アルゴリズムとプログラミング実践講座

http://akashi.ci.i.u-tokyo.ac.jp/mary/lectures/algorithm/

火曜　13:00 -- 14:30
I-REF 棟 6階　ヒロビー

稲葉真理　with 浅井大史・手塚宏史

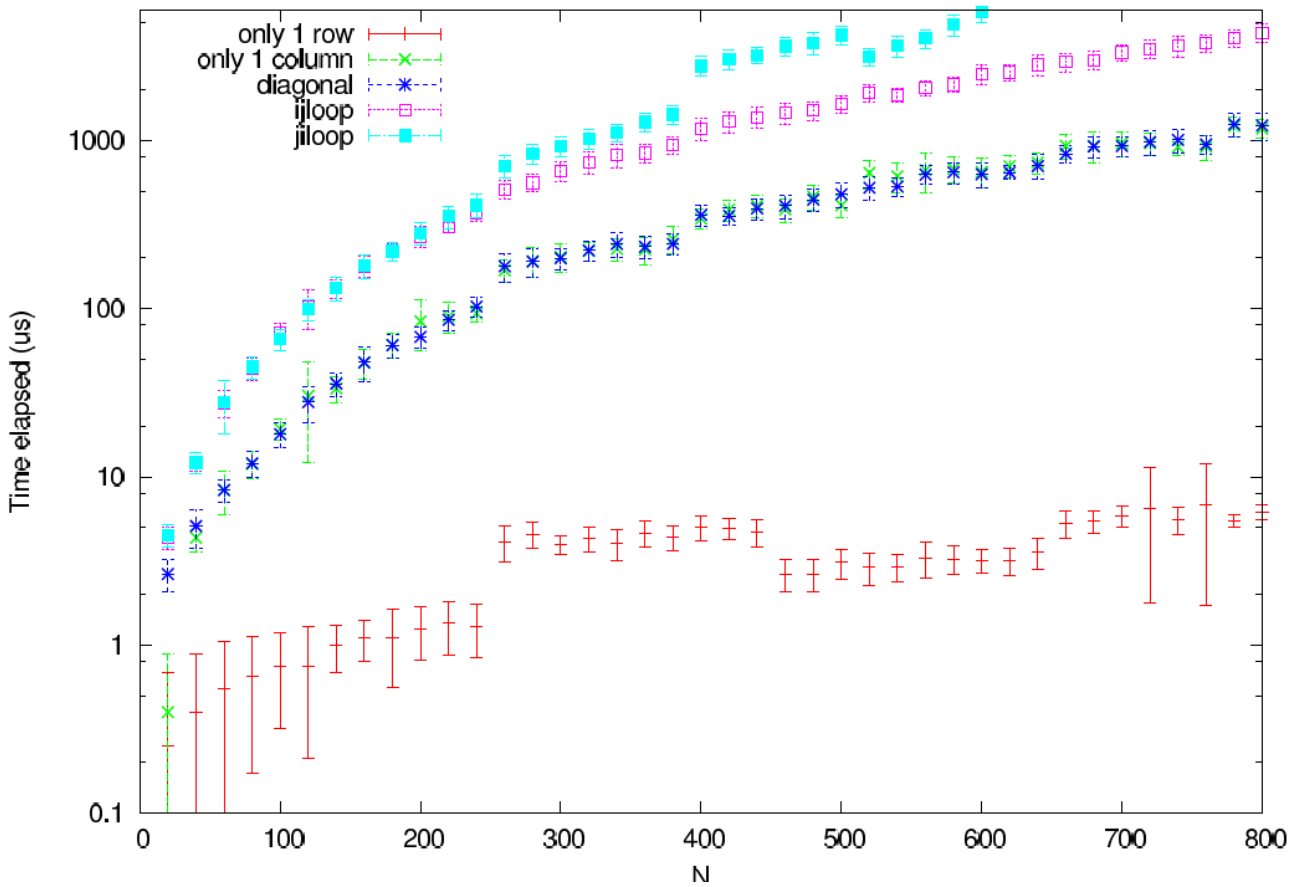
先週のレポート・紹介

3.2　方法

(a) から (e) それぞれについて、$N = 20, 40, 60, ..., 800$ の 40 パターンで時間を計測した。各試行は 20 回繰り返し、かかった平均時間とその標準偏差を求め、横軸 N に対してプロットした (計 $5 \times 40 \times 20 = 4000$ 回)。言語は C++ (with clang) を使用し、オプションは-O2 と最適化なしの 2 通りで行った。

先週のレポート　平均と標準偏差

各試行の結果は以下の図１、図２のようになった。



先週のレポート　平均と標準偏差

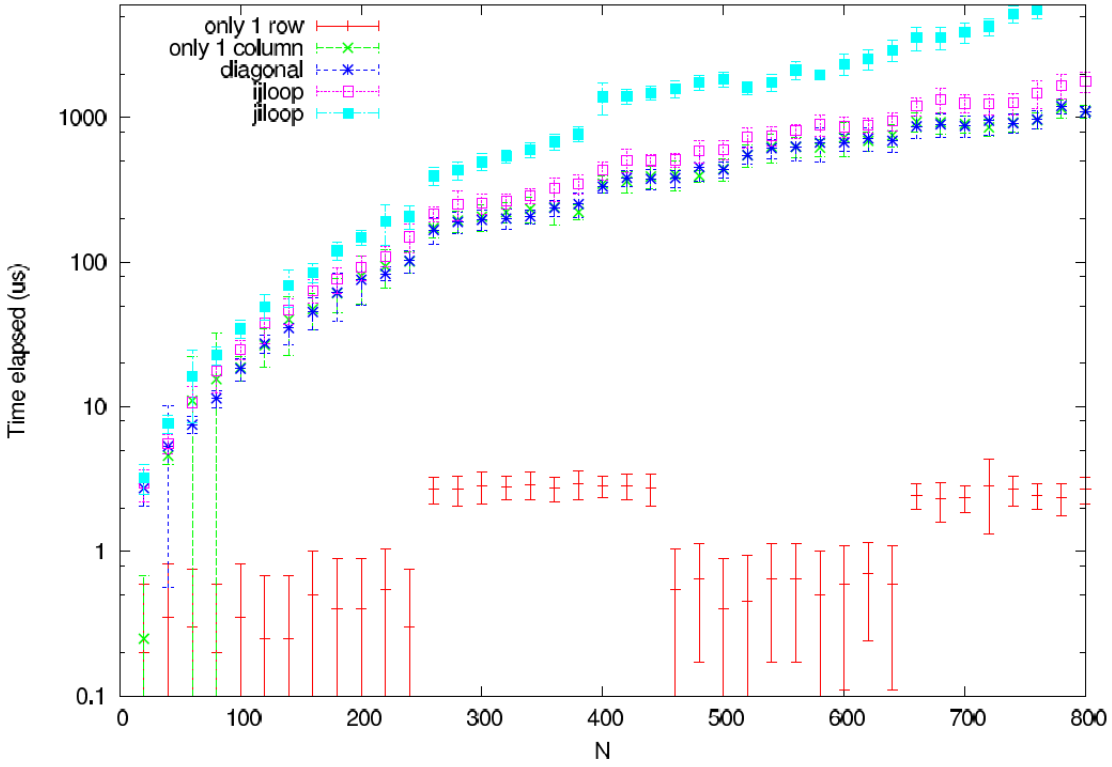
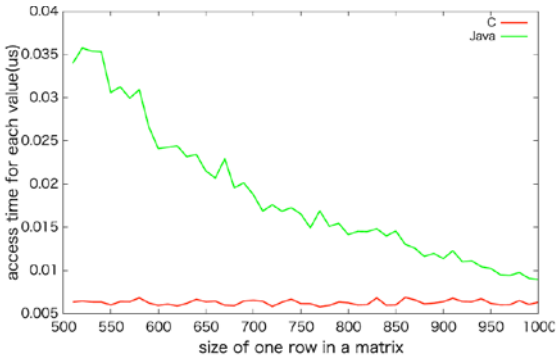


図 2: コンパイラオプションを-O2 とした場合。他は図 1 と同様。

先週のレポート　１要素ごとの時間



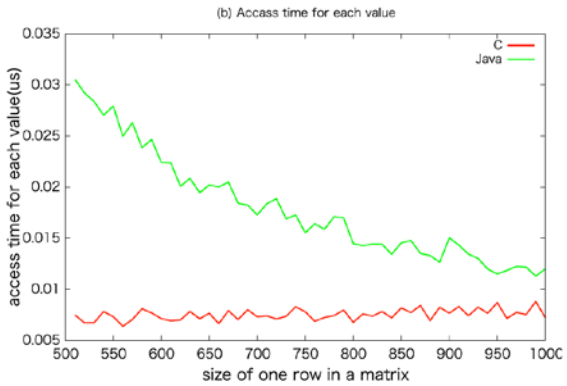


図 7　(b) パターンのアクセスにおける 1 要素あたりのアクセス時間

図 8　(c) パターンのアクセスにおける 1 要素あたりのアクセス時間

図 9　(d) パターンのアクセスにおける 1 要素あたりのアクセス時間

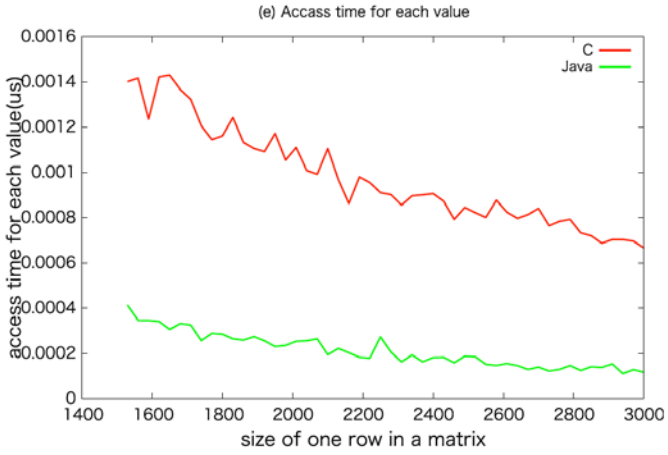


図 10　(e) パターンのアクセスにおける 1 要素あたりのアクセス時間

データと説得力

毎回、結果は同じか？

計算機実験では

「毎回、答えが同じ」の事は多い。

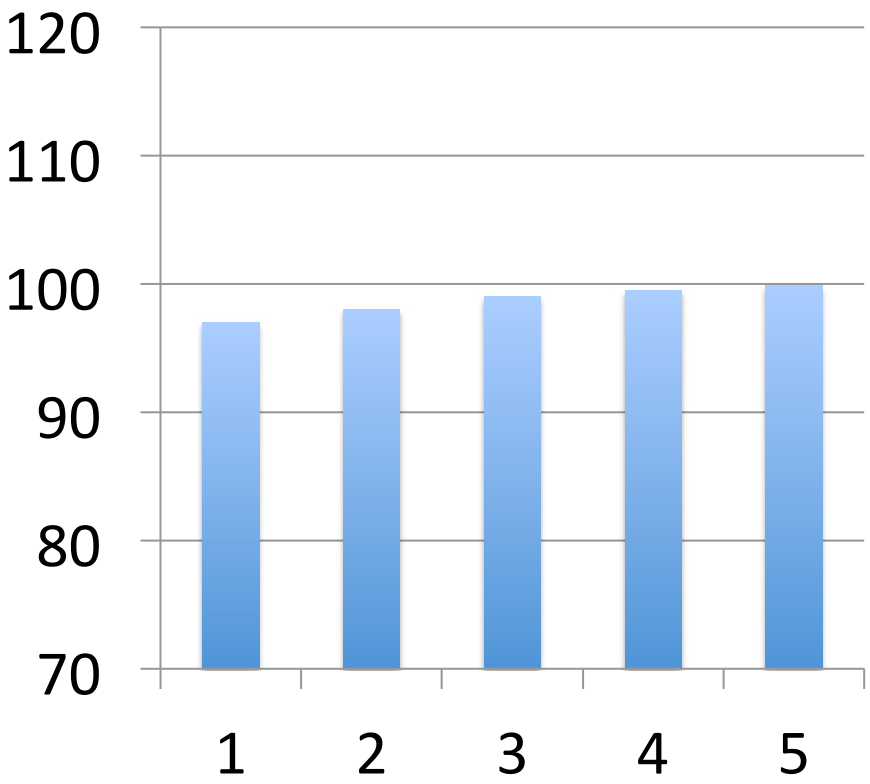
違う原因も推測できる　→　乱数とかバグとか

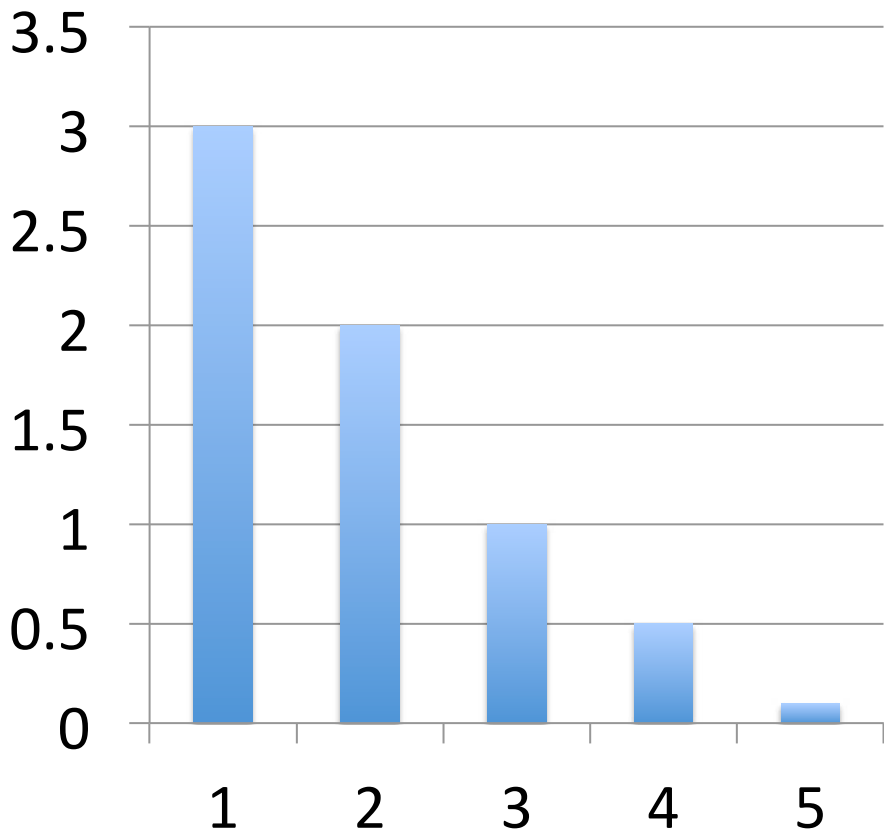
「同じ環境・プログラムでも、**時間が違う**」ことは多い。

グラフにするときに、深呼吸して考えてみましょう。

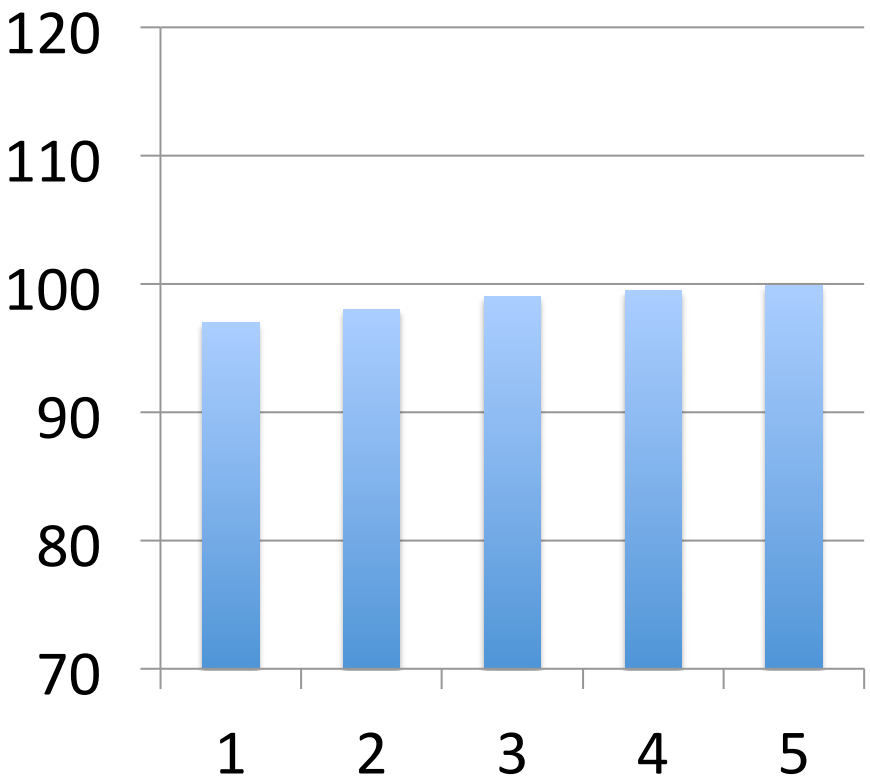
- 軸をどうするか　（対数？　そろえる？）
- 何を見せるか？　（全体？　１個あたり？）

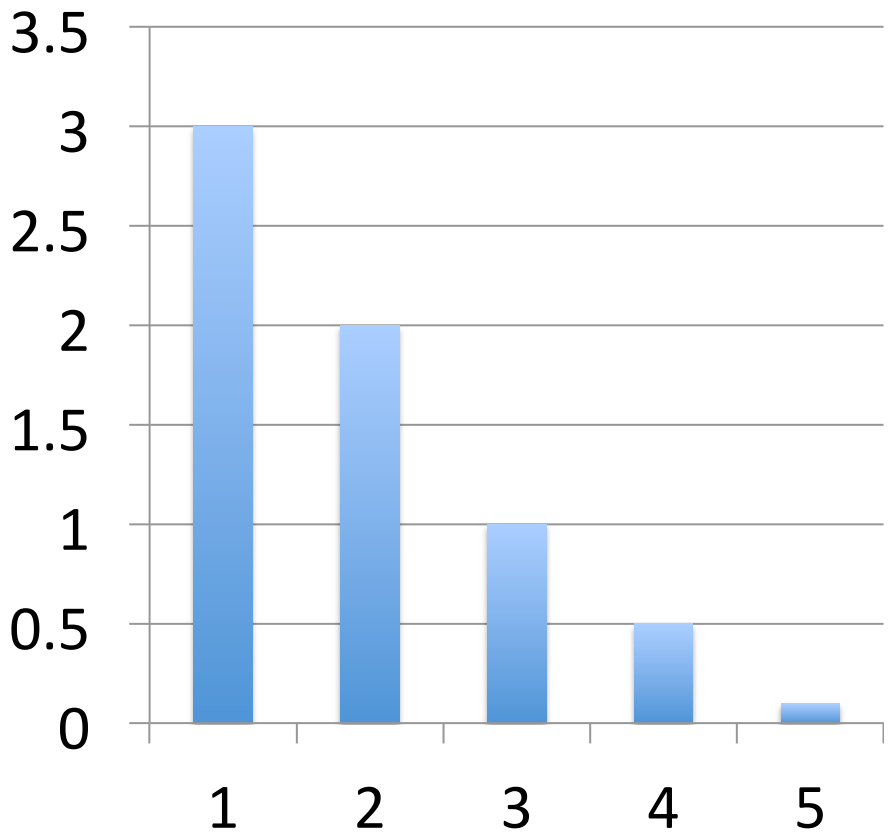
(おまけ)グラフの見せ方





(おまけ)グラフの見せ方





両方とも同じデータから作ったグラフ

1: 97,　2:98,　3:99,　4:99.5,　5:99.9

（先週のふりかえり） データ構造

実装は(ちょっと)忘れて　**抽象化したもの**

→　**データ構造　＋　オペレーション**

大雑把に分けると

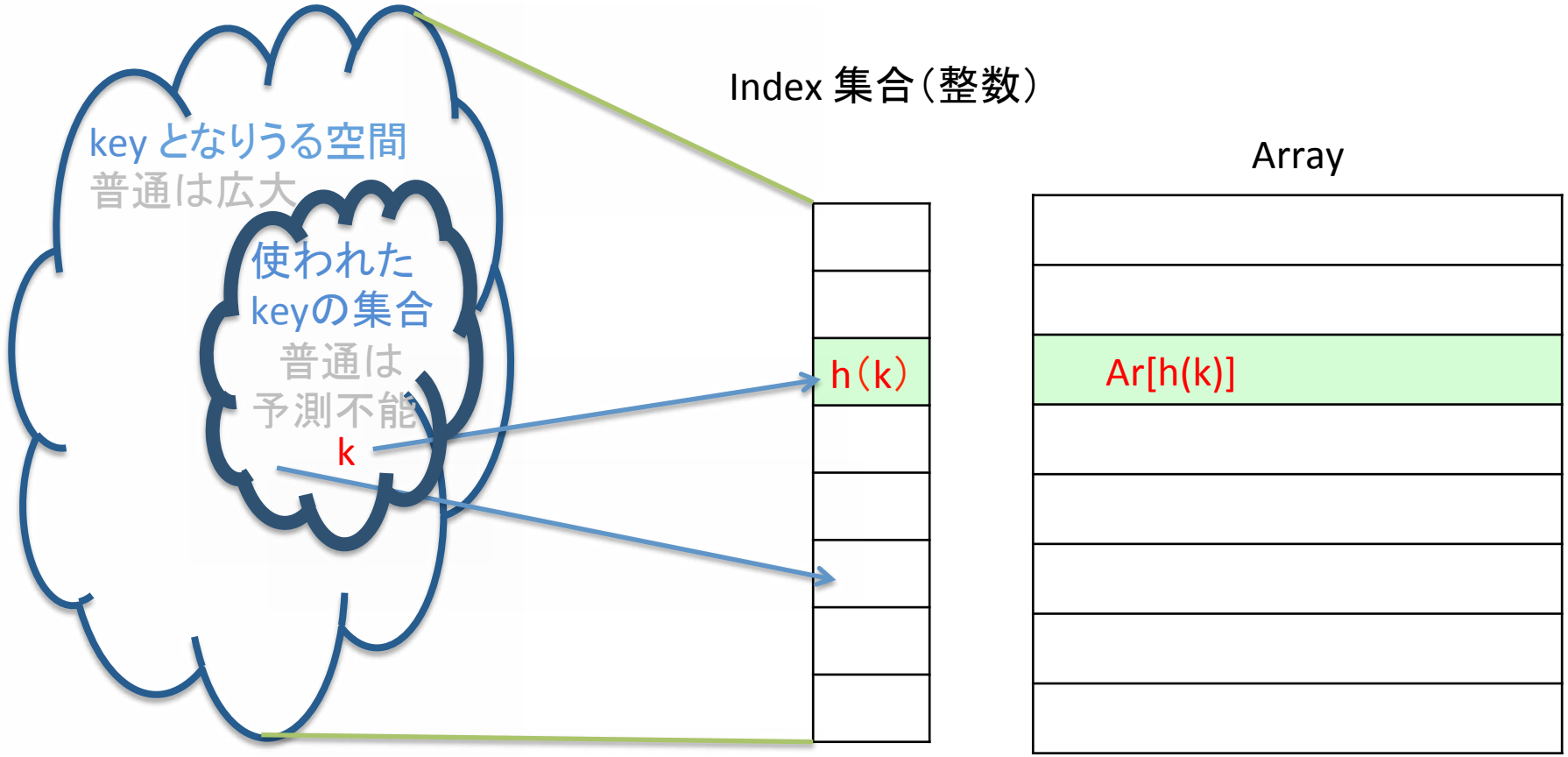
（1）　連続　（メモリ上, 定数時間アクセス）

Array,　Matrix,　Heap, Stack, Queue,　**Hash**

（2）　リンク　（insert が簡単、**overflow しない**、flexible）

List,　Tree,　Graph

HASH: 速くしまう　&　速く取り出す

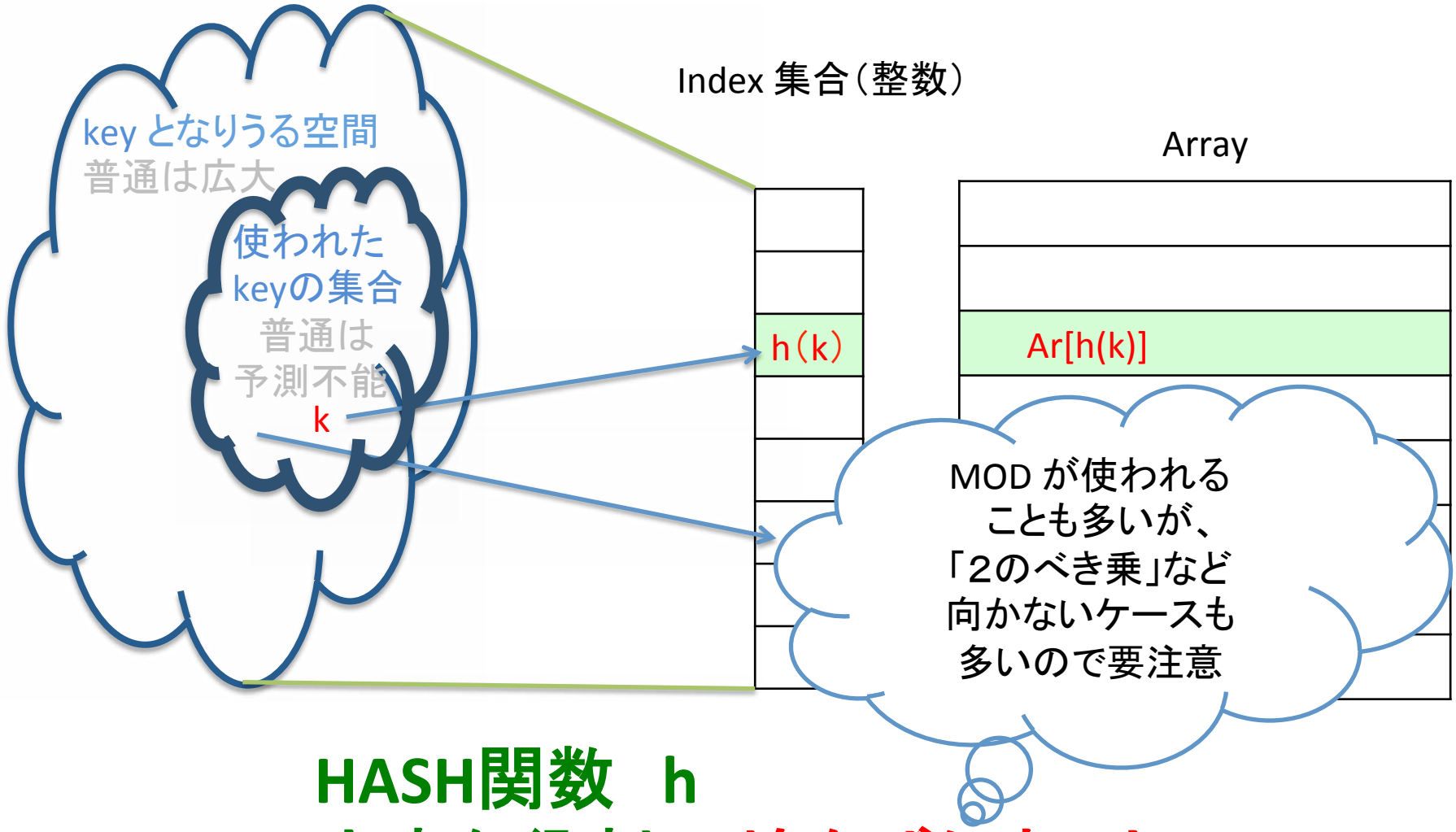


HASH関数　h

大事な役割：　塊をばらすこと

絶対必要な事：　速く計算できること！

HASH: 速くしまう　&　速く取り出す



HASH関数　h

大事な役割：　塊をばらすこと

絶対必要な事：　速く計算できること！

ハッシュの難しさ

- ハッシュ関数に何を使うか？
- 衝突したときどうするか？

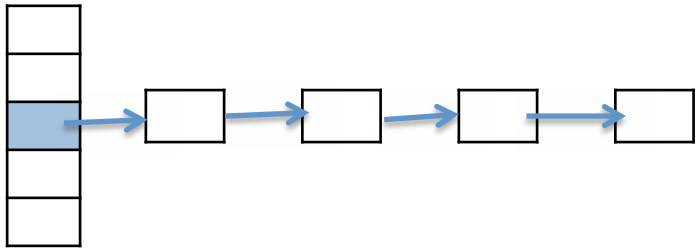
- 範囲指定の探索は？
- key の大きさが最初わからないことが多い
- 最悪ケースの保証が難しい
  - 応答速度が人命にかかわるような リアルタイムアプリケーションには使わない方が良い

    [The Art of Computer Programming, Vol.3 (Knuth)]

Hash：衝突の回避

Chaining

衝突したものをリストでつなぐ



Open Adressing

衝突したら再ハッシュ（削除が大変）

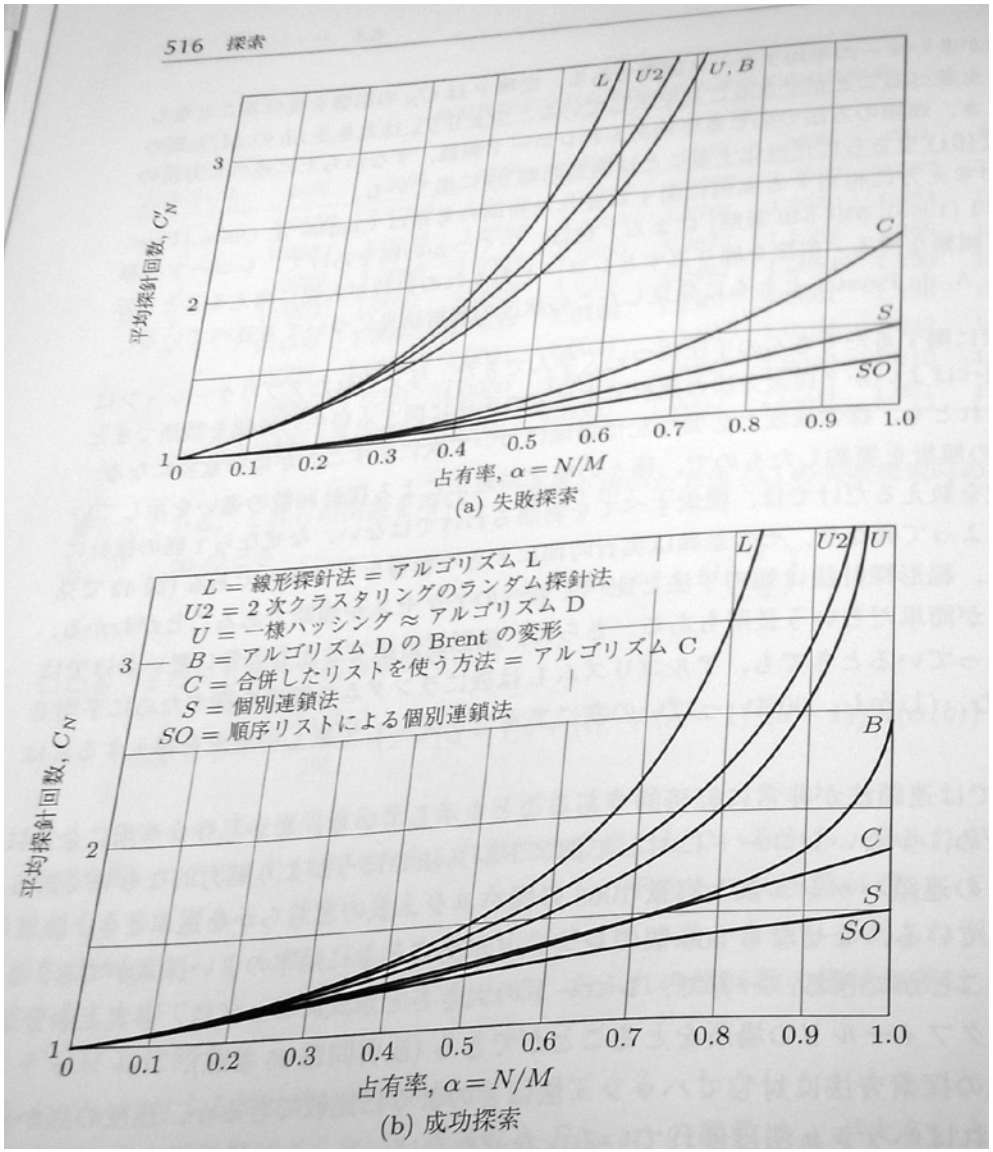
Hash → 色々なところでとても良く使われる

使うとき少し調べてから使うほうが良い

Hash については（も）、Knuth 本がとても詳しい

HASH
性能変化

興味を
持った人は
ぜひ勉強
してください♪



The art of Computer Programming (Knuth) 3 巻より

しまった物を速く探したい

（おまけ） コンパイラのシンボルテーブル

プログラムの識別子(関数名とか変数名とか）の属性（位置、型、スコープ）などを まとめた表。hash で実装することが多い。

（nm コマンドで見えることが多い）

（先週の補足のおまけ）
コンパイラのシンボルテーブル

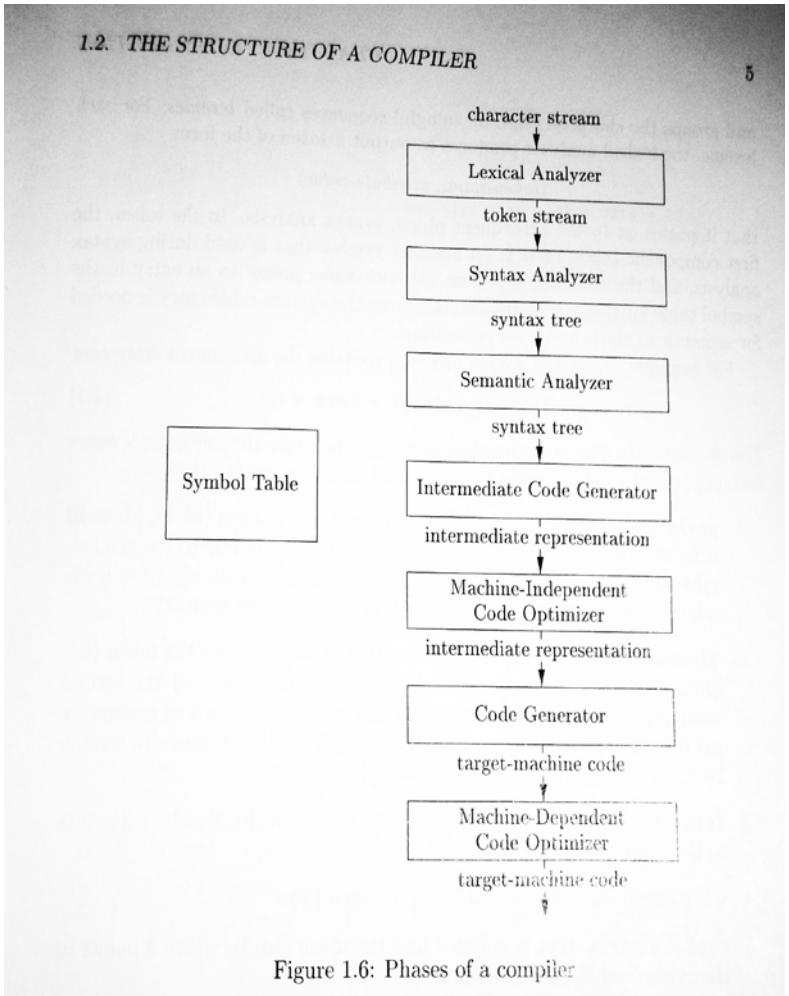


Figure 1.6: Phases of a compiler

Compilers (Dragon book) より

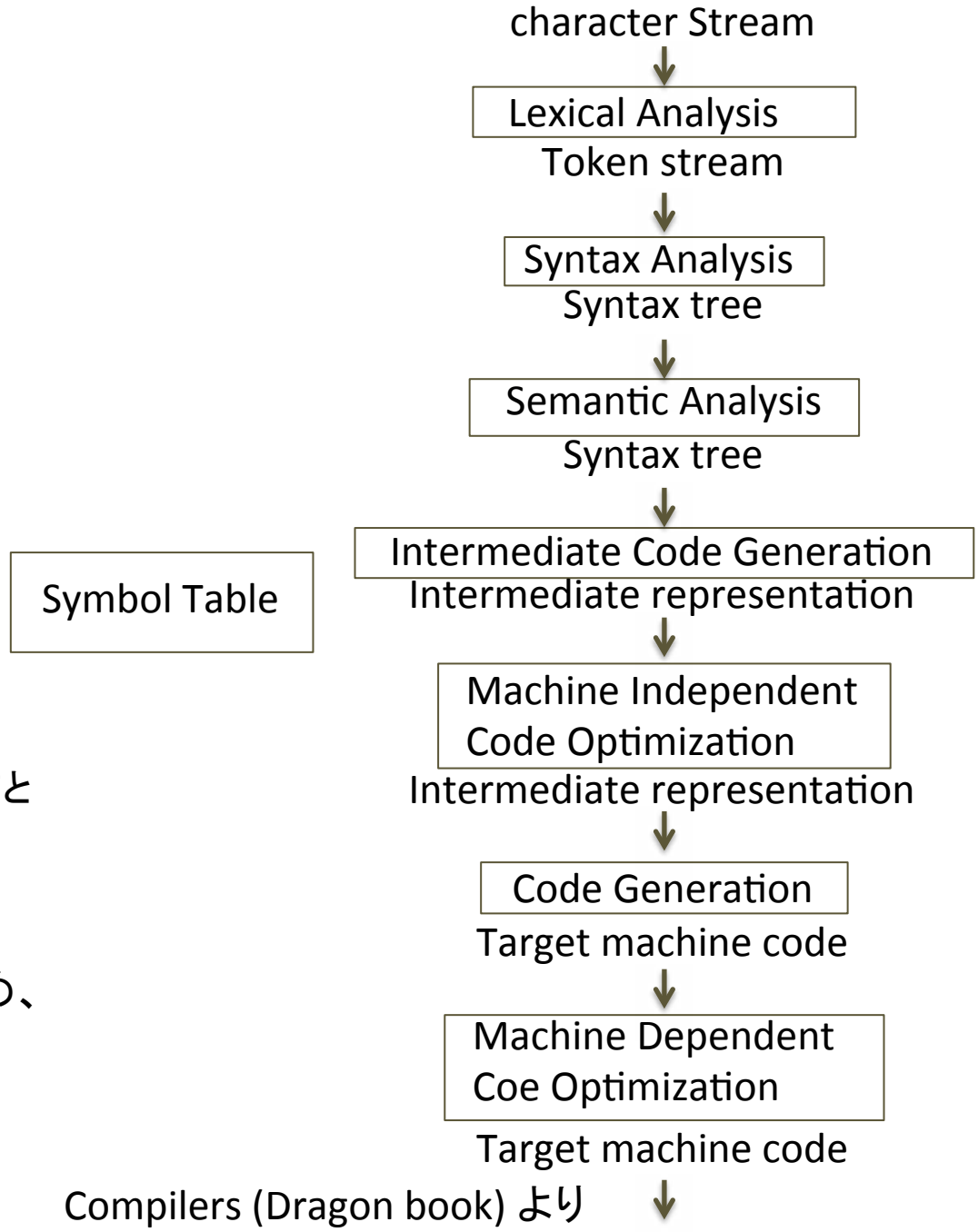
（おまけ）
コンパイラの
シンボルテーブル

（コンパイラは）
ソースコードで使われた variable の
「名前」と各名前に対する「属性」を
記録する。

「属性」とは、例えば 「型」や
「スコープ」。variable が「関数」の
場合は、これに加えて、
引数の数や型も記録。

シンボルテーブルは、各variable の名前と
属性を記録するためのデータ構造。

コンパイラは、シンボルテーブルに
記録されている情報を利用するため、
すばやく記録し、すばやく探したい。



Compilers (Dragon book) より

(続・おまけ)よろしくない C のコード

```
#include<stdio.h>

/* (注)こういう宣言をしてはいけない */
int lvar;
float fvar;

void
plus3()
{
        lvar+=3;
        fvar+=3;
}

int
main()
{
        scanf("%d",&lvar);
        scanf("%f",&fvar);

        plus3();

        printf("int = %d, float = %1f\n", lvar, fvar);
}
```

（おまけ） cc –S test.c（の一部）

```
_plus3:                                 ## @plus3
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp2:
        .cfi_def_cfa_offset 16
Ltmp3:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp4:
        .cfi_def_cfa_register %rbp

        movq    _fvar@GOTPCREL(%rip), %rax
        movabsq $3, %rcx
        cvtsi2ssq       %rcx, %xmm0

        movq    _lvar@GOTPCREL(%rip), %rcx
        movl    (%rcx), %edx
        addl    $3, %edx
        movl    %edx, (%rcx)

        movss   (%rax), %xmm1
        addss   %xmm0, %xmm1
        movss   %xmm1, (%rax)

        popq    %rbp
        ret
        .cfi_endproc
```

（蛇足のおまけ） nm a.out

```
nm a.out
0000000100000000 T __mh_execute_header
0000000100001020 S _fvar
0000000100001024 S _lvar
0000000100000ed0 T _main
0000000100000e90 T _plus3
                 U _printf
                 U _scanf
                 U dyld_stub_binder
```

strip すると消える

（蛇足） シンボルリスト

型によってターゲットコードは変わる
スコープによって階層化
動的に型が変わると、もちろん大変

昔のコンパイラの参考書
→ 「ハッシュかリストで実装」と書いてある

今のコンパイラの参考書
→ 実装は気にしない（言語のレベルで隠蔽）

データ構造

Asymptotic Notation と相性がいい♪

これも　実装は(ちょっと)忘れて　**抽象化したもの**

→　**データ構造　＋　オペレーション**

大雑把に分けると

（1）　連続　（メモリ上, 定数時間アクセス）

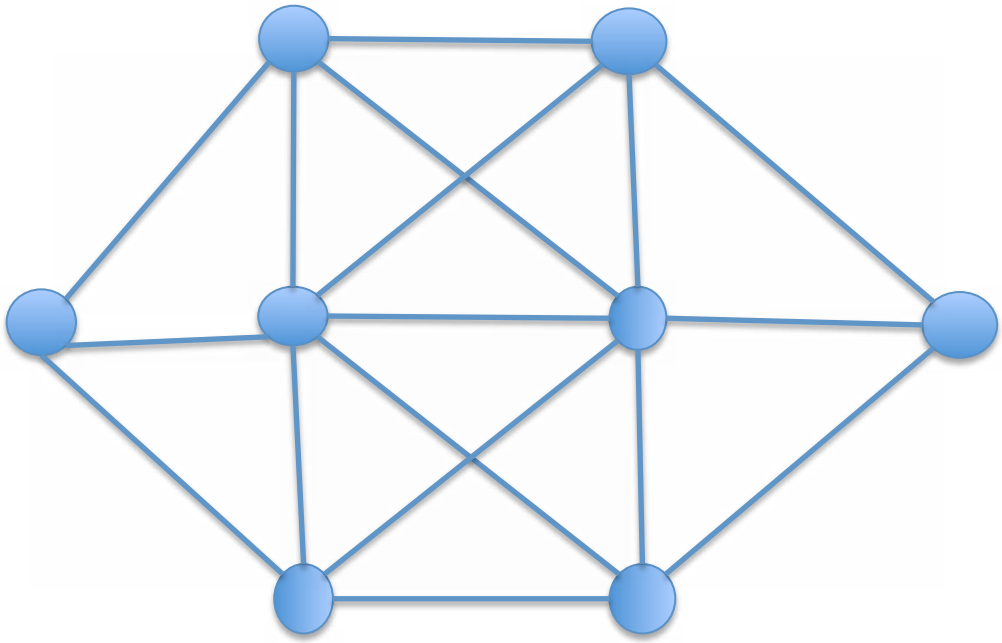
Array,　Matrix,　Hash,　Heap, Queue

（2）　リンク　（ポインタでつながっている）

List,　Tree,　Graph

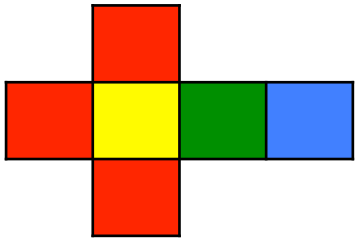
グラフ

パズル
グラフ理論入門(Wilson)より

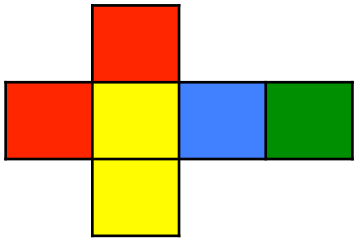


図の８個の青い丸に、ABCDEFGH の　８文字を割り付ける。
ただし、アルファベットで隣接する文字同士は線で結ばれないこと。

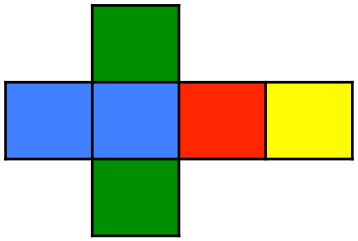
パズル（その２）
グラフ理論入門(Wilson)より



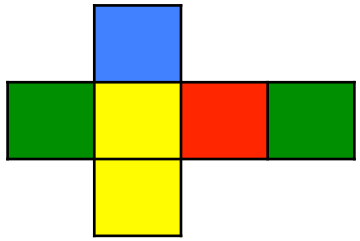
正立方体１



正立方体２

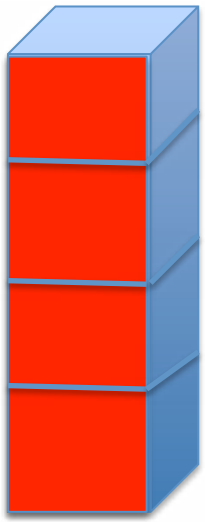


正立方体３



正立方体４

上の４つの正立方体を上手に積み上げて
各側面が一色で塗られるような柱を構成できるか？



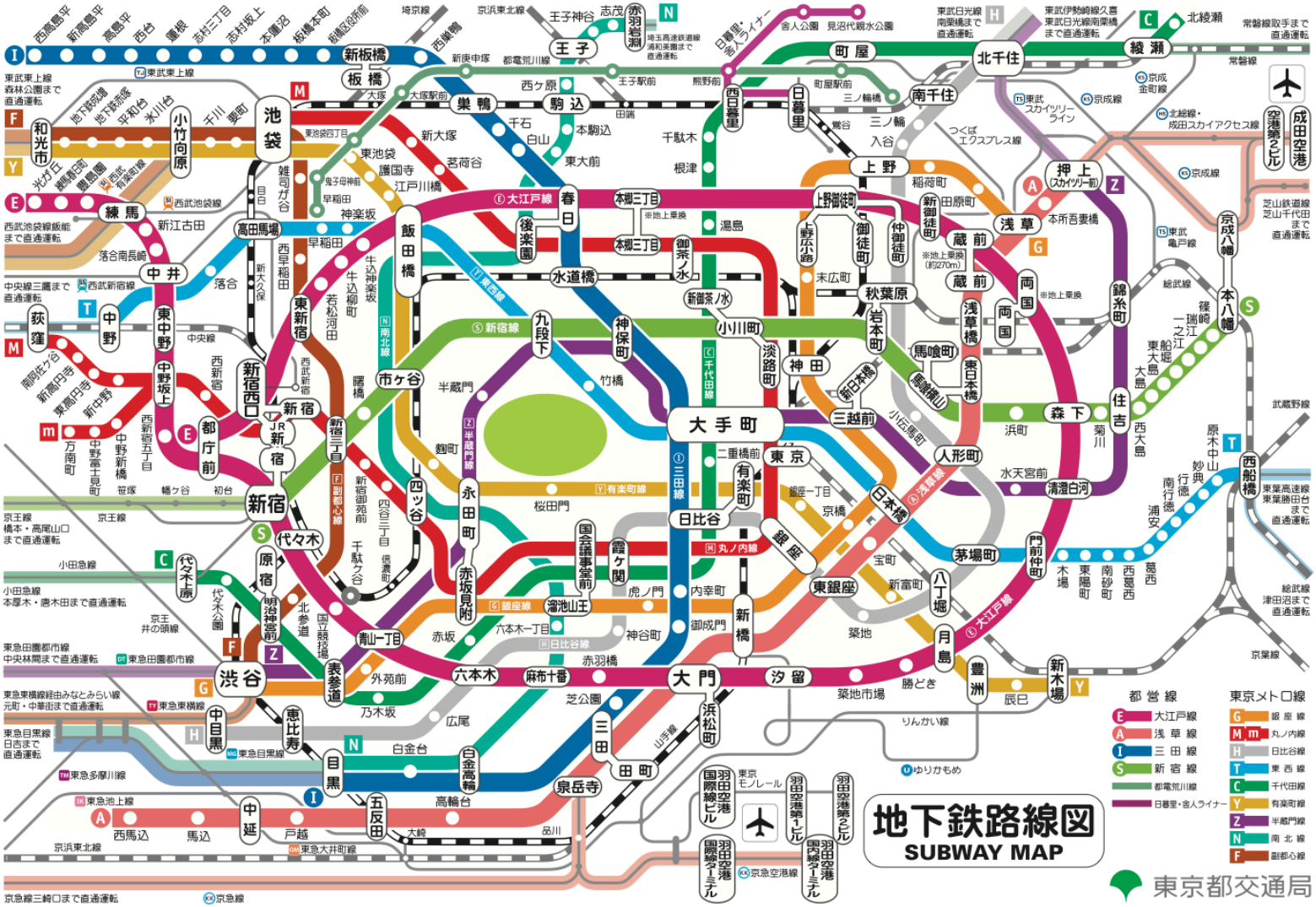
正立方体４

正立方体３

正立方体２

正立方体１

地下鉄路線図


地下鉄路線図
SUBWAY MAP
都営線
大江戸線
浅草線
三田線
新宿線
都電荒川線
日暮里・舎人ライナー
東京メトロ線
銀座線
丸ノ内線
日比谷線
東西線
千代田線
有楽町線
半蔵門線
南北線
副都心線
東京都交通局
新板橋
板橋
王子
赤羽岩淵
町屋
北千住
綾瀬
巣鴨
駒込
池袋
和光市
小竹向原
練馬
高田馬場
中井
飯田橋
春日
後楽園
水道橋
御茶ノ水
上野
秋葉原
浅草
押上
錦糸町
両国
本八幡
中野
東中野
中野坂上
都庁前
新宿西口
新宿
東新宿
市ヶ谷
九段下
神保町
神田
大手町
東京
有楽町
日比谷
銀座
日本橋
三越前
人形町
森下
住吉
西船橋
代々木
原宿
渋谷
表参道
青山一丁目
赤坂見附
四ツ谷
永田町
霞ケ関
六本木
麻布十番
大門
汐留
築地市場
豊洲
新木場
目黒
恵比寿
中目黒
白金台
白金高輪
泉岳寺
三田
田町
浜松町
中延
五反田
高輪台
西馬込
馬込
戸越
品川
羽田空港国際線ビル
羽田空港国内線ターミナル
羽田空港第1ビル
羽田空港第2ビル
羽田空港国際線ターミナル
羽田空港国内線ターミナル
成田空港
空港第2ビル
京成八幡
京成上野
日暮里
南千住
東京モノレール
ゆりかもめ
りんかい線
京浜東北線
京急線
京急空港線
東急目黒線
東急多摩川線
東急池上線
東急大井町線
東急東横線
東急田園都市線
東急田園都市線
東急東横線経由みなとみらい線
元町・中華街まで直通運転
東急目黒線
日吉まで
直通運転
東急田園都市線
中央林間まで直通運転
小田急線
本厚木・唐木田まで直通運転
京王
井の頭線
京王線
京王線
橋本・高尾山口
まで直通運転
京急線三崎口まで直通運転
東武東上線
森林公園まで
直通運転
西武池袋線飯能
まで直通運転
中央線三鷹まで
直通運転
西武新宿線
東武日光線
南栗橋まで
直通運転
東武伊勢崎線久喜
東武日光線南栗橋
まで直通運転
東武
スカイツリーライン
つくば
エクスプレス線
京成線
北総線・
成田スカイアクセス線
芝山鉄道線
芝山千代田
まで
直通運転
東武
亀戸線
総武線
武蔵野線
東葉高速線
東葉勝田台
まで
直通運転
総武線
津田沼まで
直通運転
京葉線
常磐線取手まで
直通運転
常磐線
埼京線
京浜東北線
都電荒川線
埼玉高速鉄道線
浦和美園まで
直通運転
東武東上線

Observation

- 有限個の点集合
- その点同士をつなぐ線

この二種類の構成要素で表現できるものが
（なぜか）非常に多い。（ここでも多数決の原理♪）

線の形状や長さは無視、
どの点とどの点がつながれているのか（いないのか）
だけに着目　→　グラフ　という数学的概念

計算機の発展とともに

- 弾道計算など　戦争用物理シミュレーション
- 給与計算
- 暗号解読　→　組み合わせ　離散探索問題

離散的な問題を扱う、モデルとして
この「グラフ」という概念が非常にはまった。
重要性が増した。

グラフに落してあげると計算機で計算できる
この計算する手段がアルゴリズム

たくさんあるグラフアルゴリズムの一部

- 最短路問題（２点間、１点全点、全点間）
- オイラー回路（一筆書き）
- 巡回セールスマン（全頂点を一度通る最短路）
- 最小全域木（MST,　Minimum Spanning Tree)
- 最大マッチング
- 安定結婚問題
- 最小費用流

グラフ

お約束の記法

G = (V,E)

G: グラフ

V: 頂点集合

E: 辺集合

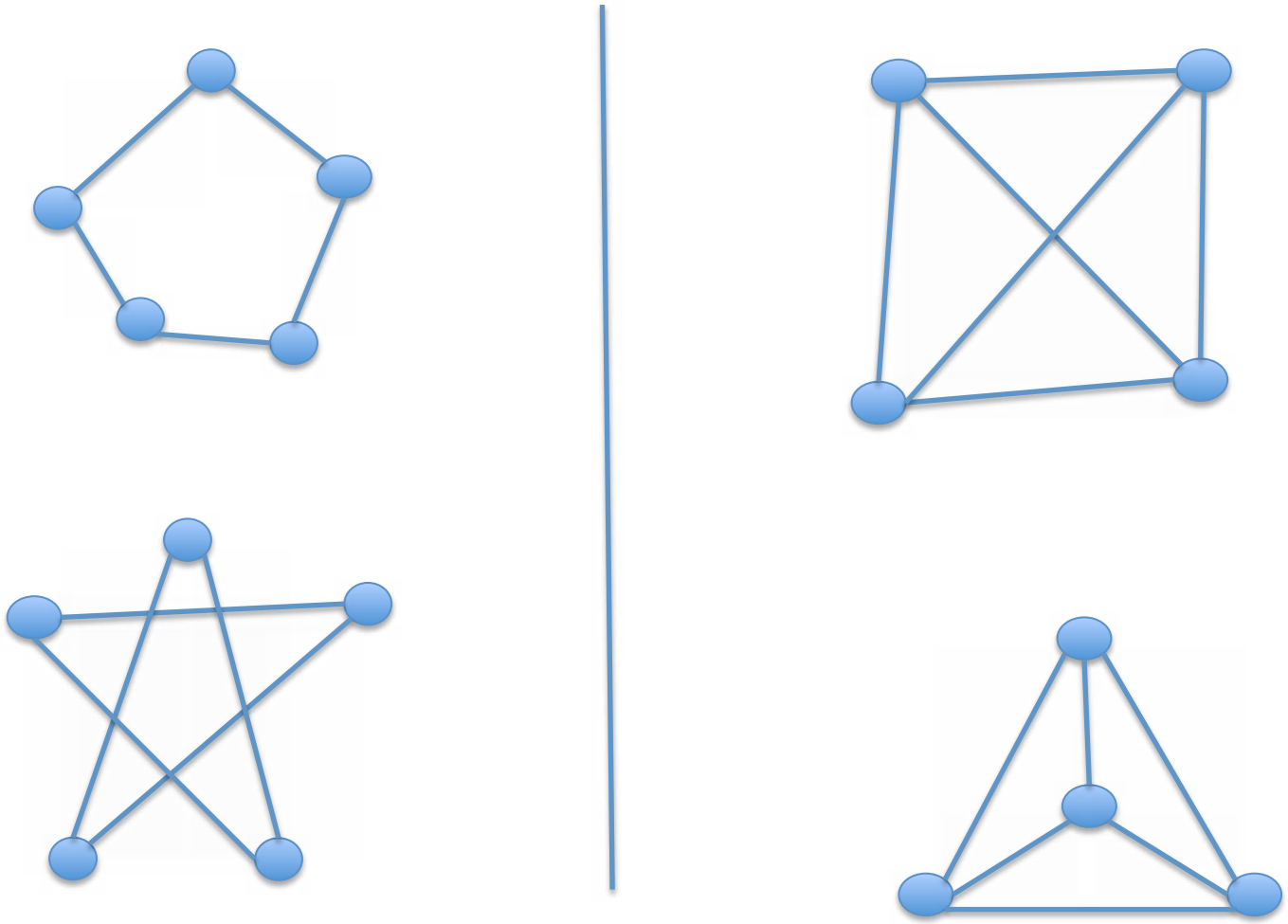
あるグラフ G

そのグラフの頂点集合 V(G)
そのグラフの変集合 E(G)

頂点u,v を結ぶ辺があるとき、u と v は隣接する

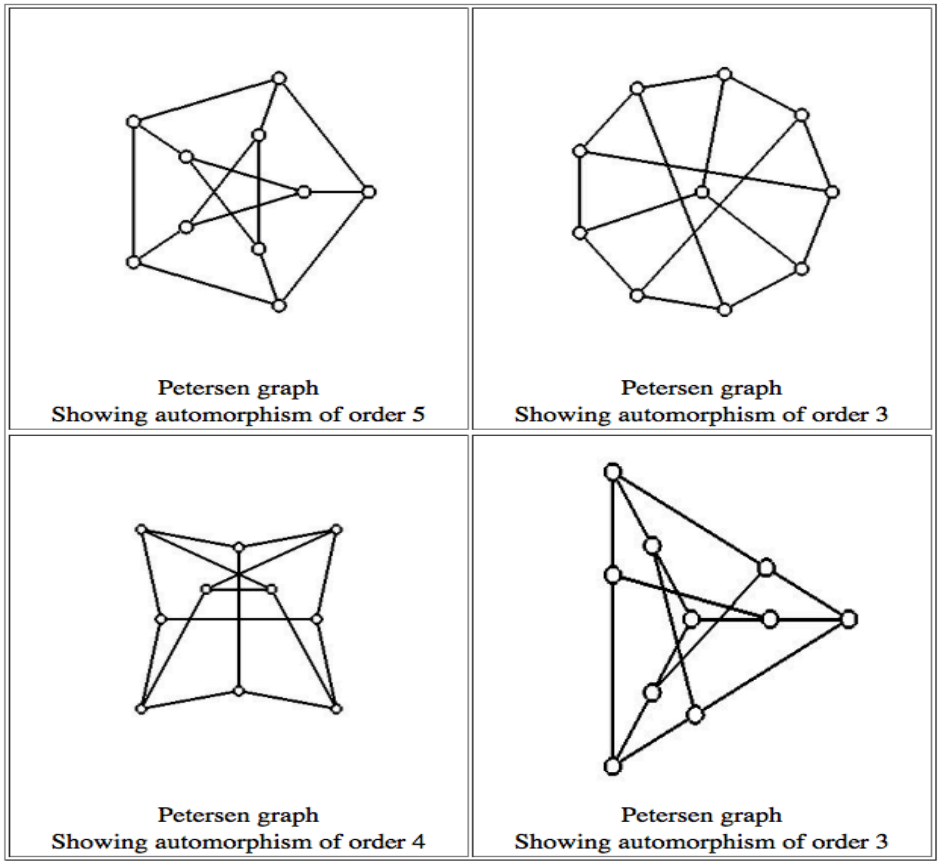
辺に向きがついている場合は、有向グラフ。

見た目は違っても、同じグラフ



見た目は違っても、同じグラフ

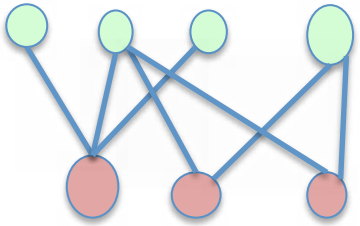
Petersen Graph（ペテルセングラフ）



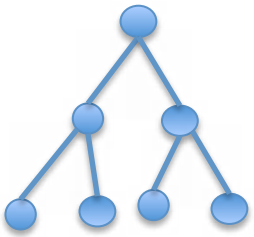
http://math.fau.edu/locke/Graphpic.htm より

良く使う大事なグラフ

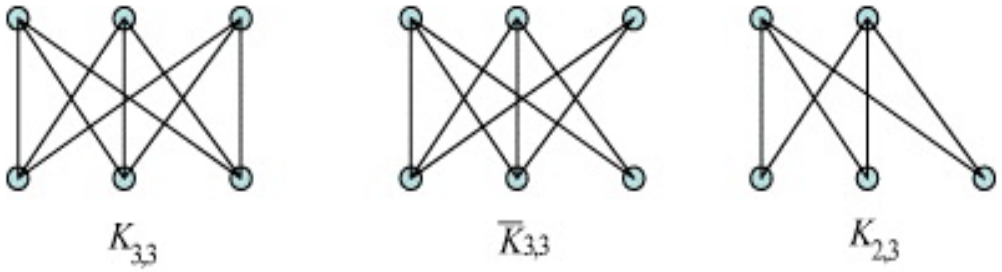
Path Graph Pn



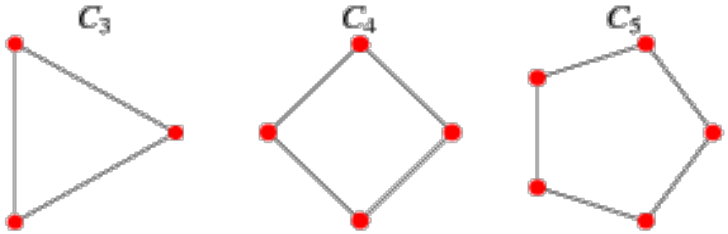
bipartite graph



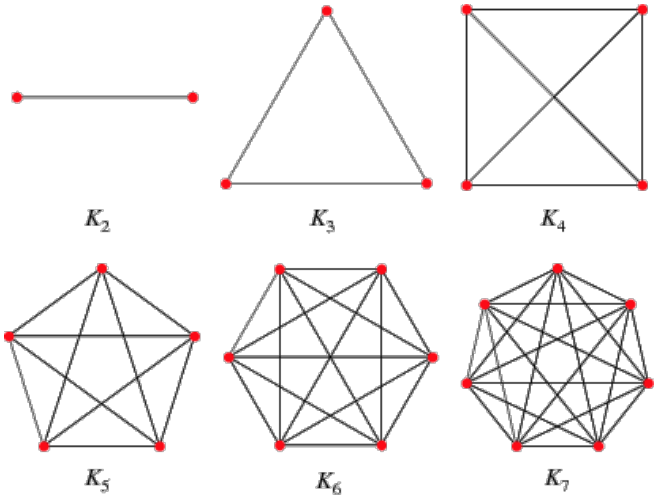
tree



Complete Bipartite Graph
(http://iopscience.iop.org より)



Cycle Graph Cn



Complete Graph Kn

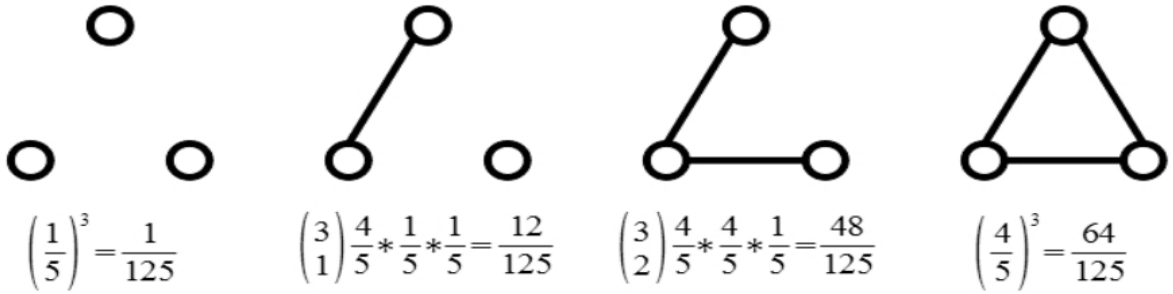
Wolfram MathWorldより

ちょっと変わったグラフ

Random Graph

- 頂点数　n
- 確率 p ，0<=p<=1
- 完全グラフの辺を　確率 pで残したグラフ

- 例　n=3, p=0.8



wikipedia より♪
http://upload.wikimedia.org/wikipedia/commons/f/fa/Random_3.png

グラフの特徴を表す数

- 次数:　ある頂点につながっている辺の数

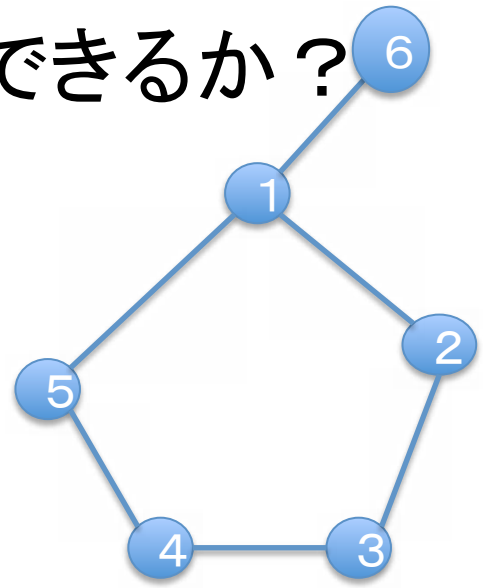
  deg(1)=3, deg(2)=deg(3)=deg(4)=deg(5)=2, deg(6)=1

- 距離:　何本の辺を通ると到達できるか？

  dist(1,3)=2

- 直径:　最遠頂点ペアの距離

  diam(G)=3



グラフのいくつかの基礎的な性質

- 連結・連結度　どのくらい、頑丈に、つながっているか？
  – いくつ**頂点**を取ってもつながっているか？
  （注：　辺ではない）

- 疎・密：　辺の数$|E|$は　$|V|^2$以下
  次数の平均が定数なら　$O(|V|)$ 程度

- 平面性　（辺を交差させずに描画できる）
  オイラーの公式　$|V|-|E|+f = 2$　（f: 面の数）
  クラウトスキーの定理：　グラフが平面なら、
  $K_{3,3}$ や K_5 のような構造を中に持たない。

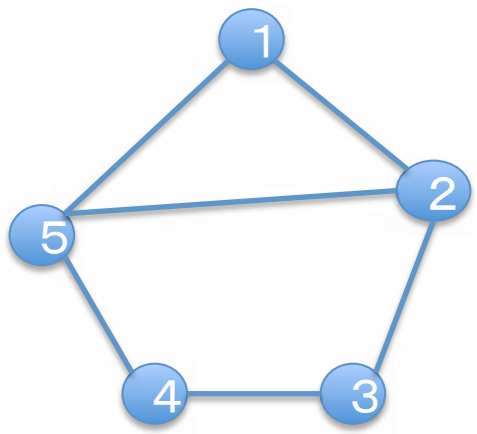
グラフの表現

- 隣接行列
  - 辺があれば1、なければ0
  - 無向グラフなら対称行列
  - **行列のかけ算ができる！**
    - 距離・到達性の計算ができる。
  - 疎の巨大グラフでは無駄
  - 動的グラフに対応できない

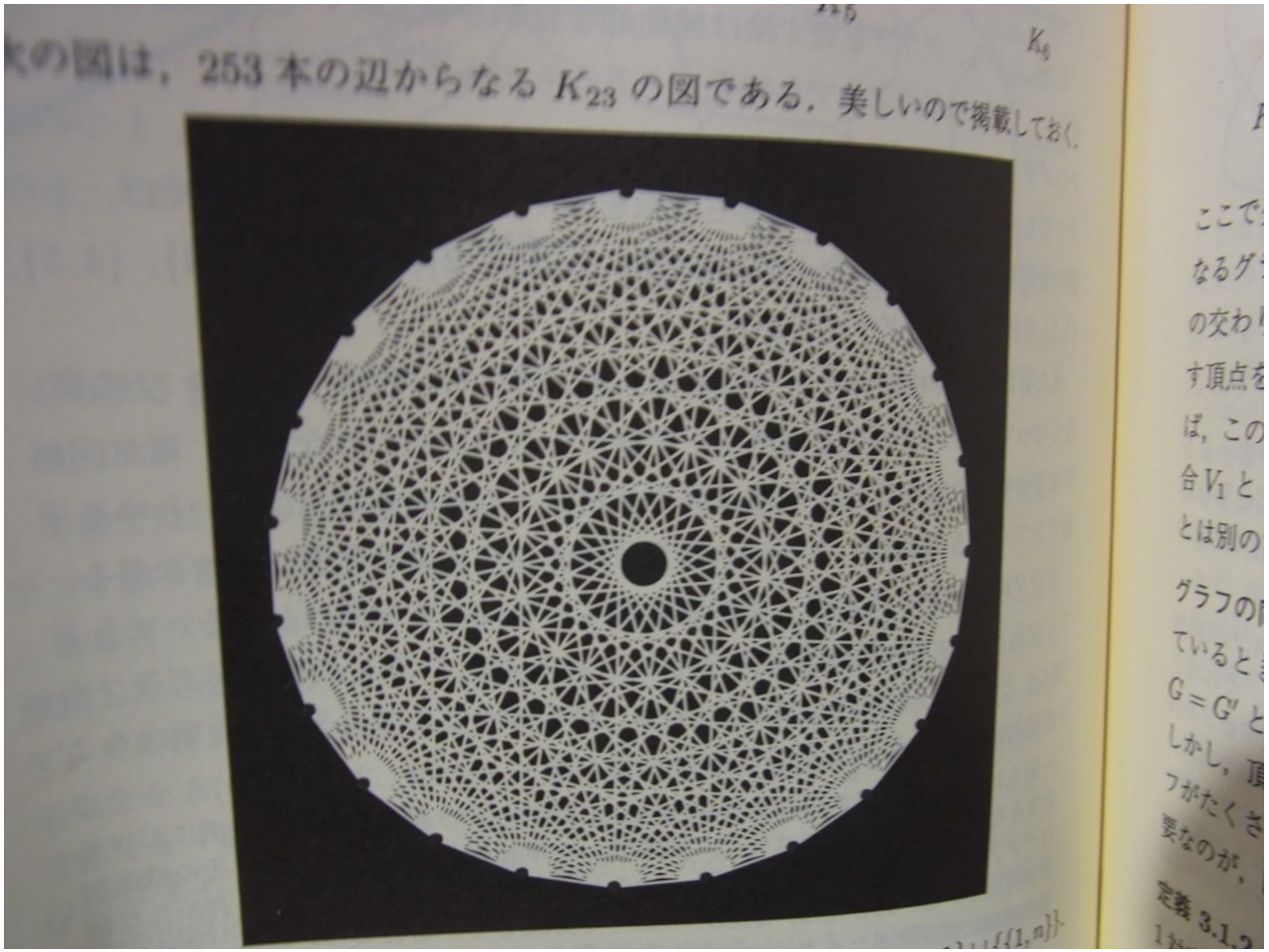
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

- 隣接リスト
  - 動的な管理

1	2,5
2	1,3,5
3	2,4
4	3,5
5	1,2,4

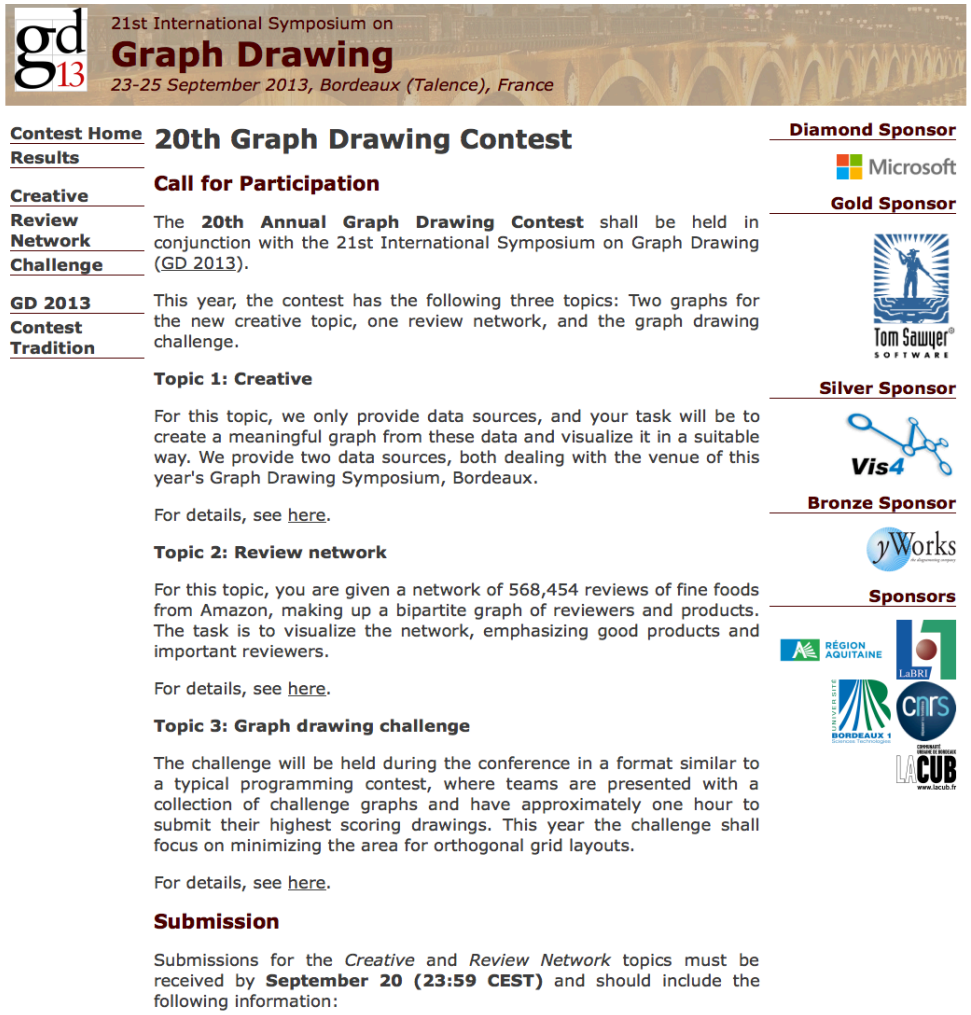


Graph Drawing



K_{23} の　描画　　離散数学への招待（Jiri Matousek, Jaroslav Nesetril)

コンテスト好きの皆様へ



21st International Symposium on
Graph Drawing
23-25 September 2013, Bordeaux (Talence), France

Contest Home
Results
Creative
Review Network
Challenge
GD 2013
Contest Tradition

20th Graph Drawing Contest

Call for Participation

The **20th Annual Graph Drawing Contest** shall be held in conjunction with the 21st International Symposium on Graph Drawing (GD 2013).

This year, the contest has the following three topics: Two graphs for the new creative topic, one review network, and the graph drawing challenge.

Topic 1: Creative

For this topic, we only provide data sources, and your task will be to create a meaningful graph from these data and visualize it in a suitable way. We provide two data sources, both dealing with the venue of this year's Graph Drawing Symposium, Bordeaux.

For details, see here.

Topic 2: Review network

For this topic, you are given a network of 568,454 reviews of fine foods from Amazon, making up a bipartite graph of reviewers and products. The task is to visualize the network, emphasizing good products and important reviewers.

For details, see here.

Topic 3: Graph drawing challenge

The challenge will be held during the conference in a format similar to a typical programming contest, where teams are presented with a collection of challenge graphs and have approximately one hour to submit their highest scoring drawings. This year the challenge shall focus on minimizing the area for orthogonal grid layouts.

For details, see here.

Submission

Submissions for the *Creative* and *Review Network* topics must be received by **September 20 (23:59 CEST)** and should include the following information:

Diamond Sponsor
Microsoft
Gold Sponsor
Tom Sawyer Software
Silver Sponsor
Vis4
Bronze Sponsor
yWorks
Sponsors
Région Aquitaine, LaBRI, Université Bordeaux 1, CNRS, LACUB

去年（2013）の　コンテストの web ページ

Graph Drawing Contest
http://
www.graphdrawing.de

Graph Drawing 併設。
サブミッション〆切は、
シンポジウム開始の
数日前。

2014年は　ドイツ
　9/24ー26

レポート・成績について

- ほぼ毎回、プログラミング課題を出題する予定
  - 効率の良い計算機実験のためのツールを使ってみる
  - アルゴリズムの実装
  - ライブラリの利用・・・　など

- ３回以上、レポートのファイルとプログラムのソースを添付し、メールで提出のこと
  - E-mail: algorithm2014@edu.jar.jp
  - サブジェクト 「アルゴリズムとプログラム実践講座・レポート」
  - 学生証番号と名前は、 メールの本文にも書いてください。
  - 〆切：次の授業の直前の日曜日深夜 （講評の都合上。〆切後も受付ます）
  - プログラムやレポートは（見本として）公開することがあります。適宜、作者名やコピーライトをいれておいてください。公開不可の場合は、プログラムの冒頭にその旨、コメントをいれておいてください。
  - 質問・作問提案も歓迎 （作問については採用の場合は別途加点）
  - サンプルプログラムは「初心者向け」です。 **上級者は無視してください。**

推奨環境など

- Linux, Mac,（Windows+Cygwin）
- 仮想マシン環境（VMware, VirtualBox, Parallels)
  - 余裕があれば、いろいろな組み合わせを試して比較してみると面白いと思います

- 言語
  - 自由。ただし、一般的でない言語については、上記いずれかのOS 上にインストール可能なもの

第三回　課題(その１）

Sample programs

trie.c　test.awk

単語数を数える

たとえば「プロジェクト・グーテンベルグ」や
「青空文庫」から、書籍のデータを収集し、
必要があれば、茶筅等、適当な前処理を行い
出現した単語のヒストグラム等を作成する。

Hash (and/or) Trie を実装、連想配列など
言語が用意する物と性能比較をする。

作成したヒストグラム等から、なんとか面白い
結果をひねり出してみる。☺

この課題の狙うところ

- Trie か hash か （両方でも♪）実装してみる

- 連想リスト等と性能を比べてみる

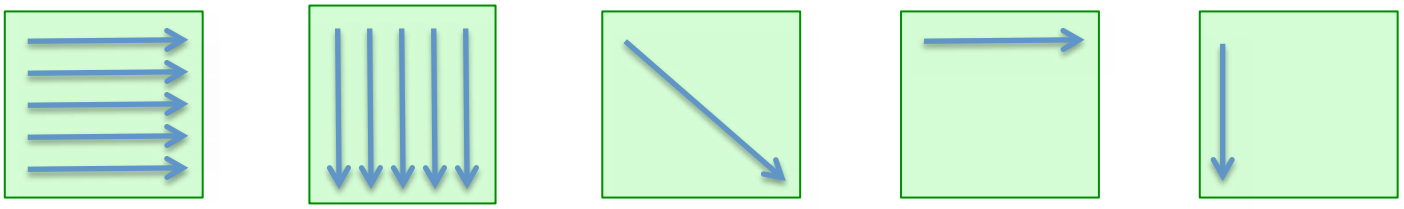
- 現実の大きめなデータを扱ってみる
（予想外に、面倒な処理が発生することもある）

- 大きなデータの中に、何か面白い性質が潜んでいないか探してみる

第三回　課題(その2）

先週の課題：行列のアクセスの時間の計測

- 正方行列に、(a)行順全要素、(b)列順全要素、(c)対角要素のみ、(d)1行のみ　(e)1列のみ代入を行ったあと、代入した要素の読出しを行うプログラムを書き、かかった時間を計測、比較考察せよ。
- 実験環境のうち、ハードウェア、OS、言語、コンパイラオプションなど実験環境のうち、いくつかを変更して、比較考察せよ。
- **実験環境について、CPU のバージョン・動作周波数・メモリサイズ・カーネルのバージョンなど、他人に再現可能なように記述すること。**



(a)　(b)　(c)　(d)　(e)

先週のレポート＋α

出題者の本音

もうひといき♪

どなたか、読んでて面白いレポート
書いてくれないかなぁ・・・

サンプルコードに関する注意

- 授業の際に提示するコードはヒントであって必ずしも良いコードでないことも多い。決してお手本というわけではない。
- たとえば、第二回の Java のサンプルは、出力にかかる時間が大きく何を計りたいかわからなくなりがち
- たとえば html への書き出しは、分散環境で複数人で使うことを真面目に考えるときは、工夫しないといけない。