サルベジオン問題

http://www.hyuki.com/codeiq/

2014 年 11 月〜12 月

目次

1　概要

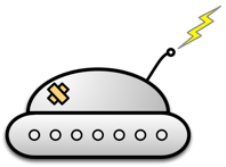
あなたは サルベジオン 社（Salvageon）にプログラマとして雇われました。サルベジオン社は、難破した無人宇宙船に格納されているデータ救出を行う会社で、遠い星の無人宇宙船と通信を行ってデータ解析を行うのを得意としています。

あなたに与えられるのは無人宇宙船のデータベースにアクセスする Web API ひとつ。この Web API を使って データを見つけ出す のがあなたのミッションなのです！

2　会話

あなた「データを見つけ出す仕事と聞きましたが、いったいどういうことでしょうか……」

依頼者「順を追って説明しましょう。わがサルベジオン社は、難破した無人宇宙船に格納されているデータ救出を行う会社です。最近、遠い星に無人宇宙船を飛ばすことが多くなってきましたが、経費削減のあおりで事故も多い。せっかく飛ばした無人宇宙船が難破してしまうこともしばしばあります」



難破した無人宇宙船

あなた「はい」

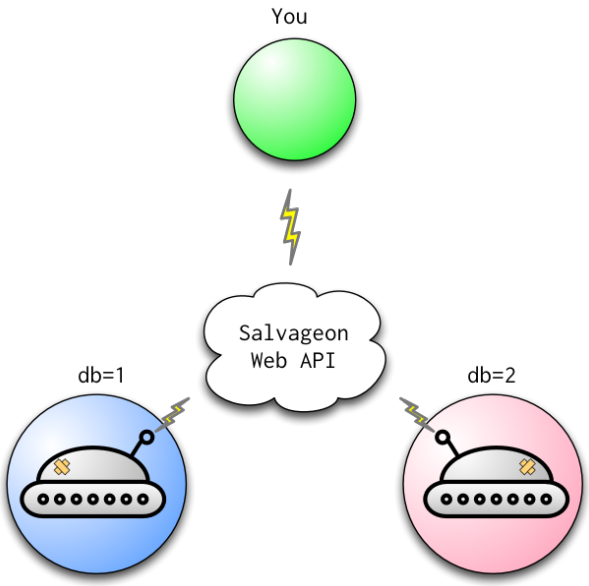
依頼者「無人宇宙船全体を修理してまた地球まで飛ばすのは費用も掛かりますから、せめてデータベースだけでもサルベージ（救出）したいという要望が多く、わが社はそこでビジネスをしているわけです」

あなた「なるほど。御社の社名、サルベジオンもそこから？」

依頼者「そういうことです。現在も二台分のデータベース・サルベージ作業を行っていて、あなたにはその業務を手伝っていただきます」

あなた「わかりました。しかし、私には無人宇宙船の知識はありませんが……」

依頼者「それはご心配なく、すでに無人宇宙船との間には通信チャネルが確立していますので、あなたには、サルベジオン社の **Web API** を使ってデータベースにアクセスしていただきます。そして、与えられたキーを持つ データを見つけ出す のがお仕事となります」



サルベジオン社 **Web API**

あなた「たとえば、SQL のような言語を使うことになりますか？」

依頼者「違います。遠い星で故障している無人宇宙船ですから、そんなに高度なアクセスはできません。アクセス方法は以下の通りです」

- アクセスには Web API を用いる。
- 個々のデータベースは、データベース番号（db）で区別する。
- 個々のデータベースは配列のように見える。
- 非負整数のインデクス (index) を与えると、キー (key) とバリュー (value) のペアが得られる。

あなた「なるほど。データベース番号とインデクスが入力で、キーとバリューのペアが出力ということですね」

依頼者「そういうことです。データベースが生きていたらキーを与えてバリューが得られるはずなのですが、あちこち壊れているために、このような原始的な形でのアクセスしかできないのです」

あなた「概要はわかりました」

依頼者「あなたにお願いしたいのは、データベースの中から、与えられたキーに対応するデータを見つけ出すことです」

あなた「与えられたキーを持つデータは、データベースの中に必ず存在するのでしょうか？」

依頼者「はい、存在します。また、一つのデータベースの中ではキーに重複はありません。バリューには重複があるかもしれませんが」

あなた「了解です。あとは具体的な情報をいただければすぐに仕事にかかれると思います」

依頼者「では具体的な例で Web API の説明をしましょう。たとえば、このようなデータベースがあるとします」

index	key	value
0	K6742	V02176333
1	K5499	V05677882
2	K3668	V79713413
3	K9396	V29970163
4	K0929	V60983729

db が 0 の例

依頼者「このデータベースのデータベース番号（db）は 0 です。このデータベースには全部で 5 個のデータ、すなわちキーとバリューのペアが 5 組格納されています」

あなた「よくわかります。たとえばインデクスが 2 のデータは、キーが K3668 で、バリューが V79713413 ということですね？」

依頼者「その通りです。このデータベースからキーが K3668 であるバリューを探してほしいといわれれば、V79713413 を答えていただければよろしいということになります」

あなた「なるほど。ところで、キーは K の後に数字列、バリューは V の後に数字列が続く形式ですね」

依頼者「そうです。それはどのデータベースも同じです」

あなた「キーの順番はどうなっているんでしょうか」

依頼者「データベースごとにルールがあるらしいのですが、ドキュメントが残されていないので判明していません」

あなた「（そんなあ！）……な、なるほど。そうですか。そこは調査が必要になりますね」

依頼者「データベース番号（db）が 0 であるデータベースへアクセスして、インデクス（index）が 2 であるデータを得るとき、Web API はこう呼び出します」

```
http://salvageon.textfile.org/?db=0&index=2
```

データベース番号が 0 で、インデクスが 2 の呼び出し

あなた「なるほど。素直な呼び出しですね」

依頼者「この結果はテキストで以下のように返されます」

```
0 2 K3668 V79713413 5
```

データベース番号が 0 で、インデクスが 2 の呼び出し結果

あなた「0 はデータベース番号、2 は与えたインデクス、K3668 は得られたキーで、V79713413 は得られたバリューですね。最後の 5 は何ですか？」

依頼者「ああ、データベースのサイズですよ。つまりこのデータベースに与えることができる最大のインデクスに 1 加えた値ですね」

あなた「なるほど、わかりました」

依頼者「一般的に書けば、入力はこうなります」

```
http://salvageon.textfile.org/?db=DB&index=INDEX
```

Web API 呼び出しの形式

依頼者「この URL で DB の部分にはデータベース番号を与え、INDEX にはインデクスを与えます。インデクスに与えるのは十進数の非負整数です」

あなた「はい」

依頼者「この URL にアクセスすると結果は、HTTP のレスポンスとしてデータベース番号 (DB)、インデクス (INDEX)、キー (KEY)、バリュー (VALUE)、そしてデータベースサイズ (SIZE) がスペース区切りで返されてきます。Content-type は text/plain です。ただしエラーの場合には ? という一文字が返されます」

```
DB INDEX KEY VALUE SIZE
```

Web API 呼び出し結果の形式

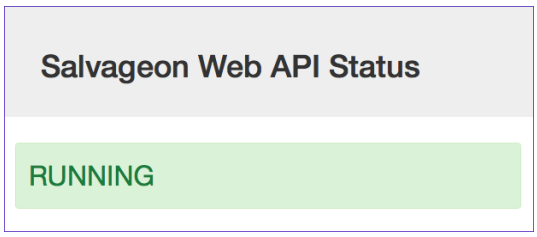
あなた「有効なインデクスの範囲は、0 以上 SIZE 未満の整数すべてですね？」

依頼者「その通りです。ただし、SIZE はデータベース番号ごとに異なります」

あなた「整理させてください。こういうことでしょうか」

- データベース番号ごとに、探すべきデータのキーが与えられる。
- Web API を使ってキーを見つけ出し、それに対応するバリューを得る。

依頼者「その通りです。なお、Web API にアクセスできないときには、以下の URL にアクセスしてみてください。通常は RUNNING とのみ表示されていますが、非常時の連絡事項がそこに表示される場合があります」



http://salvageon.textfile.org/status.html

あなた「わかりました。では、プログラムを作りますので、データベース番号とキーの一覧をいただけますか」

依頼者「探すデータは二件です。ファイル keys.txt としてお渡しします」

db	key
1	K208050656559285601386927895421059705239114932023754
2	K20236360709985574445425860 45

データベース番号と探すべきデータのキー（keys.txt）

あなた「ありがとうございます」

依頼者「そうだ、最後にもう一つ。Web API へのアクセスは過負荷にならないようにしてください。具体的には一回のアクセスごとに約 1 秒、時間をおいてくださいね」

あなた「あっ、はい」

ミッションはこのようにして始まりました……。

3　解答編

3.1　はじめに

こんにちは、結城浩です。サルベジオン問題へ多数のご解答ありがとうございました。今回のサルベジオン問題は、Web API を使って非常にたくさんのデータの中から目的のものを探し出すというものでした。楽しんでいただけたでしょうか。

今回は特に大きなトラップもなく、単純といえば単純な問題だったと思います。多くの方が最高評価 5 を得ることができました。

3.2　評価基準

評価基準は以下の通りです。

- 評価 5 サルベジオン賞（二つのバリューを正しく得ることができた）
- 評価 3 準サルベジオン賞（一つのバリューのみ正しく得ることができた）
- 評価 1 残念賞（バリューを一つも得られなかった。形式上の不備を含む）

複数回投稿した場合は、最終投稿のみを評価しました。
以下の二点は CodeIQ 運営さんからの連絡になります。

- 評価 5 の方全員には サルベジオンバッジ がサイト上で付与されます。
- 評価 5 の方には抽選で書籍 『数学文章作法　推敲編』 が当たります。

図 1　『数学文章作法　推敲編』

3.3　正解

以下に正解を示します。

```
V406435859539156181269150751031
V1101943557675920722238136981003
```

なお、バリューを求めるための Ruby プログラムを参考として以下に置いてあります。
https://gist.github.com/hyuki0000/a6535884503b0054b974

3.4　評価の分布

評価の分布は以下の表の通りです。

評価 5	180 名
評価 3	10 名
評価 1	6 名
合計	196 名

3.5　キーを求めて

それでは今回の問題を一緒に考えていきましょう。
二つのデータベースを順番に見ていきます。

3.6　データベース 1 番（db=1）の探索

何はともあれ、指定された Web API を使っていくつかデータを見ていきましょう。
最初の 5 個は次のようになります。

```
1 0 K19584500653053662232211568267 V830542486163201924733591581247 126765060022822940149670320537 6
1 1 K19623607256232418445590556076 V115157972049091669316554187614 5 126765060022822940149670320537 6
1 2 K19718984353819469994289486141 V803371842571480223738576246215 126765060022822940149670320537 6
1 3 K19781945725870318832662826826 V314778970077814367350137878130 126765060022822940149670320537 6
1 4 K19854960460600482668081543475 V526182689909684857174556484180 126765060022822940149670320537 6
```

まず注目すべき点は、126765060022822940149670320537 6 という極端に大きなデータサイズです。この時点で、しらみつぶし（ブルートフォース）の探索をあきらめることになります。なぜなら、1 秒に 1 回のアクセスでは、全数探索するのに約 8 億年以上掛かるからです。8 億年も掛かったら、〆切に間に合わなくなってしまいますね。

プログラマなら大きな数を見たら、ビットサイズを計算したくなります。このデータサイズを二進数で表すと、ちょうど 2^{100} すなわち、101 ビットの数であることが分かります。

そもそも、こんな大きなサイズのデータが実際に格納されているとは思えませんから、何らかの計算によっ

てデータを作り出しているはずだということは予想できるでしょう。そうなると、何らかのパターンがあるのではないか、と考えることになります。

ここからは、Web API を使ってあちこちを探索し、予想するしかありません。その際に、同じインデクスに対して何度もアクセスするはめにならないように、ちゃんと手元にデータを保存しておくのは良い考えです。

最初の 5 個を見ましたので、試しに最後の 5 個を表示してみましょう。桁数が大きいので適当に改行を入れています。

```
1 126765060022822940149670320537 1 K10281805306702037028226499642898951367495882500687541280 8
V72353912253864407876496545024 6 1267650600228229401496703205376
1 126765060022822940149670320537 2 K10281805306702037028226499642907096498056739472818039374 6
V98415146281545994927935827789 6 1267650600228229401496703205376
1 126765060022822940149670320537 3 K10281805306702037028226499642918408925364988842186226727 4
V68827420877513972098407613647 6 1267650600228229401496703205376
1 126765060022822940149670320537 4 K10281805306702037028226499642919488078938210267557980347 7
V98115398180933817700982342819 8 1267650600228229401496703205376
1 126765060022822940149670320537 5 K10281805306702037028226499642929664730117211031593914773 4
V76458195439642989863798809208 6 1267650600228229401496703205376
```

ここから、ははーんとわかった方もいると思います。

最初の 5 個のキーは、 K の後に 29 桁の数が続いていますが、最後の 5 個のキーは、K の後に 57 桁の数が続いています。この桁数の極端な差は何でしょう。

ここから、もしかして キーは昇順（小さい順）に並んでいる のではないか？　と予想することができます。

この予想を確実に証明することは不可能です。なぜなら、約 8 億年掛けないと全データを見ることができないからです。しかし、ランダムにアクセスすることで、その予想が確からしいことは確認できるでしょう。

そして、「キーが昇順に並んでいる」言い換えると「キーが昇順でソート済みである」ことは、この問題を解決できる大きな手がかりとなります。

なぜなら 二分探索（バイナリサーチ） が使えるからです！

Step 1. 全体の中央部分の（約 1/2 に位置する）キーを得る。
Step 2. 得たキーを目的のキーと比較する。
Step 3. 比較結果に応じて、以下のいずれかを行う。

- 目的のキーが大きいならば、後半の中央部分のキーを得て、Step 2 に戻る。
- 目的のキーが小さいならば、前半の中央部分のキーを得て、Step 2 に戻る。
- 目的のキーに等しいならば、終了する。

データが 2^n 個の場合、二分探索（バイナリサーチ）を使えば $n+1$ 回の比較で目的のキーおよびバリューが見つかります。

ですから、最悪でも 101 回の Web API のアクセスで目的のキーが見つかることが確信できます。

なお、データベース 1 番では、キーが単調増加になるように、インデクス n と対応するキーの数値 $K(n)$ の間に以下のような関係があります。

$$K(n) = \mathrm{Offset} + n \times \mathrm{Gap} + \mathrm{SHA1}(n) \bmod \mathrm{Gap}$$

Offset と Gap は定数です。SHA1 を使うことで、単調増加でありつつも少し揺らぎが出るようにしてあります。しかし、この関係を予測せずとも解けるように問題の難易度を設定してあるつもりです。

3.7 データベース 2 番（db=2）の探索

では、同じようにして、Web API を使ってデータベース 2 番を見てみましょう。

最初の 5 個は次のようになります。

```
2 0 K0 V866608106806207094188269706010 126765060022822940149670320537 6
2 1 K6338253001141147007483516026 88 V17934733055090765520159193627 1 126765060022822940149670320537 6
2 2 K3169126500570573503741758013 44 V96787456649005690576359009697 6 126765060022822940149670320537 6
2 3 K9507379501711720511225274040 32 V21007428639505136878697302990 126765060022822940149670320537 6
2 4 K1584563250285286751870879006 72 V49990388040697516791462636846 7 126765060022822940149670320537 6
```

最初の K0 に困惑します。

気を取り直してインデクス 1 から 4 まで見ていきますが、先ほどのように昇順に並んでいるわけではありません。それでは、大小関係はどうなっているでしょうか。

解説のため、

$$K(n) = \text{インデクスが } n \text{ であるキーの、整数部分}$$

という表記法を使うことにします。たとえば、

$$K(1) = 6338253001141147007483516026 88$$

です。

この表記法を使えば、大小関係は以下の式のようになります。

$$K(0) \ll K(4) < K(2) < K(1) < K(3)$$

何となく「交互」に大きくなっているような気がしますが、はっきりしません。続く 5 個も見てみましょうか。

```
2 5 K4753689750855860255612637020 16 V13761252599589734064800042811 6 126765060022822940149670320537 6
2 6 K7922816251426433759354395033 60 V10139248988293067723801702696 14 126765060022822940149670320537 6
2 7 K1109194275199700726309615304704 V12524046450564272911825330859 87 126765060022822940149670320537 6
2 8 K792281625142643375935439503 36 V52059599933957096599388307882 8 126765060022822940149670320537 6
2 9 K2376844875427930127806318510 08 V12568912849672773367141344964 24 126765060022822940149670320537 6
```

ここまでの調査でわかる大小関係はこうです。

$$K(0) \ll K(8) < K(4) < K(9) < K(2) < K(5) < K(1) < K(6) < K(3) < K(7)$$

1 から 9 までを順にたどってみると、「何となく交互」ではありますが、規則性はわかりません。

ここで、大小関係にこだわるのをやめて、$K(1)$ から $K(9)$ までの数をじっと見てみましょう。

$$
\begin{aligned}
K(1) &= 633825300114114700748351602688 \\
K(2) &= 316912650057057350374175801344 \\
K(3) &= 950737950171172051122527404032 \\
K(4) &= 158456325028528675187087900672 \\
K(5) &= 47536897508558602561263702016 \\
K(6) &= 7922816251426433759354395033 60 \\
K(7) &= 1109194275199700726309615304704 \\
K(8) &= 792281625142643375935439503360 \\
K(9) &= 237684487542793012780631851008
\end{aligned}
$$

じっと見ているうちに、$K(1)$ と $K(2)$ の関係にふと気付きます。

$$
\begin{aligned}
K(1) &= 633825300114114700748351602688 \\
K(2) &= 316912650057057350374175801344
\end{aligned}
$$

$K(1)$ は $K(2)$ のちょうど 2 倍ですよね！　つまり、

$$K(1) = 2K(2)$$

のように見えます。

そういう観点で $K(1)$ から $K(9)$ を見ると、以下のような関係を見つけることができます。

$$K(1) = 2K(2) = 4K(4) = 8K(8)$$

つまり、$K(1) = 2^n K(2^n)$ という関係です。これは大きい手がかりです。なぜなら、

$$K(2^n) = \frac{K(1)}{2^n} \qquad (n = 0, 1, 2, \ldots)$$

という形で、大きなインデクスに対応するキーを予想することができるからです。あくまで予想ですけれど。

さらに、$\dfrac{K(1)}{2^n}$ という分数が気になるので、$K(1)$ を 2 進表記してみましょう。

$$1\underbrace{000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000}_{99\text{ 個}}$$

なんと、$K(1) = 2^{99}$ という値なのですね。つまり、

$$\frac{K(1)}{2^n}$$

という分数は、$n = 0, 1, 2, \ldots, 99$ の範囲で整数になるということです。

次に $K(1), K(2), K(3)$ を見ましょう。

$$
\begin{aligned}
K(1) &= 633825300114114700748351602688 \\
K(2) &= 316912650057057350374175801344 \\
K(3) &= 950737950171172051122527404032
\end{aligned}
$$

ここから、以下のような関係がわかります。

$$K(1)+K(2)=K(3)$$

和の関係があるという観点で $K(1)$ から $K(9)$ を見ると、以下のような関係を見つけることができます。

$$\begin{aligned}
&K(1)+K(2)=K(3), && (\heartsuit)\\
&K(2)+K(4)=K(5), && (\heartsuit)\\
&K(2)+K(5)=K(6), && (\clubsuit)\\
&K(1)+K(5)=K(7), && (\clubsuit)\\
&K(4)+K(8)=K(9), && (\heartsuit)
\end{aligned}$$

つまり、♡ から、$K(2^{n-1})+K(2^n)=K(2^n+1)$ という関係がありそうです。もしこの予想が正しければ、

$$\begin{aligned}
K(2^n+1)&=K(2^n)+K(2^{n-1})\\
&=\frac{K(1)}{2^n}+\frac{K(1)}{2^{n-1}}\\
&=\frac{3K(1)}{2^n}
\end{aligned}$$

ということになります。

♣ からは何がわかるでしょうか。インデクスが 6 と 7 ですから、ちょうどこれはすでにわかっている 5 と 8 の「間の埋め方」がわかるようですね。

$$\begin{aligned}
K(6)&=K(2)+K(5)\\
&=K(2^1)+K(2^2+1)\\
&=\frac{K(1)}{2^1}+\frac{3K(1)}{2^2}\\
&=\frac{5K(1)}{2^2}\\
K(7)&=K(1)+K(5)\\
&=K(1)+\frac{3K(1)}{2^2}\\
&=\frac{7K(1)}{2^2}
\end{aligned}$$

ここまでの内容を整理してみましょう。

$$
\begin{aligned}
K(1) &= K(2^0) &&= \frac{1K(1)}{2^0} \\
\\
K(2) &= K(2^1) &&= \frac{1K(1)}{2^1} \\
K(3) &= K(2^1+1) &&= \frac{3K(1)}{2^1} \\
\\
K(4) &= K(2^2) &&= \frac{1K(1)}{2^2} \\
K(5) &= K(2^2+1) &&= \frac{3K(1)}{2^2} \\
K(6) &= K(2^2+2) &&= \frac{5K(1)}{2^2} \\
K(7) &= K(2^2+3) &&= \frac{7K(1)}{2^2} \\
\\
K(8) &= K(2^3) &&= \frac{1K(1)}{2^3} \\
K(9) &= K(2^3+1) &&= \frac{3K(1)}{2^3} \\
&\vdots
\end{aligned}
$$

$1, 3, 5, 7$ という並びで、《奇数》という規則性が見えてきました。

$n = 0, 1, 2, \ldots$ とし、$m = 1, 2, \ldots, 2^n$ とするとき、以下のように予想できそうですね。

$$
K(2^n + m - 1) = \frac{(2m-1)K(1)}{2^n} \qquad \cdots\cdots\text{《関係式》}
$$

この予想も、本当に正しいかどうか（すべてのインデクスにあてはまるかどうか）の確認はできません。データベースに含まれているデータの数が多すぎるためです。でも、Web API を使って正しそうであることは確かめられます。

ここまで予想できたところで、私たちのやりたいことを思い出しましょう。

私たちのやりたかったことは、特定のキー `K2023636070998557444542586045` に対応したバリューを求めることですから、このキーを持つインデクスを推測する必要があります。すなわち、先ほどの予想から、

$$
K(2^n + m - 1) = \frac{(2m-1)K(1)}{2^n} = 2023636070998557444542586045
$$

を満たす m, n の組を探すことになります（n は 0 以上の整数で、m は 1 以上 2^n 以下の整数)。

$$
K(1) = 633825300114114700748351602688
$$

であることから、上の式は次のように変形できます。

$$\frac{(2m-1)6338253001141147007483516026 88}{2^n} = 202363607099855744454258604 5$$

$$\frac{2m-1}{2^n} = \frac{202363607099855744454258604 5}{6338253001141147007483516026 88}$$

$$= \frac{2 \times 10118180354992787222712930 23 - 1}{2^{99}}$$

ここから、

$$\begin{cases} m &= 10118180354992787222712930 23 \\ n &= 99 \end{cases}$$

が得られます。すなわち、

$$K(2^{99} + 10118180354992787222712930 23 - 1) = 202363607099855744454258604 5$$

であると予想できます。またインデクスは、

$$2^{99} + 10118180354992787222712930 23 - 1 = 6348371181496139794706228957 10$$

であると予想できます。

さあ、このインデクスに Web API でアクセスしてみましょう！ 桁数が大きいので適当に改行を入れています。

```
2 6348371181496139794706228957 10 K202363607099855744454258604 5
V110194355767592072223813698 1003 126765060022822940149670320 5376
```

見つかりました！

3.8 データベース 2 番（db=1）の謎

ここまでで、データベース 1 番とデータベース 2 番の両方とも解決しました。

- データベース 1 番は、昇順のキーを二分探索（バイナリサーチ）する。
- データベース 2 番は、規則性を見つけて計算する。

ところで……データベース 2 番の規則性には背後に隠れたルールがあります。

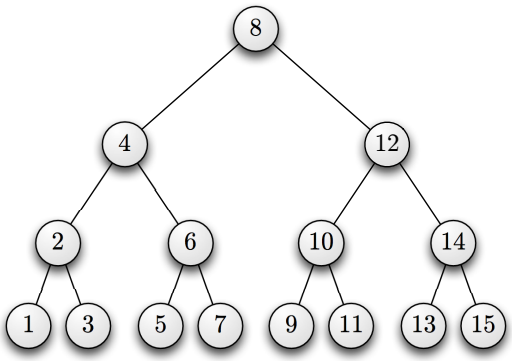
$K(1) = 2^{99}$ という大きな数では扱いにくいので、たとえば $K'(1) = 2^3$ で考えてみましょう（混乱を避けるため、$K(n)$ ではなく $K'(n)$ と表記します）。すると、以下のような《関係式》が予想できます。

$$K'(2^n + m - 1) = \frac{(2m-1)K'(1)}{2^n} = \frac{(2m-1)8}{2^n} \qquad \cdots\cdots\text{《関係式》}(K'(1) = 8 \text{ の場合})$$

これを、次のような一覧表にしても、謎は見えません。

$$K'(1) = 8$$
$$K'(2) = 4$$
$$K'(3) = 12$$
$$K'(4) = 2$$
$$K'(5) = 6$$
$$K'(6) = 10$$
$$K'(7) = 14$$
$$K'(8) = 1$$
$$K'(9) = 3$$
$$K'(10) = 5$$
$$K'(11) = 7$$
$$K'(12) = 9$$
$$K'(13) = 11$$
$$K'(14) = 13$$
$$K'(15) = 15$$

でも、こんな図にすると、謎は解けます。

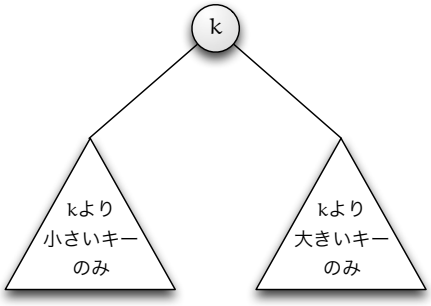


二分探索木（バイナリサーチツリー）

上図で、丸の中（ノード）に書かれた数をキーだと思ってください。そうすると、《左の子孫ノード》のキーはすべて自分のキーよりも小さく、《右の子孫ノード》のキーはすべて自分のキーよりも大きくなっていることがわかりますね。

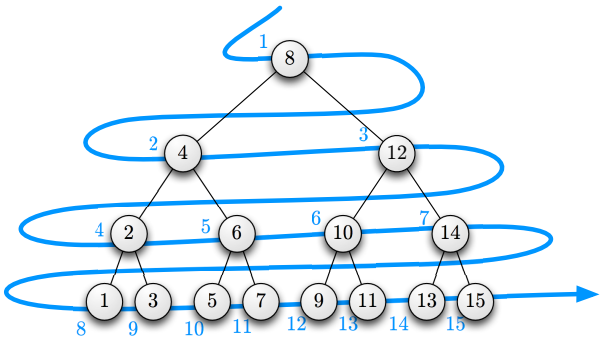
つまり、データベース 2 番は、二分探索木（バイナリサーチツリー）になっていたのです！

二分探索木には、以下のような制約があります。



二分探索木（バイナリサーチツリー）の制約

まだピンと来ない方のために、インデクスとキーの関係図を以下に示します。この図ではインデクスが青の数字で表されています。



インデクス（青の数字）とキー（丸の中の数字）の関係図

この関係図を見ると、

- 左の子供に行くためには、インデクスを 2 倍する。
- 右の子供に行くためには、インデクスを 2 倍して 1 加える。

という関係になっていることがわかりますね。

このようにインデクスを工夫して、配列のようなデータ構造の上に、木構造を組み立てたデータ構造を ヒープ と呼びます。インデクスが持っている《関係式》を使うと、ヒープを使って二分探索木を実現できるのです。

リンクを使って木構造を実装した場合には、リンクをたどることで木構造を巡りますが、ヒープを使って木構造を実装した場合には、インデクスの計算によって木構造を巡ることになります。

ここまでの図で示したのはは $n = 3$ の場合でしたが、サルベジオン問題のデータベース 2 番では、これが $n = 99$ の場合になっていたのです。

前節では、キーの値を直接的に計算しました。しかし、たとえばキーの値を直接的に計算できない場合でも、キーの値の大小関係が二分探索木の制約（左の子孫のキーは自分よりも小さく、右の子孫のキーは自分よりも大きい）を守っているなら、二分探索によって非常に高速な探索が可能になります。

謎が解けたところで、もう一度、データベース 1 番とデータベース 2 番をまとめてみましょう。

- データベース 1 番は、昇順のキーを二分探索（バイナリサーチ）する。
- データベース 2 番は、規則性を見つけて計算する。
  または、ヒープで実現した二分探索木を使って二分探索（バイナリサーチ）する。

ということになります。
今回のサルベジオン問題のテーマは 二分探索（バイナリサーチ） だったのです。

……なのですが、データベース 2 番から見つけるように指定したデータが、二分探索木の《葉》にあたるものだったため、局所的な等差数列を使って発見されてしまうことも多かったようですね。

なお、バリューを計算で推測することは想定しませんでした。バリューは、ソルトを交えた SHA1 ハッシュ値を使って作っています。

3.9 解答に使ったコード

バリューを求めるための Ruby プログラムを参考として以下に置いてあります。
https://gist.github.com/hyuki0000/a6535884503b0054b974

3.10 挑戦者が使った言語

挑戦者が使った言語などは以下の通りです。
行末の数は挑戦者数を表しています。

- C# 6
- C# .NET (.NET Framework 4.0) 1
- C#, Haskell 1
- C#と Excel 1
- C++ 3
- C++, Maxima 1
- C++11 1
- Clojure 1
- Clojure と手動 1
- D 1
- DB 1 は php、DB2 は手計算 1
- D 言語 1
- Erlang, Ruby（電卓代わり）, factor コマンド, 紙とペン 1
- F# 1
- Firefox, Python, TeraPad 1
- Groovy 1
- Haskell 4
- Java 12
- Java(Android) 1
- Java(TM) SE Runtime Environment (build 1.8.0_25-b17) 1
- Java+Xtend 1
- JavaScript 1
- Javascript(Node.js) 1

- Java と電卓 1
- Julia 1
- Microsoft Excel 2002 と Windows 関数電卓 1
- Node.js 1
- OCaml 1
- PHP 6
- Perl 6
- Perl,Excel 1
- Python 17
- Python 2.7.8 3
- Python 2.7.8 on Windows7 Professional 64bit 1
- Python 3.2.2 1
- Python 3.4.1 1
- Python, bash 1
- Python,WolframAlpha 1
- Python2.7 1
- Python3 1
- Python3, Perl5 1
- Python3.4 1
- Pyton 1
- R 1
- Ruby 40
- Ruby (2.0) 1
- Ruby + 視覚 1
- Ruby 1.9.3 1
- Ruby 2.0.0 1
- Ruby, 手計算 1
- Ruby,LibreOffice Calc 1
- Scala 3
- Scala,Haskell 1
- Swift 1
- Tcl (cygwin 版 tclsh.exe ver. 8.5.11) 1
- This is perl 5, version 20, subversion 1 (v5.20.1) built for x86_64-linux-thread-multi 1
- VB.NET 1
- curl 1
- db1:Perl db2:電卓 1
- java 2
- java + javascript 1
- objective-c 1
- php 2
- python 2
- python + 人力 1
- ruby 1
- scala 1
- しらみつぶし 1
- なでしこ 1
- ほぼ手計算（DB=2 の 2 進数化だけ Windows の電卓では桁が足りないので Java7 でコードを書きました） 1
- サクラエディタ, 電卓 1
- サクラエディタ（テキストエディタ）＋電卓（Windows 付属のもの） 1

- シェルスクリプト 1
- シェルスクリプトと irb(電卓として) 1
- テキストエディタと関数電卓 1
- メモ帳，Java，根気 1
- 人力 1
- 全手動 1
- 手 4
- 手作業 1
- 手作業＋ Wolframalpha 1
- 手動 1
- 手動、電卓 (irb) 1
- 手計算 3
- 手計算（電卓） 1
- 手＋ Windows 7 64bit ＋ Firefox ＋メモ帳＋ Dr.Racket（素因数分解、有理数除算） 1
- 最終的に「R」。途中経過で電卓，Excel, 少しだけ python 1
- 机上、取得したデータの管理と計算に C# 1
- 紙とペン 1
- 脳内プロセッサ ＋ 電卓 1
- 電卓 (Win8.1 上の) 1

3.11 挑戦者の声から

ではおまちかね、挑戦者のみなさんの声をいくつかピックアップしてご紹介します！ 全員の分は掲載していません。また、掲載しているものも全文ではなく、さらに、一部編集している部分もあります。ご了承ください。

ENV: Java+Xtend
POINT: 解答しきれませんでした　時間内でポイントや規則性を見つけることができませんでした

いつも楽しく観させていただいています　ありがとうございます　お身体ご自愛ください

ENV: 手動
POINT: DB2 は分かりませんでした。
DB1 はインデックスに対するキーの値が単調増加だったので、挟み込みで計算しました。

ENV: Erlang, Ruby（電卓代わり）, factor コマンド, 紙とペン
POINT:db1 は二分探索、db2 は 2 のべき乗がポイントです。db2 は素因数分解を駆使して解きました。

db2 は factor コマンドを主に使いました。キーの一般項が

```
2^(100 - n) (2k + 1), 1 <= n <= 100, 0 <= k <= 2^n
```

のような形をしていることに気が付きました。
探しているキー K20236360709985574445425860 45 を factor コマンドで素因数分解してみると

3 5 59 22865944305068445701046 17

となり、因数 2 を持たないことが分かります。
したがって、n = 100 のブロックを調べればよいです。
それは index = 2^99 から始まります。

2k + 1 = 20236360709985574445425860 45

を解いて k を決め、index = 2^99 + k の値を調べたところ、探していたキーを発見できました。

ちなみに隠されたテーマというのはよく分かっていません。(笑)

ENV: Ruby
POINT: 勘で解きました。

データベース番号 2
インデクス i (>0) とキー k の関係は lg を 2 を底とする対数として、
  k = (2(i-2^m)+1) * 2^(99-m)
  ただし m = floor(lg(i))
で与えられると推測できるので、これの逆関数を考えればよい (下のプログラム)。

```
def key_to_index(k)
  x = k
  v = 0
  while x.even? do
    x /= 2
    v += 1
  end
  2 ** (99 - v) + (x - 1) / 2
end

p key_to_index(20236360709985574445425860 45)
```

ENV: python + 人力
POINT: db=1 はキーが昇順のため 2 分探索。db=2 は最後のデータからキーの値が-2 ずつ減っていたので、「最大 index - (最後のキー - 検索キー) / 2」で index を求めました。
法則を見つけるのが大変でした><。

ENV: OCaml

POINT: DB1 はインデックスとキーが強い相関を持つことを、DB2 は後半のインデックスでキーが 2 ずつ増加することを、それぞれデータから推定しました。

今回、DB1 の法則はすぐに見抜けたのですが、DB2 は最初の方のデータを注視してしまい法則がわかりませんでした。
最初と最後のインデックスでインデックスとキーの数値が一致していること、インデックス 1 のデータがデータ数のちょうど半分になっていることを考えれば十分に気づける仕組みになっていたのですね。
自分が確認した限りでは、インデックスの最大値に対して 0/1，1/2，1/4，3/4，1/8，... と、ファレイ数列で新たに出てくる値（から 1/1 を除いたもの）を掛けた結果がキー値になっている、で合っているでしょうか。
さすがの問題設定です。面白い問題、ありがとうございました。

ENV: C#
POINT: Binary Search と Modular。

DB2 について、自分の作成したプログラムの計算が
あまりに早く終わったため、間違えたかな？ と思いましたが、
1 回で見事正解を引き当てていました。

ENV: Clojure
POINT: Index と Key の関係が単調増加だと気付けたことです。そのため、2 分探索が可能だと思い至りました。

ENV: 紙とペン
POINT:ルールの発見と多倍長演算プログラミングのスキル
db=1 はキーの値が増加傾向にあったので二分法で解けるはず！ と後回しにして結局時間切れでした。
db=2 はルール発見後、一瞬途方に暮れましたが、末尾が奇数である事に気が付き、index=2^99 以降しかありえない！ と地道に手計算したら意外と簡単に答えに辿り着きました。

ENV: C#
POINT:

■ db=2 について
こちらも index=0〜index=1000 までプログラムで探索してから解きました。
結果のデータを眺めただけではよくわかりませんでした。
単調増加でもなく増加・減少に規則性が見つからなかったからです。
そこで Excel にデータをおいたあと、グラフを書いてみることにしました。
index を横軸とし、縦軸を key とします。
key は数値部分のみをとりだしそれを大きな数で割った値をプロットしました。
(大きな数はいくつか忘れてしまいました)
グラフを見ると以下のことに気が付きました。
* 単調増加する区間がいくつか連なっている
* 区間は index が増加するにつれて長くなっている

* 区間開始時の key は index が増加するにつれて小さくなっている
区間の長さがわからなかったのですが、区間の長さが index の関数であるという予想がつけば
db=1 のときと同じく二分探索で解けそうだと予想しました。
key が増加していないときの index を調べると 2^n〜(2^n - 1) で区間が形成されているようでした。
そこで二分探索を実施しました。

ENV: 手計算
POINT: あきらめずに数列とにらめっこする力
結局、API をたたくためのスクリプトは作成しましたが、
db1 はある程度検討をつけてあとは力技で探索、
db2 は部分的に（？ ）規則性が分かったので手計算で求めました。
全体がどんな数列だったのかまでは把握しきれませんでした。

ENV: C++11
POINT: 1 つ目はキーが昇順、2 つ目はキーの偶奇で前後に分かれてそれぞれで昇順に並んでいるため、二分探索により
キーの位置を求めて対応するバリューを出力する。

C++ は HTTP リクエストや多倍長整数が標準ではないため、そこの実装で一番苦労しました。

ENV: objective-c
POINT: 二分探索だと思います

1 問目は手動で解きました。
2 問目は法則が分かったのでプログラムを作成しましたが、
検索が終わりません・・・。
解説を楽しみにしています。

ENV: Ruby
POINT: 今回は解けませんでした。完敗です。

ENV: Haskell
POINT: 二分探索？
二回目の挑戦です。
データベース 1 は普通に二分探索で見つけました。

データベース 2 は最初解なしかと思いました。
しかし、その根拠となる index から key への関数の間違いに気づきました。
結局電卓で解きました。

ENV: Ruby
POINT: 睡眠がポイントです。睡眠を沢山とって解きました。

夜中にやっていたら寝落ちしました・・・。
それはさておき、各 DB から 10 個ほど持ってきて傾向を読んで、
1 はバイナリサーチ、2 は規則に従って index を計算しました。
答えがばしっと出るので、合っているか悩まなくて済み、
またよく眠れそうです。

ENV: Ruby
POINT: DB1 のキーは昇順、DB2 のキーはキーの数字が奇数の時は後半に等差数列になっていることがポイントです。

・データベース番号 1 のキーは昇順のようなので二分探索で求めました。
・データベース番号 2 のキーはインデクスを昇順に並べると、前半のキーは偶数、後半のキーは奇数となり、
  さらに奇数部分は初項 1 公差 2 の等差数列になるので、キーの位置を計算で求めました。
  電卓で求めたという人のツイートの意味がわかりました。

ENV:This is perl 5, version 20, subversion 1 (v5.20.1) built for x86_64-linux-thread-multi
POINT:db=1 は単調増加なので二分探索、db=2 はデータベースの先頭付近だけで規則性を考えず、末尾、ど真ん中も見てみる。

●２問目
index が 8 * 2**n の時に桁が減る、その時の key は index が 4 * 2**n + 2 の時の末尾の 0 を削ったもの、
…などなど何かしら規則性があるようでしたが、そこから指定の key を求める方法が分からず。。

先頭の 100 個ではなく、末尾の 100 個を見てみたところ、奇数の連番になっていたので、
じゃあ真ん中はどうなんだ？ と思って見てみたところ、コードを書かずに算数で求められました。

ENV: なでしこ
POINT: ポイントはわかりませんでした。
DB2 は解けませんでしたが
なでしこで WebAPI を使うやり方を覚えるよいきっかけとなりました。

ENV: Python
POINT: インデックスの小さな値を出力して、データのパターンを見つけ出すこと

いつも楽しい問題をありがとうございます。
データベース番号 1 は、インデックスとキーが昇順になっていることに気づき、
2 分探索で答えを見つけることができました。

データベース番号 2 は、
・キーがのこぎり型になっていること
・インデックスに対するキーの値がデータベースのサイズと関係があること
までは分かったのですが、その関係式を見つけるところまでには至りませんでした。

ENV: Swift
POINT: 「Key があることが証明できればよい」がポイントです。「Key の順序ルールの推定が正しいかは特に検証せず」解きました。
CodeIQ で初めての回答になります。Swift の勉強も兼ねてこの問題を楽しんでみようと考えました。

ENV: Python 3.2.2
POINT: db1 は力技(バイナリサーチ)、db2 は奇数キーが後半に並んでいることがわかれば、あとは index=(key+max)/2 という計算のみでインデックスが求まる。

ENV: Ruby
POINT: 法則性が見いだせず db1 は二分探索で求めています。db2 はどうアプローチしていいのかさえ分かりませんでした。

Twitter 上の挑戦者たちの「簡単だった」等の感想を見るにつけ、まったく分からなかった自分の数的感覚が鈍いことに愕然としてしまいます。
db2 が未回答ですが解法を知りたかったので参加させていただきました。お手数おかけして申し訳ありません。

サーバーにアクセスしてデータを取得するプログラムは初めてでしたので勉強になりました。
取得したデータをコンソール画面上に順次表示していると本当に難破船にアクセスしてデータを取得してる気分になり楽しかったです。

ENV:Java と電卓
POINT:Key の並び方がポイントです。Key の値を探索して解きました。
面白い問題ありがとうございました。
振り返ってみれば本当によくできた問題でどういう手法で解けばいいのかということが試されている気がしました
気になったのがこの問題を解く過程はどういうものを想定して書かれたのでしょうか
複数の方法、人気のあった解き方などがあれば教えて欲しいです。

ENV: Ruby
POINT: 規則性がポイントです？？？
■感想
１問目は key,index の変換規則がわからず、昇順になっているようなので二分探索をしました。
２問目は昇順になっていませんでしたが、２進数にすると規則性がみえたので変換規則が推定できました。

ENV: Ruby 1.9.3
POINT: 二分探索しました。
db2 はデータの並びに気づくまで時間がかかりました。

ENV: Microsoft Excel 2002 と Windows 関数電卓
POINT:数列の問題だと思います。db2 は群数列の問題として解きました。

ENV: scala
POINT: Q1 は力技 (二分探索), Q2 は bit 演算で idx<=>key が変換出来ることに
気付いて簡単に解けました。

ENV: Java
POINT: 1 問目は二分探索でゴリ押し。2 問目は（キー）=（DB サイズ）*（奇数）/（2 の冪乗）の規則性が見えた。

ENV: Python
POINT: db1 は index が増えるに従い必ず key が増えるようになっており、db2 は奇数であれば index は全データベースの後半に存在することに注目して解きました
db2 のキーの値が 1bit 目が 1 の値でなく、した 4bit が 1000 になる値とかにするともう少し問題が難しくなったかも

ENV: Java
POINT: データベース 1 に関しては、キーが昇順であることから 2 分探索法を用いました。データベース 2 に関してはインデックスが 2^99 の時、キー番号が 1 になること。また、以降はインデックスが増える毎にキー番号が 2 ずつ上がっていくことを利用しました。

ENV: Scala,Haskell
POINT: 根性
番号 1 は二分探索、
番号 2 は法則を見つけた後は手で解きました。

ENV: Ruby
POINT: 2 の 100 乗

どちらもインデックスの総数が 2 の 100 乗になっていてそこからいろいろ発想が生まれました。
DB2 のほうは法則性が見えやすいけど、すこし考えないと機械的にはインデックスがわからなくなっていました。
BD1 のほうが法則性はないけれど、とにかく値が増加していれば、バイナリサーチでなんとか求められそうと
思い当たり、求めることができました。

ENV: Ruby
POINT: DB1 は二分探索で見つけられる。DB2 のキーはインデックスを 100 桁の二進数表示にしたものに比較的簡単なビット操作を施すと得られる。

DB1 は、「おそらくキーは、インデックスに対する増加関数だろう」という推測のもとやっています。なので、もし例外があっても確かめる方法を持ちません。ワナが仕掛けられていないかとビクビクしながらやりました。
インデックスからキーを生成するような関数は見つけられませんでした。もしかしたら下半分の桁は乱数が入ってるかもと思っていますが、実際はどうだったのでしょう。回答に書かれていることを期待します。

DB2 に関しては、インデックスの最大値から 1 ずつさかのぼって行ったら回答はすぐ発見できてしまいました。インデックスを 1 減らすごとにキーは 2 減っていき、かつキーの最大値で奇数のキーは。
一般のキーを見つけるのは少し手こずりましたが、「僕」の原則、「特殊な値を確かめる」を使ってうまく求められました。

ENV: Python
POINT: 二分探索で解きました。DB2 は対象のキーが奇数なので、最下点または連続する 2 数の差が奇数になる範囲を求めてから探索しました。

ENV: Ruby
POINT: 単純に探す事がポイントなのだろうか。もう何だか泥縄でした。一つ目は兎も角二つ目は何故解けたのか？
今日読んでいた伊坂幸太郎さんの PK という小説に
ファインマンの
「物理学者たちは、チェスの試合を見て、チェスのルールを探り当てようとしている」
っていう言葉が使われてるんですけど、ちょうどそんな感じ？ かなあと。
この場合チェスっていうのは「神様が決めた宇宙の原理」みたいな感じなわけです。
今回の問題の「神様」はもちろん結城さんというわけです。
いやまあ時間かかりました。探すだけなんですけど。

ENV: VB.NET
POINT: DB1 は Index に対する Key が単調増加。DB2 は「2 の n 乗で極小値、(2 の n+1 乗）-1 で極大値」をノコギリ型に繰り返すもの。ノコギリの歯部分は一次直線。
感想など：
2 回目の挑戦で失礼いたします。DB2 の時間がかかり過ぎるので見直しました。

DB1 は単調増加ながら Index（以下「I」と表記）の +1 変化に対する K の増分規則が解明できなかったので、I の最上位桁から 1 桁ずつ確定させるアルゴリズムを VB.NET の WebAPI でコーディングしました。

DB2 は I の 0〜16 をサンプル＆プロットしたところで上記 POINT に記した特徴に気づきました。同時に極小→極大の間の増分が等しい事から一次直線であると推定しました。
ここで、API から返る DB のサイズが 2 の 100 乗であることから、上記の推定に従って最後のノコギリの歯を調べると、2 の 99 乗を「a」、100 乗を「2a」として、I に a を与えると K に 1 が戻り、2a-1 を与えると 2a-1 が戻ることが判りました。従って I と K の関係は、K=2I-2a+1 であり、I について解くと、I=(K+2a-1)/2　が得られます。既知である K と 2a を代入して得られた I を DB2 に対する INDEX として WEB API に渡すと

2 634837118149613979470622895710 K20236360709985574445425860 45
V110194355767592072223813698 1003 126765060022822940149670320 5376

が戻り、所望の K に対する V が得られていることが確認できました。

ENV: Python
POINT: 大まかな予測で数値を絞った。db2 は 16 進数に変換して求めました

著作の方、愛読させていただいています
基本的なアルゴリズムで回答がでるものの
現実的な時間で回答を得られるように
人の手と PC に任せるところとを工夫するのは楽しかったです。

ENV: Javascript(Node.js)
POINT: db1 は KEY が INDEX に対して単調増加と仮定し、二分探索を行いました。db2 は数列を特定しました。

ENV: DB 1 は php、DB2 は手計算
POINT: 大きな数の演算がポイントです。DB1 は大きな数 index を 2 分探索して解きました。DB2 は 2 の累乗のデータから類推して筆算で解きました。

ENV: Perl,Excel
POINT: 解き明かさなくても見つかりさえすればよい。

db1 は、キーの値に対してバリューの値がほぼ直線で短調増加しているように見えたので、
二分探索を行なった結果、index=256507035290138824303049 1 で目的のキーが見つかりました。
二分探索を開始してから、97 回 API にアクセスしました。

db2 は、キーが 2^n 〜 2^(n+1)-1 の範囲で、バリューの値がほぼ直線で短調増加しているように見えたので、
各範囲ごとにまず最小値（範囲の最初の値）と最大値（範囲の最後の値）を取得し、
比例計算によっておよその index を求め、その付近を探していきました。
その結果、index=6348371181496139794706228957 10 で目的のキーが見つかりました。
この方法での探索を開始してから、388 回 API にアクセスしました。

ENV: ruby
POINT: db=2 は、index の数字が 2 の累乗の周期で変動することに気付くのが大事。

ENV: C++
POINT: 2 分木探索とヒープソート的な格納された完全 2 分木ですね！

私なりの解説記事を下記 Wiki に書きました。
詳細はこちらに書いておきます。
http://tessy.org/wiki/index.php?CodeIQ%2F%A5%B5%A5%EB%A5%D9%A5%B8%A5%AA%A5%F3%CC%E4%C2%EA

ENV: Python3.4
POINT: 数回の試行による規則推定。db1 は 2 分探索、db2 は因数分解。

久しぶりの CodeIQ 出題、嬉しいです。
今回も楽しく解かせていただきました。
db1、db2 ともに比較的単純な規則だったので、さくっと解けました。
API にアクセスせずに VALUE を計算する方法もあるのかなと思いつつ。。

ENV: Python
POINT:
データベース 1 はインデクスの値を大きくするとキーの値も単調増加するようだったので二分法でキーを特定しました。
データベース 2 はインデクスの値を大きくするとキーの値はジグザグと単調増加→激減→単調増加を繰り返しており、ある自然数 n について、x=633825300114114700748351602688 として、インデックスの値が 2^n から 2^(n+1)-1 の間は、キーの値を x/2^n+x/2^(n-1)*α （αは 0<=α<=2^n-1 の整数）で表せることが分かったので、目的のキーの値を x/2^n で引いた値が x/2^(n-1) で割り切れるかどうかを n を 0 からインクリメントしていって調べることでキーを特定しました。データベースの範囲内では問題のキーは 1 つのみ存在するようです。

ENV: java + javascript
POINT: db1 はバイナリーサーチ。db2 は index と key の関係（数列になっている。）から key の位置（index）を予想する。

・db1 については、数列の法則が一般化できなかったので、自然増加しているので単純にバイナリサーチを選択した。
おそらく一般化できるのではないかと思うので、残念。

・db2 については、key が 2 の乗数をもとにした数列であることが分かったので、index を計算し取得できた。
一つの数列の数式にするのは難しかったが、
index 2^n 番目（ 0<= n <=100 を満たす自然数）ごとに、等差数列になっている。
2^n ごとに　index m　を与えるとすると
index 2^n + m　：　2^(100- n- 1) + 2^(100-n) * m　{ 0 <= m　<= 2^(n-1)}

ある程度数列がわかった時点で、与えられた key が奇数であったので、奇数となり得るのは index 2^99　〜 2^100 -1 の間に絞られおおよその位置はわかったが、数列を式にすることにすることにこだわってしまい、時間が非常に掛かった。

ENV: Java
POINT: 二分探索がポイントです。探索範囲の中央のインデックスのキー値と目的のキー値を比較することで探索範囲の絞込みをおこないました。
　　データベース 1 についてはキー値が全て昇順のため、
　　全範囲で二分探索をおこないました。
データベース 2 については奇数のキー値については
データベースの配列の中央以降に昇順で並んでいるため、
配列の中央以降で二分探索をおこないました。

ENV: Scala
POINT: db1 は昇順なのでバイナリサーチ、db2 はインデックスが 2^x から左は 2^x の倍数、右は 2^x 以外が昇順で並んでいる。

ENV: Julia
POINT: キーに重複がないことが大事。実装面では、多倍長整数や有理数を扱いやすい言語だととても楽

1 番、2 番ともに最初の 10 個ぐらいを表示して、キー順序法則の予想をした

1 番は、詳細は不明だが少なくともソート済みだと予想したので、
二分探索に掛けたところ無事に見つけることができた

2 番は 2^99（index=1）との比を取ると前から
0/1, 1/1, 1/2, 3/2, 1/4, 3/4, 5/4, 7/4, 1/8, 3/8, ...
のように、約分した分母と分子に関してソートされている
欠損値がないと仮定して、目的のキーが何番目にあるのかを計算し、
実際にそのインデックスに目的のキーがあることを確認した

ENV: D
POINT: ソフトウェアの試験と同様に、境界（先頭、末尾）および中央などのインデックスのキーをあらかじめ確認し、キーの規則性をつかむ。

■感想
何回か手動で API を動作させ、規則性を見抜くことを重視して問題にあたりました。
規則性を見つけられるかどうか、自信はなかったのですが、
運よく、特徴的な結果が出たので、思ったよりもすんなりと解答（おそらく）にたどり着けました。

ENV: Java
POINT: DB の内容を見て Key の並びを推測し、二分探索ではなく内挿探索にしたところです。
謎解きのようで楽しかったです。
DB2 は index が前の方ばかり見ていて最初絶望しかけましたが、Key が偶数ばかりなのに気づけて良かったです。

ENV: Ruby
POINT: DB1 は INDEX 全体、DB2 も INDEX が [2^n, 2^(n+1)-1] の範囲で KEY が昇順に並んでいるところがポイントです。昇順部分を二分探索して解きました。DB2 は昇順部分が等差数列であることに注目すると時間短縮ができました。

ENV: Java
POINT: インデクスとキーの関係を推測することがポイントです。1 はプログラムで、2 は手作業で解きました。
■感想など
「スペーストーキー問題」と同様に簡単に答え合わせができるので、解答提出を早く決断できました。
「チケットゴブル問題」のような簡単には解答に自信が持てない問題の方が長く楽しめます。
でも、今回、2^100 個ものデータをサーバに格納しているとは思えないので、インデクスからキーを、そして
インデクスとキーからバリューを導出できる計算式が何かあるはずで、これを見つけるのが解答提出後にも
用意された楽しみなのだと思っています。とはいえ、今のところさっぱり分かりそうにないのですが。

ENV: Python
POINT: バイナリサーチ

ENV: Java
POINT: 巨大整数の規則性。最初に 100 件手動で取得して，並び替えたり，掛けたり割ったりしてみました。

感想：
規則を見つけてからも，巨大整数の演算に苦労しました。
一番馴染みのある文字列型で計算式を作ってみたのですが，繰り上がり処理ひとつ作るのにも試行錯誤…
普段何気なく頭で行なっている計算は，プログラムに書くと意外と高度な処理になって新鮮でした。

ENV: Python
POINT: db1 ではすべての key が昇順になっていて、db2 では index が 2 の n 乗になるごとにある規則に従って key が並ぶということがポイントです。db1 については二分探索、db2 については n を決めたときの式を解いて n について全探索しました。

1 回のアクセスごとに 1 秒待たないといけないということで、アルゴリズムの性能がわかりやすく時間に反映されて面白い問題だなと思いました。

私の解答は対数オーダなので約 100 秒ぐらいで終わるのでは無いかと思います。

ENV: Ruby
POINT:2 分探索を使ってとくことができるかがポイントです。特に DB2 では DB 全体に 2 分探索をかけられないため, キーのルールを発見することが必要です.

ENV: Python 2.7.8
POINT: db1 は、バイナリサーチで計算量を抑えつつ、API へのアクセスを必要最小限にすることがポイントです。
db2 は、インデックス最大値のレコードを起点に、インデックスを 1 デクリメントするとキー値は 2 デクリメントするという規則性から算出できました。
以下は db1 用のコードです。

ENV:
POINT: すべてを把握しようとせず解答を求めることがポイントでしょうか。

ENV: Ruby
POINT: 各データベース毎のキーの並びの規則。db1 はキーが昇順に並んでいるようだったので二分探索でアクセスしながら、db2 はキーからインデックスを求めて値を得ました。

ENV: Python2.7
POINT: 単調増加ー>２分探査。２進数。
db2:
　　２進数でみるとよく分かる！
　　i のインデックスを持つ key の値は n を 2^n <= i < 2^(n+1) として
　　key = 2^(99-n) * (2*i - 2^(n+1) + 1)
　　の関係にある。
　　key=20236360709985574445425860 45 は奇数なので n=99 がわかる。
　　すると i = (key + 2^100 - 1)/2 = 6348371181496139794706228957 10
　　がわかる！

ENV: Python3
POINT: データベース 1 index 増加で必ずキーが増加するのがポイント。二分探索が使えた。
データベース 2 キー生成のアルゴリズムに法則性があり、そのアルゴリズムをプログラムで実装する事で、キーの index(=6348371181496139794706228957 10) を取得。

ENV: 最終的に「R」。途中経過で電卓，Excel，少しだけ python
POINT: DB1 はキーが単調増加。DB2 はキーを 100bit の 2 進数で表しいちばん下の 1 の桁を上下反転させたものに残りの桁を加えたものの順に並んでいる。(キー Q を割り切れる最大の 2 のべき乗を 2^k(k>=0) とした時，2^(99-k)+(Q-2^k)/2^(k+1) が index になる)
#最初の 20 個と最後の 1 個を見て，データの個数が 2 の 100 乗で index と key が重複しないことから 2 進数の 100 桁の数字の桁の入れ替えで作られているようだとわかりました。

ENV: Node.js
POINT: 総当たりでは膨大な計算量になる問題を、いかに端折って最短でデータを特定できるかが重要だと思いました。

ENV: C++
POINT: 1つめは単調増加と二分探索がポイントです。2つめは二進法と階差がポイントです。

二分探索をしたら値がピタリと出てきたのは、予測どおりとはいえ気持ちよかったです。
2つめの規則性がなかなか分からず手こずりました。
整数が long long int に入りきらなかったので、多倍長演算を組むことになり、そのあたりが面倒でした。
多倍長をサポートしているような言語も学んでみようと思います。

ENV: R
POINT: DB1 と DB2 の先頭 100 件出してあたりをつけました。DB1 は法則がわからなかったので二分探索、DB2 は法則がわかったので R の gmp を使って計算を一回するだけでわかりました。

ENV: Python
POINT: db1 はキーが昇順に並んでいる。db2 は 2^n ~ 2^(n+1)-1 までのキーが等差数列になっている。

両データベース共に最初の 100 個を取ってきて眺めて、ルールを探しました。

ENV: C#
POINT:始めに db1、db2 それぞれ 0 番から連続した 60 個前後のデータを取得、内容を確認。db1 はキーの値が単調増加しているので、二分探索を用いる。db2 は配列のような連続したアドレスに収められた 2 分探索木なので、それに従って探索。データベースのサイズはどちらもおよそ 10^30 ≒ 2^100 個であり、二分探索はおよそ 100 回以内に成功する。実際、どちらも 100 回ほどで目的のキーを発見できた。
データベースの構造は基本的なデータ構造を知っていれば容易に推察できたので、探索するアルゴリズムは難なく書けたのだが、むしろデータベースにアクセスする HTTP リクエスト部分や、decimal 型にも収まらない巨大な数字を扱う部分が厄介で時間がかかった。

ENV: D 言語

POINT: 二分探索と奇数がポイントです。手でいくつか試して法則を推測したあとプログラムで探索しました。

ENV: Ruby
POINT: 業務として行う設定だからあまり美しくなくてもなんとかして答を得ることが重要。

ENV: サクラエディタ, 電卓
POINT: db1 は key は昇順で並んでいるとだけ分かり手で探索。db2 は後半の値の法則から式をたてた

ENV: C#
POINT: System.Numerics.BigInteger がポイントです。似たようなクラスを自作した後で存在に気付いて唖然としました。
ENV: Python 2.7.8
POINT: db1 はキーの値が単調増加なので、二分探索で。db2 はインデックス区間 [2^p,2^(p+1)-1] でキーの値が単調増加なので、区間ごとに二分探索を繰り返しました。

それぞれ先頭 100 個取得してデータの傾向が読め、ある程度時間をかければ二分探索で答えにたどり着ける目途がついたので、あとは、何度も動作検証する際の効率化するために、一度検索した結果を sqlite に保存するようにしました。

ENV: Ruby
POINT: 最初の 10 データから index に対する key の法則を推測するのがポイント。DB1 はキー番号昇順なので 2 分探索、DB2 は数列的な法則だったので手計算後の 1 アクセスで求めることができた。

ENV: Java
POINT: DB1 は key が昇順であることから 2 分探索。DB2 は 0〜10,max-10〜max までの値を眺めて、要素数が 2^100 である所から、中間点の 2^99〜2^99+10 を見たところで解けました。
いつも楽しい問題をありがとうございます。
DB1 を線形探索したときの絶望感。DB2 をバイナリサーチしたときのしてやられた感。いつも一歩先に仕掛けが置いてあり、ワクワクしながら解いています。
今回は、DB2 の一般解から答えに辿り着くことができなかったのが悔やまれました。

ENV: Python
POINT: キーの桁数に怯えるな！ パターンを見抜け！

db1 は昇順に並んでいるようなので二分木探索でいいかな、と思ったものの、
意外と見つけるまでに時間がかかる（1 回接続する度に 1 秒待つので）のは引っかかる…。
結城先生のことだから、そんな単純なものではなく、何かパターンがあるはず！ と、
とりあえず、それぞれどれくらいずつ増えていくのか、適当な index で、

(key の値 - index=0 の key の値) / index
を出してみたところ、ある数字が出てきました。
index を他の数に変えてみても答えは同じ。
毎回綺麗に割りきれるわけでは無いので、その数値ごと増えている訳では無いのが不思議ですが、
この数字を使って、答えに行き着きました。

db2 の方は、20 個ほどデータを持ってきたところ、
これはこれの 2 倍、ここは 1/2 とか考えていくうちに、
どういうパターンなの閃きました。
個人的には、db2 より db1 の方が難しかったです。

ENV: Ruby
POINT: キー順序のルールの把握がポイント。1 は昇順、2 は 2 のべき乗個ずつで昇順。
データベース番号 2 は、キー値のグラフを描くことでルールが把握できました。
おそらく「2 のべき乗個ずつ」に何か数学的な意味があると思い、
考えをめぐらせたのですが、それ以上のことは把握できませんでした。
キーの探索には二分探索を用いました。

ENV: Ruby
POINT: ほぼ人力。 エレガントに解きたい...。

ENV: Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
POINT: データベース１のポイントはバイナリサーチです。データベース２のポイントは循環です。

ENV: Ruby
POINT: DB1 は二分探索で、DB2 は対象キーが何番目の奇数かなどを求めて探しました。

ENV: C#
POINT: 二分探索がポイントです。データベースを昇順ブロックに分解して解きました。
ENV: Ruby 2.0.0
POINT: 手作業で WebAPI を叩いて、Index と Key の関係を見つけ出すことができるかがポイントです。
db1 の法則は簡単で、Index を 1 から順に入力することで、Index が増えれば Key も増えることがすぐにわかりました。
db2 はすぐにはわかりませんでした。Index を 1 から入力しても、Key に法則は見つからず。
試しに、SIZE/2 あたりを入力すると、633825300114114700748351602688 から Key が増えることに気づきました。
まあ、当てずっぽうが功を奏した、といった感じです。
求める Key に近い Index を探して範囲を絞れば、後は二分探索でサーチすればいいだけなので、コーディングは簡単でした。

ENV: Java
POINT: 任意の長さの 10 進数を格納できる BigDecimal クラスを使うのがポイントです。Excel のグラフで index と key との関係を推測し、Java でプログラムを組んで解きました。

ENV: Java
POINT: db2 は全体の規則性が分かりませんでした。根気良く探ってたら奇数のキーに限って二分探索が使えるのを発見して、何とかなりました。

ENV: Ruby
db 1 : 単調増加っぽいので二分探索した。
db 2 : f(x):=index が x のときの key の値として、g(x) := f(x) - f(x-1) のような差を取っていくと、g(2^k) だけが負で、それ以外の 2^k<=x< 2^(k+1) の範囲で g(x) は 2^(100 - k) ずつ増える増加関数となっている。よって key - f(2^k) が 2^(100-k) で割り切れてかつ 0 <= (key - f(2^k))/2^(100-k) < 2^(k+1) となる k を見つければ 2^k + (key - f(2^k))/2^(100-k) が答えの key の index となる。index の範囲は 2^100 未満なので単純に k を 1 から 99 まで見ていった。

ENV: Perl
POINT: DB1 は二分探索，DB2 は規則性
DB1 は key がソートされていると予想して二分探索した。

ENV: Ruby
POINT: index の範囲がとても大きいので最初は 1~100 くらいの小さい範囲で API を呼び出して規則性がないか探してみること。
db1 では index の値が昇順に並んでいたので単純な二分探索で、
db2 では index の値が [2^n, 2^(n+1)) の範囲ごとに昇順に並んでいたので各区間で二分探索をしました。

問題文を読み終えた直後はどうすればいいかすぐには思いつかなかったですが、まずは小さい値から試してみることで規則性に気づくことができました。
とても楽しみながら解くことが出来ました。
ありがとうございました。

ENV: C#
POINT: [0, 100) の範囲のインデックスのキーを列挙してみて、diff を取ってキーの並び方のパターン性を考えました。

ENV: 手動、電卓（irb）

POINT: db1 は key が規則性不明ながら単調増加っぽいので手動で二分探索、db2 は key が群数列になっていて index との関係が求められる。

db1 はもっとスマートに解ける気がするんですけどね。

ENV: Ruby + 視覚
POINT: 目視で確認してその気になるのがポイントです。法則を確認して解きました。

結城浩先生へ

救出したデータの意味を解読して、悪の組織の陰謀を阻止することも考えましたが、
任務はデータの救出のみなので、そのようにいたしました。(^^)

ENV:人力
POINT:根気よく手打ちです。
ひたすら手打ちして、パターンを読取りました。数学的な事は一切使っていません。

ENV: Python
POINT: データベース 1 は key の値が単調増加となっており、データベース 2 は key の値が 2 のべき乗ごとに単調増加となっている規則性を利用して key を探しました。大雑把に範囲を絞り込み、最終的には二分探索を使って目標となる key を見つけ出しました。
ENV:
POINT: DB1：キーの数値が増加数列であり、増加量が一定の範囲内でおさまっていそうだったので、大きなインデックスの値から平均の増加量を算出し、そこから類推しました。　DB2：キーの数値が 1,2,4,8,16 と半減していったのと、2 の累乗でない数の場合、直近の 2 の累乗の値の奇数倍であるので、そこから算出しました。

ENV: 手
POINT: DB1 はキーがインデクスに対し昇順。DB2 はキーの 2 進表記からインデクスが定まる。

DB1 は、インデクスを上の桁から順に確定させていけば、手作業でも現実的に求めることが可能。
二分探索をすればより効率的に求まるが、手作業ではつらいのでやめた。

DB2 は、キーを 100 ビットの二進数で表記し、先頭に'1' を付加したのち、
末尾のすべての'0' および 1 つの'1' を除去し、最後に十進変換でインデクスが定まる。
基本的に手作業だが、電卓として Ruby を使用した。

ENV: PHP
POINT: KEY の順序を推測すること
DB=1 は、INDEX0-9 を取得した結果より、KEY が昇順にソートされていると仮定して、

最小（=0）と最大の INDEX（=SIZE-1）から、二分探索を行いました。約 100 回で発見しました。
DB=2 は、INDEX0-19 を取得した結果より、（INDEX=0 以外は）
KEY=(2 ^ (100 - n)) * (2k - 1)
k = 1, 2, 3, ... , 2 ^ (n - 1)
n = 1, 2, 3, ...
と仮定して、2 の累乗の和 + 探す KEY/2 の近辺を探しました。

ENV: PHP
POINT: DB1 はバイナリサーチ．DB2 は???

DB1 はすぐに解法を思いつきましたが，DB2 の方は規則を見いだせずに苦労しました．

ENV に PHP と書きましたが，PHP を使ったのは DB1 だけになります．
DB2 は計算して Key を求めました．

ENV: Ruby（2.0）
POINT: 二分探索と等差数列

ENV: 脳内プロセッサ + 電卓
POINT: DB である以上、key と index の間には些少の計算量で値を求められるという関係があるはず。その関係を解き明かせば、後は計算するのみです。

ENV: Ruby
POINT: Key の並びを効率よく探すのがポイントです。等間隔に 100 個 Key を出力して、Key の並びを推測しました。
ENV: 電卓（Win8.1 上の）
POINT: キーのパターンの仮説を用意してその検証を行うこと。
DB1 はキーが昇順にソートされていると仮定して、単純にバイナリサーチでキーを探り当てる方針で。
Windows の電卓が関数電卓モードで本設問のキーの桁数が扱えたので、力技で。
(なので、ソート済み、という仮定以外キーの法則性については未検証です)
DB2 については、いくつかキーを見てみたところ、リスト後半が奇数の連番になっているとデータパターンに見えたので、その仮説に基づいて検証 (今回提示されたキーは奇数だったので、後半部に順番に並んでいると仮定)。
リスト上のキーの位置は上記仮定に基づくと単純な計算で分かるので、こちらも電卓を使用して計算。
双方の結果は API に問い合わせたところ、問題ないようなので、それを回答としました。

ENV: 手計算（電卓）
POINT: キーの数字の増え方をよく見ると何となく分かるかも。。。

・db2 について
こっちはキーの数値が増えたり減ったりしていましたが、
しばらく見るとしばらく増え続けて急に減るというのを周期的に繰り返しているのが分かって、
さらに見ていくとその周期はだんだん長くなって最後は延々と 2 ずつ増えている状態になっていて、
さらに調べると周期は２倍ずつ増えていて、増え幅は半分ずつに減っているというパターンまで見えてきました。
設問のキーは奇数なので、最後に延々と続く奇数のキーの中にあることが分かり、
インデックスが計算できました。

ENV: C#, Haskell
POINT: 二分検索がポイントだと思いました
一つ目はキーの値が単調に増加していると予想し二分検索の要領でキーの範囲を絞り込みました
大きい桁から順に数値を指定して目的のキーの前後を調べていきました
二つ目のキーを求める計算を Haskell で書きましたので添付しておきます

ENV: Ruby
POINT: インデックスの取り得る範囲と前後のキーとの関係に着目して考えました。
感想:
db1 はキーが単純に増えていっているのには気付いたのですが、規則性はハッキリとは分かりませんでした。
適当にインデックスの桁数を増やしたり、半分にしてみたりして、一桁ずつ近い値に寄せて求めました。
我ながら無理やり脳筋スタイルでがんばりました。

db2 は規則性を見つけられたので Ruby で求めました。
前後の差に着目したところ、等差数列っぽい特徴があったのでそこから考えました。

ENV: Python
POINT: 規則性を見付けることがポイントです。
いくつか試行錯誤することで 1 は直線であることがわかり、
2 は 2 のベキのときに指数関数的に減衰し、それ以外は線形であることがわかったので、
それぞれに対する式を求めて、解を予測して解きました。

ENV: Ruby
POINT: db1 は index の増加に合わせて key も増加していることがポイントで、二分探索法を使用することで現実的な探索回数で解きました。db2 は、key の数値に 2 の倍数がいくら含まれているかで index を逆引きできることがわかり、これを使用して解きました。

V406435859539156181269150751031
V1101943557675920722238136981003
ENV: Python
POINT: 1 番は Key の 2 分探索で解きました。2 番は key を 100bit の値として、末尾 0 をカットして論理右シフトする

と index の値になりました。

面白かったです。有難うございました。

ENV: Perl
POINT: キーの値が増加する範囲を求め、二分探索しました。

キー分布のアルゴリズムはわかりませんでした。

db=1 は index 順にキーの値が増加していたので、
手操作で求める値を含む範囲を狭め、残りは二分探索しました。

db=2 は一定範囲でキーの値が増加していますが、
つぎの範囲では前よりもキーの値が小さくなっていることもあり、
単純な二分探索は使えないようでした。

そのため、求める値が含まれる範囲（桁数が同じになるところ）を
ランダムに求め、求めた範囲で二分探索しました。

ENV: しらみつぶし
POINT: 正直、二分探索法で検討つけてました。後で送られる解説でアルゴリズムを理解してみようと思います。

ENV: Groovy
POINT: キーの規則性を見つけることがポイントだと思いますが、結果として Try&Error の側面が強くなってしまいました。1 件目は 2 分探索で、2 件目は Key の Chain を辿って見つけました。

ENV: Ruby
POINT: 2 分探索を使いました。
db=1 は難なく解けたのですが、db=2 が最初はうまく行きませんでした。結城さんが作成される問題なので、両方とも同じようには解けないとは思っていたので (^^)、db=2 は他の探索アルゴリズムを使うのかとも思ったのですが、2 分探索の左端を index=1 で始めることで無事、解くことができました。

ENV: Ruby
POINT: 二分探索

ちょうど数学ガールの乱択アルゴリズムを読んでいたのですぐにピンときました。
宇宙全体ですら 300 ビットそこそこで表現できるんですからこのくらいは余裕ですね。
とはいえ、ジョン・ベントリー先生の仰るとおり、二分探索は考え方が簡単な割に
（正しい）実装は難しく、できるまでちょっと苦戦してしまいました。

ENV: Ruby
POINT:
データベース 1 はキーの番号がインデックス順で単調増加していることが分かったので、
二分探索で目的のキーを持つインデックスを求めました。
データベース 2 はインデックスが 2^n から 2^(n+1)-1 の間でキーの番号が等差数列になっていることが
分かったので、目的のキーが 2^n から 2^(n+1)-1 の間の等差数列に含まれているかどうかを
判定し、含まれていなければ n を増やす、という方法で目的のキーを持つインデックスを求めました。

ENV: python
POINT: 2**100 is large!

ENV: Ruby
POINT: DB1 は二分探索で、DB2 は index と key の一般項から逆算しました。

ENV: Haskell
POINT: 一問目はキーが単調増加であることを利用して二分探索で解きます。二問目はキーからインデックスを計算して解きます。

ENV: Java
POINT: db=1 はキーの数値がソートされているため二分探索で、db=2 は index が 2^n+k（n = 0〜100、k = 0〜2^(n+1)-1）の時、キーの数値が 2^(100-n)+k*(2^(101-n)) となるため、計算で解けます。
db=2 のルールを見つけるのに苦労しました。
db=1 の増分にルールがあるかどうか分からないのが悔しいです。

ENV: Python,WolframAlpha
POINT:大きい数の規則性を見つけること

DB1 はキーが単調増加している（らしい）ことに気づき、適当な探索方法で
計算オーダーを落とせばよいと考えました。実際うまくいきました。
DB2 はキーは全てが単調増加ではないのですが、単調増加している区間が
2 の累乗の長さであり、また DB のサイズが 2^100, キーの最初の数個の値が
0,2^99,2^98,2^98*3,…であったので、規則性に気づきました。
DB1 で二分探索を使ったので、DB2 の規則性を見つけた時は、なるほどと思いました。

ENV: 全手動
POINT: めげない心

ENV: 手
POINT: 1 問目は WebAPI を 2 分探索しました。2 問目は index - (index - key) / 2 です。

ENV: C++, Maxima
POINT: バイナリサーチを使えば強引に解けますね（笑）
Visual Studio では_int128 がサポートされていないので自分で桁数の多い計算をするプログラムを組んだりして楽しかったです。
db2 はパターンが分かったので後はプログラム任せでしたが、db1 はパターンが分からなかったのでバイナリサーチを掛けました。それも手動で（笑）
Maxima なら桁数の多い計算もやってくれるので、それで計算しながら決めて行きました。頑張った私！
なので、db1 はプログラムの問題をコンピュータの手助けこそ借りましたが手動で解いたことになります。何じゃそりゃ。

ENV: Haskell
POINT: 手動で api をコールして、インデックスの順に Key が並んでいるような気がして、 二分探索を行なうことがポイントです。db1 はインデックスが 0〜最大値の範囲で 2 分探索をして解きました。 db2 はインデックスが 0〜最大値の範囲で 2 分探索行なったろころ、 0〜633825300114114700748351602687 の範囲内には答えがなさそだったので、 インデックスが 633825300114114700748351602687〜最大値の範囲で 2 分探索をして解きました。

ENV: Python
POINT: DB1 は二分サーチ、DB2 は key のルールさえわかれば簡単。

DB1 は key が昇順に並んでいるようなので、二分サーチすれば多少時間は掛りますがすぐに見つかりました。
DB2 は初めはルールがわからずなやみましたが、一項前の key と差分を取った所ルールがわかり、解にたどり着きました

ENV:手
POINT:DB1 は二分岐探索,DB2 は前半偶数, 後半奇数が並んでいるのでそこから探す

ENV: Python
POINT: 1 つ目は先頭の桁から順番に、2 つ目は 2 の累乗を超えた時に隣り合う key の差を利用して答えの index を探すのがポイント

ENV: Python3, Perl5
POINT: DB1 のキーは数字の小さい順，DB2 のキーは 100 ケタの 2 進数にしてビット逆順にした数値の小さい順に並んでいる
最初に題意をとりちがえて全データベースで同じ規則を使っているものと勘違いしており時間を溶かしてしまいました．

適当にサンプルをとってきたところ DB1 は小さい順だったようなので，素直に二分探索を行ないました．

DB2 は最初の 10 個をとってきて，インデックス 1 の 633825300114114700748351602688 で割ってみたところ，
0，1，1/2，3/2，1/4，3/4，5/4，7/4，1/8...
と規則性がみつかったのでそこから逆算しました．
（答えを見つけたあとで 633825300114114700748351602688=2^99 に気がつきました）

今回の CodeIQ は手作業で答えの確認ができるのがいいですね．

ENV: Python
POINT: キー値のおおよその分布を掴むこと、仮定の一般性を検証することが重要だったと考えます。

答えを得るまでの過程を簡潔に付します。
db1
・概観を得るため、データベースのサイズを把握した上でランダムなインデクスで 100 回サンプリングするプログラムを作成
・インデクスでソートしたデータを見て、キー値が昇順になっていると仮定
・インデクスとキー値がおおよそ比例していることを利用し、探すキー値近傍のインデクスを得てから二分探索を行うプログラムで目的のデータをゲット

db2
・同じくランダムなインデクスで 100 回サンプリングして、キー値に何らかの周期性があると気づく
・インデクス 1~100 を取り出してみると 2^n 個ごとのグループにまとまっていたので一般項を考える
・これだと偶数のキー値しか出現しないことに気づく（探すキーも奇数だし）
・最初の概観を再確認して奇数は全て後半に集まっていると仮定
・インデクスがちょうど真ん中のデータを確認すると K1 が得られたので、前半のキー値 +1 で並んでいると仮定
・真ん中から始めて順番にいくつかキー値を確認してみるとただの奇数列
・後半部分のデータを 100 回サンプリング
・どうやら後半は単純に奇数列になっていたようなので探すキー値からインデクスを逆算して目的のデータをゲット

というわけで規則の考察には至っていませんが、楽しませて頂きました。

ENV: Python 3.4.1
POINT: インデックスとキーの関係がポイントです。DB1 はキーが単調増加しているので二分探索で、DB2 はインデックスの後半に奇数キーが固まっていることを利用して解きました。

「数学文章作法　基礎編」持っています。推敲編も執筆されたのですね。当選しなかったら買います。

ENV: Python 2.7.8
POINT: db1 は二分探索で解き，db2 は index と key の関係性を導き出しました。

db1 について:
いくつか API を叩いてみたところ，key は昇順に並んでいるようだったので二分探索が使えると考えました。データベースのサイズは 2^{100}なので比較回数は最大で 100 回，API を叩く間隔を考慮すると 100 秒程度で答えを求めることができるはずです。早速プログラムを組んで実行してみると，98 回目の比較で答えが見つかりました。

db2 について:
このデータベースについては，key が昇順に並んでいないようです。しかし key の並び方に法則性が見られました。先頭の 20 件の key を書き出し，紙とペンを使ってその法則性をいろいろと考えたところ，key をデータベースのインデックスから生成する式が分かりました。

インデックスが n の時の key を key_n と表すと，
key_0 = 0
key_n = key_{i+j} = 2^{99} / i * (2*j + 1)
ただし，i は n 以下で最大の 2 のべき乗数，j は 0 及び i 未満の自然数

となります。与えたれた key からそのインデックスを導く時は，key が必ず 2^{99}/i で割り切れることを利用してまず i を求めます。あとは上記の式を j について解くだけです。この方法をプログラムで実装したところ，無事に答えが見つかりました。

感想:
db2 が特に面白かったです。数学ガールの一番最初の内容を思い出しました。いち，いち，に，さん。答えを出したあとでも，どうしたらこれをきれいな式で表現することができるかあれこれ考えていました。

ENV: Java
POINT: BigInteger 優秀 v(^_^)v

ENV: C#と Excel
POINT: DB1 は単調増加の特性から二分探索，DB2 は 2 の累乗と基数の変化に注目
DB1
単調増加であることがわかったので二分探索を使いました.
n=100 桁だろうがアクセスの 1 秒待ちがあっても log(n) なので余裕ううううう

ENV: PHP
POINT: キーの順番のルールがポイントなので、サンプルに 100 件ほど収集し、ルールを確認してからスクリプトで結果を求めた。

ENV: Ruby, 手計算
POINT: db1 は二分法, db2 は群数列の項数を手計算
ENV: C# .NET（.NET Framework 4.0）
POINT: クエリを使用せずにいかにデータを検索するかが最重要ポイントだと思います。2 分木検索で解きました。

学生のころからデータベースといえば SQL だったので、
SQL を用いずにデータを検索したことがなく、概念として知っていた 2 分木での検索を初めて実装しました。
データがソートされていて良かったです。

また、Web API から返されるデータベースのサイズが大きすぎて、
扱ったことのない桁数の数値だったのでその点でも苦労しました。

勉強になりました！
ありがとうございました！

ENV: Scala
POINT: 二分探索木

とても面白かったです。WebAPI の勉強にもなりました。
ところでデータベース 0 のキーの並びにも法則性はあるのでしょうか。

［結城：ありません］

ENV: メモ帳, Java, 根気
POINT: 桁数が多いですね…多倍長整数を扱える環境が必要です.

db1 の問題は, index に対して単純増加だと分かったので, メモ帳使って一桁ずつ求めていきました.
db2 の問題は, 最初は, 2 進数に直して逆から読むのかなぁ…とか思ったのですが, どうやってもダメっぽいので,
index の最後の方から確かめていくと, key が奇数の場合は何か簡単に求まることが分かったので（説明がめんどくさい）
求めることが出来ました.
ありがとうございました.

ENV: 手計算
POINT: bd2 は Index と Key に簡単な対応関係があったので計算は楽でした。が、db1 はソート済であることから手作業で 2 分木探索しました...

ENV: Java(Android)
POINT: 先頭のいくつかのデータから並びの規則性を判断することがポイントだったが、共通するキーワードは「半分」かなと思った。

ENV: php
POINT: db1 は index と key の値が昇順になっているので 2 分岐で調べた。db2 は index の最大 (126765060022822940149670320537 5) から-1 する毎に key の値は-2 されている規則性があったので K20236360709985574445425860 45 との差分を計算した

個人的に丁度良い難易度で面白かったです。

ENV: Python, bash
POINT: KEY を INDEX の先頭から少量取得して法則性を特定し、KEY に対応する INDEX を求めるコードを書きました

ENV: Perl
POINT: db 1 は key が単調増加していること、db 2 は key と index の関係を調べて解きました

ENV: Python
POINT: Database 1 は increment に key が増える、Database 2 は、奇数の key は index が 2^n の key と index が 6338253001141147007483516026 87 から 126765060022822940149670320537 5 の間にしか存在しない。

ENV: ほぼ手計算（DB=2 の２進数化だけ Windows の電卓では桁が足りないので Java7 でコードを書きました）

感想ですが、正直なところ、前問『チケットゴブル』のほうが面白かったですね。
本問は、なんというか「面倒くさい IQ テスト」をやってるみたいでした。

[結城：すみませんです……次回はもっとおもしろく！ (^^)]

ENV: Haskell
POINT: index を 0 から見てみたところ、Key の昇順になっているようだったので、二分探索しました。

ENV: Clojure と手動
POINT: Index1 はバイナリーサーチしました。Index2 は最後の値から引いた数と同じ数を引いた分だけが Key になり

ます。
バイナリーサーチはある程度計算機でできるところまで手動で探しました。
Index2 は最後から２つ３つ探して回答を見つけました。

ENV: C++
POINT: データの並び方を予測してアルゴリズムを決めました。db=1 は二分探索を用い、db=2 は等差数列を解きました。

ENV: PHP
POINT: 大きな整数の扱いがポイントです。PHP では、文字列化することと BCMath 任意精度数学関数を使うことで突破しました。

ENV: php
POINT: 二分探索法で解きました。

ENV: Ruby
POINT: とりあえず 100 件ずつ取得してみたところ、db1 は K が昇順だったので 2 分探索で解きました。db2 は 2 の累乗数番目の K がやたらと小さかったので、2 分木で表していることに気が付きました。

ENV:Ruby
POINT:(1)2 分探索 (2)O(logn) で探索できるまで規則を単純化すること

ENV: Ruby
POINT: 2 つ目の DB のインデックスは 2^99+202363607099855744454258604 5/2 で求まります。

1 つ目の DB では単純に二分探索すればいいので簡単でした。
2 つ目の DB ではインデックスとキーの法則を複雑に考えてしまい、無駄に時間がかかってしまいました。
気づいてしまえばシンプルな法則ですね。

ENV: サクラエディタ（テキストエディタ）＋電卓（Windows 付属のもの）
POINT: 1 は増加関数だから繰り返して目的の値に近付ける。2 は素因数分解して 2 のべきが多い順に増加関数。

ENV: Perl
POINT: db1 は二分探索、db2 は規則の推測。

db1 は、キーが昇順に並んでいるので、二分探索で探しました。
db2 は、キーの値が奇数なので、インデックスは 2**99+(キーの値)/2 です。

ENV: Java
POINT: DB1 は昇順に並んでいるのがポイントです。0〜size-1 の範囲をバイナリサーチして解きました。DB2 は 2 の階乗ごとに個別の昇順一次関数の値が入っているのがポイントです。一次関数ごとに範囲チェックして線形補間して解きました。

ENV: 手＋ Windows 7 64bit ＋ Firefox ＋メモ帳＋ Dr.Racket（素因数分解、有理数除算）
POINT:ソートと列挙法

ENV: テキストエディタと関数電卓
POINT: Key 生成のアルゴリズムを見出すこと
データベース 1 の方は正確なアルゴリズムを見出せず、何らかの法則で増えていってると言う所だけ分かったため、二分探索のようなことをしました。
データベース 2 の方はアルゴリズム分かったので計算機で KEY を求めました。後ろから一定の数で引いていって引けなくなったら最初の数字から引いて、その後また新しい値 (2 の倍数かな) で引いていくような形ですね。
1 の方のアルゴリズムがよく分からなかったのが悔しい…

ENV: 手作業
POINT: データベース番号 1 は index に対してキーが直線的に単調増加すること、データベース番号 2 は長さが 2^100 で、インデックス 2^n+m のキーが n + m * 2 ^ n となることに気づくこと。

ENV: Perl
POINT: 二分探索、ビットローテート

db2 の法則を見つけるのに苦労しました。

ENV: シェルスクリプトと irb(電卓として)
POINT: データベース 1 は、キーが昇順で、かつ概ね均等な分布らしく、最初のキー k1 と最後のキー k2 を取得して、データ件数 *（探すキー - k0）/（k1 - k0）でほぼ正確にインデックスが求まりました (1 つ隣でした)。データベース 2 は、最初の数件を取得すると
(0, 2^99, 2^98 * 1, 2^98 * 3, 2^97 * 1, 2^97 * 3,

2^97 * 5, 2^97 * 7, 2^96 * 1, ..., 2^96 * 15, 2^95 * 1, ...)
という規則的な並びとなっていることがわかり、キーが奇数であることから、2*99 + (探すキー-1)/2 をインデックスにすれば良いことが分かりました。

ENV: Ruby
POINT: db=1 は key が昇順に並んでおり、db=2 は等差数列を並べたものになっている。db=1 は二分探索で、db=2 は該当の key が何番目の項に現れるかを計算して求めた。

ENV: Ruby
POINT:
■データベース 1
キーの K の後の数字は、インデクスとともに増加しているようなので、
インデクスを二分探索して求めました。

ENV: Firefox, Python, TeraPad
POINT: db1 は二分探索、db2 は最後から 2 ずつ引く→4 ずつ引く→8 ずつ引く→…というパターン

「挑戦者の声から」は以上です！

3.12 コード集

以下に、〆切後、結城浩（@hyuki）あてに送っていただいた挑戦者さんのコードをまとめました。

- http://togetter.com/li/758302

3.13 最後に

さて今回のサルベジオン問題はお楽しみいただけたでしょうか。
結城が出題した問題については、ブログやツイッターなどでの言及、参照、引用は大歓迎です。
ぜひ、このフィードバックに対するご感想も @hyuki までお聞かせくださいね。
結城はこれからも CodeIQ で出題していきますので、またチャレンジしてください。
今回は挑戦ありがとうございました。ではまた！

http://www.hyuki.com/codeiq/

今回のミッションはこのようにして終了しました……