```

考察をもう少し頑張ると、$1 \leq d_1 \times d_{10} \leq M$ であることと $d_1 \times d_{10} = m$ なる $1 \leq m \leq M$ が存在することは同値なので、次の実装例ように 1 重ループで書くこともできます。$d_1, d_{10} \geq 2$ のとき、$d_1 \times d_{10} \geq 1$ であることに注意してください。

```
#include <bits/stdc++.h>

using namespace std;

```

```cpp
int main() {
  int M, D; cin >> M >> D;
  int ans = 0;
  for (int d = 10; d <= D; ++d) {
    int d_1 = d % 10, d_10 = d / 10;
    if (d_1 >= 2 && d_10 >= 2 && d_1 * d_10 <= M) {
      ++ans;
    }
  }
  cout << ans << endl;
}
```

B: Kleene Inversion

B を構成する K 個の A を順に A_1, A_2, ..., A_K とします。B の転倒数は、ある A_i の内部で発生するものと A_i と A_j $(i \neq j)$ の間で発生するものに分かれます。前者の総和は A の転倒数に K を掛けたものに等しく、A の転倒数は愚直な全探索で $O(N^2)$ で求めることが可能です。後者の総和は 2 つの A の間で発生する転倒数に、K 個のものから 2 つを選ぶ選び方 $(= \frac{K \times (K-1)}{2})$ を掛けたものに等しいです。ここで 2 つの A の間で発生する転倒数は、A の各要素について、それより小さい養素が A にいくつ存在するかを調べることで求めることができます。愚直に計算してもこれは $O(N^2)$ の計算量で実行可能です。以上より、全体で $O(N^2)$ の計算量で B の転倒数を計算することができます。

C: Cell Inversion

まず、操作の順番を入れ替えても結果は変わりません。l 番目のマスを左側に、r 番目のマスを右側に選んだ操作を操作 (l, r) と表すことにします。すると、2 つの操作 $(l_1, r_1), (l_2, r_2)$ について、これらの操作をする代わりに操作 $(l_1, r_2), (l_2, r_1)$ をしても結果は変わりません。すなわち、各マスが左側として選ばれたのか右側として選ばれたのかのみによって結果が変わります。実際、隣合うマスの色の関係 (同じか異なるか) に注目すると、i 番目のマスを左側として選んだ場合は、$i-1$ 番目のマスと i 番目のマスの色の関係のみが変化し、i 番目のマスを右側として選んだ場合は、i 番目のマスと $i+1$ 番目のマスの色の関係のみが変化します。

d_i を、i 番目のマスを左側として選ぶべきなら L、右側として選ぶべきなら R とします。まず、左端のマスは必ず左側として選ばれるので、$d_1 = \mathrm{L}$ です。$i = 2, 3, ..., 2N$ について、i 番目のマスは $S_{i-1} = S_i$ ならば $d_i \neq d_{i-1}$、$S_{i-1} \neq S_i$ ならば $d_i = d_{i-1}$ です。例えば、S が BWWWWB のとき、$d_1 d_2 ... d_{2N}$ は LLRLRR となります。S が WBBBW のときも LLRLRR となりますが、左端や右端が W のときはそのマスを白色に戻せません。また、L の数と R の数が異なるときも全マスを白色にすることはできません。

さて、残るは、$d_1 d_2 ... d_{2N}$ となるような選び方の個数の求め方です。$d_i = \mathrm{R}$ のマスについて、左から順に決めていくことにします。i 番目のマスまで (i 番目のマスを含まない) に登場する L の個数を L_i、R の個数を R_i とします。すると、$d_i = \mathrm{R}$ となる最も左のマスについては相方として選べる $d_j = \mathrm{L}$ のマスは L_i 通り、その次の $d_i = \mathrm{R}$ のマスは $L_i - 1$ 通りです。すなわち、$d_i = \mathrm{R}$ となる各マスについて、$L_i - R_i$ 通りの選択肢があり、これらの積が操作 $(l_1, r_1), (l_2, r_2), ..., (l_{2N}, r_{2N})$ の $r_1 < r_2 < ... < r_{2N}$ のとき (操作を集合としてみたとき) の場合の数になります。操作は必ず N 回行うので、操作の順番を入れ替えた場合も考慮すれば、これに $N!$ を掛けたものが答えになります。

時間計算量は $O(N)$ です。

D: Classified

簡単のため，レベル $1 \sim k$ のみを使って全通路を設定することを k-分類 と呼ぶことにします．

結論から言うと，k-分類 できる最大の N は $N = 2^k$ です．つまり，以下のことが言えます．

- $N = 2^k + 1$ のとき，k-分類できない．
- $N = 2^k$ のとき，k-分類できる．

前者を数学的帰納法により示します．$k = 1, N = 2^k + 1 = 3$ のときは明らかなので，$k = x - 1$ のとき成り立つと仮定して，$k = x, N = 2^x + 1$ のとき成り立つことを示します．

さて，与えられた条件は「任意のレベルについて，そのレベルの通路のみに注目すると，それらは二部グラフをなす」と言い換えられます．二部グラフについて言えることとして「頂点を適切に 2 色に塗り分けることで，同色の 2 頂点の間に辺が無いようにできる」というものがありますが，頂点数が $2^x + 1$ のとき，どう塗り分けてもある $2^{x-1} + 1$ 個の頂点が同色となる (なぜなら $2^x + 1 > 2^{x-1} \times 2$ なので) ということから，$N = 2^x + 1$ のときに x-分類できるためには $N = 2^{x-1} + 1$ のときに $(x-1)$-分類できることが必要と分かります．ところが，これは不可能と仮定してあるので，結局のところ $N = 2^x + 1$ のときは x-分類できません．

後者は単純に示せます．部屋番号を $0 \sim 2^k - 1$ とします．通路 i, j 間のレベルを「i, j を 2 進法で書いて，片方のみで立っている (桁が 1 である) ビットを好きに 1 つ選んだとき，そのビットは下から何番目か」として決めればよいです．たとえば，部屋 $1101_{(2)}, 0111_{(2)}$ 間の通路のレベルとしては $2, 4$ があり得ます．これが条件を満たすことは容易に納得できます．

上に書いた構成方法をそのまま使うことで実装も簡単にできます．

E: Card Collector

各行に対応した頂点と各列に対応した頂点を用意し、R_i 行目に対応した頂点と C_i 列目に対応した頂点に重み A_i の辺を張ったグラフ G を考えます。すると、この問題は G 上で各頂点についてそれに接続する辺を 1 つずつまで選び、その合計重みの最大値を求める問題と解釈できます。なお、G は二部グラフですが、一般のグラフについても解くことができます。

辺集合 E' の辺がすべて選べるための条件を考えてみます。すると、E' に含まれる辺だけで構築した $G = (V, E)$ の部分グラフ $G' = (V, E')$ におけるどの連結成分についても、その連結成分に含まれる辺の数が頂点数以下であれば必ず全ての辺 (カード) を選ぶ (取る) ことができます。これは、葉から順に割り当てていくことを考えれば (サイクルが 1 つ残る場合がありますが) 可能なことが分かると思います。もしくは、マッチングを考え、Hall の結婚定理を適用すれば得られます。逆に、辺の数が頂点数より大きくなるような連結成分があれば、明らかに全ての辺を選ぶことができません。

さらに、重みの大きい辺から順に選べるなら選ぶことを繰り返す戦略が最適であることがクラスカル法と同様に示せます (マトロイドを知っていれば一発です)。

したがって、重みの順にソートし、選んだ辺だけで構築されるグラフの各連結成分の辺の数と頂点数を UnionFind などで管理しながら重みの大きい辺から順に追加できるなら追加することを繰り返すことで答えを求めることができます。時間計算量は $O(NlogN + N\alpha(\min\{N, H + W\}))$ です[*1]。

[*1] α はアッカーマン関数の逆関数です。

F: Candy Retribution

常識として，「n 人に r 個以下配る」組合せは ${}_{n+r}C_r$ 通りです [*2].

以下では，和が K 以下で，2 つ目の条件を**満たさない**ものを数えることにします (これを $K = L-1, R$ についてそれぞれ求めればあとは簡単です).

2 つ目の条件を満たさないことは，とある整数 x に対し，ある M 要素の最小値が x で，残りの $N-M$ 要素が $x-1$ 以下であることと同値です．この個数は，「ある M 要素が x 以上で…」の数から「ある M 要素が $x+1$ 以上で…」の数を引くことで求まります．

「ある M 要素が a 以上で，残りの $N-M$ 要素が b 以下である $(a > b)$」ものの個数は，以下のようにして求まります．

- 大きい方の M 要素の位置の選び方は ${}_NC_M$ 通り．
- 小さい方の $N-M$ 要素に関する条件はそのままでは扱いにくいので，代わりに「この i 個は約束を破って $b+1$ 以上 (他の要素はどうでもいい)」という条件で解き，包除原理を適用する．
- 選ぶ個数 i に対して，${}_{N-M}C_i$ 通りある選び方のいずれも結果は同じ．
- 各 i に関して，あらかじめ大きい M 要素に a ずつ，約束を破る i 要素に $b+1$ ずつ配ることで，ただの重複組合せと上 2 つの積となり容易に産出できる．

各 $b = x-1$ に対する計算量は $O(\min(\frac{K}{b+1}, N-M))$ となります．$b = x-1$ は高々 K なので，雑に見積もっても $O(K \log K)$ 時間と，十分高速です．

[*2] ゴミ箱も 1 人に数えて「$n+1$ 人にちょうど r 個配る」と言い換えることで，重複組合せの公式が適用できます．

A: Takahashi Calendar

We can found the answer by, for each $m = 1, 2, ..., M$ and $d = 1, 2, ..., D$, checking if the date m-d is a Product Day and incrementing the answer by 1 each time we found a Product Day. To do so, we need to extract d_1 and d_{10} from d, which can be found as the quotient and remainder when dividing d by 10. Sample C++ implementation follows:

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  int M, D; cin >> M >> D;
  int ans = 0;
  for (int m = 1; m <= M; ++m) {
    for (int d = 10; d <= D; ++d) {
      int d_1 = d % 10, d_10 = d / 10;
      if (d_1 >= 2 && d_10 >= 2 && d_1 * d_10 == m) {
        ++ans;
      }
    }
  }
  cout << ans << endl;
}
```

With a little more thought, we can implement it with a single loop, by noticing that $1 \leq d_1 \times d_{10} \leq M$ if and only if there exists m $(1 \leq m \leq M)$ such that $d_1 \times d_{10} = m$. Note that, when $d_1, d_{10} \geq 2$, $d_1 \times d_{10} \geq 1$.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  int M, D; cin >> M >> D;
  int ans = 0;
  for (int d = 10; d <= D; ++d) {
    int d_1 = d % 10, d_10 = d / 10;
    if (d_1 >= 2 && d_10 >= 2 && d_1 * d_10 <= M) {
      ++ans;
    }
  }
  cout << ans << endl;
}
```

B: Kleene Inversion

Let the K copies of A that form B be $A_1,\ A_2,\ ...,\ A_K$, in the order they occur in B. There are two kinds of inversions in B: those that occur in a single A_i, and those that occur between A_i and A_j. The number of the former kinds of inversion is the number of inversion in A multiplied by K, which can be naively found in $\mathrm{O}(N^2)$ time. The number of the latter kinds of inversion is the number of inversion occurring between two copies of A multiplied by $\frac{K\times(K-1)}{2}$ (the number of ways to choose two of the K copies). This can be found by counting, for each element in A, the number of elements in A smaller than that, which can be done naively in $\mathrm{O}(N^2)$ time. Thus, we can compute the inversion number of B in a total of $\mathrm{O}(N^2)$ time.

C: Cell Inversion

First, changing the order of operations does not affect the result. Let (l, r) denote the operation of choosing Squares l and r $(l < r)$. Then, for two operations $(l_1, r_1), (l_2, r_2)$, doing them in the order $(l_1, r_1), (l_2, r_2)$ or $(l_1, r_2), (l_2, r_1)$ does not matter. That is, what only matters is whether each square is chosen as l or r. Actually, if we choose Square i as l, it only changes whether the colors of Square $i-1$ and Square i are the same, and if we choose Square i as r, it only changes whether the colors of Square i and Square $i+1$ are the same.

Let us define d_i as follows: if we should choose Square i as l, d_i is L; if we should choose it as r, d_i is R. First, the leftmost square is always chosen as l, so $d_1 =$ L. Then, for $i = 2, 3, ..., 2N$, $d_i \neq d_{i-1}$ if $S_{i-1} = S_i$, and $d_i = d_{i-1}$ if $S_{i-1} \neq S_i$. For example, if S is BWWWWB, $d_1 d_2 ... d_{2N}$ will be LLRLRR. If S is WBBBBW, $d_1 d_2 ... d_{2N}$ will be again LLRLRR, but if the leftmost or the rightmost square is white, we cannot make that square white again. We cannot make all the squares white when the number of L and the number of R differ, either.

What remains is to count the number of ways resulting in $d_1 d_2 ... d_{2N}$. Let us consider the squares with $d_i =$ R from left to right. Let L_i and R_i be the number of L and R to the left of Square i (not counting itself), respectively. Then, for the leftmost square with $d_i =$ R, there are L_i candidates for its counterpart with $d_j =$ L, and for the next square with $d_i =$ R, there are $L_i - 1$ candidates. More generally, for each square with $d_i =$ R, there are $L_i - R_i$ candidates for its counterpart. The product of these number is the number of possible sequences of operations $(l_1, r_1), (l_2, r_2), ..., (l_{2N}, r_{2N})$ such that $r_1 < r_2 < ... < r_{2N}$ (that is, possible sets of operations). Considering the possible orders in which we perform these operations, the answer is the above number multiplied by $N!$.

The time complexity of our solution is $O(N)$.

D: Classified

Let us call a configuration of levels *k-classifying* when it only uses levels 1 through k.

The conclusion is, the maximum N for which k-classifying is possible is $N = 2^k$, that is:

- If $N = 2^k + 1$, k-classifying is impossible.
- If $N = 2^k$, k-classifying is possible.

We will show the former inductively. It obviously holds for $k = 1, N = 2^k + 1 = 3$, so let us assume that it holds for $k = x - 1$ and show that it also holds for $k = x, N = 2^x + 1$.

The given condition can be rephrased as follows: for any level, the passages of that level forms a bipartite graph. A bipartite graph has the following property: we can paint the vertices in a bipartite graph with two colors so that there is no edge between vertices of the same color. When there are $2^x + 1$ vertices, there always exist $2^{x-1} + 1$ vertices painted in the same color (because $2^x + 1 > 2^{x-1} \times 2$), so in order for the case $N = 2^x + 1$ to be x-classifiable, the case $N = 2^{x-1} + 1$ must be $(x - 1)$-classifiable. However, we assumed that it was impossible, so the case $N = 2^x + 1$ is not x-classifiable.

The latter statement can be shown easily. Assume that the room are numbered 0 through $2^k - 1$. We can set the level of the passage connecting Room i and j to be any x such that the x-th lowest bits of i and j differ. For example, we can set the level of the passage connecting Room $1101_{(2)}$ and $0111_{(2)}$ to be 2 or 4. We can easily see that this strategy satisfies the condition, and implementing it is also easy.

E: Card Collector

Consider a graph G where each vertex corresponds to a row or a column, and for each i, there is an edge of weight A_i connecting the vertex corresponding to the R_i-th row and the C_i-th column. Then, the problem can be rephrased as follows: for each vertex, we choose at most one of the edges incident to that vertex, and we want to maximize the total weight of the chosen edges. G is bipartite, but we can solve the problem for general graphs.

Consider the condition necessary for an edge set E' to be chosen. If, in every connected component of the subgraph $G' = (V, E')$ of $G = (V, E)$, the number of edges contained in it is at most the number of vertices, we can choose all the edges in E' (or, take all the corresponding cards). We can see this by assigning an edge to a vertex one by one starting from the leaves (possibly ending up with a cycle), or applying Hall's theorem. On the other hand, if there is a connected component where the number of edges contained in it is more than the number of vertices, we obviously cannot choose such an edge set.

Additionally, we can show that the following strategy is optimal: consider the edges one by one in decreasing order of weight, and take the edge whenever possible. The proof is similar to that of Kruskal's algorithm (if you know matroids, it's trivial).

Thus, we can solve the problem in this manner, maintaining the number of edges and vertices in each of the connected components of the subgraph formed by the chosen edges, using UnionFind or such, with the time complexity of $\mathrm{O}(N \log N + N\alpha(\min\{N, H + W\}))$ [*3].

[*3] α is the inverse ackermann function.

F: Candy Retribution

Sorry, being translated. Will be ready in an hour.