# رمزگشایی یک اشتباه دردسرساز

ایلشات یک مهندس نرم افزار است که روی موضوع داده‌ساختارهای بهینه کار می‌کند. یک روز، او داده ساختار جدیدی ابداع کرد. این داده‌ساختار می‌تواند مجموعه‌ای از اعداد صحیح $n$ بیتی نامنفی را ذخیره کند که $n$ توانی از دو است. یعنی عدد صحیح نامنفی $b$ وجود دارد که $n = ۲^b$.

داده‌ساختار در ابتدا خالی است. برنامه‌ای که از این داده‌ساختار استفاده می‌کند باید قوانین زیر را رعایت کند:

- برنامه می‌تواند عناصر را که به شکل اعداد صحیح $n$ بیتی هستند، یکی یکی، به کمک تابع `add_element(x)` به داده‌ساختار اضافه کند. اگر برنامه سعی کند عنصری را اضافه کند که پیش از این در داده‌ساختار وجود داشته است، هیچ اتفاقی رخ نمی‌دهد.
- پس از افزودن آخرین عنصر، برنامه باید تابع `compile_set()` را دقیقا یک‌بار صدا کند.
- در نهایت، برنامه می‌تواند با فراخوانی تابع `check_elemnt(x)` وجود عنصر $x$ در داده‌ساختار را بررسی کند. این تابع می‌تواند چندین بار استفاده شود.

هنگامی که ایلشات برای اولین‌بار این داده‌ساختار را پیاده‌سازی کرد، در پیاده‌سازی تابع `compile_set()` مرتکب اشتباهی شد. بر اثر این اشتباه، ارقام دودویی هر عنصر درون مجموعه به ترتیب یکسانی جابه‌جا می‌شوند. ایلشات از شما می‌خواهد تا ترتیب دقیق جابه‌جایی ارقام ناشی از اشتباه را پیدا کنید.

به عبارت دقیق‌تر، دنباله‌ی $p = [p_۰, \ldots, p_{n-۱}]$ را در نظر بگیرید که هر یک از اعداد ۰ تا $n - ۱$ دقیقاً یک‌بار در آن آمده‌اند. به چنین دنباله‌ای یک جایگشت می‌گوییم. عنصری از مجموعه را در نظر بگیرید که ارقام آن در نمایش دودویی $a_۰, \ldots, a_{n-۱}$ است ($a_۰$ پرارزش‌ترین بیت است). هنگامی که تابع `compile_set()` صدا زده می‌شود، این عنصر با عنصر $a_{p_۰}, a_{p_۱}, \ldots, a_{p_{n-۱}}$ جایگزین می‌شود.

جایگشت یکسانی برای جابه‌جایی ارقام تمام عناصر استفاده می‌شود. هر جایگشتی امکان پذیر است، از جمله جایگشت $p_i = i$ به ازای هر $۰ \le i \le n - ۱$.

برای مثال، فرض کنید $n = ۴$، $p = [۲, ۱, ۳, ۰]$، و شما اعدادی را در مجموعه اضافه کرده‌اید که نمایش دودویی آن‌ها 0000، 1100 و 0111 است. فراخوانی تابع `compile_set` این عناصر را به ترتیب به 0000، 0101 و 1110 تبدیل می‌کند.

وظیفه‌ی شما نوشتن برنامه‌ای است که جایگشت $p$ را به کمک تعامل با داده‌ساختار پیدا کند. برنامه باید (به ترتیب زیر):

۱. مجموعه‌ای از اعداد صحیح $n$ بیتی را انتخاب کند،

۲. این اعداد را به داده‌ساختار اضافه کند،

۳. تابع `compile_set` را صدا کند تا موجب رخ دادن اشتباه شود،

۴. وجود برخی از عناصر را در مجموعه‌ی تغییر یافته بررسی کند،

۵. از این اطلاعات برای تعیین و بازگرداندن جایگشت $p$ استفاده کند.

توجه کنید که برنامه‌ی شما تنها می‌تواند یک‌بار تابع `compile_set` را صدا کند.
به‌علاوه، در تعداد دفعات فراخوانی توابع کتابخانه توسط برنامه‌ی شما محدودیت وجود دارد. یعنی برنامه‌ی شما می‌تواند

- تابع `add_element` را حداکثر $w$ بار صدا بزند ($w$ ابتدای کلمه‌ی writes به معنای نوشتن‌ها است)
- تابع `check_element` را حداکثر $r$ بار صدا بزند ($r$ ابتدای کلمه‌ی reads به معنای خواندن‌ها است)

## جزئیات پیاده‌سازی

شما باید یک تابع را پیاده‌سازی کنید:

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: تعداد بیت‌های نمایش دودویی هر عنصر درون مجموعه (و همچنین طول $p$)
  - `w`: بیشترین تعداد عملیات‌های `add_element` که برنامه‌ی شما می‌تواند انجام دهد.
  - `r`: بیشترین تعداد عملیات‌های `check_element` که برنامه‌ی شما می‌تواند انجام دهد.
  - تابع باید جایگشت بازیابی شده‌ی $p$ را برگرداند.

برای زبان C تعریف تابع کمی متفاوت است:

- `void restore_permutation(int n, int w, int r, int* result)`
  - معنای `n`، `w` و `r` با بالا یکسان است.
  - تابع باید جایگشت بازیابی شده‌ی $p$ را با ذخیره کردن آن در آرایه‌ی داده‌شده‌ی `result` برگرداند: به ازای هر $i$، باید مقدار $p_i$ را در `result[i]` ذخیره کند.

## توابع کتابخانه

برای تعامل با داده‌ساختار، برنامه‌ی شما باید از سه تابع زیر استفاده کند:

- `void add_element(string x)`

این تابع یک عنصر که با $x$ نمایش داده می‌شود را به مجموعه اضافه می‌کند.

- `x`: رشته‌ای از `'0'` و `'1'` که نمایش دودویی عدد صحیحی است که باید به مجموعه اضافه شود. طول `x` باید $n$ باشد.

- `void compile_set()`

این تابع باید دقیقا یک‌بار صدا زده شود. برنامه‌ی شما نمی‌تواند `add_element()` را بعد از فراخوانی این تابع صدا بزند. برنامه‌ی شما نمی‌تواند `check_element()` را پیش از فراخوانی این تابع صدا بزند.

- `boolean check_element(string x)`

این تابع وجود عنصر $x$ در مجموعه‌ی تغییر یافته را بررسی می‌کند.

- `x`: رشته‌ای از `'0'` و `'1'`، نمایش دودویی عدد صحیحی که باید مورد بررسی قرار بگیرد. طول `x` باید $n$ باشد.

- اگر x در مجموعه‌ی تغییر یافته باشد true برمی‌گرداند و درغیراین‌صورت، false برمی‌گرداند.

توجه کنید که اگر برنامه‌ی شما هر یک از محدودیت‌های بالا را رعایت نکند، نتیجه‌ی ارزیابی آن «پاسخ غلط ـ Wrong Answer» خواهد بود.
برای تمامی رشته‌ها، اولین کاراکتر پرارزش‌ترین بیت عدد متناظر رشته است.
ارزیاب نمونه جایگشت $p$ را پیش از فراخوانی تابع restore_permutation تعیین می‌کند.
لطفا از فایل‌های قالب ارائه شده برای جزئیات پیاده‌سازی در زبان برنامه‌نویسی خود استفاده کنید.

## مثال

ارزیاب تابع را به شکل زیر صدا می‌زند:

- restore_permutation(4, 16, 16). در این حالت $n = ۴$ و برنامه می‌تواند حداکثر ۱۶ «نوشتن» و ۱۶ «خواندن» انجام دهد.

برنامه تابع‌ها را به ترتیب زیر فراخوانی می‌کند:

- add_element("0001")
- add_element("0011")
- add_element("0100")
- compile_set()
- check_element("0001") مقدار false برمی‌گرداند
- check_element("0010") مقدار true برمی‌گرداند
- check_element("0100") مقدار true برمی‌گرداند
- check_element("1000") مقدار false برمی‌گرداند
- check_element("0011") مقدار false برمی‌گرداند
- check_element("0101") مقدار false برمی‌گرداند
- check_element("1001") مقدار false برمی‌گرداند
- check_element("0110") مقدار false برمی‌گرداند
- check_element("1010") مقدار true برمی‌گرداند
- check_element("1100") مقدار false برمی‌گرداند

تنها یک جایگشت با مقادیر برگردانده‌شده توسط check_element() سازگار است: جایگشت $p = [۲, ۱, ۳, ۰]$.
بنابراین، restore_permutation باید $[۲, ۱, ۳, ۰]$ را برگرداند.

## زیرمسئله‌ها

۱. (۲۰ امتیاز): $n = ۸$، $w = ۲۵۶$، $r = ۲۵۶$، حداکثر برای ۲ اندیس $i$ ، $p_i \neq i$ $(۰ \leq i \leq n - ۱)$

۲. (۱۸ امتیاز): $n = ۳۲$، $w = ۳۲۰$، $r = ۱۰۲۴$.

۳. (۱۱ امتیاز): $n = ۳۲$، $w = ۱۰۲۴$، $r = ۳۲۰$.

۴. (۲۱ امتیاز): $n = ۱۲۸$، $w = ۱۷۹۲$، $r = ۱۷۹۲$.

۵. (۳۰ امتیاز): $n = ۱۲۸$، $w = ۸۹۶$، $r = ۸۹۶$.

## ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: اعداد صحیح $n$، $w$، $r$،
- خط ۲: $n$ عدد صحیح، عناصر $p$.