課題２：石選び問題

講義ノート

１．単純なアルゴリズム（プログラム stone1）

単純なアルゴリズムの考え方について述べる．可能な石の選び方をすべて，順に考えていって，その選び方で選んだ石の重さの合計が目標の重さ W になるかを調べればよい．つまり，次のように調べるのが単純なアルゴリズムである．

stone1 のアルゴリズム（入力：石の重さ $s_1, ..., s_n$ と目標の重さ W）

すべての石の選び方 $i_1, ..., i_k$ に対し，次のことを行う

(1) $W' = 0 + s_{i_1} + s_{i_2} + \cdots + s_{i_k}$ を求める．
(2) $W' = W$? を調べる．
(3) もし等しければその選び方を出力して計算を終える．

注： ここでは 0 に足すような無駄なことをしている.（その方が理論式が簡単になるので.）

ここでの石の選び方だが，0 から $2^n - 1$ の２進数に対応して選んでいくようにする．つまり，２進数の対応するビット位置が 1 になっている番号の石を選び，その石の重さを W' に足し込んでいくのである．

さて，このアルゴリズムの計算時間について考えよう．今回の場合は，石の重さの和を計算するのが基本だろう．そこで，計算終了までに行われる（石の重さの）足し算の回数をもってその計算の時間と考えることにする．最悪の入力例は，すべての選び方を試してみなければならない場合だが，その場合の，このアルゴリズムの計算時間（足し算の回数）は，総数で $n2^{n-1}$ になる．（確認しておこうね！）

２．賢いアルゴリズム：アイデア編（プログラム stone2）

賢いアルゴリズムの考え方を説明する．以下では石の数 n を 20 として説明しよう．

先に説明したように，stone1 では 石 1 ～ 石 20 のすべての石に対して，その選び方を考えた．そのために，2^{20} 通りを調べなければならなかった．それではもし，前半の石，石 1 ～ 石 10 の中から石を選んだ時点で，その選び方に見込みがあるかどうかを直ちに判定できるとしたらどうだろう．その場合，次のような調べ方が考えられる．

stone2 のアイデア（入力：石の重さ $s_1, ..., s_n$ と目標の重さ W）

石 1 ～ 石 10 からの選び方 $i_1, ..., i_k$ すべてに対し，次のことを行う

(1) $W' = 0 + s_{i_1} + s_{i_2} + \cdots + s_{i_k}$ を求める．
(2) W' に見込みがあるかを調べる．
(3) もし見込みがあれば，残りの石 11 ～ 石 20 の選び方を求める．

そうすると，調べる回数は 2^{10} 通りと大幅に減る.（もちろん，その分，(2) や (3) に時間がかかるかもしれないが，そこが素早くできたとするとかなりの時間節約になる.）

では「見込みがあるか？」をどうやって調べればよいだろう．たとえば，途中までの和 W' が目標の重さ W より大きければ（つまり $W' > W$ ならば），もう見込みがないことは明らかである．したがって，$W'' = W - W'$ は正でなければならない.（ちなみに $W'' = 0$ の場合には，すでに選んだ石が答えである.）もちろん，W'' が正だからといって，必ずしもよいわけではない．本当に見込みがあるのは，残りの石 11 ～ 石 20 の中から適当に選ぶと，その重さの総和が W'' になる場合である.

したがって，上記のアルゴリズムをさらに詳しく書くと次のようになる.

stone2 のアイデア（入力：石の重さ $s_1, ..., s_n$ と目標の重さ W）

石 1 ～ 石 10 からの選び方 $i_1, ..., i_k$ すべてに対し，次のことを行う

(1) $W' = 0 + s_{i_1} + s_{i_2} + \cdots + s_{i_k}$ を求める.

(2) $W'' = W - W'$

(3) $W'' \geq 0$ で，しかも残りの石 11 ～ 石 20 から適当に石を選ぶと，その総和が W'' になるか？を調べる.

(4) もしそうならば $i_1, ..., i_k$ と残りの石の選び方を出力して計算を終える.

ここで肝心なのは，上の (3) のテストを素早く行うことである．そのための最初のアイデアは，計算に先だって，石 11 ～ 石 20 からの石の選び方すべてに対して，その選び方のもとでの重さの総和を求め，それを表にしておくことである.（もちろん，この総和の表を計算する時間も考慮に入れる.）

重さの総和の表

0	⟵ 何も選ばない
957238	⟵ 11 を選んだとき
29410	⟵ 12 を選んだとき
986648	⟵ 11,12 を選んだとき
2886419	⟵ 13 を選んだとき
3843657	⟵ 11,13 を選んだとき
2915829	⟵ 12,13 を選んだとき
3873067	⟵ 11,12,13 を選んだとき
⋮	⋮

ただし，上図のように，ただ単純に表にしたのでは，W'' がその表の中にあるかどうかを探すのに，いちいち表の上から探さなければならなくなり，これでは時間がかかってしまう．ところが，これを「ハッシュ表」と呼ばれる表にすると,（ほとんど）一発で，W'' が表中に出てくるかが調べられる．しかも「その重さを得るのに，石 11 ～ 石 20 のどれを組み合わせればよいか？」ということも，わかってしまうのである．このハッシュ表を使うのが，今回のアルゴリズムの鍵

となるアイデアだったのである.（この「ハッシュ」について興味のある人は，このプリントの最後に添付した説明も読んでみて下さい.）

３．問題例の難しさ：最良（？），平均，最悪の時間計算量

問題例のサイズ n が同じでも，問題例によっては計算時間が大きく異なる場合がある．今回考えたアルゴリズム stone1, stone2 でも，サイズ（石の数 n）が同じでも，問題例によって計算時間が大きく変わってくる．同じサイズで，最も計算時間が長くなる問題例を 最悪例 といい，各サイズ n に対し，その最悪例に対する計算時間を与える関数を 最悪時間計算量 という．アルゴリズムの効率を評価する場合，この最悪時間計算量を考えることが多い．これとは反対に，最も計算時間が短い問題例（最良例）も考えられるが，最良例はアルゴリズムの効率の評価にはあまり用いられない.

最悪時間計算量は悪いのだが，大抵はうまくいってしまう，というアルゴリズムもある．そこで「大抵」の効率を評価するために，平均時間計算量 というのも，最近では用いられるようになってきている．ただ,「平均」といった場合，何に対する平均を取るかがいろいろ考えられる．また，問題例に対する確率分布も無数に考えられる．その問題が使われる状況に応じて「平均」を考えるべきだろう.

【練習】（レポートに関係あり．考えておいて下さい）

問 1 アルゴリズム stone1, stone2 に対する最悪例，最良例とは何か，例をあげて説明せよ.

問 2 アルゴリズム stone1, stone2 の（加算の回数の基づく）最悪時間計算量を，それぞれ $tmax_1(n)$, $tmax_2(n)$ とする．これら 2 つの計算量を予想せよ.

問 3 石選び問題に対する各種アルゴリズムの平均時間計算量を考える際に，どのような平均の取り方があるか？妥当と思われる平均の取り方（実験方法）を 2 種類以上示せ.

問 4 アルゴリズム stone2 では，重さの残りにちょうど一致するような石の組み合わせが,「後半の表」にあるかないかが一発でわかることが大変重要である．もし，そのような石の組み合わせがあるかないかを調べるのに，表を全部しらべなければならないとしたら，計算時間はどのくらいになるか？やはり，最悪の場合を想定して評価せよ.

実験ノート

１．実験課題

石選び問題に対する 2 つのプログラム (stone1, stone2) に対し，以下のような時間計算量（ただし，単位は石の重さの加算回数）を 実験により 求める.

(1) 最悪時間計算量 $tmax_1(n), tmax_2(n)$．ただし，問題例のサイズ n は石の個数（以下，同様）.

(2) ある種の平均時間計算量 $tave_1(n), tave_2(n)$．各自，妥当な設定のもとでの「平均」を考えて実験してみること.

(3) （もし時間があれば）stone1 の方が stone2 よりも平均計算時間が短くなるような平均の状況を考え，そのもとでの平均時間計算量 $tave'_1(n), tave'_2(n)$ を測る.

具体的には，実験を $n = 2 \sim 30$ の範囲で行い，それにより得られた計算時間（加算の回数）を測定し，その結果を gnuplot などを使って解析すること.（必ずしも gnuplot を使わないといけないわけではない.）

注意！ stone1 では，$n \geq 20$ になると，データによってはとてつもなく時間がかかる場合がある．ほどほどの時間で実験が終了するように計画して欲しい.

２．レポートの構成

レポートは以下のような構成にすること.（前回のレポートで注意されたような点には特に気を付けること.）

1. 実験手順：実験手順についての説明．どのような実験を，どのような種類のデータに対して，それぞれ何回行なったか，などを明確に書く．今回はとくに，どのような平均時間を測ったかがわかるように！
2. 実験結果のまとめ：実験で得られた計算時間（加算回数）のデータの要約（例：平均）を表にして示す．次に，そのデータに基づき gnuplot で，計算量 $tmax_1, tmax_2$，$tave_1, tave_2$ などの予想式を求める.（どうやってデータから式を推定したかも説明すること.）さらに，stone1 の最悪時間のデータに基づくグラフと予想式 $tmax_1$ のグラフを一枚に書いたもの．同様に，それぞれ，stone2 の最悪時間のグラフと予想式 $tmax_2$ のグラフを，stone1 の平均時間のグラフと予想式 $tave_1$ のグラフを，そして，stone2 の平均時間のグラフと予想式 $tave_2$ のグラフを一枚に書いたもの．これら 4 枚の図を添付する.（$tave'_1, tave'_2$ についても同様.）
3. 考察：$tmax_1, tmax_2$ の予想式の妥当性，$tave_1, tave_2$ の予想式の妥当性．なぜ，そのような計算量になるのか，あるいはなりそうなのかの説明．アルゴリズムや平均の取り方から，なぜ，そのような予想式（実験データ）が得られたかを説明する.（$tave'_1, tave'_2$ についても同様.）
4. 感想（もしあれば．評価の対象外）：今回の課題や実験，あるいは講義についての感想.

３．使用ソフト

今回説明したプログラム（ソースならびに実行形式）は，ディレクトリ `~owatanab/pub/stone` に用意してある.

実験データ生成用

gen（プログラムのソースは gen.c）

[自分のプロンプト] `./gen -1` n $seed$ `>` $file1$

- n は石の個数．$seed$ は乱数の種（シード），自分の学籍番号（00_xxxx-y の xxxx の部分など）を使うとよい.
- $file1$ は出力先のファイル名．好きな名前を使ってよい.
- 解がつねに一つ以下になるように，n 個の石の重さを作り，それを $file1$ へ出力する.

[自分のプロンプト] `./gen -2` n x $seed$ `>` $file2$

- n は石の個数．x は石の重さの上限．$seed$ は乱数の種（シード），自分の学籍番号（00_xxxx-y の xxxx の部分など）を使うとよい.

◦ $file2$ は出力先のファイル名．好きな名前を使ってよい．

• n 個の石の重さとして，$1 \sim x$ の間のランダムな数を n 個選び $file2$ へ出力する．

sum（プログラムのソースは sum.c）

[自分のプロンプト] ./sum n k $seed$ < $file1$ > $file3$

◦ n は石の個数．k は問題例の数．$seed$ は乱数の種（シード）．

◦ $file1$ は石の重さが書かれているファイル名．このファイルには各行に 1 つずつ（計 n 行）に，石の重さ書かれていなければならない．

◦ $file3$ は出力先のファイル名．

• $file1$ に与えられている n 個の石の重さを用い，石をランダムに選んで，目標の重さ W を計算する．これを k 回行い，k 通りの目標の重さを求め，結果を $file2$ へ出力する．

[自分のプロンプト] ./sum -a n k $seed$ < $file1$ > $file3$

• 上とほぼ同様．ただ，それぞれの目標の重さ W に対して，その答えも出力する．

実験対象プログラム

stone1, stone2（プログラムのソースは各々 stone1.c, stone2.c）

[自分のプロンプト] ./stone$i$ n k < $file$

◦ n は石の個数．k は問題例の数．

◦ $file$ は 石の重さと目標の重さが書かれたファイル．具体的には，n 個の石の重さと，k 通りの目標の重さが次のように書かれていなければならない．

• それぞれの目標の重さに対して，石の選び方を出力する．ただし，該当する解がないときは「解なし」と答える．

• 毎回の計算時間（加算回数）を出力し，最後には，その平均を出力する．

```
345732
1234
1457
1111900
  ⋮
234567
```
n 個の石の重さ

（空行）

```
32113
56348129
  ⋮
2347197
```
目標の石の重さ（k 個）

参考文献

今回，少しふれた公開鍵暗号を始め，最新の情報セキュリティの技術には，いろいろとおもしろいものが多い．[1] は，この情報セキュリティ技術の原理や応用について，わかりやすく解説し

た（つもり）の本である．それより少しレベルは高くなるが，もう少し深く勉強したい人には [2] をお勧めする．暗号などでよく使われる整数論の結果などについても丁寧に解説してある．また，ナップサック暗号に関しては [3] が詳しい．

[1] 太田和夫，黒澤馨，渡辺治，情報セキュリティの科学，講談社ブルーバックス，1995.

[2] 岡本龍明，太田和夫，暗号・ゼロ知識証明・数論，共立出版，1995.

[3] A. Salomma, Public-Key Cryptography, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1990.

一方,「P $\neq$ NP 予想」について，もう少し詳しく知るには [4] などを読むとよいだろう．教科書なども紹介されている．

[4] 戸田誠之助，P $\neq$ NP 予想，数学セミナー９月号, Vol.37, No.9, 1998.

（付録）

ハッシュ法について

１．ハッシュ表，ハッシュ関数

いま，$W_1, ..., W_N$ の N 個の数が与えられているとする．これを表にしておき，後で与えられた W'' がその表の中にあるかどうかを素早くチェックしたい．どうすればよいだろうか？

まず，コンピュータにおける表の扱い方から説明しよう．コンピュータでは，データを表にしてしまうことが簡単にできるようになっている[1]．簡単にいえば，たとえばデータ $a_1, ..., a_7$ を下図 (a) のような表としてしまうことができ（その表を A と呼ぶことにする），表 A の上から 6 番目に格納されているデータ（この場合 a_6）を見たり，逆に A の 3 番目に新しいデータ b を格納すること（下図 (b)）などが簡単にできる．

(a) データの参照

1	a_1	
2	a_2	
3	a_3	
$\vdots$	$\vdots$	
6	a_6	$\leftarrow ?$
7	a_7	

表 A

(b) データの格納

1	a_1	
2	a_2	
3	b	$\Leftarrow b$
$\vdots$	$\vdots$	
6	a_6	
7	a_7	

表 A

以下の説明では，左図のような表 A に対し，$A[i]$ で，表 A の上から i 番目の場所にあるデータ（左図 (a) では a_i）を表すことにする．また，$A[i] \leftarrow b$ と書いたら,「表 A の上から i 番目の場所に b を入れる」ことを意味することにしよう．

したがって，たとえば $A[3] \leftarrow b$ の後では，$A[3] = b$ である．

このような表に，$W_1, ..., W_N$ を素直に格納した場合（下図 (a)），後で与えられた W'' がその中にあるかを調べるには時間がかかってしまう．ここで考え方を変えてみる．$W_1, ..., W_N$ を表に

[1] そもそも，コンピュータは，データを記憶する装置（俗にいうメモリ）と，データを処理する演算装置（俗にいうCPU）から成っている．だから，メモリからデータを取ってきたり，メモリにデータをしまうのは朝飯前のことなのである．

入れるデータと考えるのではなく，表の場所を示す値としてみなしてはどうだろう．たとえば，$W_1 = 5, W_2 = 8, W_3 = 2, W_4 = 10, ...$ の場合，それぞれ $5, 8, 2, 10, ...$ 番目の場所に 1 を，それ以外の場所には 0 を入れた表（下図 (b)）を作るのである．

(a) 普通の表

1	W_1
2	W_2
3	W_3
4	W_4
5	W_5
6	W_6
7	W_7
⋮	⋮
N	W_N

表 X

(b) 発想の逆転

1	0
2	1
3	0
4	0
5	1
6	0
7	0
8	1
⋮	⋮

表 Y

左図のような表 Y を作るのは簡単だ．まず，すべての i について $Y[i]$ を 0 にしておき，その後で

$$\begin{aligned} Y[W_1] &\leftarrow 1, \\ Y[W_2] &\leftarrow 1, \\ Y[W_3] &\leftarrow 1, \\ &\vdots \end{aligned}$$

とやればよい．

また，このような表 Y があれば，W'' が $W_1, ..., W_N$ の中にあるかどうかのチェックもたちどころにできる．$Y[W''] = 1$ かどうかを確かめればよいからだ．

ただし，この方法には重大な問題点がある．表が大きくなりすぎてしまうのである．たとえば，データ数 N が 1,000 だったとしよう．一方，$W_1, ..., W_N$ の最大値が一億だったとする．すると素直な表 X の大きさが 1,000 であるのに対し，表 Y の大きさは一億になってしまう．たった千個のデータをしまうのに，表の大きさが一億になってしまうのはではもったいない．

実際，$n = 20$ のとき，重さの種類が $N = 2^{20/2} = 1024$ なのに対し，最初に例として示した石の重さだと一億を越えるものができてしまう．一億もの表を作るのは忍びない．（コンピュータによってはメモリが足りなくなってしまう[2]．）

この問題を解消するために，W_i を直接使わずに W_i を N で割った余りを使って表 Z を作ってみよう．（以下の説明では，$h(x)$ で「x を N で割った余り」を表すことにする．）たとえば，$N = 10$ で $W_1 = 12302, W_2 = 66630, W_3 = 93278, W_4 = 533, ...$ だとする．

0	66630
1	0
2	12302
3	533
4	0
⋮	⋮

表 Z

このとき，$W_1 = 12302$ に対して表の W_1 番目を使うのではなく，$h(W_1) = h(12302) = 2$ 番目を使うのである．ただし，表 Y では $Y[W_1]$ に 1 を入れていたのに対し，表 Z では $Z[2]$ には W_1 を格納しておく．

同様に，$W_2 = 66630$ は $Z[h(W_2)] = Z[0]$ へ，$W_3 = 93274$ は $Z[h(W_3)] = Z[4]$ へ,... と格納するのである（左図）．

こうしておけば，たとえば 533 が $W_1, ..., W_N$ 中にあるかを調べるには，$h(533)$ を求め,「$Z[h(533)] = 533?$」をチェックすればよい．こうすれば，表 Y と同様に（ほぼ）瞬時にチェックできるのである．

さらに，h の値は 0 から $N - 1$ までなので，表 Z の大きさは N である．つまり，Z は普通の表と同じ大きさで，しかも Y と同じように，チェック（ほぼ）瞬時にできる魔法の表なのだ．

[2]最近のコンピュータはそんなヤワではないですが...

一般には，h はどんな関数でも構わない．生データだと値のばらつきが大きくなってしまうのを，適当な範囲の値に変換する関数であれば何でもよい．そのような目的で使われる関数をハッシュ関数といい，ハッシュ関数を用いて，いま説明したような考え方で作られた表をハッシュ表というのである．

【練習】

問 5 表 Y では 0 か 1 しか入れなかったのに，ハッシュ表 Z には値 W そのものを入れていた．なぜか？

２．世の中そんなに甘くない！

本当に Z は魔法の表なのだろうか？残念ながらそうとは限らない．世の中そんなに甘くないのである．

問題は，ハッシュ関数の値がばらつかないときに生じる．たとえば，先ほどの例で，たとえば $W_5 = 2346$, $W_6 = 2006$, $W_7 = 432906$, ... だったらどうだろう．$h(x)$ は x を 10 で割った余りなので，$h(W_5) = h(W_6) = h(W_7) = 6$ となってしまう．これをハッシュ値の衝突（collision）という．

この場合にも，表 Z の 6 番目に W_5, W_6, W_7 の 3 つの値を格納しておけば，一応対処できる．$h(W'') = 6$ となったとき，その 3 つの値に対して「$W'' = W_5$? or $W'' = W_6$? or $W'' = W_7$?」とチェックすればよいからだ．しかし，もし仮に 10 個すべてのハッシュ値が同じ値（たとえば 0）だったらどうだろう．表の一カ所にすべての値が格納されることになる．しかも，さらに $h(W'')$ も 0 だったとすると，結局,「$W'' = W_1$? or $W'' = W_2$? or $\cdots$ $W'' = W_{10}$?」のすべてを調べなければならなくなる．普通の表を使っているのと変わらなくなってしまうのだ！

(a) $h(W_5) = h(W_6) = h(W_7) = 6$

0	66630
1	0
2	12302
3	533
4	0
5	0
6	2346, 2006, 432906
⋮	⋮

表 Z

(b) 最悪の場合

0	10230,20,520,2340,10,90,110,4890,330,12020
1	0
2	0
3	0
4	0
5	0
6	0
⋮	⋮

表 Z

だが，そう悲観しなくてもよい．上図 (b) のように，一カ所に集まってしまう場合も稀だからである.「世の中そう甘くもないけれど，世の中そう捨てたものでもない」といったところだろうか.

このことを，適当な想定のもとにもう少し詳しく分析してみよう．データ数を N とし，今までの例と同様，$h(x) =$（x を N で割った余り）をハッシュ関数として用いることにする．さらに，$W_1, ..., W_N$ が 1 以上 D 以下の値をランダムに取ると仮定する．このとき，ハッシュ表の 1 つの場所（ここでは 0 番目とする）に格納される W_i の個数の期待値を求めてみよう．

期待値とは平均値のことである．では，どんな場合の，何の平均値だろうか？$W_1, W_2..., W_N$ が $1, 1, ..., 1$ から $D, D, ..., D$ まで，すべての値の組み合わせをとったときの，$h(W_i) = 0$ となる W_i の個数の平均である．しかし，これは

W_1 が 0 番目に格納される回数の平均 ＋ W_2 が 0 番目に格納される回数の平均
＋ W_3 が 0 番目に格納される回数の平均 ＋ W_4 が 0 番目に格納される回数の平均
＋ ⋯
＋ W_{N-1} が 0 番目に格納される回数の平均 ＋ W_N が 0 番目に格納される回数の平均

に他ならない．

一方，W_1 を 1 から D まで変化させたときの，W_1 が 0 番目に格納される回数は，$h(W_1) = 0$ となる回数なので D/N 以下（正確には，ちょうど D/N の切り捨て）．したがって，平均回数は $(D/N) \div D$ で $1/N$ 以下となる．他の $W_2, W_3, ..., W_N$ も同様．したがって，$h(W_i) = 0$ となる W_i の個数の期待値は $(1/N) \times N = 1$ 以下，つまり，表の同じ位置に来るデータの個数（の平均）は，一個以下におさまるのである．

実際，stone2 は 4 つ以上の値が衝突した場合に “Too many collisions!” と言って止ってしまうように設計してあったが，今回の実験で，それに遭遇した人はそう多くなかったはずだ．

そうは言っても，衝突が多発するような $W_1, ..., W_N$ もある．しかも，その $W_1, ..., W_N$ を，どうしても処理しなければならない場合にはどうしよう．その場合には，今度はハッシュ関数の方をランダムに選ぶ．実は，関数の方をランダムに選ぶ方法があるのだ．（何をランダムにするのだろうね？）そうすると，衝突を避けてくれるようなハッシュ関数を，高い確率で選ぶことができるのである．

３．参考文献

ハッシュ法やハッシュ関数，とくに最後に述べたランダムなハッシュ関数の選び方については，次の参考文献の 12 章に丁寧な説明がある．

[1] T. Cormen, C. Leiserson, and R. Rivest（浅野他訳），アルゴリズム・イントロダクション（第１巻），近代科学社 1990.