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(4/12提出分まで)

- 言語・コンパイルオプションの比較（clang はすごい）
- メモリアクセスパターンの差
- 環境比較 （仮想PC 特有の問題など）

- アクセス回数に関する考察 （行数=n ） → １人
  - |ijloop| = |jiloop| = n×|１列 | = n× |１行|= n×|対角|

- 行列のサイズを確認した人 → ０
- 複数回の試行を行った人 → ０

先週のふりかえり

- アルゴリズムの良さの計り方いろいろ
  - 計算量 （実装や現実の機械を忘れ抽象化したもの）
    - 時間計算量（入力サイズに対する最悪値評価）← 一番多い
      - ループの数とか木の形とか (易しい、解ける、難しい）
    - 空間計算量（メモリーをどれだけ使うか？）
    - randomized algorithm (高い確率で・・・など）
  - 収束回数（今回の授業の範囲外）

- 難しいと言われる問題もなんとか解きたい
  - 「厳密解にどこまで近い解が出せるか」なども一つの尺度

計算がスケールするか？

たとえば、入力のサイズ n に対して

線形時間の時間計算量

データのサイズが10倍になったら時間も10倍 → 速い

準線形時間 O(n log n) の時間計算量 → 速い

多項式時間アルゴリズム → 計算できる

二乗時間アルゴリズム

データのサイズが 10倍になったら計算時間は 100倍

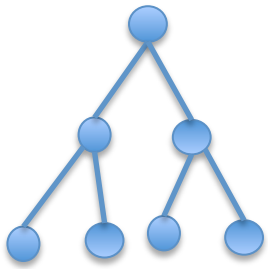
指数時間アルゴリズム → 難しい

データサイズが1増えても 厳しくなることも → 組み合わせ爆発

大雑把な話

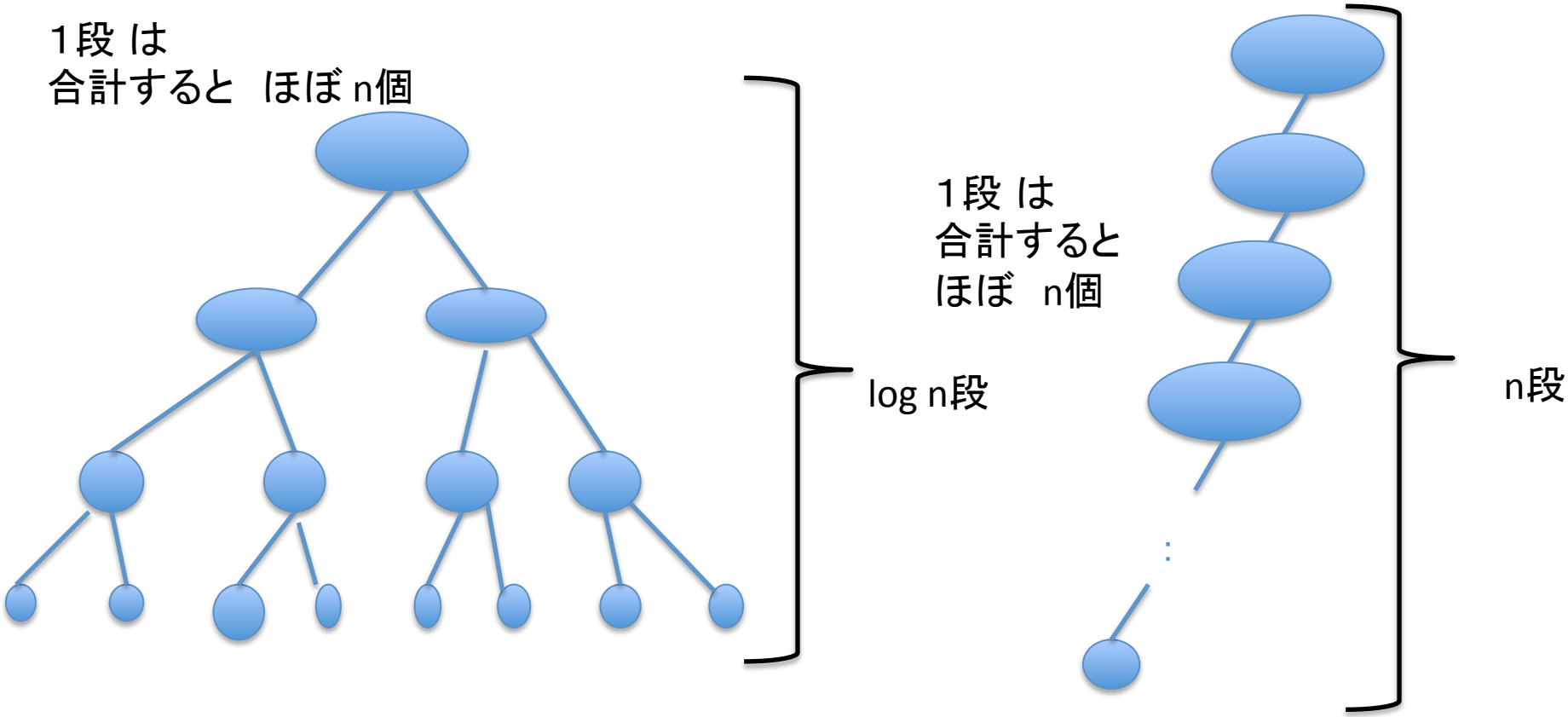
繰り返しがなければプログラムはすぐ終る。
時間がかかるのは、「繰り返しの演算」の部分

- For 文の中身が何度実行されるか？
  - たとえば　1..n の　３重ループなら　n^3回
- Tree のサイズがどのくらいか？
  - たとえば、leaf がn 個なのか
  - たとえば、高さが n なのか



先週の補足　Tree の形

n個の要素を比較するためには、O(n)時間が必要



基礎的なパラダイム（考え方）
分割統治　(Divide and Conquer)

（１）　Divide（ subproblem に分割する）

（２）　Conquer

（３）　Combine

Merge Sort

（１） Divide（ subproblem に分割する）
半分にわける

（２） Conquer
半分にわけたものをMerge Sort する

（３） Combine
二つのグループを順に一つにする
（時間は　サイズ分かかる）

→ O(n log n)　　$T(n)= 2 \cdot T(n/2)+O(n)$

お約束のように良くでてくる式

Binary Search
（sort 済の配列から x を探す）

（１） Divide（ subproblem に分割する）
真ん中と x を比べる
（２） Conquer
大小関係に従って、どちらかに
（３） Combine
何もしない
→ O (logn)　　$T(n)=T(n/2)+O(1)$
お約束のように良くでてくる式

Randomized Algorithm
（かっこいいなと思ったアルゴリズム）
（先週の復習）

Random Quick Sort Algorithm

- クイックソートは速く動くことが多い
- ただし入力がすでにソートされていると$O(n^2)$
- ピボットにする要素をランダムに選べばいいじゃない！　→　**高い確率でうまくいく**

Random selection

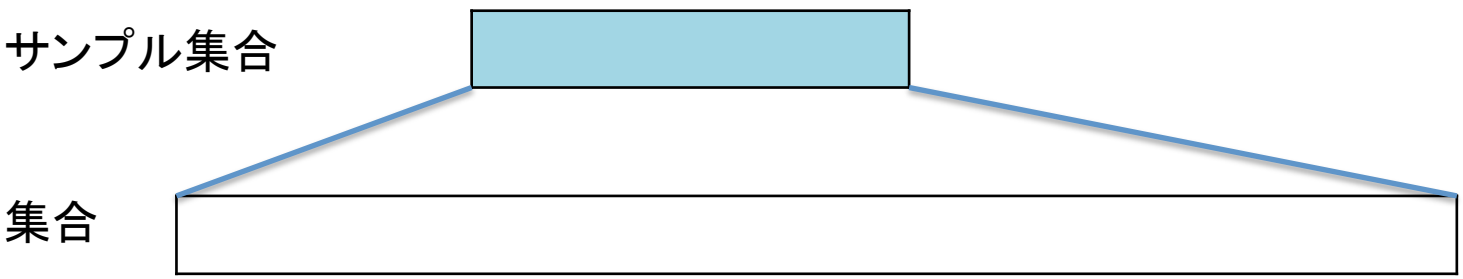
（k 番目の大きさの要素をO(n)で求めたい）

(1) Random にサンプル集合を選ぶ

(2) サンプル集合をソートし、k番目あたりの上と下で少しだけ幅をもたせた要素を選ぶ。

(3) 元々の集合を、選んだ要素をセパレータに大・中・小の集合にわける。

(4) k番目が中の集合にあれば、中の集合だけソートしk番目を求める。→中の集合が十分小さければ成功



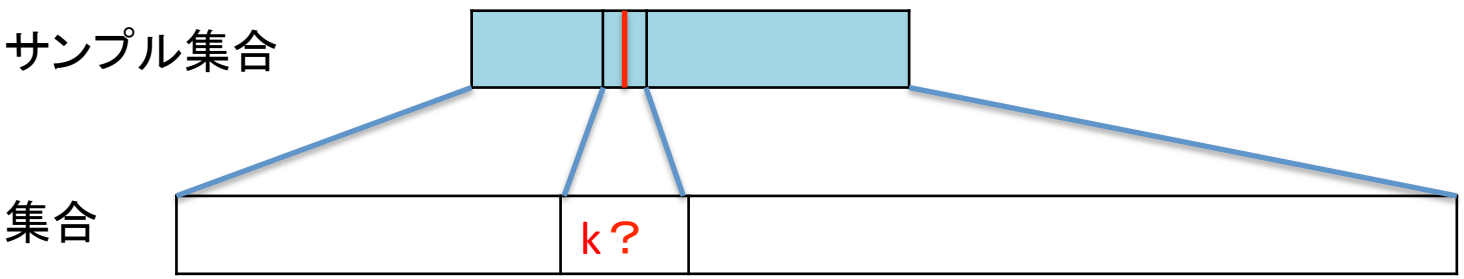
Random selection
（k 番目の大きさの要素を求める）

(1) Random にサンプル集合を選ぶ

(2) サンプル集合をソートし、k番目あたりの上と下で少しだけ幅をもたせた要素を選ぶ。

(3) 元々の集合を、選んだ要素をセパレータに大・中・小の集合にわける。

(4) k番目が中の集合にあれば、中の集合だけソートしk番目を求める。→中の集合が十分小さければ成功



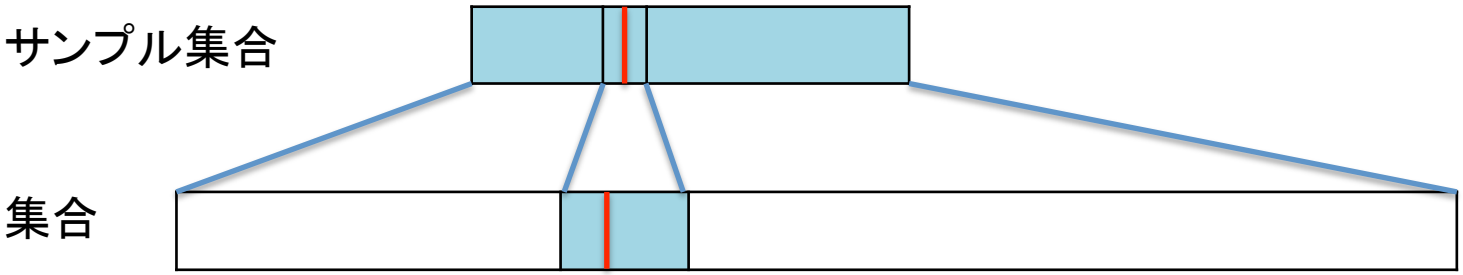
Random selection
（k 番目の大きさの要素を求める）

(1) Random にサンプル集合を選ぶ

(2) サンプル集合をソートし、k番目あたりの上と下で少しだけ幅をもたせた要素を選ぶ。

(3) 元々の集合を、選んだ要素をセパレータに大・中・小の集合にわける。

(4) k番目が中の集合にあれば、中の集合だけソートしk番目を求める。→中の集合が十分小さければ成功



Randomized Algorithm については、Raghavan 本が、本当にお薦め

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

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）

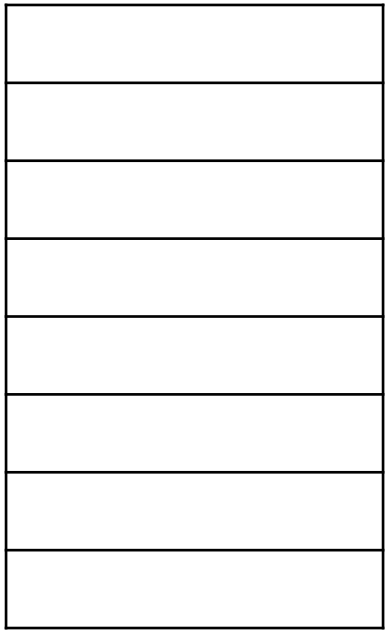
Array,　Matrix,　Heap, **Stack,** Queue,　Hash

（2）　リンク　（ポインタでつながっている）

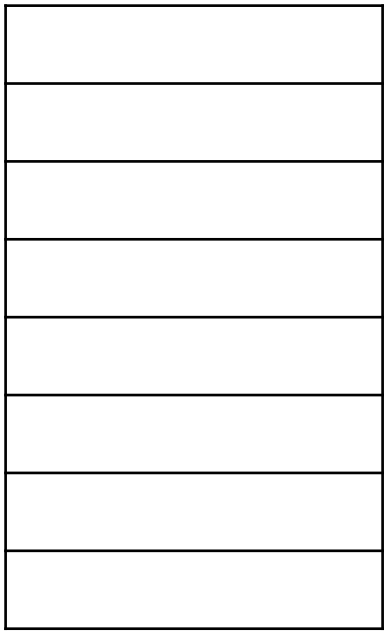
List,　Tree,　Graph

insert が簡単、**overflow しない**、flexible

Data 構造と Operation

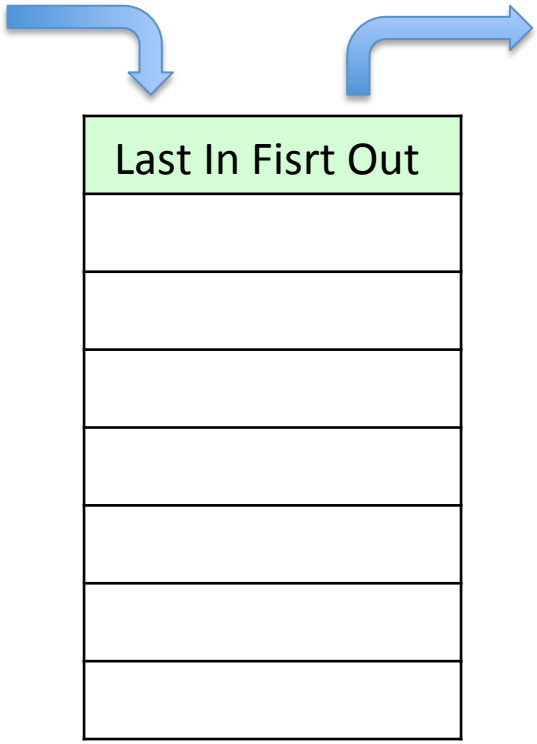


Stack

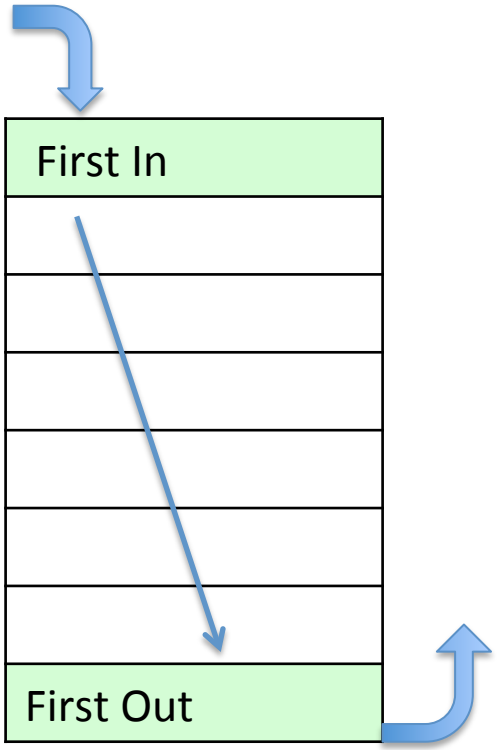


Queue

Data 構造と Operation



Stack　Push(x,s)
　　Pop(s)

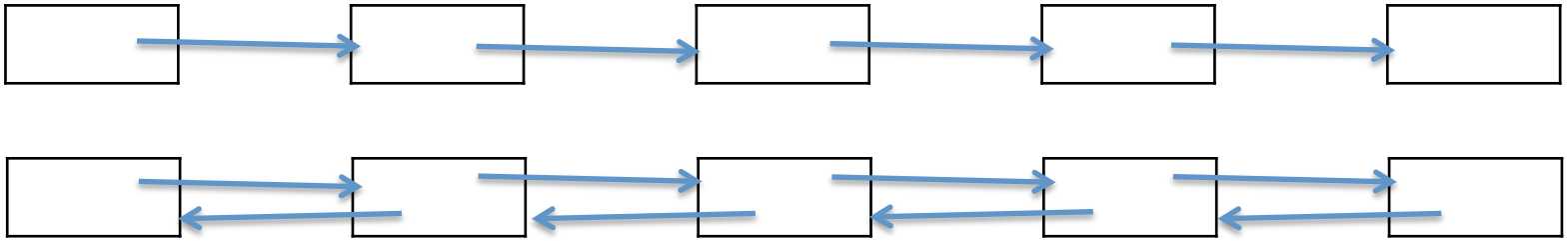


Queue Enqueue(x,q)
　　Dequeue(q)

４種類のLinked Listを使って　Dictionary を作る

1方向

双方向



オペレーション	1方向	双方向	１方向ソート	双方向ソート
Search(L,k)	O(n)	O(n)	O(n)	O(n)
Insert(L,x)	O(1)	O(1)	O(n)	O(n)
Delete(L,x)	O(n)	O(1)	O(n)	O(1)
Successor(L,x)	O(n)	O(n)	O(1)	O(1)
Predecessor(L,x)	O(n)	O(n)	O(n)	O(1)
Minimum(L)	O(n)	O(n)	O(1)	O(1)
Maximu(L)	O(n)	O(n)	O(1)	O(1)

参考　アルゴリズム設計マニュアル　by S. Skiena

Array

- 単位時間アクセス
- メモリースペースは、データしか使わない
- データがつながっている
  - **cache, prefetch が効きやすい**

（おまけ）cache　とか prefetch とか

cache は　ライン単位でデータをもってくる

1ライン 32 or 64 バイトが多い

L1キャッシュ：ページサイズ×セットアソシアティブのウェイ数が上限

4way 4 Kバイトのページなら16KB が 最大 →　色々な解決が提案

メモリアクセスのローカリティーが効く（ことが多い）

プリフェッチ：「そろそろ使いそうなデータ」を予測、
メモリからキャッシュに（勝手に）あらかじめ持ってきておく。
あたれば　200 – 300 サイクルお得。
history を使うことも多いが、時間制約はきびしい。
また、ハードウェアリソースの制限もきびしい。

Array

- 単位時間アクセス
- メモリースペースは、データしか使わない
- データがつながっている
  - cache, prefetch が効きやすい

困りそうなことは？

Array

- 単位時間アクセス
- メモリースペースは、データしか使わない
- データがつながっている
  - cache, prefetch が効きやすい

サイズがあらかじめ、わかってないと困る

→ **動的に Array のサイズを変えたい**

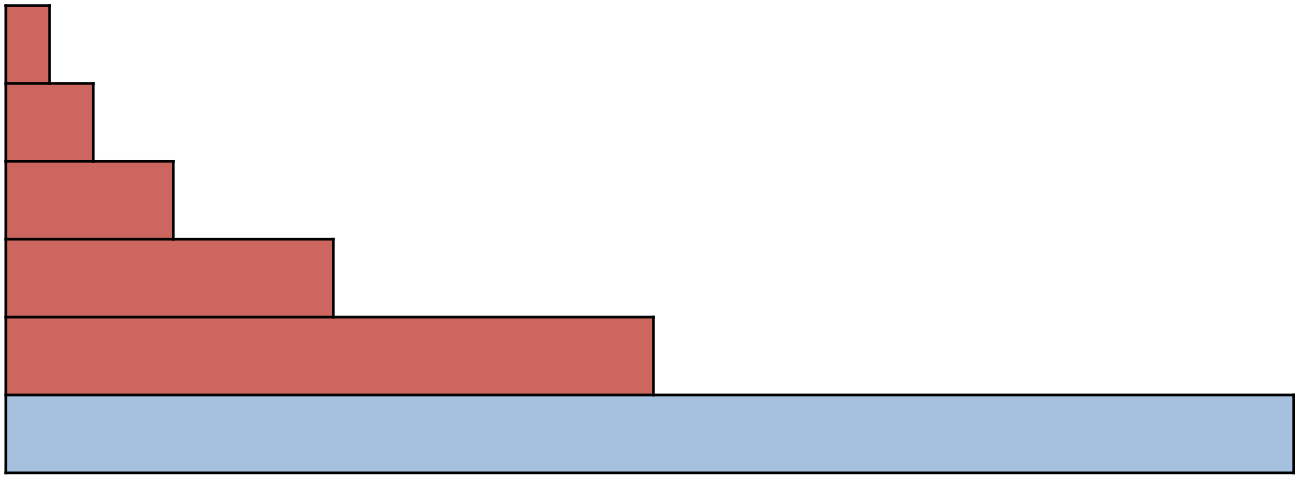
Dynamic Array

（0） 適当なサイズの配列をとる
（1） 配列のサイズが足りなくなったら
- 大きさが倍の配列を作る
- データを新しい配列にコピーする
- 旧い配列を捨てる　→（1）へ

何度もコピーされ、効率が悪そうに見えるが、
全部でたかだか O(n)しかコピーされない。
サイズの予想できないデータにはとても有効

Dynamic Array のコピーの回数

何度もコピーされて無駄そうに見えるけれど、
毎回コピーされるのは前の方の塊のみ
毎回、倍・倍に、Array のサイズを増やしていけば
コピーをされた回数は、全部でたかだか　O(n)。
たいした量ではない。



ちょっとした調整

たとえば、

- dynamic array で、倍・倍に 大きさを増やす

  （毎回、1でつ増やすでは駄目）

- random selection のサンプル集合の大きさやセパレータの幅の取り方

  （サンプル集合のソート　O(m log m)は　O(n)より小さく！）

こういうところの、パラメータの値の決め方が、けっこう大事。

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

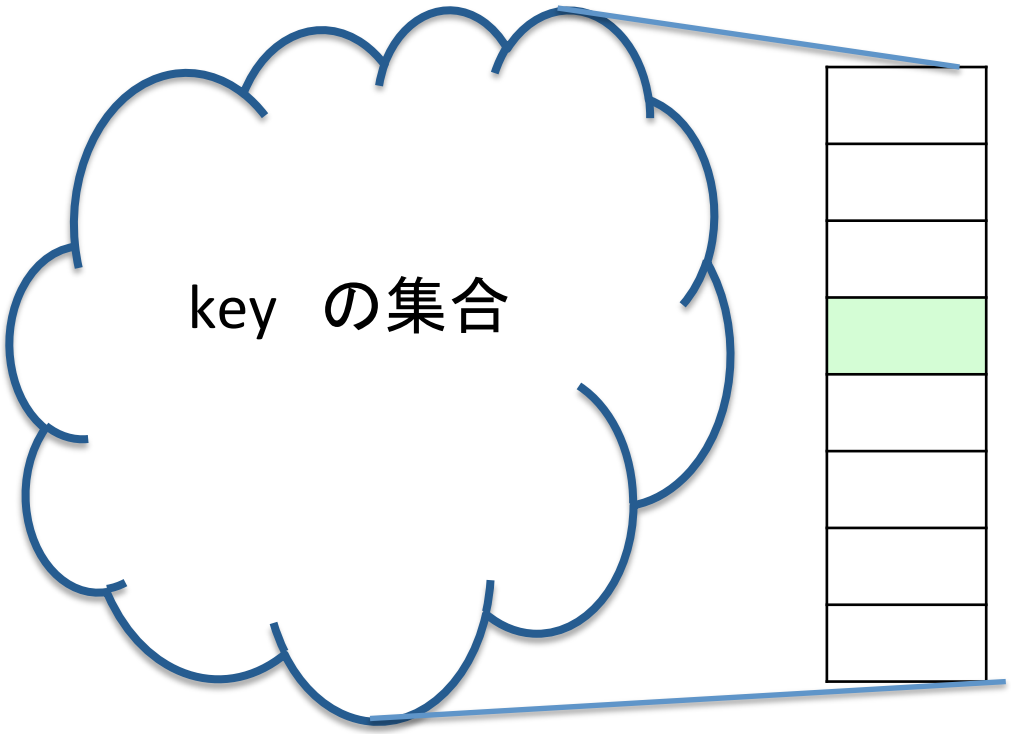
プログラムの識別子(関数名とか変数名とか）の属性（位置、型、スコープ）などを まとめた表。hash で実装することが多い。

（nm コマンドで見えることが多い）

しまった物を速く取り出したい

Array → index がわかれば O(1) で取り出せる。

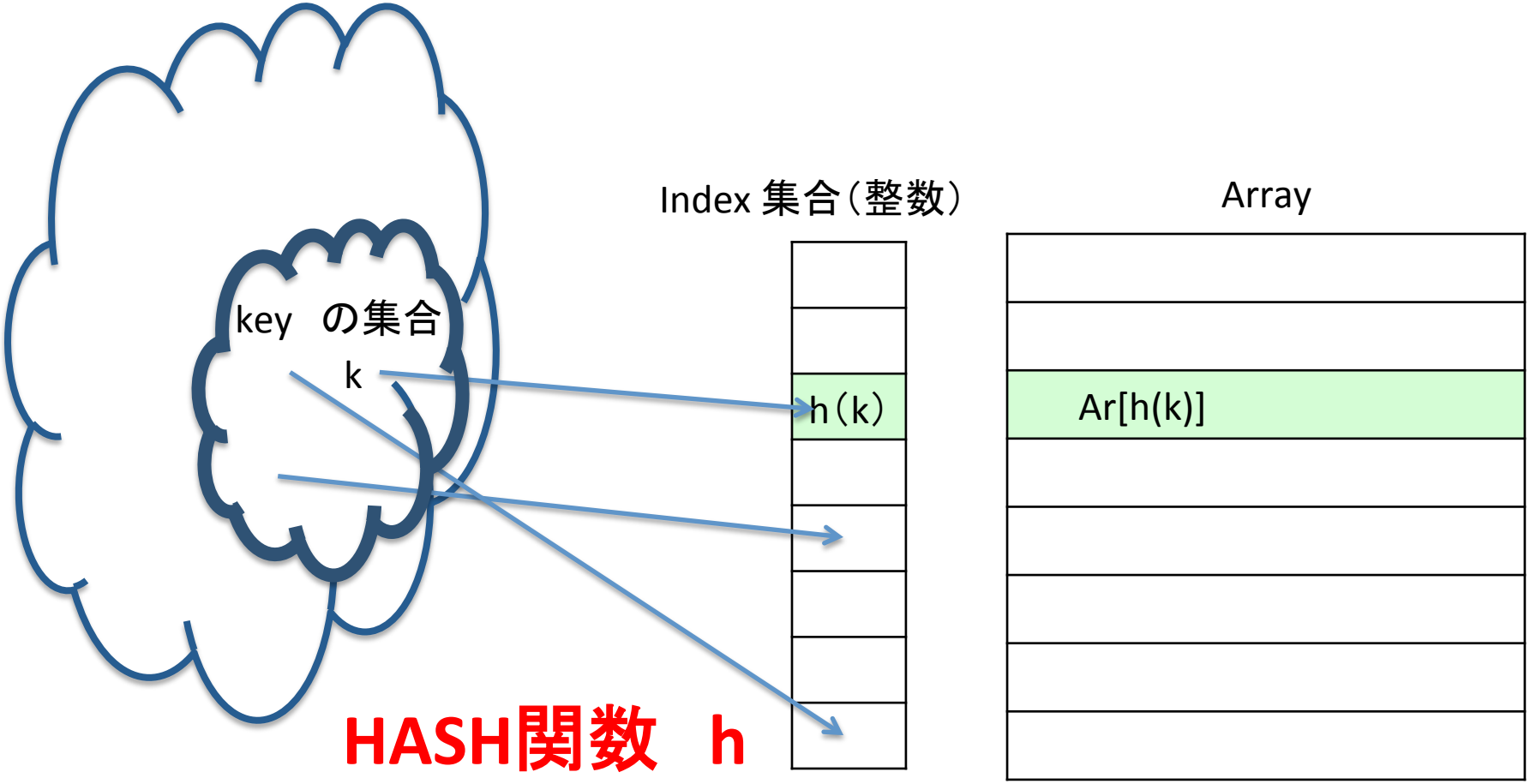
Hash Table → index の代りに KEY を使う。



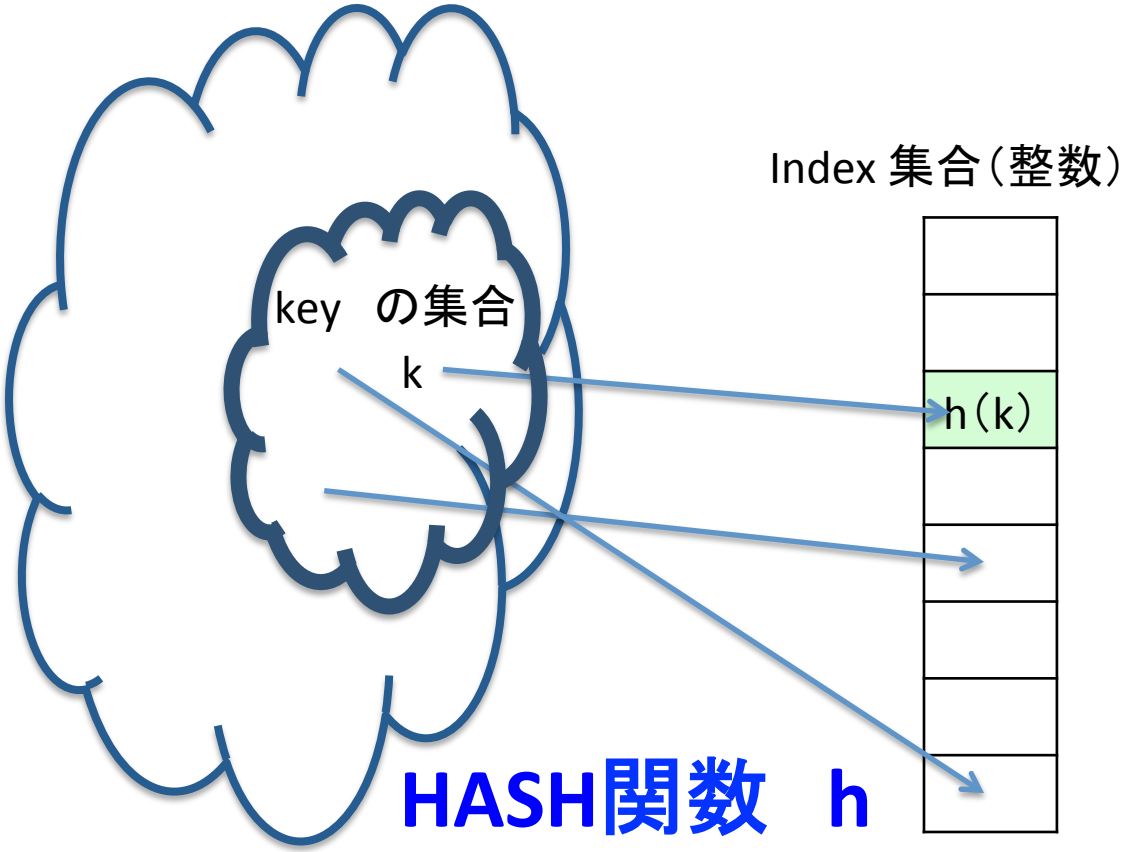
Hash

Insert(S, x)　S ← S U {x}
Delete(S,x)　S ← S - {x}
Search(S,k)　存在すれば　Ar[k]を返す

しまった物を速く取り出したい



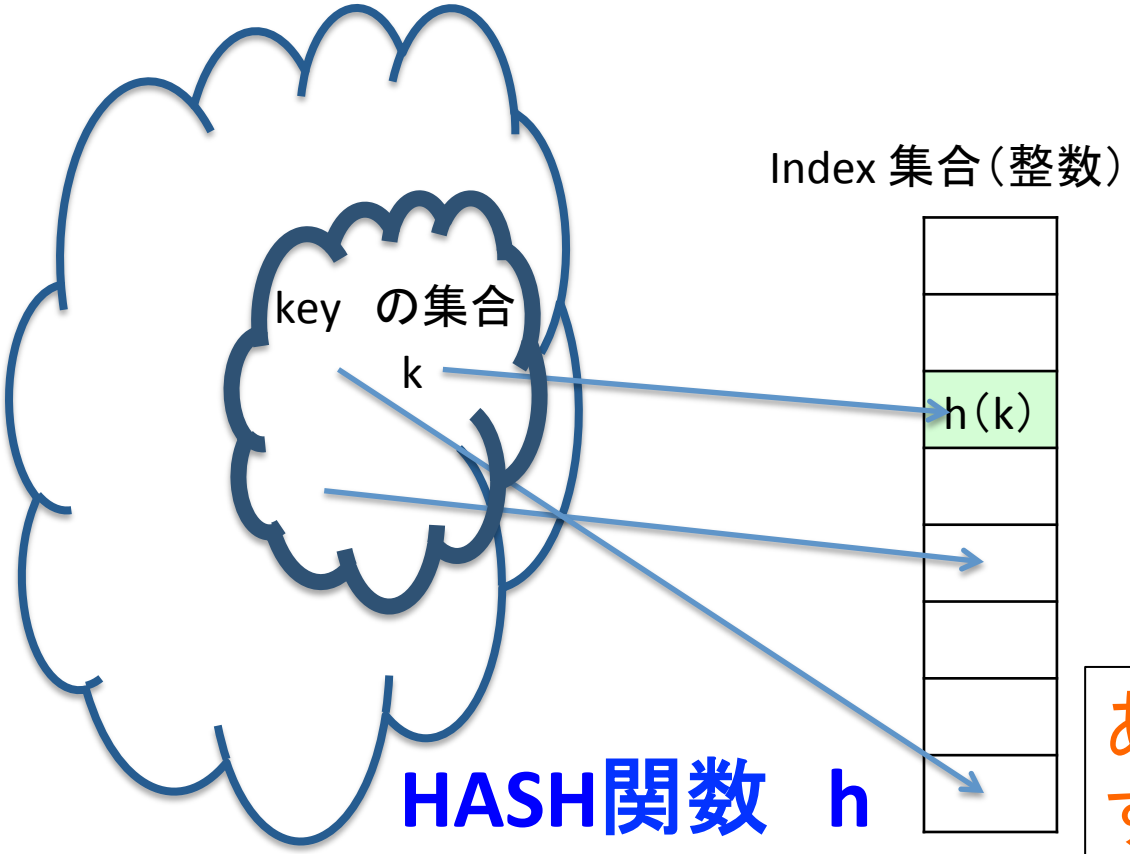
hash　関数に望まれる性質は？



違う key は
違う整数に写る

Index集合は
Array のサイズで決まる

hash　関数に望まれる性質は？

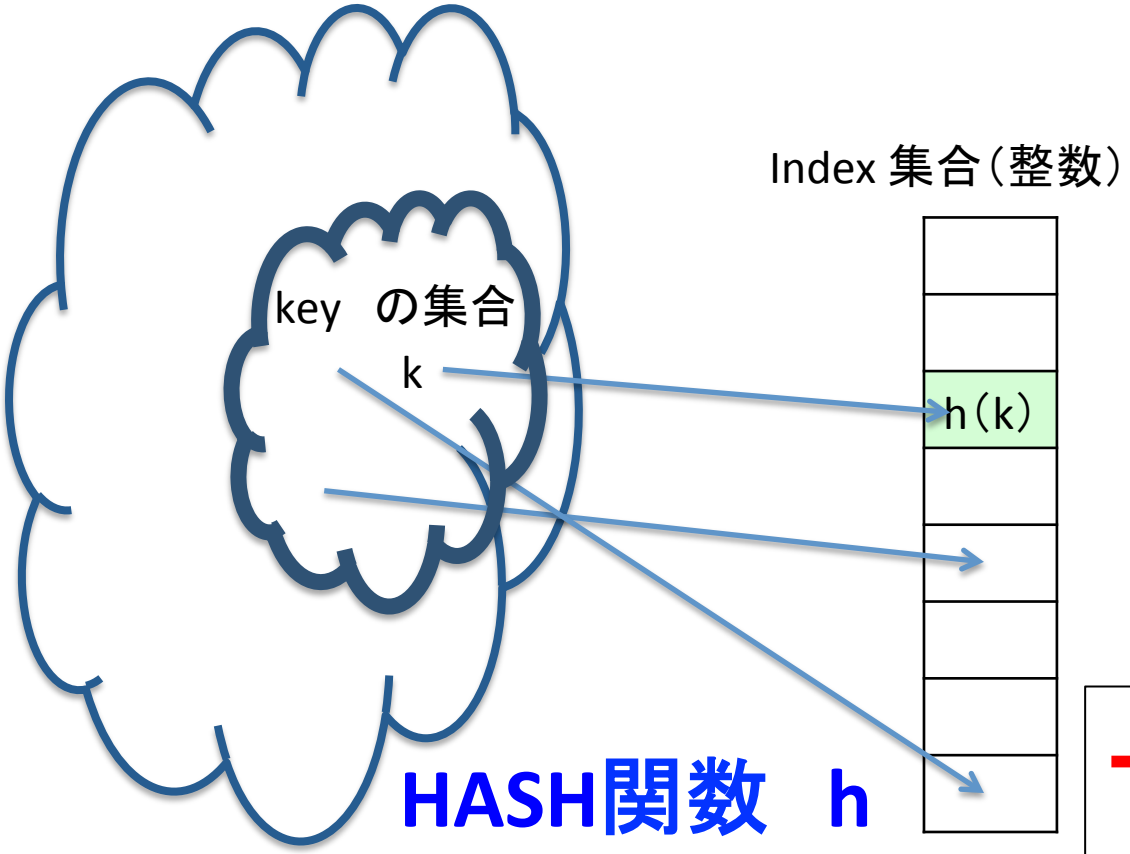


違う key は
違う整数に写る

あらかじめ　key が
すべてわかっていれば
作ることが可能

パーフェクトハッシュ

hash　関数に望まれる性質は？



違う key は
違う整数に写る

一般には無理

すべての　h（k）が
同じ　index をさす事も

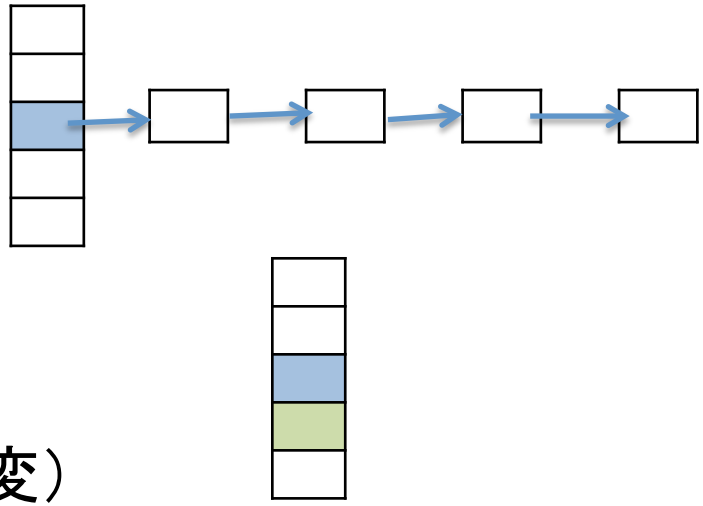
Hash：衝突の回避

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

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、レポートとプログラムのソースを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：次の週の日曜日深夜 （講評の都合上。〆切後も受付ます）
  - プログラムは（お手本として）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

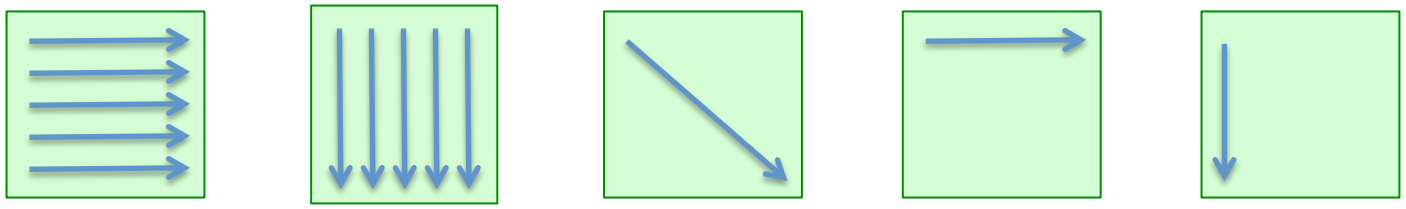
- 言語
  - 自由。ただし、一般的でない言語については、上記いずれかのOS 上にインストール可能なもの

第二回　課題

Sample programs
gettime.h gettime.c ijloop.c
test.sh plot.gp Makefile
ExArray1.java test.sh Makefile

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 の結果のグラフ化

- 言語・コンパイルオプションの比較
- メモリアクセスパターンの差
- 環境比較 （仮想PC 特有の問題など）

- アクセス回数に関する考察 （行数=n とする）
  - |ijloop| = |jiloop| = n×|１列 | = n× |１行|= n×|対角|

- 行列サイズの変更
- 複数回の試行
- 遅いルーチンの細かい時間を計る

- グラフの作成は、（ある程度）自動化すること

この課題の狙うところ

- 前回の課題から、面白そうなところを探してみる。
- 仮説をたて、あたってるか実験してみる。
- 計りたいものが実際本当に計れているか考える。

- 続：実験の自動化
  - シェルスクリプトの利用 （stdout, stderr, hear-documentなど）

- 取得したデータの処理
  - グラフを作ってみる （gnuplot)
  - ヘテロな環境からデータを集める仕組みを考える

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。