۱ برنامه‌نویسی

از آن‌جایی که $0 \leq a_i < m$ پس بعد از حداکثر m مرحله طبق اصل لانه‌ی کبوتری به یک عدد تکراری می‌رسیم. فرض کنید $a_j = a_i (j > i)$ پس $a_{j+k} = a_{i+k}$. بنابراین بعد از رسیدن به اولین عدد تکراری، دیگر به عددی خارج از مجموعه‌ی اعدادی که تاکنون به آن رسیده‌ایم نمی‌رسیم. پس کافی است دنباله‌ی a_i را تا زمانی که یا به a_t برسیم یا به عنصر تکراری محاسبه کنیم و بیشترین عدد دربین اعداد محاسبه شده را خروجی دهیم. از آن‌جا که بعد از حداکثر m مرحله حتما به عدد تکراری می‌رسیم الگوریتم بالا از $O(min(m, t))$ است.

۲ شیرینی

در این سوال هیکاپ باید تا جایی که می تواند به رستوران شازززز برود تا مقدار غذایی که هر بار می خورد به ماکسیمم حالت خود برسد سپس تا جایی که می تواند به رستوران زززاش برود.

برای اثبات درستی این روش طبق برهان خلف فرض کنید روش بهتری وجود داشته باشد آنگاه بهترین روش را در نظر بگیرید. اگر در این روش یک روز وجود داشته باشد که هیکاپ به رستوران زززاش رفته سپس بعد از آن اولین روزی که غذا خورده در رستوران شازززز بوده است، می توانیم به روش بهتری دست پیدا کنیم. برای این کار فرض کنید قبل از این روز x کیلو غذا خورده باشد. بعد از این دو وعده که اولی در زززاش بوده و $\frac{x}{۲}$ کیلو غذا خورده و دومی در شازز بوده و x کیلو غذا خورده در مجموع $\frac{۳x}{۲}$ واحد غذا خورده و همین طور مقدار غذایی که آخرین بار خورده x کیلو است.

اما اگر به زززاش نرود و مستقیما به شازززز برود در آنجا $۲x$ واحد که از $\frac{۳x}{۲}$ بیش تر است غذا می خورد و آخرین بار هم $۲x$ تا غذا خورده پس در آینده هم اشتهای بیش تری دارد. پس به روش بهتری دست پیدا می کنیم.
پس هیکاپ حتما باید ابتدا چندبار به رستوران شازززز برود و سپس چندبار به رستوران زززاش برود. هر روش را با یک جفت (x,y) مشخص می کنیم که یعنی هیکاپ ابتدا x بار به شازززز رفته و سپس y بار به زززاش می رود.
در این صورت مقدار غذا خوردن در روش (x,y) را با $f(x,y)$ نشان می‌دهیم. می‌دانیم:

$$f(x,y) = m * (۲^۱ + ۲^۲ + ... + ۲^x) + m * ۲^x * (\frac{۱}{۲} + \frac{۱}{۲^۲} + ... + \frac{۱}{۲^y})$$

همچنین می دانیم به ازای هر $q \in \mathbb{N}$

$$\frac{۱}{۲} + \frac{۱}{۲^۲} + ... + \frac{۱}{۲^q} < ۱$$

از این دو گزاره نتایج زیر به دست می آید:

$$f(x,۰) > f(x-۱,y) x,y \in \mathbb{Z}$$

$$f(x,y) > f(x,y-۱)$$

پس روش بهینه روشی است که در آن هیکاپ بیشترین تعداد ممکن به شازززز برود. در بین تمام چنین روش‌هایی روش بهینه روشی است که در آن بیشترین تعداد ممکن به زززاش رفته است.
حال برای پیاده سازی این راه ابتدا یک تابع تعریف می کنیم.به این تابع اسم رستوران و یک $t۱$ می دهیم و به ما می گوید اگر بخواهیم از زمان $t۱$ به بعد فقط به رستوران داده شده برویم حداکثر چند بار می توانیم برویم و مقدار $t۲$ که حداقل زمان لازم که در آن می توانیم حداکثر بار به آن رستوران برویم را نیز حساب می کند. همین طور ترتیب دعوت هایی که باید قبول کنیم تا این حالت پیش بیاید را بدهد. سپس ابتدا رستوران شازززز و $t۱ = ۰$ را به این تابع می دهیم سپس رستوران زززاش و همین طور $t۱ = t۲ + ۱$ را به این تابع می دهیم. حال اگر دفعه ی اول تابع دنباله ی $a_۱, a_۲, ..., a_{k۱}$ و دفعه ی بعد دنباله ی $b_۱, b_۲, , b_{k۲}$ را بدهد آنگاه هیکاپ باید به ترتیب ابتدا به رستوران های $a_۱, a_۲, ..., a_{k۱}$ و سپس به رستوران های $b_۱, b_۲, , b_{k۲}$ برود.
برای پیاده سازی تابع مورد نظر ابتدا دعوت های برای رستوران داده شده را بر اساس شروعشان مرتب می‌کنیم.
حال فرض کنید تا زمان قبل p یک سری از رستوران‌ها را انتخاب کرده‌ایم و تمام رستوران هایی که می توانستیم در زمان $p-۱$ انتخاب کنیم(به جز آن‌هایی که تا الان انتخاب کردیم) را در s که یک set است قرار داده ایم. حال تا زمانی که دعوتی در s بود که پایانش حداکثر p بود آن را حذف می کنیم(دعوت هایی که در s قرار دارند به ترتیب پایانشان از کوچک به بزرگ قرار دارند.) اگر s خالی بود p را آنقدر جلو می بریم تا به شروع اولین دعوتی که تا

به قبل p نمی توانستیم برویم برسیم. حال تا زمانی دعوتی باشد که زمان شروعش p باشد آن را درون s قرارداده و از لیستمان حذفش می کنیم. حال در بین تمام دعوت های درون s یکی را انتخاب می کنیم که زود ترین شروع را داشته باشد که با توجه به اینکه زودتر از بقیه تمام می شود برداشتن آن از بقیه بهتر است. آن را انتخاب کرده و در لیست جواب قرار می دهیم و آن را از s حذف می کنیم و p را یکی افزایش می دهیم. مقدار p هنگامی که آخرین دعوت را قبول می کنیم همان $t۲$ است که تابع می خواست بدست آورد. زمان این راه حل از $O(nlogn)$ است.

۳ پل

فرض کنید افراد با شماره های ۰ تا $n-1$ هستند. مجموعه‌ی متناظر با یک عدد $x(0 \leq x < 2^n)$ را مجموعه‌ای تعریف می‌کنیم که شامل تمام افرادی مانند y است که اگر x را در مبنای ۲ بنویسیم در مکان yم عدد یک نوشته شده باشد.

برای حل این سوال ابتدا dp_i را تعریف می کنیم. dp_i برابر است با حداقل زمان لازم برای آنکه تمام افراد درون مجموعه‌ی متناظر i از پل عبور کنند. جواب برابر dp_{2^n-1} است که معادل با حداقل زمان عبور همه از پل می باشد.حال برای پر کردن $dp_0, dp_1, ..., dp_{2^n-1}$، ابتدا dp_0 را به عنوان پایه صفر می گذاریم و سپس بعد از آن برای هر $0 < i < 2^n$

$$dp_i = min(dp_{i-mask} + f_{mask})$$

این مینیموم‌گیری بین تمام $mask$هایی که مجموعه‌ی متناظر آن زیرمجموعه‌ی مجموعه‌ی متناظر i است و مجموع وزن افراد درون این مجموعه کمتر از w است انجام می‌شود. f_{mask} نیز زمان لازم برای عبور این افراد از پل است(که برابر با ماکسیمم زمان عبور هر یک از آن‌ها از پل است).

یعنی به ازای هر $mask$ که مجموعه‌ی متناظرش زیرمجموعه‌ی مجموعه‌ی متناظر i است اگر بتوانیم تمام آن ها را یکباره از پل رد کنیم و مجموع وزنشان از w کمتر باشد آنان را عبور دهیم و بقیه را با مینیمم حالت عبور بدهیم. در این حالت تمام راه هایی که می توان عبور داد را حساب کرده و مینیمم آن ها را می گیریم پس حالت بهینه را هم حساب کردیم و تمام حالت هایی که حساب می کنیم هم قابل اجرا هستند.

تعداد جفت‌های $(i, mask)$ که مجموعه‌ی متناظر $mask$ زیرمجموعه‌ی مجموعه‌ی متناظر i است $O(3^n)$ است(چرا؟) پس اگر بتوانیم تمام چنین جفت‌هایی را در $O(3^n)$ محاسبه کنیم به راحتی می‌توانیم dp_{2^n-1} را محاسبه کنیم. برای این‌کار می توانید از حلقه‌ی زیر استفاده کنید:

```
for(int mask = i; mask; mask = (mask - 1) & i)
```

اثبات آن‌که این حلقه تمامی $mask$هایی را که مجموعه‌ی متناظرش زیرمجموعه‌ی مجموعه‌ی متناظر i است تولید می کند، بر عهده‌ی شماست. روش دیگر برای تولید چنین $mask$هایی محاسبه‌ی مجموعه‌ی متناظر i و سپس استفاده از تابع بازگشتی برای محاسبه‌ی تمام زیرمجموعه‌های آن است.

۴ بازی

به ازای یک زیرمجموعه‌ی S از رنگ‌ها$f(S)$ را برابر جفت (x,y) تعریف می‌کنیم که x و y به شکل زیر به دست می‌آیند:

$$(\sum_{i\in S} a_i) = x$$

$$(\sum_{i\notin S} b_i) = y$$

یک جفت (x,y) را زشت می‌نامیم اگر به ازای یک زیرمجموعه‌ی S $f(S) = (x,y)$. همچنین S را مجموعه‌ی متناظر این جفت می‌نامیم.(اگر یک جفت چند مجموعه‌ی متناظر داشت یکی را به دلخواه مجموعه‌ی متناظر آن در نظر می‌گیریم).

می‌خواهیم ثابت کنیم اگر هیکاپ جفت عدد (a,b) را به مسئول مسابقه اعلام کند حتما می برد اگر و تنها اگر هیچ جفت زشت (c,d) نباشد که $a \le c, b \le d$ و همچنین جفت مورد نظر جفت معتبری باشد یعنی حداقل a توپ در کیسه‌ی اول و b توپ در کیسه‌ی دوم موجود باشد و همچنین $a,b \ge ۱$. در صورتی که چنین جفت زشتی وجود داشته باشد، مسئول مسابقه می‌تواند توپ‌های کیسه‌ی اول را از توپ‌هایی که رنگشان در S آمده بردارد و توپ‌های کیسه‌ی دوم را از توپ‌هایی که رنگشان در S نیست بردارد. پس این شرط، لازم است.

برای اثبات کافی بودن فرض کنید مسئول مسابقه بتواند طوری a توپ از کیسه‌ی اول و b توپ از کیسه‌ی دوم بردارد که توپی از کیسه‌ی اول با توپی از کیسه‌ی دوم هم‌رنگ نشود. اگر مجموعه‌ی رنگ‌های توپ‌های برداشته شده از کیسه‌ی اول را S بنامیم $f(S)$ جفت زشتی است که خاصیت گفته شده را دارد که این تناقض است. پس این شرط کافی نیز هست. پس باید جفت با چنین خاصیتی را پیدا کنید که مجموع دو عدد آن کمینه شود.

اول از همه باید تنها جفت‌هایی مانند (a,b) که را بررسی کنیم که $c \in \mathbb{Z}$ وجود داشته باشد که $(a-۱,c)$ جفت زشت باشد. چرا که در غیر این صورت حتما $(a-۱,b)$ نیز این خاصیت را دارد ولی مجموع دو عدد درون آن کمتر است که درنتیجه جفت بهتری است. پس کافی است تمامی جفت‌های زشت را به دست بیاوریم و آن‌ها را براساس عنصر اولشان از بزرگ به کوچک مرتب کنیم. سپس روی آن‌ها حرکت کنیم و وقتی به یک جفت (a,b) می‌رسیم اگر بیشینه‌ی دومین عنصر در جفت‌های قبلی mx باشد جفت $(a+۱,mx)$ را به گزینه‌ها اضافه کنیم و سپس از بین گزینه‌ها یکی از بهترین جفت‌ها را خروجی دهیم. چون تعداد جفت‌های زشت حداکثر برابر زیرمجموعه‌ی رنگ‌ها یعنی ۲^n است پس در کل الگوریتم از $O(۲^n)$ است.