ABC 142 解説

beet, drafear, kort0n, ynymxiaolongbao

2019 年 9 月 28 日

For International Readers: English editorial starts on page 7.

A: Odds of Oddness

実数 x について,x を超えない最大の整数を $floor\left(x\right)$ と定義します.N 以下の正の奇数は $N - floor\left(\frac{N}{2}\right)$ 個数存在しますから, 答えは $\left(N - floor\left(\frac{N}{2}\right)\right)/N$ です.
ただし, 実装には注意が必要です. 例えば C++ では,N を int 型の変数とした場合 $floor\left(\frac{N}{2}\right)$ は $N/2$ で計算出来ますが,$\left(N - N/2\right)/N$ のように書くと, 二度目の除算も int 型の変数同士の除算となり, 正しい計算結果が得られません.
これを回避する手法はいくつかあります。例えば, 二度目の除算が double 型の変数同士の除算として実行されるように, 型変換を行えば良いです.
より詳しくは, AtCoder Programming Guide for beginners Y-3.01. 数値型 をご参照ください。
C++ による解答例:https://atcoder.jp/contests/abc142/submissions/7731173

B: Roller Coaster

N 個の整数を入力し、そのうち K 以上の整数がいくつあるかを数える問題です。
各整数について K 以上であるかを判定すればいいです。
以下は C++ における実装例です。

```cpp
#include <iostream>
using namespace std;
int main(){
  int N,K;
  cin>>N>>K;

  int cnt=0;
  for(int i=0;i<N;i++){
    int h;
    cin>>h;
    if(h>=K) cnt++;
  }

  cout<<cnt<<endl;
  return 0;
}
```

C: Go to School

与えられた順列に対し、各要素のソート後のインデックスを求める問題です。
$rev[A_i] = i$ となるような配列 rev を求めればいいです。
以下は C++ における実装例です。

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
  int N;
  cin>>N;
  vector<int> A(N);
  for(int i=0;i<N;i++) cin>>A[i];

  vector<int> rev(N);
  for(int i=0;i<N;i++) rev[A[i]-1]=i+1;
  for(int i=0;i<N;i++) cout<<rev[i]<<endl;
  return 0;
}
```

D: Disjoint Set of Common Divisors

x, y が互いに素であるのと、x, y の最大公約数が 1 であることは同値です。さらに、$gcd(x, y)$ を x と y の最大公約数とすれば、$gcd(a, x) \leq gcd(ab, x)$ です。したがって、整数 ab $(a, b > 1)$ を選ぶよりも、a を選んだほうが良いです (注意: ab が公約数ならば a も公約数です)。すなわち、P を最適解とすると、P で選んだ公約数の中に ab $(a, b > 1)$ と表せる整数が存在したとすると、代わりに a を選ぶことができます。

よって、1 または素数のみを選んだ最適解が存在します。さらに、1 または素数である公約数を全て選ぶことができるため、これらを全て選ぶのが最適です。

さて、そのような公約数はどのように探せば良いでしょう。考察も実装も簡単な方法としては、A, B をそれぞれ素因数分解し (時間計算量はそれぞれ $O(\sqrt{A}), O(\sqrt{B})$)、共通の素因数を探す方法があります。素因数の個数はそれぞれ $O(\log A), O(\log B)$ 個なので、共通の素因数を愚直に求めても十分間に合います。1 と共通の素因数を選ぶのが最適なので、共通の素因数の個数に 1 を加えた値が答えになります。

考察をもう少し進めると、x が A と B の公約数であることと、x が $gcd(A, B)$ の約数であることは同値なので、$gcd(A, B)$ を素因数分解し、素因数の個数を数え上げる方法もあります。

いずれの方法でも、時間計算量は $O(\sqrt{A} + \sqrt{B})$ となります。

E: Get Everything

動的計画法により解くことが出来ます.
0 以上 2^N 未満の整数 j が,「1 以上 N 以下の整数 k について, 現在購入している鍵で宝箱 k を開けられることと,j を 2 進数表記した際に $k-1$ 桁目が 1 であることが同値」という状態を表すこととします.
dp テーブルを以下のように定義します.

$$dp_{i,j} = i+1 \text{ 番目以降の鍵は購入せずに状態を } j \text{ にする為に必要な最小費用}$$

1 以上 N 以下の整数 i について,0 以上 2^N 未満の整数 d_i を,「1 以上 N 以下の整数 k について, 鍵 i で宝箱 k を開けられることと,d_i を 2 進数表記した際に $k-1$ 桁目が 1 であることが同値」となるように定めます. このとき, 動的計画法の遷移式は

$$dp_{i,j \text{ or } d_i} = \min\left(dp_{i-1,j \text{ or } d_i}, dp_{i-1,j} + a_i\right)$$

となります. 時間計算量は $O\left(2^N M\right)$ です.
C++ による解答例:https://atcoder.jp/contests/abc142/submissions/7731316

F: Pure

閉路が存在しないとき、条件を満たす誘導部分グラフは存在しません。そうでないとき、以下の手順を実行します。

手順 1 ： 閉路をひとつ探し、閉路に含まれる頂点の誘導部分グラフを残す。

手順 2 ： そのグラフが条件を満たすとき、終了する。

手順 3 ： 閉路に用いられない辺が存在するので、その辺の終点から閉路をたどり、その辺の始点に到達したらやめる。このとき通過した頂点集合の誘導部分グラフを残し、手順 2 に戻る。

手順 3 を行うたび誘導部分グラフの頂点数が必ず減少し、また手順 2, 3 を通して誘導部分グラフは常に閉路を持ちます。閉路を持つ頂点数が 2 のグラフは条件を満たすため、必ず条件を満たす誘導部分グラフが得られます。愚直に実装すれば計算量は $O(N(N+M))$ となり、正答できます。

A: Odds of Oddness

For a real number x, let's define $floor\left(x\right)$ as a maximum integer that does not exceeds x. Since there are $N - floor\left(\frac{N}{2}\right)$ odd numbers less than or equal to N, so the answer will be $\left(N - floor\left(\frac{N}{2}\right)\right)/N$.

However, you have to be careful of implementation. For example, in C++, if you define N as a variable of type int, you can calculate $floor\left(\frac{N}{2}\right)$ by $N/2$, but if you write such like $\left(N - N/2\right)/N$, the second division will be a division of ints, so you cannot obtain a right answer.

There are several ways to avoid this. For example, you can apply a type conversion so that the second division will be a division of doubles.

For more information, please google about integer types in C++.

An example solution in C++:https://atcoder.jp/contests/abc142/submissions/7731173

B: Roller Coaster

In this problem you are asked to input N integers and find the number of integers that is more than or equal to K.

You can judge if each integer is more than or equal to K.

The following is an implementation example in C++.

```cpp
#include <iostream>
using namespace std;
int main(){
  int N,K;
  cin>>N>>K;

  int cnt=0;
  for(int i=0;i<N;i++){
    int h;
    cin>>h;
    if(h>=K) cnt++;
  }

  cout<<cnt<<endl;
  return 0;
}
```

C: Go to School

In this problem you are asked to sort a given permutation and find the indices of the elements after sorted.

You can find an array rev such that $rev[A_i] = i$.

The following is an implementation example in C++.

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(){
  int N;
  cin>>N;
  vector<int> A(N);
  for(int i=0;i<N;i++) cin>>A[i];

  vector<int> rev(N);
  for(int i=0;i<N;i++) rev[A[i]-1]=i+1;
  for(int i=0;i<N;i++) cout<<rev[i]<<endl;
  return 0;
}
```

D: Disjoint Set of Common Divisors

x, y are coprime if and only if GCD of x, y is 1. Furthermore, let $gcd(x, y)$ be the GCD of x and y, then $gcd(a, x) \leq gcd(ab, x)$ holds. Therefore, instead of choosing an integer ab $(a, b > 1)$, it is better to choose a (Note: if ab is a common divisor, then a is also a common divisor). That is, given an optimal solution P, if an integer ab $(a, b > 1)$ was contained in the common divisors selected in P, then you can choose a instead.

Therefore, there exists an optimal solution only consiting of 1 and primes. Furthermore, you can choose all the common divisors that is 1 or a prime, so it is optimal to choose all of them.

Then how can we find such common divisors? One way that consideration and implementation are both easy is to factorize A, B into prime factors (it will take a total of $O(\sqrt{A}), O(\sqrt{B})$ time each), and find the common prime factors. The number of prime factors are $O(\log A), O(\log B)$ each, so you can find the common prime factors naively in time. It is optimal to choose 1 and common prime factors, so the answer will be the number of common prime factors added by 1.

With a little more consideration, it appears that x is a common divisor of A and B if and only if x is a divisor of $gcd(A, b)$, so you can factorize $gcd(A, B)$ into prime factors and count the number of prime factors.

Anyway, it can be solved in a total of $O(\sqrt{A} + \sqrt{B})$ time.

E: Get Everything

It can be solved by Dynamic Programming.
Assume that an integer j in the range from 0 (inclusive) to 2^N (exclusive) represents the state that "for each integer k from 1 to N, you can open the treasure box k if and only if the $k-1$-th bit of j is 1 when j is binary notated."
Let define a dp table as follows:

$$dp_{i,j} = \text{minimum cost to achieve the state} j \text{without using} i + 1\text{-th and later keys}$$

For each integer i from 1 to N, let's define an integer d_i in the range from 0 (inclusive) to 2^N (exclusive) such that "for each integer k from 1 to N, you can open the treasure box k if and only if the $k-1$-th bit of j is 1 when d_i is binary notated." Then the recurrence formula will be

$$dp_{i,j \text{ or } d_i} = \min\left(dp_{i-1,j \text{ or } d_i}, dp_{i-1,j} + a_i\right).$$

This solution costs a total of $o\left(2^n m\right)$ time.
An example solution in C++:https://atcoder.jp/contests/abc142/submissions/7731316

F: Pure

If the graph does not contain a cycle, then induced subgraph that satisfies the conditions does not exist. Otherwise, perform the procedure below.

Step 1 : Find a cycle from the graph, and retain the induced subgraph consisting of the vertices on the cycle.

Step 2 : If the graph satisfies the condition, exit the procedure.

Step 3 : There exists an edge that is not part of the cycle. Walk through the cycle from the ending vertex of the edge, until reaching the starting vertex of the edge. Retain the induced subgraph consisting of the vertices you passed through, and go back to Step 2.

Every time Step 3 is performed the number of vertices decreases, and through the Steps 2 and 3, the induced subgraph always contains a cycle. Since a graph consisting of two vertices that contains a cycle satisfies the conditions, so you can always obtain a induced subgraph that satisfies the conditions. If you implement it naively, the time complexity will be $O(N(N+M))$, so you can get accepted.