رقابت با زمان

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

شرلوک هلمز در پی موری‌آرتی است و توانسته مکان اختفای او را کشف کند. خیابان‌های لندن شلوغ است و او برای رسیدن به موری‌آرتی باید از n ساختمان که به هم متصل‌اند گذر کند تا او را در پس آخرین ساختمان بیابد. نحوه حرکت شرلوک بدین شرح است که در هر ثانیه می‌تواند حداکثر مسافت k متر به بالا، پایین یا جلو حرکت کند تا به ساختمان بعدی برسد. در صورتی که مسافت مانده تا رسیدن به سقف ساختمان بعدی یا زمین کمتر از k متر باشد، این مسافت باقیمانده را نیز در یک ثانیه طی می‌کند. همچنین در ابتدا شرلوک و موری‌آرتی روی زمین (در ارتفاع صفر) هستند. عرض هر ساختمان نیز برابر با k متر است؛ به این معنا که شرلوک عرض هر ساختمان را در یک ثانیه طی می‌کند. شما باید بگویید حداقل زمان مورد نیاز شرلوک برای رسیدن به موری‌آرتی چقدر است؟

ورودی

ورودی شامل سه خط است. در خط اول، k حداکثر مسافتی که هلمز می‌تواند در یک ثانیه طی کند، داده می‌شود. در خط دوم، عدد n (تعداد ساختمان‌ها) آمده است. در خط سوم ورودی، n عدد آمده که نشان‌دهنده ارتفاع ساختمان‌هاست.

$$1 \leq k, n, a_i \leq 1000$$

خروجی

در خروجی مدت زمانی که طول می‌کشد تا هلمز در سریع ترین حالت به موری‌آرتی برسد را چاپ کنید.

مثال

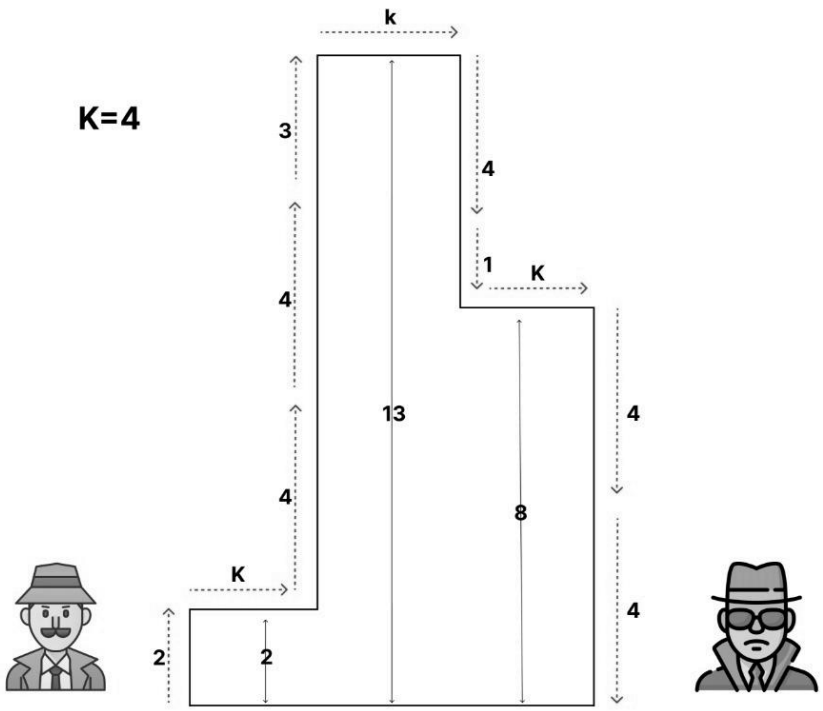
ورودی نمونه ۱

```
4
3
2 13 8
```

خروجی نمونه ۱

```
11
```

▼ توضیحات نمونه ۱



- مقدار k برابر ۴ است. به این معنی که شرلوک، در هر ثانیه حداکثر چهار متر حرکت می‌کند. اگر مسافت باقی‌مانده تا رسیدن به سقف ساختمان بعدی یا زمین کمتر از چهار متر باشد، آن مسافت

را در ۱ ثانیه طی می‌کند تا در سریع ترین حالت به موری‌آرتی برسد.

- در این مثال، هلمز از ساختمان اول که دو متر ارتفاع دارد، طی ۱ ثانیه بالا می‌رود. سپس در ۱ ثانیه عرض ساختمان اول را طی می‌کند. در ۳ ثانیه بعدی یازده متر دیگر از ساختمان دوم بالا می‌رود و ۱ ثانیه از روی آن عبور می‌کند. پس از آن در ۲ ثانیه پنج متر پایین می‌آید تا روی ساختمان سوم برسد و در ۱ ثانیه عرض آن را طی می‌کند. در نهایت هشت متر طول ساختمان سوم را در ۲ ثانیه پایین می‌آید و به موریاتی می‌رسد. بنابراین کل این مسیر $1 + 1 + 3 + 1 + 2 + 1 + 2 = 11$ ثانیه طول می‌کشد.

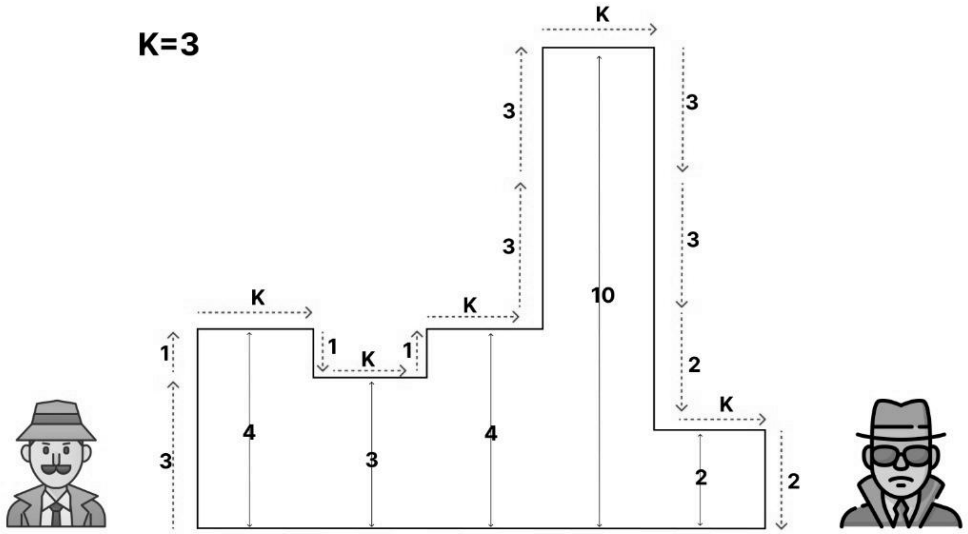
ورودی نمونه ۲

```
3
5
4 3 4 10 2
```

خروجی نمونه ۲

```
15
```

▼ توضیحات نمونه ۲



در این مثال، شرلوک در تلاش برای عبور از پنج ساختمان است و حداکثر مسافتی که شرلوک در یک ثانیه می‌تواند طی کند، برابر سه متر است. همانطور که در تصویر این مثال مشخص شده‌است، مقدار زمانی که نیاز است تا شرلوک تمام ساختمان‌ها را بپیماید، برابر تعداد بردارهای قرمز در تصویر، یعنی ۱۵ ثانیه می‌شود.

جستجوی هالیس

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

شرلوک نامه‌ای از یکی از دوستانش دریافت می‌کند. متن نامه چنین می‌گوید: «شرلوک عزیز، امیدوارم این نامه به موقع به دستتان برسد. من در یک شهر کوچک واقع در سواحل آنتارکتیکا گیر کرده‌ام. قبلا از مسیری گذشته‌ام و دوباره به این شهر رسیده‌ام. اکنون نمی‌دانم از چه مسیری عبور کنم و گم شده‌ام. لطفا سریعا به من کمک کنید. با تشکر، هالیس.»

شرلوک با بررسی دقیق متوجه می‌شود که بعضی از شهرهای آنتارکتیکا با یک جاده به هم متصل هستند و از هر شهر می‌توان با طی کردن تعدادی جاده به هر شهر دیگر رفت. همچنین او یافته است شهری که هالیس در آن قرار دارد متعلق به یک **دور در گراف** آنتارکتیکا است. این گراف بدین گونه ساخته می‌شود: شهرهای آنتارکتیکا رئوس گراف هستند و اگر بین دو شهر جاده ای وجود داشته باشد بین رئوس متناظر آنها یال است.

اگر شهر هایی که روی دور قرار دارند را شهرهای حلقوی بنامیم (رئوسی که متعلق به دور هستند)، کوتاه ترین مسیر از شهر فعلی شرلوک به یک شهر حلقوی را چاپ کنید. اگر چند جواب برای مسئله وجود داشت، یکی از آنها را به دلخواه انتخاب کنید.

ورودی

ورودی شامل $m+3$ خط است. در خط اول، k، شماره شهری که شرلوک در آن قرار دارد می‌آید. در خط دوم، n که تعداد شهرها را نشان می‌دهد آمده است. در خط سوم عدد m داده می‌شود. در m خط بعدی، در هر خط دو عدد با فاصله از هم آمده است که شماره شهرهایی که بین آن‌ها جاده وجود دارد آمده است.

▼ نکته: تضمین می‌شود گراف داده‌شده حداکثر $n+1$ یال داشته باشد و شهری که شرلوک روی آن قرار دارد روی دور نباشد.

$$4 \leq n, m \leq 100$$

خروجی

در خروجی، کوتاه ترین مسیری که شرلوک را به یک شهر حلقوی می‌رساند چاپ کنید. در صورت وجود چند مسیر، یکی از آنها به دلخواه چاپ شود.

مثال

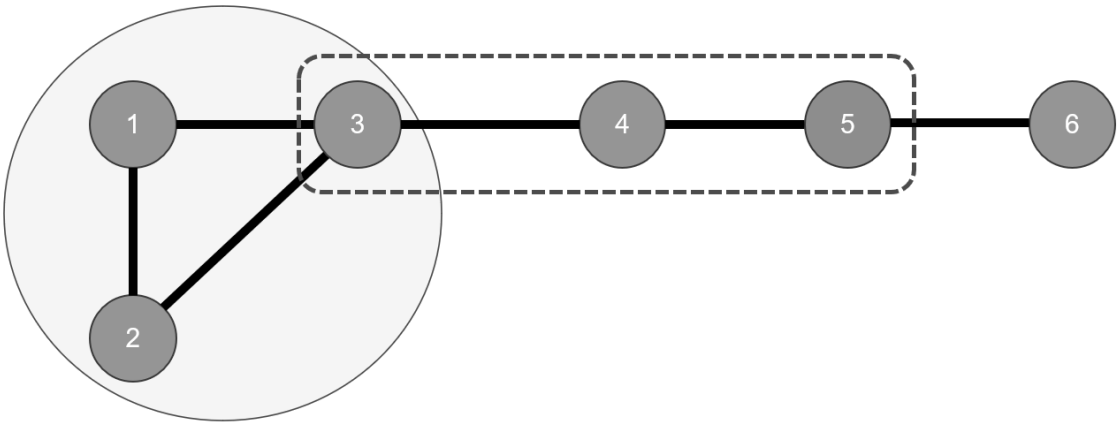
ورودی نمونه ۱

```
5
6
6
1 2
1 3
2 3
3 4
4 5
5 6
```

خروجی نمونه ۱

```
5 4 3
```

▼ توضیحات نمونه ۱



اگر گراف را رسم کنیم، می‌بینیم که یک دور با رئوس ۱، ۲ و ۳ داریم و کوتاه‌ترین مسیر از رأسی که شرلوک روی آن قرار دارد (رأس ۵) به نزدیک‌ترین رأس دور (رأس ۳) مسیر زیر می‌باشد:

```
5 4 3
```

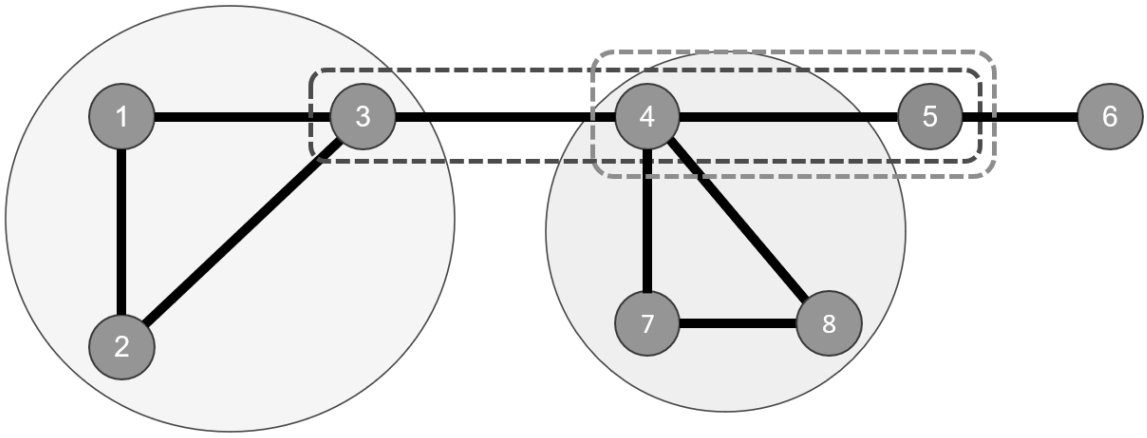
ورودی نمونه ۲

```
5
8
9
1 2
1 3
2 3
3 4
4 7
4 8
7 8
4 5
5 6
```

خروجی نمونه ۲

```
5 4
```

▼ توضیحات نمونه ۲



اگر گراف را رسم کنیم، می‌بینیم که یک دور با رئوس ۱، ۲و ۳ و یک دور با رئوس ۴، ۷ و ۸ داریم و کوتاه‌ترین مسیر از رأسی که شرلوک روی آن قرار دارد (رأس ۵) به نزدیک‌ترین رأس نزدیک‌ترین دور (رأس ۴) مسیر زیر می‌باشد:

5 4

کوتاه‌ترین مسیر به دور دیگر طول ۲ دارد که از مسیر انتخابی ما که طول ۱ دارد بلندتر است پس انتخاب ما دور ۴، ۷ و ۸ می‌باشد.

بازی استخدامی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

شرلوک و موری‌آرتی پس از جنجال‌های بسیار به این نتیجه رسیدند که تبهکاری و حل معما آخر عاقبت ندارد؛ بنابراین تصمیم گرفتند به اصفهان سفر کرده و در شرکت مهیمن مشغول به مهندسی نرم‌افزار شوند. مدیر تیم برای اینکه بتواند مسائل *HR*ی که در آینده بین این دو به وجود می‌آید را کنترل کند، تصمیم گرفته که آن‌ها را در یک بازی دونفره شرکت دهد و برنده را استخدام کند.

بازی بدین صورت است که n جعبه داریم که در هرکدام از آن‌ها مقداری گز وجود دارد. دو نفر با شروع از شرلوک به ترتیب بازی می‌کنند و هرکس در نوبت خود باید یک جعبه را انتخاب کند و k تا گز از جعبه بردارد، به صورتی که k را بتوان به صورت p^a نوشت، که p یک عدد اول و a یک عدد حسابی است. کسی که در نوبت خود نتواند گزی بردارد، می‌بازد. می‌دانیم که هر دونفر به اندازه کافی باهوش هستند و هردو بهترین بازی خود را انجام می‌دهند، همچنین تضمین می‌شود بازی دقیقاً یک برنده دارد. شما باید بگویید که در نهایت چه کسی برنده بازی می‌شود.

ورودی

در خط اول ورودی t آمده که نشانگر تعداد تست‌ها است. سپس در $2 \times t$ خط بعدی تست ها آمده اند. هر تست شامل دو خط است که در خط اول آن عدد طبیعی n که نشان‌دهنده تعداد جعبه‌هاست آمده و در خط دوم a_1 تا a_n با فاصله از هم آمده‌اند؛ a_i نشان‌دهنده تعداد گزهای جعبه iام است.

$$1 \leq t \leq 500$$

$$1 \leq n \leq 1000$$

$$1 \leq a_i \leq 10^9$$

خروجی

در خروجی شما باید نام برنده را خروجی دهید، اگر شرلوک برنده بازی بود، `Sherlock` و در صورتی که موری‌آرتی برنده بود، `Moriarty` را چاپ کنید.

مثال

ورودی نمونه ۱

```
2
1
9
2
1 1
```

خروجی نمونه ۱

```
Sherlock
Moriarty
```

در تست اول، شرلوک می‌تواند در یک حرکت 3^2 گز از تنها جعبه موجود بردارد و از آن‌جا که گزها تمام می‌شوند، دیگر موری‌آرتی نمی‌تواند حرکتی انجام دهد و شرلوک برنده بازی می‌شود.

در تست دوم شرلوک و موری‌آرتی هرکدام در نوبت خود باید p^0 یعنی یک گز از جعبه‌ای بردارند، بنابراین پس از حرکت موری‌آرتی گزها تمام می‌شوند و دیگر شرلوک نمی‌تواند حرکتی انجام دهد و موری‌آرتی برنده بازی می‌شود.

رمزنگار

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- **زبان‌های مجاز:** `Java, C#, C++`

شرلوک پس از کشف راز قتل اندرسون توسط موری‌آرتی تصمیم می‌گیرد اسرار این جنایت مخوف را برای پلیس افشا کند به همین دلیل می‌خواهد نامه‌ای برای بازرس لستراد بنویسد و این راز را برملا کند. موری‌آرتی در شبکه پست مزدورانی دارد که نامه‌ها را به صورت مخفیانه بررسی می‌کنند؛ از آنجا که شرلوک نمی‌خواهد موری‌آرتی متوجه کشف این راز توسط او بشود، تصمیم می‌گیرد که نامه را به صورت رمزنگاری شده برای بازرس ارسال کند. در این سوال شما باید در رمزنگاری نامه به شرلوک کمک کنید!

نگاشت حروف به اعداد

ابتدا باید هر حرف در زبان انگلیسی را مستقل از بزرگ یا کوچک بودن، به یک عدد از 0 تا 25 نگاشت کنیم و در محاسبات رمزنگاری از آن عدد استفاده کنیم.

▼ **مشاهده جدول نگاشت**

حرف	عدد
A	0
B	1
C	2
D	3
E	4

عدد	حرف
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R
18	S
19	T
20	U
21	V
22	W

حرف	عدد
X	23
Y	24
Z	25

الگوریتم‌های رمزنگاری

برای تبدیل متن انگلیسی معنادار (`plaintext`) به متن رمزشده (`ciphertext`) می‌توان از الگوریتم‌های مختلفی استفاده کرد:

▼ **الگوریتم `Additive Cipher`**

توضیح: در این روش، هر حرف متن اصلی با شیفت دادن به تعداد مشخصی از حروف در الفبا، به رمز تبدیل می‌شود.

پارامترها:

- `text` (متن اصلی)
- `key` (کلید، که یک عدد صحیح است و مقدار شیفت را مشخص می‌کند)

مثال:

- `text` : "hello"
- `key` : 3

رمزنگاری:

```
h -> K ((7 + 3) % 26 = 10 -> K)
e -> H ((4 + 3) % 26 = 7 -> H)
l -> O ((11 + 3) % 26 = 14 -> O)
```

```
l -> O ((11 + 3) % 26 = 14 -> O)
o -> R ((14 + 3) % 26 = 17 -> R)
```

نتیجه رمز شده: `"KHOOR"`

دستور:

```
additive-cipher -text "hello" -key 3
```

▼ **الگوریتم `Multiplicative Cipher`**

توضیح: در این روش، هر حرف متن اصلی با ضرب در یک کلید به رمز تبدیل می‌شود.

پارامترها:

- `text` (متن اصلی)
- `key` (کلید، که یک عدد صحیح است و با 26 نسبت به هم اول‌اند)

مثال:

- `text` : "hello"
- `key` : 3

رمزنگاری:

```
h -> V ((7 * 3) % 26 = 21 -> V)
e -> M ((4 * 3) % 26 = 12 -> M)
l -> H ((11 * 3) % 26 = 7 -> H)
l -> H ((11 * 3) % 26 = 7 -> H)
o -> Q ((14 * 3) % 26 = 16 -> Q)
```

نتیجه رمز شده: `"VMHHQ"`

دستور:

```
multiplicative-cipher -text "hello" -key 3
```

▼ **الگوریتم** `Affine Cipher`

توضیح: این روش ترکیبی از رمزنگاری جمعی و ضربی است، به این صورت که هر حرف متن اصلی ضرب در یک کلید می‌شود و سپس به نتیجه کلید دوم اضافه می‌شود.

پارامترها:

- `text` (متن اصلی)
- `a` (کلید ضربی، که یک عدد صحیح است و با 26 نسبت به هم اول‌اند)
- `b` (کلید جمعی، که یک عدد صحیح است)

مثال:

- `text` : "hello"
- `a` : 5
- `b` : 8

رمزنگاری:

```
h -> R ((7*5 + 8) % 26 = 43 % 26 = 17 -> R)
e -> C ((4*5 + 8) % 26 = 28 % 26 = 2 -> C)
l -> L ((11*5 + 8) % 26 = 63 % 26 = 11 -> L)
l -> L ((11*5 + 8) % 26 = 63 % 26 = 11 -> L)
o -> A ((14*5 + 8) % 26 = 78 % 26 = 0 -> A)`
```

نتیجه رمز شده: `"RCLLA"`

دستور:

```
affine-cipher -text "hello" -a 5 -b 8
```

▼ **الگوریتم `Mapping Cipher`**

توضیح: در این روش، هر حرف متن اصلی به یک حرف دیگر در الفبا مطابق با یک نگاشت از پیش تعریف شده تبدیل می‌شود.

پارامترها:

- `text` (متن اصلی)
- `mapping` (نگاشت حروف، به صورت رشته‌ای که ترتیب جدید حروف الفبا را نشان می‌دهد)

مثال:

- `text` : "hello"
- `mapping` : "zyxwvutsrqponmlkjihgfedcba" (نگاشت معکوس الفبا)

رمزنگاری:

```
h -> S
e -> V
l -> O
l -> O
o -> L
```

نتیجه رمز شده: `"SVOOL"`

دستور:

```
mapping-cipher -text "hello" -mapping "zyxwvutsrqponmlkjihgfedcba"
```

همان‌طور که در مثال‌ها آمده است، فرمت کلی هر دستور به شکل زیر است:

```
<cipher-type> -text "<text>" [-key <key>] [-a <a-value>] [-b <b-value>] [-mapp
```

که در آن `cipher-type` ، نوع الگوریتم رمز و سایر موارد، پارامترهای آن الگوریتم رمز هستند. پارامترهایی که بین `[]` آمده‌اند، بسته به الگوریتم رمزنگاری ممکن است در دستور وجود نداشته باشند.

نکات تکمیلی (مهم)

▼ نکته اول

ترتیب پارامترها می‌تواند متفاوت باشد. برای مثال دو دستور زیر معادل هستند:

```
affine-cipher -text "hello" -a 5 -b 8
affine-cipher -a 5 -text "hello" -b 8
```

▼ نکته دوم

فاصله‌های اطراف متن معنادار نادیده گرفته می‌شوند و در متن رمزشده مشاهده نمی‌شوند. اما فاصله‌های بین کلمات دقیقا در متن رمز شده می‌آید:

```
Plaintext: " please help me    "
Ciphertext: "qmfbtf ifmq nf"
```

▼ نکته سوم

بزرگی یا کوچکی حروف در متن معنادار مهم نیست. بنابراین متن رمزشده برای دو متن معنادار `hello` و `Hello` یکسان خواهد بود.

▼ توجه: لازم است اصول برنامه نویسی شیءگرا، کد تمیز و اصول SOLID تا حد ممکن در نظر گرفته شود.

▼ توجه: امکان پیاده‌سازی بخشی از سوال و کسب نمره آن بخش از سوال وجود دارد.

▼ توجه: تنها یک فایل شامل کدهای نوشته شده آپلود کنید.

ورودی

ورودی شامل $n+1$ خط است. در خط اول n، تعداد دستورات می‌آید. در n خط بعدی، در هر خط یک دستور رمزنگاری آمده‌است.

$$1 \leq n \leq 50$$

خروجی

در خروجی به ازای هر دستور، نتیجه رمزنگاری را **با حروف بزرگ** در یک خط چاپ کنید.

مثال

ورودی نمونه ۱

```
2
additive-cipher -text "HELP me" -key 1
additive-cipher -text " HELP me " -key 1
```

خروجی نمونه ۱

```
IFMQ NF
IFMQ NF
```

▼ توضیحات نمونه ۱

در دستور اول، متن معنادار از نظر بزرگی یا کوچکی حروف فرقی نمی‌کند. بنابراین اگر `help me` را بخواهیم با الگوریتم `Additive Cipher` رمز کنیم، محاسبات به صورت زیر است:

```
h -> I ((7 + 1) % 26 = 8 -> I)
e -> F ((4 + 1) % 26 = 5 -> F)
l -> M ((11 + 1) % 26 = 12 -> M)
p -> Q ((15 + 1) % 26 = 16 -> Q)
m -> N ((12 + 1) % 26 = 13 -> N)
e -> F ((4 + 1) % 26 = 5 -> F)
```

- در این مثال، `key` برابر یک است. بنابراین هر کاراکتر در متن معنادار با عدد 1 جمع می‌شود. چون عدد حاصل باید بین 0 تا 25 باشد، باقیمانده‌ی حاصل جمع را بر عدد 26 بدست می‌آوریم. سپس در جدول نگاشت حروف انگلیسی به اعداد، حرف انگلیسی متناظر عدد بدست آمده را به عنوان کاراکتر رمزشده درنظر می‌گیریم. کاراکتر فاصله بین کلمات نیز دقیقا در متن رمز شده می‌آید. بنابراین متن رمز شده، `IFMQ NF` خواهد بود.
- دستور دوم مشابه دستور اول است، با این تفاوت که فاصله‌های اضافی اطراف متن معنادار نادیده گرفته شده‌است. بنابراین متن رمزشده در هر دو دستور یکسان خواهد بود.

ورودی نمونه ۲

```
2
multiplicative-cipher -text "danger" -key 3
multiplicative-cipher -key 3 -text "danger"
```

خروجی نمونه ۲

```
JANSMZ
JANSMZ
```

▼ توضیحات نمونه ۲

- در دستور اول، اگر `danger` را بخواهیم با الگوریتم `Multiplicative Cipher` و کلید با مقدار 3 رمز کنیم، محاسبات به صورت زیر است:

```
d -> J ((3 * 3) % 26 = 9 -> J)
a -> A ((0 * 3) % 26 = 0 -> A)
n -> N ((13 * 3) % 26 = 13 -> N)
g -> S ((6 * 3) % 26 = 18 -> S)
e -> M ((4 * 3) % 26 = 12 -> M)
r -> Z ((17 * 3) % 26 = 25 -> Z)
```

در دستور دوم، صرفا جای پارامترها عوض شده و محاسبات مانند دستور قبل است.

ورودی نمونه ۳

```
1
affine-cipher -text "Hi" -a 3 -b 1
```

خروجی نمونه ۳

```
WZ
```

▼ توضیحات نمونه ۳

در این مثال، محاسبات رمزنگاری به صورت زیر است:

```
h -> W ((7*3 + 1) % 26 = 22 -> W)
i -> Z ((8*3 + 1) % 26 = 25 -> Z)
```

در این مثال، پارامتر `a` کلید ضربی و پارامتر `b` کلید جمعی است. بنابراین حروف متن معنادار در `a` ضرب و با `b` جمع می‌شوند. بنابراین متن رمز شده، `WZ` خواهد بود.

ورودی نمونه ۴

```
1
mapping-cipher -text "hello" -mapping "zyxwvutsrqponmlkjihgfedcba"
```

خروجی نمونه ۴

```
SVOOL
```

▼ **توضیحات نمونه ۴**

در این مثال، `mapping` داده شده برابر `"zyxwvutsrqponmlkjihgfedcba"` است. به عبارت دیگر نگاشت حروف انگلیسی به متن رمز شده به صورت زیر است:

```
a -> Z
b -> Y
c -> X
d -> W
e -> V
f -> U
g -> T
h -> S
i -> R
j -> Q
k -> P
l -> O
m -> N
n -> M
o -> L
p -> K
q -> J
r -> I
s -> H
t -> G
u -> F
v -> E
```

```
w -> D
x -> C
y -> B
z -> A
```

با توجه به نگاشت بالا، متن `hello` به این صورت رمز می‌شود:

```
h -> S
e -> V
l -> O
l -> O
o -> L
```

بنابراین متن رمز شده، `SVOOL` خواهد بود.

معمای مرموز

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- **زبان‌های مجاز:** `Java, C#, C++`

در خانه‌ی مشهور شماره ۲۲۱ بیکر استریت، جاناتان کولمز، یک برنامه‌نویس با استعداد و دستیار وفادارش دکتر واتسون، به دنبال حل یک معمای جدید هستند. یک روز وقتی که جاناتان به دفترچه‌ی پوشه‌دار خود مراجعه می‌کند، با اسنادی مرموز و عجیب مواجه می‌شود.

این اسناد به زبان انگلیسی نوشته شده‌اند و به نظر می‌رسد برای نظریه‌پردازی درباره‌ی یک پرونده‌ی مخفی تهیه شده‌اند. جاناتان و دکتر واتسون، با تصمیم به حل این معما، به ساختن ابزاری سریع و قدرتمند می‌پردازند که به آن‌ها کمک می‌کند تا اطلاعات مورد نیاز خود را از متون مرموز استخراج کرده و معمایی که پیش رویشان قرار دارد را حل نمایند.

گام اول: پیش‌پردازش اسناد

اولین گام، پیش‌پردازش متون اسناد است. این پیاده‌سازی باید شامل موارد زیر باشد:

▼ **تبدیل متن به حروف کوچک**

برای جلوگیری از مشکلات مربوط به حساسیت به بزرگی و کوچکی حروف، همه حروف موجود در متن باید به حروف کوچک تبدیل شوند.

برای مثال متن سند پیش از پردازش:

```
Please HELP! Halis said.
```

پس از این پردازش به متن زیر تبدیل خواهد شد:

```
please help! halis said.
```

▼ **پاکسازی متن**

هرگونه کاراکتر غیر از حروف انگلیسی و اعداد که برای جستجوی متنی اهمیتی ندارند، باید حذف شوند.

برای مثال متن سند پیش از پردازش:

```
please help! halis said.
```

پس از این پردازش به متن زیر تبدیل خواهد شد:

```
please help halis said
```

▼ **حذف کلمات توقفی**

کلمات توقفی شامل موارد زیر هستند که در جستجوها نادیده گرفته می‌شوند و باید از متن حذف شوند:

```
"a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "
```

برای مثال متن پیش از پردازش:

```
halis is a programmer
```

پس از این پردازش به متن زیر تبدیل خواهد شد:

```
halis programmer
```

▼ **حذف فاصله‌های اضافی**

نیاز است هرگونه فاصله اضافی بین کلمات حذف شود.

برای مثال متن پیش از پردازش:

```
please help    halis  said
```

پس از این پردازش به متن زیر تبدیل خواهد شد:

```
please help halis said
```

گام دوم: طراحی Inverted Index

ساختار داده Inverted Index، ساختاری است که امکان نگهداری اطلاعات برای جستجوی سریع متنی را فراهم می‌کند. در ادامه با این ساختار داده بیشتر آشنا می‌شوید.

▼ **ایجاد لیست کلمات**

در این مرحله، باید تمامی کلمات موجود در هر سند را شناسایی کرده و لیستی از این کلمات ایجاد کنیم. این لیست، پایه‌ای برای ساخت Inverted Index است.

برای مثال در سند زیر:

```
"please help halis said"
```

لیست کلمات به این شکل خواهد بود:

```
["please", "help", "halis", "said"]
```

▼ **ایجاد Inverted Index**

در این مرحله، یک دیکشنری ایجاد می‌شود که کلیدهای آن کلمات موجود در لیست کلمات هستند و مقادیر آن‌ها لیستی از شماره اسنادی است که هر کلمه در آن‌ها ظاهر شده است.

برای مثال دو سند پیش‌پردازش شده زیر را در نظر بگیرید:

```
Document 1: "please help halis said"
Document 2: "halis writes code"
```

بنابراین Inverted Index به این شکل خواهد بود:

```
{
  "please": [1],
  "help": [1],
  "halis": [1, 2],
  "said": [1],
  "writes": [2],
  "code": [2]
}
```

همانطور که می‌بینید، کلمه `halis` هم در سند 1 و هم در سند 2 وجود دارد؛ در حالی که کلمه `please` تنها در سند 1 و کلمه `code` تنها در سند 2 وجود دارند.

▼ نکته مهم: Inverted Index از روی اسناد پیش‌پردازش شده ایجاد می‌شود.

▼ بازیابی اسناد

اکنون می‌توانیم به کمک ساختار داده ایجاد شده، اسناد دلخواه را بازیابی کنیم.

برای مثال Inverted Index زیر را درنظر بگیرید:

```
{
  "sherlock": [1],
  "halis": [1, 3, 4],
  "watson": [1, 2],
```

```
  "moriarty": [2, 3, 4]
}
```

می‌خواهیم بدانیم کلمات `halis` و `watson` هم‌زمان در چه اسنادی آمده‌است. برای این منظور، کافی است اشتراک شماره اسناد برای این دو کلمه را بررسی کنیم:

```
intersect([1, 3, 4], [1, 2]) = [1]
```

همانطور که می‌بینید، می‌توان نتیجه گرفت کلمات `halis` و `watson` هم‌زمان در سند 1 آمده اند.

گام سوم: جستجوی متنی

در این گام، کوئری‌هایی طراحی می‌شوند که به کمک آن‌ها می‌توان جستجوی متنی پیشرفته انجام داد و شماره اسناد مربوطه را بازیابی کرد. ساختار این کوئری‌ها با فرمت `JSON` است که در ادامه معرفی خواهند شد.

کوئری‌های فرزند

▼ کوئری `match`

با استفاده از این کوئری، اسنادی که شامل یک عبارت مشخص هستند، بازیابی می‌شوند.

```
{
    "match": "john watson"
}
```

در مثال بالا، اسنادی که شامل عبارت `john watson` هستند بازیابی می‌شوند؛ یعنی این دو کلمه پشت سر هم در متن پیش پردازش شده سند وجود داشته باشد.

▼ کوئری `all`

با استفاده از این کوئری، اسنادی که شامل تمام عبارات هستند، بازیابی می‌شوند.

```
{
    "all": ["john watson", "moriarty"]
}
```

در مثال بالا، اسنادی که شامل هر دو عبارت `moriarty` و `john watson` هستند بازیابی می‌شوند.

▼ کوئری `any`

با استفاده از این کوئری، اسنادی که شامل حداقل یکی از عبارات هستند، بازیابی می‌شوند.

```
{
    "any": ["john watson", "moriarty"]
}
```

در مثال بالا، اسنادی که شامل حداقل یکی از عبارت‌های `moriarty` یا `john watson` باشند بازیابی می‌شوند.

▼ **توجه: منظور از عبارت، یک یا چند کلمه با حروف کوچک است که با فاصله از هم جدا شده‌اند.**

کوئری‌های والد

▼ کوئری `and`

با استفاده از این کوئری، اسنادی بازیابی می‌شوند که با تمام از کوئری‌های فرزند به طور همزمان مطابقت دارند.

```
{
    "and": [
        {
            "match": "rainy"
        },
        {
            "match": "sherlock"
        },
```

```
        {
            "match": "death"
        }
    ]
}
```

در مثال بالا، اسنادی بازیابی می‌شوند که شامل هر سه کلمه‌ی `rainy` ، `sherlock` و `death` باشند.

یک نمونه کوئری دیگر شامل `and` :

```
{
    "and": [
        {
            "match": "rainy night"
        },
        {
            "and": [
                {
                    "all": ["sherlock holmes", "dr john watson", "moriarty"]
                },
                {
                    "any": ["death"]
                }
            ]
        }
    ]
}
```

▼ کوئری `or`

با استفاده از این کوئری، اسنادی بازیابی می‌شوند که با حداقل یکی از کوئری‌های فرزند مطابقت داشته باشد.

```
{
    "or": [
        {
```

```
            "match": "rainy night"
        },
        {
            "all": ["sherlock holmes", "dr john watson", "moriarty"]
        },
        {
            "any": ["death", "blood"]
        }
    ]
}
```

در مثال بالا، اسنادی بازیابی می‌شوند که یا شامل عبارت `rainy night` باشند، یا اینکه همه عبارات `sherlock holmes` ، `dr john watson` و `moriarty` در آن اسناد آمده باشد، و یا اینکه حداقل یکی از کلمات `death` یا `boold` در آن اسناد آمده باشد.

▼ کوئری `not`

با استفاده از این کوئری، تمام اسنادی که با کوئری فرزند مطابقت نداشته باشند بازیابی می‌شوند.

```
{
    "not": {
        "and": [
            {
                "match": "knife"
            },
            {
                "match": "gun"
            }
        ]
    }
}
```

در مثال بالا، اسنادی بازیابی می‌شوند که هر دو عبارت `knife` و `gun` همزمان در آن اسناد نیامده باشد.

نکات تکمیلی (مهم)

▼ نکته اول

کوئری‌های والد، خود نیز می‌توانند فرزند کوئری والد دیگری باشند.

```
{
    "and": [
        {
            "or": [...]
        },
        {
            "and": [...]
        },
        {
            "not": {...}
        }
    ]
}
```

▼ نکته دوم

در ریشه کوئری، ممکن است متغیر `size` بیاید که نشان دهنده حداکثر تعداد اسنادی است که باید بازیابی شوند. درصورتی که اندازه مشخص نشده بود، همه نتایج بازیابی می‌شوند.

```
{
    "and": [...],
    "size": 10
}
```

جمع‌بندی

هدف این است که ابزاری ساخته شود تا متون اسناد را پیش پردازش کرده، سپس ساختار داده Inverted Index ایجاد شود. در نهایت به کمک این ساختار داده جهت افزایش سرعت جستجو، کوئری‌هایی با هدف

بازیابی اسناد مختلف، پردازش شوند.

▼ **توجه: لازم است اصول برنامه نویسی شیءگرا، کد تمیز و اصول SOLID تا حد ممکن در نظر گرفته شود.**

▼ **توجه: امکان پیاده‌سازی بخشی از سوال و کسب نمره آن بخش از سوال وجود دارد.**

▼ **توجه: تنها یک فایل شامل کدهای نوشته شده آپلود کنید.**

ورودی

ورودی شامل $n+q+2$ خط است. در خط اول، n تعداد سندها، داده می‌شود. در n خط بعدی، در هر خط متن سند iام داده می‌شود. سپس، عدد q (تعداد کوئری‌ها) آمده است. در q خط بعدی، در هر خط یک کوئری با فرمت `JSONPath` می‌آید.

$$1 \leq n \leq 150$$

$$1 \leq q \leq 1500$$

▼ **راهنمایی: برای تبدیل کوئری از فرمت JSONPath به فرمت JSON، می‌توانید از این کدها استفاده نمایید.**

▼ **کد Java**

```
import java.util.*;

class JsonObjectCreator {
    private void setValue(Map<String, Object> json, String path, Stri
        String[] keys = path.split("\\.");
        for (int i = 0; i < keys.length; i++) {
            String key = keys[i];
            if (key.contains("[") && key.contains("]")) {
                String keyPart = key.substring(0, key.indexOf('['));
                int index = Integer.parseInt(key.substring(key.index(
```

```
                if (!json.containsKey(keyPart)) {
                    json.put(keyPart, new ArrayList<>());
                }
                List<Object> list = (List<Object>) json.get(keyPart);
                while (list.size() <= index) {
                    list.add(new HashMap<>());
                }
                if (i == keys.length - 1) {
                    list.set(index, value);
                } else {
                    json = (Map<String, Object>) list.get(index);
                }
            } else {
                if (i == keys.length - 1) {
                    json.put(key, value);
                } else {
                    if (!json.containsKey(key)) {
                        json.put(key, new HashMap<>());
                    }
                    json = (Map<String, Object>) json.get(key);
                }
            }
        }

    public Map<String, Object> createFromPath(String jsonPath) {
        Map<String, Object> json = new HashMap<>();
        String[] paths = jsonPath.split(",");
        for (String path : paths) {
            String[] parts = path.split("=");
            String keyPath = parts[0].trim();
            String value = parts[1].trim();
            setValue(json, keyPath, value);
        }
        return json;
    }
}
```

▼ کد #C

```
public class JsonObjectCreator
{
    private void SetValue(Dictionary<string, object> json, string pat
    {
        var keys = path.Split('.');
        for (var i = 0; i < keys.Length; i++)
        {
            var key = keys[i];
            if (key.Contains('[') && key.Contains(']'))
            {
                var keyPart = key.Substring(0, key.IndexOf('['));
                var index = int.Parse(key.Substring(key.IndexOf('[')
                if (!json.ContainsKey(keyPart))
                {
                    json[keyPart] = new List<object>();
                }

                var list = (List<object>)json[keyPart];
                while (list.Count <= index)
                {
                    list.Add(new Dictionary<string, object>());
                }

                if (i == keys.Length - 1)
                {
                    list[index] = value;
                }
                else
                {
                    json = (Dictionary<string, object>)list[index];
                }
            }
            else
            {
                if (i == keys.Length - 1)
                {
                    json[key] = value;
```

```
                }
                else
                {
                    if (!json.ContainsKey(key))
                    {
                        json[key] = new Dictionary<string, object>();
                    }

                    json = (Dictionary<string, object>)json[key];
                }
            }
        }
    }

    public Dictionary<string, object> CreateFromPath(string jsonPath)
    {
        var json = new Dictionary<string, object>();
        var paths = jsonPath.Split(',');
        foreach (var path in paths)
        {
            var parts = path.Split('=');
            var keyPath = parts[0];
            var value = parts[1];
            SetValue(json, keyPath, value);
        }

        return json;
    }
}
```

▼ کد C++

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <string>
#include <sstream>
#include <any>
```

```
using namespace std;

class JsonObjectCreator {
private:
    void SetValue(unordered_map<string, any> &json, const string &pat
        vector<string> keys;
        stringstream ss(path);
        string key;
        while (getline(ss, key, '.')) {
            keys.push_back(key);
        }

        unordered_map<string, any> *currentJson = &json;
        for (size_t i = 0; i < keys.size(); i++) {
            const string &key = keys[i];
            if (key.find('[') != string::npos && key.find(']') != str
                string keyPart = key.substr(0, key.find('['));
                int index = stoi(key.substr(key.find('[') + 1, key.fi
                if (currentJson->find(keyPart) == currentJson->end()
                    (*currentJson)[keyPart] = vector<any>();
                }
                vector<any> &list = any_cast<vector<any> &>((*current
                while (list.size() <= index) {
                    list.push_back(unordered_map<string, any>());
                }
                if (i == keys.size() - 1) {
                    list[index] = value;
                } else {
                    currentJson = &any_cast<unordered_map<string, any
                }
            } else {
                if (i == keys.size() - 1) {
                    (*currentJson)[key] = value;
                } else {
                    if (currentJson->find(key) == currentJson->end()
                        (*currentJson)[key] = unordered_map<string, a
                    }
                    currentJson = &any_cast<unordered_map<string, any
                }
```

```
            }
        }
    }

public:
    unordered_map<string, any> CreateFromPath(const string &jsonPath)
        unordered_map<string, any> json;
        vector<string> paths;
        stringstream ss(jsonPath);
        string path;
        while (getline(ss, path, ',')) {
            paths.push_back(path);
        }
        for (const auto &path: paths) {
            vector<string> parts;
            stringstream pathSS(path);
            string part;
            while (getline(pathSS, part, '=')) {
                parts.push_back(part);
            }
            string keyPath = parts[0];
            string value = parts[1];
            SetValue(json, keyPath, value);
        }
        return json;
    }
};
```

خروجی

در خروجی به ازای هر کوئری، شماره اسنادی که بازیابی می‌شوند را به ترتیب شماره اسناد از کوچک به بزرگ با فاصله در یک خط چاپ کنید. در صورتی که هیچ سندی در نتیجه اجرای کوئری پیدا نشد، عبارت NO RESULT را چاپ کنید.

مثال

ورودی نمونه ۱

```
3
The solution is simple.
Another mystery solved.
The mystery deepened with every clue.
3
match=mystery
any[0]=mystery,any[1]=solution
all[0]=mystery,all[1]=clue
```

خروجی نمونه ۱

```
2 3
1 2 3
3
```

▼ **توضیحات نمونه ۱**

پس از مرحله پیش‌پردازش، اسناد به صورت زیر خواهند بود:

```
Document 1: solution simple
Document 2: another mystery solved
Document 3: mystery deepened every clue
```

همچنین کوئری‌ها پس از تبدیل به شکل زیر خواهند بود:

```
{"match": "mystery"}
{"any": ["mystery", "solution"]}
{"all": ["mystery", "clue"]}
```

- در کوئری اول، به دنبال اسنادی هستیم که عبارت `mystery` در آن‌ها وجود دارد. این عبارت تنها در اسناد 2 و 3 آمده است.

- در کوئری دوم، اسنادی را می‌خواهیم بازیابی کنیم که حداقل یکی از عبارات `mystery` یا `solution` در آن اسناد آمده است. عبارت `mystery` در سند 2 و 3 و عبارت `solution` در سند 1 آمده است.
- در کوئری سوم، دنبال اسنادی می‌گردیم که هر دو عبارت `mystery` و `clue` در آن اسناد آمده باشد. سند 3 هردوی این عبارات را دارا است.

ورودی نمونه ۲

```
8
Watson: "Holmes, are you sure?"
Holmes: "Quite certain, Watson."
Holmes: "Observe, Watson, and learn."
Watson: "I'm always learning, Holmes."
Watson: "Holmes, you’re brilliant!"
Holmes: "I have my moments."
Holmes: "The truth is stranger than fiction."
Watson: "And you prove it daily."
5
or[0].match=watson,or[1].match=holmes
or[0].match=watson,or[1].match=holmes,size=6
and[0].any[0]=watson,and[0].any[1]=holmes,and[1].match=learn
and[0].all[0]=watson,and[0].all[1]=holmes,and[1].match=learning
not.match=watson
```

خروجی نمونه ۲

```
1 2 3 4 5 6 7 8
1 2 3 4 5 6
3
4
6 7
```

▼ توضیحات نمونه ۲

پس از مرحله پیش‌پردازش، اسناد به صورت زیر خواهند بود:

```
Document 1: watson holmes you sure
Document 2: holmes quite certain watson
Document 3: holmes observe watson learn
Document 4: watson im always learning holmes
Document 5: watson holmes youre brilliant
Document 6: holmes i have my moments
Document 7: holmes truth stranger than fiction
Document 8: watson you prove daily
```

همچنین کوئری‌ها پس از تبدیل به شکل زیر خواهند بود:

```
{"or": [{"match": "watson"}, {"match": "holmes"}]}
{"or": [{"match": "watson"}, {"match": "holmes"}], "size": 6}
{"and": [{"any": ["watson", "holmes"]}, {"match": "learn"}]}
{"and": [{"all": ["watson", "holmes"]}, {"match": "learning"}]}
{"not": {"match": "watson"}}
```

- در کوئری اول، اسنادی را می‌خواهیم بازیابی کنیم که حداقل یکی از عبارات `watson` یا `holmes` در آن اسناد آمده است. عبارت `watson` در همه اسناد به جز اسناد 6 و 7 آمده است. از طرفی عبارت `holmes` در سند 6 و 7 هم دیده می‌شود. بنابراین همه اسناد بازیابی می‌شوند.
- کوئری دوم همانند کوئری اول است با این تفاوت که `size` برابر 6 است. بنابراین تنها 6 سند اول بازیابی می‌شوند.
- در کوئری سوم، دنبال اسنادی می‌گردیم که حتما عبارت `learn` در آن اسناد وجود داشته باشد، به علاوه اینکه حداقل یکی از عبارات `watson` یا `holmes` هم وجود داشته باشد. عبارت `watson` و `holmes` در کنار عبارت `learn` تنها در سند 3 آمده‌اند.

ورودی نمونه ۳

```
3
Crime is common, logic is rare.
```

```
The little things are infinitely the most important.
The world is full of obvious things which nobody by any chance ever observes.
5
match=little things
match=common logic rare
match=common rare
any[0]=little things,any[1]=obvious things,any[2]=common logic
all[0]=things
```

خروجی نمونه ۳

```
2
1
NO RESULT
1 2 3
2 3
```

▼ **توضیحات نمونه ۳**

پس از مرحله پیش‌پردازش، اسناد به صورت زیر خواهند بود:

```
Document 1: crime common logic rare
Document 2: little things infinitely most important
Document 3: world full obvious things which nobody any chance ever observes
```

همچنین کوئری‌ها پس از تبدیل به شکل زیر خواهند بود:

```
{"match": "little things"}
{"match": "common logic rare"}
{"match": "common rare"}
{"any": ["little things", "obvious things", "common logic"]}
{"all": ["things"]}
```

- در کوئری اول، به دنبال اسنادی هستیم که عبارت `little things` در آن‌ها وجود دارد. کلمات `little` و `things` تنها در سند 2 پشت سر هم دیده می‌شوند.
- کوئری دوم نیز مشابه کوئری اول است، با این تفاوت که به دنبال 3 کلمه پشت سر هم هستیم.
- در کوئری سوم، عبارت `common rare` در هیچ سندی دیده نمی‌شود.

ورودی نمونه ۴

```
10
You see, but you do not observe.
Crime is common,  logic is rare.
There is nothing more deceptive than an obvious fact.
The world is full of obvious things which nobody by any chance ever observes.
Data, data, data! I can't make bricks without clay.
It's a capital mistake to theorize before one has data.
I am lost without my Boswell.
Eliminate all other factors, and the one which remains must be the truth.
The truth is what I pursue.
London, at its best, is a labyrinth of intrigue and danger.
1
or[0].all[0]=london,or[0].all[1]=intrigue,or[0].all[2]=thing,or[1].any[0]=psyc
```

خروجی نمونه ۴

```
1 2 4
```