تعطیلات

جیان-جیا می‌خواهد برای تعطیلات بعدی خود در تایوان برنامه‌ریزی کند. در طی تعطیلات، جیان-جیا از شهری به شهر دیگر می‌رود و از مکان‌های دیدنی شهرها بازدید می‌کند.

در تایوان n شهر وجود دارد که همگی در طول یک بزرگراه قرار گرفته‌اند. شهرها به ترتیب با شماره‌های ۰ تا $n-۱$ شماره‌گذاری شده‌اند. شهر i (برای $۰ < i < n-۱$) با شهرهای $i-۱$ و $i+۱$ همسایه (مجاور) است. تنها همسایه‌ی شهر ۰، شهر ۱، و تنها همسایه‌ی شهر $n-۱$، شهر $n-۲$ می‌باشد.

هر شهر دارای تعدادی مکان دیدنی است. جیان-جیا d روز تعطیلات دارد و می‌خواهد تعداد مکان‌های دیدنی که بازدید می‌کند را تا حد ممکن افزایش دهد. جیان-جیا از قبل مشخص کرده است که از کدام شهر، تعطیلات خود را شروع می‌کند. در هر روز از تعطیلات، جیان-جیا یا از همه‌ی مکان‌های دیدنی شهری که در آن قرار دارد بازدید می‌کند یا از آن شهر به شهر مجاورش می‌رود، اما هر دو کار را نمی‌تواند در یک روز انجام دهد. جیان-جیا هرگز مکان‌های دیدنی یک شهر را دوبار بازدید نمی‌کند، حتی اگر چند روز در آن شهر بماند. به جیان-جیا کمک کنید که تعطیلاتش را به گونه‌ای برنامه‌ریزی کند که از بیش‌ترین تعداد مکان‌های دیدنی بازدید کند.

مثال

فرض کنید جیان-جیا یک تعطیلات ۷ روزه دارد و ۵ شهر (طبق جدول زیر) وجود دارد و او می‌خواهد از شهر ۲ شروع کند. در اولین روز، جیان-جیا از ۲۰ مکان دیدنی شهر ۲ بازدید می‌کند. در روز دوم، او از شهر ۲ به شهر ۳ می‌رود و در روز سوم همه‌ی ۳۰ مکان دیدنی شهر ۳ را بازدید می‌کند. سپس جیان-جیا سه روز بعدی را صرف رفتن به شهر ۰ می‌کند و در روز ششم از ۱۰ مکان دیدنی شهر ۰ بازید می‌کند. جیان-جیا در مجموع از $۶۰ = ۱۰ + ۳۰ + ۲۰$ مکان دیدنی بازدید می‌کند که بیش‌ترین تعداد مکان‌های دیدنی است که جیان-جیا می‌تواند در طی ۷ روز تعطیلات با شروع از شهر ۲ از آن‌ها بازدید کند.

city	number of attractions
0	10
1	2
2	20
3	30
4	1

day	action
1	visit the attractions in city 2
2	move from city 2 to city 3
3	visit the attractions in city 3
4	move from city 3 to city 2
5	move from city 2 to city 1
6	move from city 1 to city 0
7	visit the attractions in city 0

مسئله

تابع `findMaxAttraction` را به منظور محاسبه‌ی بیش‌ترین تعداد مکان‌های دیدنی که جیان-جیا می‌تواند در تعطیلاتش بازدید کند، پیاده‌سازی کنید.

- `findMaxAttraction(n, start, d, attraction)`
  - `n`: تعداد شهرها.
  - `start`: شماره‌ی شهر شروع.
  - `d`: تعداد روزهای تعطیلات.
  - `attraction`: آرایه‌ای به طول `n`؛ `attraction[i]` نشان‌دهنده‌ی تعداد مکان‌های دیدنی شهر `i` به ازای $0 \leq i \leq n-1$ است.
  - این تابع باید بیش‌ترین تعداد مکان‌های دیدنی را که جیان-جیا می‌تواند از آن‌ها بازدید کند برگرداند.

زیرمسئله‌ها

در همه‌ی زیرمسئله‌ها می‌دانیم $0 \leq d \leq 2n + \lfloor n/2 \rfloor$، و تعداد مکان‌های دیدنی هر شهر نامنفی است. محدودیت‌های بیش‌تر در جدول زیر آمده است.

subtask	points	n	max number of attractions in a city	starting city
1	7	$2 \leq n \leq 20$	$1,000,000,000$	no constraints
2	23	$2 \leq n \leq 100,000$	100	city 0
3	17	$2 \leq n \leq 3,000$	$1,000,000,000$	no constraints
4	53	$2 \leq n \leq 100,000$	$1,000,000,000$	no constraints

جزئیات پیاده‌سازی

شما باید یک فایل به نام `holiday.c`، `holiday.cpp` یا `holiday.pas` ارسال کنید. این فایل باید تابعی که در بالا توضیح داده شد را به شکل زیر پیاده‌سازی کند. همچنین شما باید هدرفایل `holiday.h` را در برنامه‌ی C/C++ خود درج کنید.

دقت کنید که خروجی ممکن است بزرگ باشد و نوع خروجی `findMaxAttraction` عدد صحیح ۶۴ بیتی است.

برنامه‌ی C/C++

```
long long int findMaxAttraction(int n, int start, int d, int attraction[]);
```

برنامه‌ی پاسکال

```
function findMaxAttraction(n, start, d : longint; attraction : array of
    longint): int64;
```

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: `n`، `start`، `d`
- خط ۲: `attraction[0]`، ...، `attraction[n-1]`

ارزیاب نمونه خروجی تابع `findMaxAttraction` را چاپ می‌کند.

دوست

می‌خواهیم یک شبکه‌ی اجتماعی شامل n نفر که از ۰ تا $n-1$ شماره‌گذاری شده‌اند ایجاد کنیم. برخی از افراد این شبکه با یک‌دیگر دوست هستند. اگر شخص x با شخص y دوست باشد، شخص y نیز با x دوست خواهد بود.

افراد در n مرحله به شبکه اضافه می‌شوند، که این مراحل نیز به ترتیب از ۰ تا $n-1$ شماره‌گذاری شده‌اند. شخص i در مرحله‌ی i اضافه می‌شود. در مرحله‌ی ۰، شخص ۰ به عنوان تنها فرد شبکه اضافه می‌شود. در هر یک از $n-1$ مرحله‌ی بعدی، یک فرد توسط یک **میزبان** به شبکه اضافه می‌شود، که این میزبان می‌تواند هر شخص از اعضای فعلی شبکه باشد. در مرحله‌ی i $(0 < i < n)$، میزبان آن مرحله می‌تواند شخص تازه‌وارد i را با یکی از سه پروتکل زیر به شبکه اضافه کند:

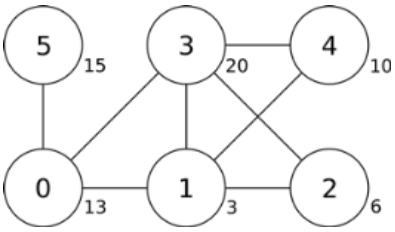
- IAmYourFriend: شخص i را با شخص میزبان دوست می‌کند.
- MyFriendsAreYourFriends: شخص i را با تمام اشخاصی که در آن لحظه با میزبان دوست‌اند، دوست می‌کند. توجه کنید که این پروتکل شخص i را با خود میزبان دوست نمی‌کند.
- WeAreYourFriends: شخص i را با میزبان و تمام اشخاصی که در آن لحظه با میزبان دوست‌اند، دوست می‌کند.

بعد از ساخت شبکه می‌خواهیم یک **نمونه** از افراد را برای نظرسنجی انتخاب کنیم، این نمونه یک زیرمجموعه از کل اعضای شبکه است. از آن جایی که دوستان معمولا علائق مشترکی دارند، در این نمونه نباید هیچ دو فردی که با یک‌دیگر دوست‌اند، انتخاب شوند. هر فرد یک **ضریب اطمینان** دارد که به صورت یک عدد صحیح مثبت نشان داده می‌شود. می‌خواهیم نمونه‌ای را بیابیم که بیش‌ترین مجموع ضریب اطمینان را داشته باشد.

مثال

stage	host	protocol	friend relations added
1	0	IAmYourFriend	(1, 0)
2	0	MyFriendsAreYourFriends	(2, 1)
3	1	WeAreYourFriends	(3, 1), (3, 0), (3, 2)
4	2	MyFriendsAreYourFriends	(4, 1), (4, 3)
5	0	IAmYourFriend	(5, 0)

در ابتدا، شبکه فقط شامل شخص ۰ است. میزبان مرحله‌ی ۱ (شخص ۰) شخص جدید ۱ را با پروتکل IAmYourFriend دعوت می‌کند، و در نتیجه این دو شخص با هم دوست می‌شوند. میزبان مرحله‌ی ۲ (مجددا شخص ۰) شخص ۲ را با پروتکل MyFriendsAreYourFriends دعوت می‌کند، که باعث می‌شود شخص ۱ (تنها دوست میزبان) با شخص ۲ دوست شود. میزبان مرحله‌ی ۳ (شخص ۱) شخص ۳ را با پروتکل WeAreYourFriends اضافه می‌کند، که باعث می‌شود شخص ۳ با شخص ۱ (میزبان) و اشخاص ۰ و ۲ (دو دوست میزبان) دوست شود. مراحل ۴ و ۵ نیز در جدول بالا نشان داده شده‌اند. شبکه‌ی نهایی در شکل زیر نشان داده شده است. در این شکل، اعداد داخل دایره نشان‌دهنده‌ی شماره‌ی اشخاص، و اعداد کنار دایره نشان‌دهنده‌ی ضریب اطمینان اشخاص است. نمونه‌ی متشکل از اشخاص ۳ و ۵ دارای مجموع ضریب اطمینان ۳۵ = ۱۵ + ۲۰ است که بیش‌ترین مقدار ممکن در این شبکه است.



مسئله

با داشتن توصیف تمامی مراحل و ضریب اطمینان اشخاص، نمونه‌ای بیابید که بیش‌ترین مجموع ضریب اطمینان را داشته باشد. شما باید تابع `findSample` را پیاده‌سازی کنید.

- `findSample(n, confidence, host, protocol)`

- `n`: تعداد اشخاص.
- `confidence`: آرایه‌ای به طول n؛ `confidence[i]` نشان‌دهنده‌ی ضریب اطمینان شخص i است.
- `host`: آرایه‌ای به طول n؛ `host[i]` نشان‌دهنده‌ی میزبان مرحله‌ی i است.
- `protocol`: آرایه‌ای به طول n؛ `protocol[i]` پروتکلی است که در مرحله‌ی i ($0 < i < n$) استفاده شده است: `0` برای IAmYourFriend، `1` برای MyFriendsAreYourFriends و `2` برای WeAreYourFriends.
- از آن جایی که مرحله‌ی ۰ هیچ میزبانی ندارد، `host[0]` و `protocol[0]` تعریف نشده‌اند و نباید توسط برنامه‌ی شما مورد دست‌رسی قرار گیرند.
- تابع باید بیش‌ترین مجموع ضریب اطمینان ممکن برای نمونه‌ها را برگرداند.

زیرمسئله‌ها

برخی از زیرمسئله‌ها فقط از زیرمجموعه‌ای از پروتکل‌ها استفاده می‌کنند که در جدول زیر مشخص شده‌اند.

subtask	**points**	n	**confidence**	**protocols used**
1	11	$2 \le n \le 10$	$1 \le \text{confidence} \le 1,000,000$	All three protocols
2	8	$2 \le n \le 1,000$	$1 \le \text{confidence} \le 1,000,000$	Only MyFriendsAreYourFriends
3	8	$2 \le n \le 1,000$	$1 \le \text{confidence} \le 1,000,000$	Only WeAreYourFriends
4	19	$2 \le n \le 1,000$	$1 \le \text{confidence} \le 1,000,000$	Only IAmYourFriend
5	23	$2 \le n \le 1,000$	All confidence values are 1	Both MyFriendsAreYourFriends and IAmYourFriend
6	31	$2 \le n \le 100,000$	$1 \le \text{confidence} \le 10,000$	All three protocols

جزئیات پیاده‌سازی

شما باید دقیقا یک فایل با نام `friend.c`، `friend.cpp` یا `friend.pas` ارسال کنید. این فایل باید تابعی که در بالا توضیح داده شد را به شکل زیر پیاده‌سازی کند. همچنین شما باید هدرفایل `friend.h` را در برنامه‌ی C/C++ خود درج کنید.

برنامه‌ی C/C++

```
int findSample(int n, int confidence[], int host[], int protocol[]);
```

برنامه‌ی پاسکال

```
function findSample(n: longint, confidence: array of longint, host:
   array of longint; protocol: array of longint): longint;
```

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: `n`.
- خط ۲: `confidence[0]`، ...، `confidence[n-1]`
- خط ۳: `host[1]`، `protocol[1]`، `host[2]`، `protocol[2]`، ...، `host[n-1]`، `protocol[n-1]`

ارزیاب نمونه مقدار خروجی تابع `findSample` را چاپ می‌کند.

تله‌کابین

تله‌کابین مائو-کونگ یک جاذبه‌ی گردشگری مشهور در تایپه است. این تله‌کابین از یک ریل دایره‌ای، یک ایستگاه، و n کابین تشکیل شده است که به ترتیب از ۱ تا n شماره‌گذاری شده‌اند و در یک جهت ثابت روی ریل در حال گردش هستند. در ابتدای کار، جهت حرکت به این صورت است که وقتی کابین i از ایستگاه گذشت، کابین بعدی $i+1$ خواهد بود مگر این که $i = n$ که در این حالت کابین بعدی ۱ خواهد بود.

کابین‌ها ممکن است خراب شوند. خوشبختانه ما به تعداد نامتناهی کابین یدکی داریم که با شماره‌های $n+1$، $n+2$ و الی آخر شماره‌گذاری شده‌اند. وقتی یک کابین خراب می‌شود، ما آن را (در همان موقعیت از ریل) با اولین کابین یدکی موجود (کابین با کم‌ترین شماره) جایگزین می‌کنیم. به‌عنوان مثال، اگر ۵ کابین وجود داشته باشد و کابین ۱ خراب شود، ما آن را با کابین ۶ جایگزین می‌کنیم.

شما دوست دارید که در ایستگاه بایستید و گذشتن کابین‌ها را تماشا کنید. یک **دنباله‌ی کابین**، دنباله‌ای از n شماره‌ی کابین است که از ایستگاه می‌گذرند. ممکن است قبل از این که شما برسید، تعدادی از کابین‌ها خراب و جایگزین شده باشند، اما هنگامی که شما در حال تماشای کابین‌ها هستید هیچ کابینی خراب نخواهد شد.

توجه داشته باشید یک وضعیت کابین‌های روی ریل می‌تواند چندین دنباله‌ی کابین تولید کند، بسته به این‌که چه کابینی ابتدا از مقابل شما عبور کند. به‌عنوان مثال اگر هیچ یک از کابین‌ها خراب نشده باشد، هر دو دنباله‌ی کابین $(۱, ۲, ۳, ۴, ۵)$ و $(۳, ۴, ۵, ۱, ۲)$ ممکن است تولید شوند، ولی $(۱, ۵, ۲, ۳, ۴)$ ممکن نیست (چرا که کابین‌ها به ترتیب نادرست ظاهر شده‌اند).

اگر کابین ۱ خراب شود، ممکن است دنباله‌ی کابین $(۳, ۲, ۶, ۵, ۴)$ را مشاهده کنیم. اگر بعد از آن کابین ۴ خراب شود، آن را با کابین ۷ جایگزین می‌کنیم و ممکن است دنباله‌ی $(۵, ۷, ۳, ۲, ۶)$ را مشاهده کنیم. اگر بعد از آن کابین ۷ خراب شود، آن را با کابین ۸ جایگزین می‌کنیم و ممکن است دنباله‌ی $(۲, ۶, ۵, ۸, ۳)$ را مشاهده کنیم.

broken gondola	new gondola	possible gondola sequence
1	6	(4, 5, 6, 2, 3)
4	7	(6, 2, 3, 7, 5)
7	8	(3, 8, 5, 6, 2)

یک **دنباله‌ی جایگزینی**، دنباله‌ای از شماره‌ی کابین‌های خراب‌شده به ترتیب خراب شدن آن‌ها می‌باشد. در مثال قبلی، دنباله‌ی جایگزینی $(۷, ۴, ۱)$ است. می‌گوییم دنباله‌ی جایگزینی r، دنباله‌ی کابین g را **تولید** می‌کند، اگر بعد از خراب شدن کابین‌ها به ترتیب دنباله‌ی جایگزینی r، مشاهده‌ی دنباله‌ی کابین g ممکن باشد.

بررسی دنباله‌ی کابین

در سه زیرمسئله‌ی اول شما باید بررسی کنید که آیا دنباله‌ی ورودی یک دنباله‌ی کابین است یا خیر. در جدول زیر چند نمونه دنباله را مشاهده می‌کنید که بعضی از آن‌ها دنباله‌ی کابین هستند و بعضی دیگر نیستند. شما باید تابع `valid` را پیاده‌سازی کنید.

- `n`: طول دنباله ورودی.
- `inputSeq`: آرایه‌ای به طول n؛ `inputSeq[i]` عنصر i ام دنباله‌ی ورودی را نشان می‌دهد.
- تابع در صورتی که دنباله‌ی ورودی یک دنباله‌ی کابین است باید مقدار ۱ و در غیر این صورت مقدار ۰ را برگرداند.

زیرمسئله‌های ۱، ۲ و ۳

subtask	**points**	n	**inputSeq**
1	5	$n \leq 100$	has each number from 1 to n exactly once
2	5	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq n$
3	10	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 250,000$

مثال‌ها

subtask	**inputSeq**	**return value**	**note**
1	(1, 2, 3, 4, 5, 6, 7)	1	
1	(3, 4, 5, 6, 1, 2)	1	
1	(1, 5, 3, 4, 2, 7, 6)	0	1 cannot appear just before 5
1	(4, 3, 2, 1)	0	4 cannot appear just before 3
2	(1, 2, 3, 4, 5, 6, 5)	0	two gondolas numbered 5
3	(2, 3, 4, 9, 6, 7, 1)	1	replacement sequence (5, 8)
3	(10, 4, 3, 11, 12)	0	4 cannot appear just before 3

دنباله‌ی جایگزینی

در سه زیرمسئله‌ی بعدی شما باید یک دنباله‌ی جایگزینی بسازید که دنباله‌ی کابین داده‌شده را تولید می‌کند. هر دنباله‌ی جایگزینی با این خاصیت پذیرفته می‌شود. شما باید تابع `replacement` را پیاده‌سازی کنید.

- `replacement(n, gondolaSeq, replacementSeq)`
  - `n`: طول دنباله‌ی کابین داده‌شده است.
  - `gondolaSeq`: آرایه‌ای به طول n؛ می‌دانیم که `gondolaSeq` حتما یک دنباله‌ی کابین است، و `gondolaSeq[i]` عنصر i ام دنباله است (برای $0 \leq i \leq n-1$).
  - این تابع باید مقدار l، طول دنباله‌ی جایگزینی، را برگرداند.
  - `replacementSeq`: آرایه‌ای که به اندازه‌ی کافی بزرگ است تا دنباله‌ی جایگزینی را ذخیره کند. شما باید دنباله‌ی جایگزینی خود را با گذاشتن عنصر i ام آن در درایه‌ی `replacementSeq[i]` برگردانید ($0 \leq i \leq l-1$).

زیرمسئله‌های ۴، ۵ و ۶

subtask	points	n	gondolaSeq
4	5	$n \leq 100$	$1 \leq$ gondolaSeq[i] $\leq n+1$
5	10	$n \leq 1,000$	$1 \leq$ gondolaSeq[i] $\leq 5,000$
6	20	$n \leq 100,000$	$1 \leq$ gondolaSeq[i] $\leq 250,000$

مثال‌ها

subtask	gondolaSeq	return value	replacementSeq
4	(3, 1, 4)	1	(2)
4	(5, 1, 2, 3, 4)	0	( )
5	(2, 3, 4, 9, 6, 7, 1)	2	(5, 8)

شمردن دنباله‌های جایگزینی

در چهار زیرمسئله‌ی بعد، شما باید تعداد دنباله‌های جایگزینی ممکن که یک دنباله‌ی داده‌شده را تولید می‌کنند، به پیمانه‌ی ۱,۰۰۰,۰۰۰,۰۰۹ بیابید (دنباله‌ی داده‌شده می‌تواند یک دنباله‌ی کابین باشد یا نباشد). برای این منظور شما باید تابع countReplacement را پیاده‌سازی کنید.

- countReplacement(n, inputSeq)
- n: طول دنباله‌ی ورودی.
- inputSeq: آرایه به طول n؛ inputSeq[i] عنصر i ام از دنباله‌ی ورودی است ($0 \leq i \leq n-1$).
- اگر دنباله‌ی ورودی یک دنباله‌ی کابین باشد، تعداد دنباله‌های جایگزینی را بیابید که این دنباله‌ی کابین را تولید می‌کنند (که این تعداد ممکن است خیلی بزرگ باشد)، و مقدار آن را به پیمانه‌ی ۱,۰۰۰,۰۰۰,۰۰۹ برگردانید. اگر دنباله‌ی ورودی یک دنباله‌ی کابین نباشد، این تابع باید مقدار 0 را برگرداند. اگر دنباله‌ی ورودی یک دنباله‌ی کابین باشد، اما هیچ کابینی خراب نشده باشد، تابع باید مقدار 1 را برگرداند.

زیرمسئله‌های ۷، ۸، ۹ و ۱۰

subtask	points	n	inputSeq
7	5	$4 \leq n \leq 50$	$1 \leq$ inputSeq[i] $\leq n+3$
8	15	$4 \leq n \leq 50$	$1 \leq$ inputSeq[i] ≤ 100, and at least $n-3$ of the initial gondolas $1, \ldots, n$ did not break down.
9	15	$n \leq 100,000$	$1 \leq$ inputSeq[i] $\leq 250,000$
10	10	$n \leq 100,000$	$1 \leq$ inputSeq[i] $\leq 1,000,000,000$

مثال‌ها

subtask	inputSeq	return value	replacement sequence
7	(1, 2, 7, 6)	2	(3, 4, 5) or (4, 5, 3)
8	(2, 3, 4, 12, 6, 7, 1)	1	(5, 8, 9, 10, 11)
9	(4, 7, 4, 7)	0	inputSeq is not a gondola sequence
10	(3, 4)	2	(1, 2) or (2, 1)

جزئیات پیاده‌سازی

شما باید دقیقا یک فایل با نام gondola.c، gondola.cpp یا gondola.pas ارسال کنید. این فایل باید هر سه زیربرنامه‌ای که در بالا توضیح داده شد را به صورت زیر پیاده‌سازی کند (حتی اگر قصد دارید فقط بعضی از زیرمسئله‌ها را حل کنید.) همچنین شما باید هدرفایل gondola.h را در برنامه‌ی C/C++ خود درج کنید.

برنامه‌ی C/C++

```
int valid(int n, int inputSeq[]);
int replacement(int n, int gondolaSeq[], int replacementSeq[]);
int countReplacement(int n, int inputSeq[]);
```

برنامه‌ی پاسکال

```
function valid(n: longint; inputSeq: array of longint): integer;
function replacement(n: longint; gondolaSeq: array of longint; var
   replacementSeq: array of longint): longint;
function countReplacement(n: longint; inputSeq: array of longint):
   longint;
```

ارزیاب نمونه

ارزیاب نمونه ورودی را در قالب زیر می‌خواند:

- خط ۱: T، شماره‌ی زیرمسئله‌ای که برنامه‌ی شما قرار است حل کند ($۱ \leq T \leq ۱۰$).
- خط ۲: n، طول دنباله‌ی ورودی.
- خط ۳: اگر T برابر ۴، ۵، یا ۶ باشد، این خط شامل gondolaSeq[0]، ...، gondolaSeq[n-1] خواهد بود. در غیر این صورت این خط شامل inputSeq[0]، ...، inputSeq[n-1] می‌باشد.