データベースシステム（11）

探索問題・探索アルゴリズム

探索（Searching）問題と探索アルゴリズム

表に数字が書かれたカードがある
（カードは裏にして並んでいる）

1	3	4	7	8	10	12	16	17	19
0	1	2	3	4	5	6	7	8	9

任意の数字Xが入力され，
その数字Xが何番目のカードかを答える．

「8はどこですか？」→「4番目です」

「9はどこですか？」→「ありません」

このような問題を「探索問題」とよぶ．（1次元探索）

探索アルゴリズム

任意の数字が入力され，
その数字が何番目のカードかを答える．

カードがでたらめに並んでいる場合

6	3	1	11	8	10	9	16	2	17
0	1	2	3	4	5	6	7	8	9

入力された数字（探索する数字を）Xとする

0番目，1番目，2番目と順番にカードをめくり，
カードがXならその位置の番号を答える

探索アルゴリズム1

i=0,1,2,...9の各iに対し，i番目の数字YがXであるか調べる．
もしYがXならiを答えて終了．最後のiまで
Xと同値のYがみつからなければ「ありません」と答え終了

身近なケーススタディー

問題

あなたは，今，辞書を手にして，
ある単語（例えば「Penny」）の意味を調べようとしています．
どういう手順で辞書から所望の単語を見つけ出しますか？

仮定

・辞書にある単語はアルファベット順に（AからZへと）並んでいる

・辞書の中には
「A」，「B」，．．．，「Z」のような見出しがついているが
ここでは見出しのない辞書を想定する

・辞書にある単語数は「n」個とする
（nは数万個程度のとても大きな数字と考えてください）

ケーススタディー（1）

愚者の手順（辞書のひきかた）

辞書にある単語を最初から（つまり「A」から）順番に探してゆく

賢者の手順（辞書のひきかた）

辞書の真ん中を開き，
そこにある単語（例えば「Let」）と「Penny」を比較
比較の結果
　もっと前を探すべきだったら
　　「Let」より前の単語集合に対し，
　もっと後ろを探すべきだったら
　　「Let」より後ろの単語集合に対し，
　同じような操作を繰り返す

探索アルゴリズム

任意の数字Xが入力され，
Xが何番目のカードかを答える．

カードが小さい順（あるいは大きい順）に並んでいる場合

1	3	4	7	8	10	12	16	17	19
0	1	2	3	4	5	6	7	8	9

探索アルゴリズム1

i=0,1,2,...9の各iに対し，i番目の数字がXであるか調べる．
もしXならiを答え処理を終わる．最後のiまでXがみつから
なければ「ありません」と答えて終了する．

（辞書を最初から順に探す「愚者の…」）

探索アルゴリズム

任意の数字Xが入力され，
Xが何番目のカードかを答える．

カードが小さい順（あるいは大きい順）に並んでいる場合

1	3	4	7	8	10	12	16	17	19
0	1	2	3	4	5	6	7	8	9

探索アルゴリズム2 （「賢者の…」）

集合を真ん中で2つに分ける

真ん中と比較して「前半」か「後半」を探す

なんで

前者のアルゴリズムが愚かで，
後者のアルゴリズムが賢いのか？

アルゴリズム（この場合の「辞書のひき方」）の良し悪し

所望の単語にたどり着くまでの手間（比較回数）が
少ないアルゴリズム（辞書のひき方）のほうがよい．

愚者のアルゴリズム

辞書にある単語を最初から（つまり「A」から）順番に探してゆく

愚者のアルゴリズムを検証

例えば，所望の単語を見つけるまでの比較回数で考えると

調べる単語が「Penny」で，　⟶　やってられねぇ！
「A」から「Penny」までに3万語あった場合
3万回の比較が必要

調べる単語が「About」で，　⟶　楽勝！！
「A」から「About」までに20語だった場合
20回の比較が必要

愚者のアルゴリズムはほんとうに愚かなのか？

所望の単語を見つけるまでの比較回数の場合
「アルゴリズムの良し悪し」だけでなく

調べる単語の位置に影響される
（調べる単語によって，
各アルゴリズムに対する比較回数が変わる）

そこで ⟶ 「計算量」でアルゴリズムを評価する

厳密には計算量には「時間計算量」と「空間計算量」があるが
ここでは「**時間計算量**」について議論する．
また，厳密には時間計算量にも
平均計算量，最悪計算量とがあり，この2つがよく使われる

計算量

ある問題に対するアルゴリズムの応答時間

入力がサイズ n（この場合，辞書の単語数）のとき
目的を達するまでの
平均，あるいは，最悪の比較回数を
n （nの関数） で表したもの

愚者のアルゴリズム

辞書にある単語を最初から（つまり「A」から）順番に探してゆく

愚者のアルゴリズムの解析

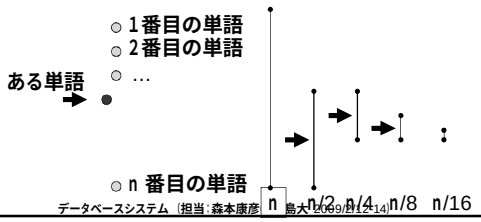
ある単語 ➡
- ○ 1番目の単語
- ○ 2番目の単語
- ○ …
- ○ n 番目の単語

平均計算量 n/2
最悪計算量 n

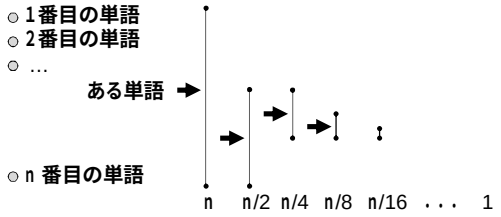
平均で，およそ $\frac{1}{2}n$ 回で単語が見つかるので

賢者の手順（辞書のひきかた）
辞書の真ん中を開き，
そこにある単語（例えば「Let」）と「Penny」を比較
比較の結果
もっと前を探すべきだったら「Let」より前の単語集合に対し，
もっと後を探すべきだったら「Let」より後ろの単語集合に対し，
同じような操作を繰り返す

賢者のアルゴリズムの解析



賢者のアルゴリズムの解析



単語数 n の辞書から所望の単語を見つける問題の
計算量を C_n とおく

1回のステップで，1回比較し，辞書のサイズが半分になる

$$C_n = C_{n/2}+1$$

賢者のアルゴリズムの解析（つづき）

単語数 n の辞書から
所望の単語を見つける問題の計算量を C_n とおく

1回のステップで，1回比較し，辞書のサイズが半分になる

$$C_n = C_{\frac{n}{2}} + 1$$

$n = 2^m$ とおく

$$C_{2^m} = C_{2^{m-1}} + 1$$

$$C_{2^m} = C_{2^{m-1}} + 1 = (C_{2^{m-2}} + 1) + 1$$
$$= C_{2^{m-2}} + 2 = (C_{2^{m-3}} + 1) + 2$$
$$= C_{2^{m-3}} + 3$$
$$\cdots$$
$$= C_{2^{m-m}} + m = C_{2^0} + m = C_1 + m = 1 + m$$

$$C_{2^m} = 1 + m$$

$n = 2^m$ とおいた　$(\log n = m)$

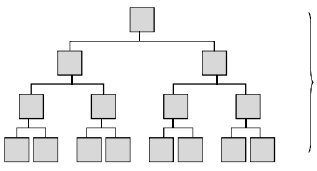
ここで n にもどす　　対数の底は2

$$C_n = 1 + \log n$$

最悪計算量は1+log n

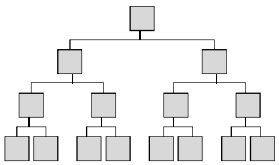
このアルゴリズムの別の見方（右脳を使った考え方）

トーナメント表を思い浮かべてください！



log n 回戦
（高さが log n）

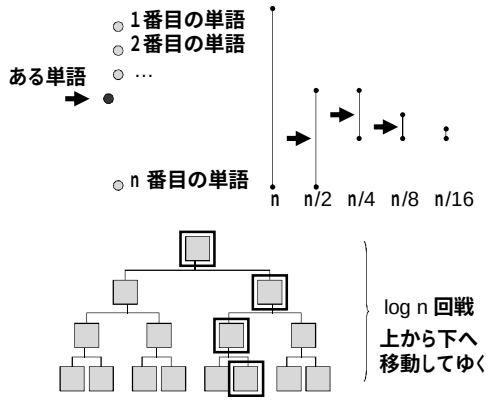
チーム数 n のトーナメントは「log n 回戦」ある！！



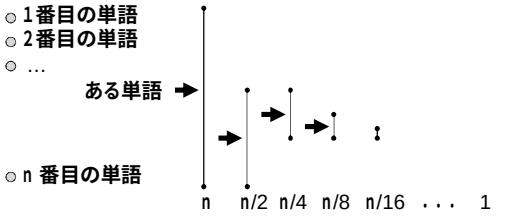
log n 回戦
（高さが log n）

チーム数2のトーナメントは「1回戦」まである
チーム数4のトーナメントは「2回戦」まである
チーム数8のトーナメントは「3回戦」まである
. . .
チーム数 2^n のトーナメントは「n回戦」
チーム数 n のトーナメントは「log n 回戦」ある！！

賢者のアルゴリズムの解析



賢者のアルゴリズムの計算量



平均計算量 (1 + log n)/2
最悪計算量 1 + log n

賢者のアルゴリズムの計算量　愚者のアルゴリズムの計算量

入力 n	1+log n	n
10	4	10
100	7	100
1,000	10	1,000
10,000	14	10,000
100,000	17	100,000

「1次元探索問題」に対する，この（「賢者の」）アルゴリズムを「2分探索（Binary Search）」と呼びます．

探索アルゴリズム（1）

任意の数字Xが入力され，
その数字が何番目のカードかを答える．

カードがでたらめに並んでいる場合

6	3	1	11	8	10	9	16	2	17
0	1	2	3	4	5	6	7	8	9

0番目のカードから順番にめくり，
カードがXならその位置の番号を答える

探索アルゴリズム1　「逐次探索（Sequential Search）」　O(n)

i=0,1,2,...9の各iに対し，i番目の数字YがXであるか調べる．
もしYがXならiを答えて終了．最後のiまで
Xと同値のYがみつからなければ「ありません」と答え終了

探索アルゴリズム（1）

任意の数字Xが入力され，
その数字が何番目のカードかを答える．

カードが小さい順（あるいは大きい順）に並んでいる場合

1	3	4	7	8	10	12	16	17	19
0	1	2	3	4	5	6	7	8	9

探索アルゴリズム1　「逐次探索（Sequential Search）」　O(n)

i=0,1,2,...9の各iに対し，i番目の数字YがXであるか調べる．
もしYがXならiを答えて終了．最後のiまで
Xと同値のYがみつからなければ「ありません」と答え終了

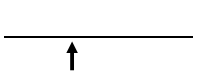
このアルゴリズムで正しいが
カードが順番に並んでいる場合は
これよりもはるかに効率的なアルゴリズムがある

探索アルゴリズム（2）

任意の数字Xが入力され，
Xが何番目のカードかを答える．

カードが小さい順（あるいは大きい順）に並んでいる場合

1	3	4	7	8	10	12	16	17	19
0	1	2	3	4	5	6	7	8	9



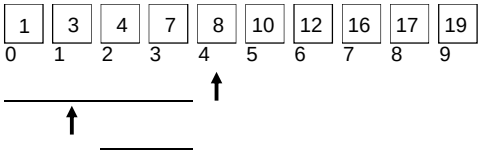
探索アルゴリズム2　（「賢者の…」）

集合を真ん中で2つに分ける
真ん中と比較して「前半」か「後半」を探す

探索アルゴリズム(2)

任意の数字Xが入力され，
Xが何番目のカードかを答える．

カードが小さい順(あるいは大きい順)に並んでいる場合



探索アルゴリズム2 (「賢者の…」)

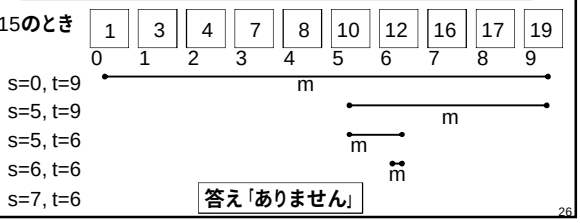
集合を真ん中で2つに分ける

真ん中と比較して「前半」か「後半」を探す

探索アルゴリズム2のふるまい

探索アルゴリズム2 (「2分探索」と呼ばれている)
「sとtの間でXを探索」
s>tなら「ありません」と答えて終了，
s<=tならm= (s+t)/2としてm番目の数字YとXを比較
XとYが一致したら，mと答えて終了
X>Yならs=m+1として，「sとtの間を探索」
X<Yならt=m-1として，「sとtの間を探索」

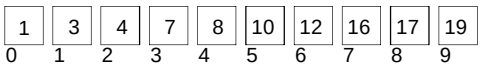
X=15のとき



探索アルゴリズム(2)

任意の数字Xが入力され，
Xが何番目のカードかを答える．

カードが小さい順(あるいは大きい順)に並んでいる場合



探索アルゴリズム2 「2分探索」 **O(log n)**

「sとtの間でXを探索」
s>tなら「ありません」と答えて終了，
s<=tならm= (s+t)/2としてm番目の数字YとXを比較
XとYが一致したら，mと答えて終了
X>Yならs=m+1として，「sとtの間を探索」
X<Yならt=m-1として，「sとtの間を探索」

2分探索(探索アルゴリズム2)の最悪計算量

逐次探索(探索アルゴリズム1)の最悪計算量

入力 n	1+log n	n
10	4	10
100	7	100
1,000	10	1,000
10,000	14	10,000
100,000	17	100,000

**「計算量」(入力サイズnのとき)を表す関数に
よく出てくる代表的なnの項**

入力 n	log n	n	n log n	n^2
10	3	10	30	100
100	6	100	600	10,000
1,000	9	1,000	9,000	1,000,000
10,000	13	10,000	130,000	100,000,000
100,000	16	100,000	1,600,000	10,000,000,000

アルゴリズムが少し違えばnの関数は変わってくる．

アルゴリズムの性能を分類する場合には
(性能への影響の少ない)わずかな関数の違いは
同一視したほうがよい．

O記法

O記法:
「オーダ」

適当な定数c_0とn_0が存在して，
$n>n_0$である任意のnに対して
$g(n)<=c_0 f(n)$であるなら，
関数g(n)はO(f(n))である

計算量 O(f(n)) **のアルゴリズム**
(およそ f(n) に比例する計算時間で動作する)

O記法のメリット
n によらない定数や定数係数を無視できる

利用するコンピュータによる違い
個別の実現方法(プログラム)による違い
によらずアルゴリズム(計算手順)の良し悪しを解析できる！

O記法:
「オーダ」

適当な定数c_0とn_0が存在して，
$n>n_0$である任意のnに対して
$g(n)<=c_0f(n)$**であるなら，**
関数$g(n)$**は**$O(f(n))$**である**

あるアルゴリズムの計算量

g(n)		c_0	f(n)	オーダ
$5n$	<=	5	n	O(n)
$\frac{1}{2}n+3$	<=	1	n	O(n)

c_0=1**のとき，**n_0=6 **(6以上のnで不等式が成り立つ)**

$2n^2+n+1$	<=	3	n^2	O(n^2)

c_0=3**のとき，約2以上のnで不等式が成り立つ)**

(直感的な) O記法の求め方

あるアルゴリズムの計算量	**オーダ**
$5n$	O(n)
$\frac{1}{2}n+3$	O(n)
$2n^2+n+1$	O(n^2)

nの一番重い項 (第1主要項) の係数を取り除く

(nがある程度以上に大きいならば)
nの一番重い項以外を無視しても影響は少ない．

入力 n	log n	n	n log n	n^2	n^2+n
10	3	10	30	100	110
100	6	100	600	10,000	10,100
1,000	9	1,000	9,000	1,000,000	1,001,000
10,000	13	10,000	130,000	100,000,000	100,010,000
100,000	16	100,000	1,600,000	10,000,000,000	10,000,100,000

nの一番重い項 (第1主要項) が
アルゴリズムの性能を一番忠実に表す部分

n^2 も n^2+n も **O (n^2)**
この2つともアルゴリズムの性能は似ている
(入力が10倍になったら，時間はほぼ100倍)

nの一番重い項 (第1主要項) の係数を取り除く

入力 n	log n	n	10*log n	100*log n
10	3	10	30	300
100	6	100	60	600
1,000	9	1,000	90	900
10,000	13	10,000	130	1300
100,000	16	100,000	160	1600
1,000,000	19	1,000,000	190	1900

この2つとも
「O (log n)」の仲間

列の壁は，係数を操作しても
(大きなnの場合は) 超えられない大きな壁

係数を無視してシンプルに
(係数分は計算機の性能でカバーできるが
それ以外はアルゴリズムを直さなければ直らない)

nの (係数をはずした) 関数部分が
アルゴリズムの良し悪しの本質を表す

nの一番重い項以外を無視してシンプルに
(nが大きい場合，一番重い項以外は
計算時間の大勢に影響を与えない)

nの一番重い項がアルゴリズムの本質的計算時間を
象徴している

アルゴリズムはO記法による計算量で評価する

計算量は「サイズnの入力」に対する解を
どのくらいの手間で求めることができるかを
「nの関数」で示したもの．

nの関数にはいろいろある

3n+1
2n
$2n^2$+2
n^2+2n+5

nが小さいときは
アルゴリズムが悪くても
それなりの時間で解ける
(アルゴリズムの良し悪しによる
差もそれほどおおきくならない)

しかし

nが大きいときは
アルゴリズムが悪いと悲惨

アルゴリズムは大きなnに対する効率で比べるべし！

nの関数にはいろいろある

3n+1　10倍になれば ほぼ10倍

2n

nが小さいときは アルゴリズムが悪くても それなりの時間で解ける (アルゴリズムの良し悪しによる 差もそれほどおおきくならない)

$2n^2+2$

n^2+2n+5　10倍になれば ほぼ100倍

しかし

nが大きいときは アルゴリズムが悪いと悲惨

アルゴリズムは大きなnに対する効率で比べるべし！

nを大きくしてゆけば行くほど...

この線の上下の差がぐんぐん広がる！

一方で，上の2つ，下の2つは大きくなる傾向が似ている (nが大きくなったときの影響が似ている)

nの関数にはいろいろある　オーダー

3n+1　10倍になれば ほぼ10倍　O (n)

2n

$2n^2+2$

n^2+2n+5　10倍になれば ほぼ100倍　$O(n^2)$

アルゴリズムは大きなnに対する効率で比べるべし！

nを大きくしてゆけば行くほど...

この線の上下の差がぐんぐん広がる！

上の2つ，下の2つは大きくなる傾向が似ている (nが大きくなったときの影響が似ている)

nが大きくなったときの影響が似ているかどうかで アルゴリズムの性能 (nの関数) を分類するとき O記法 (「オーダー」) が便利

O記法：「オーダ」

適当な定数c_0とn_0が存在して，$n>n_0$である任意のnに対して

$g(n)<=c_0 f(n)$であるなら，

関数$g(n)$は$O(f(n))$である

アルゴリズムの応答時間				オーダ
g(n)		c_0	f(n)	
$5n$	<=	5	n	O(n)
$\frac{1}{2}n+3$	<=	1	n	O(n)
$2n^2+n+1$	<=	3	n^2	$O(n^2)$

計算量

	平均計算量	最悪計算量
逐次探索	(½)*n	n
2分探索	(½)*(log n + 1)	log n + 1
O記法では		
逐次探索	O(n)	O(n)
2分探索	O(log n)	O(log n)

サーチエンジンに検索語 (キーワード) を入力してクリックすると，一瞬のうちに，その検索語を含むページが列挙される．

なんでだ？

・約80億ページが検索対象となっている
・検索語としては約1000万語が登録されている (Googleのホームページによる)

当然，毎回，世界中の80億ページの中身を調べてるわけではない

サーチエンジンのテクノロジーの基本は「インデックス」

全文検索 (ロボット型検索エンジン) では

ウエブページを自動的に収集

クローラ (crawler)，検索ロボット

DBに登録されているウエブページを自動巡回

巡回しているウエブページからリンクしているページにも巡回してゆく

→

ウエブページのインデックスを作成

ウエブページのインデックス (1)

ウエブページ

ページ 1

「ウエブページのインデックス」. . .

1. 各ページ内に記述されている文字列の分割

「ウエブページのインデックス」

ウエブページ の インデックス

ウエブページのインデックス (2)

ウエブページ

ページ 1

ページ 2

ページ 3

2. インデックスの作成

文字列	ページ番号
海	2, 10, 30
川	8
山	4, 2
. . .	. . .

インデックスの例

インデックスはインターネットの(巨大な)辞書・電話帳のようなもの

ページ 1

赤い花

ページ 2

青い花

ページ 3

白い雲

文字列は順番に並んでいる

文字列	ページ番号
青い	2
赤い	1
雲	3
白い	3
花	1, 2

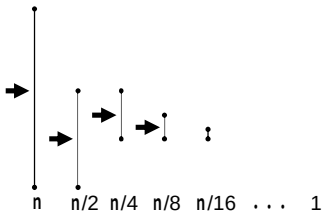
検索の例(「花」を検索語とした場合)

文字列	ページ番号 (ID)
青い	2
赤い	1
雲	3
白い	3
花	1, 2
. . .	. . .

1. インデックス表の「花」を探す O (log n)

インデックスにある約1000万語のキーワードからの「2分探索 (binary search)」

- 1番目の単語
- 2番目の単語
- …
- n 番目の単語



log n 回, 2分割すると1になる
(log n 回以内の比較回数で見つかる)

log 1千万 = 約23

検索の例(「花」を検索語とした場合)

文字列	ページ番号 (ID)
青い	2
赤い	1
雲	3
白い	3
花	1, 2
. . .	. . .

1. インデックス表の「花」を探す
2. 「花」に対応するページ1, 2を答える