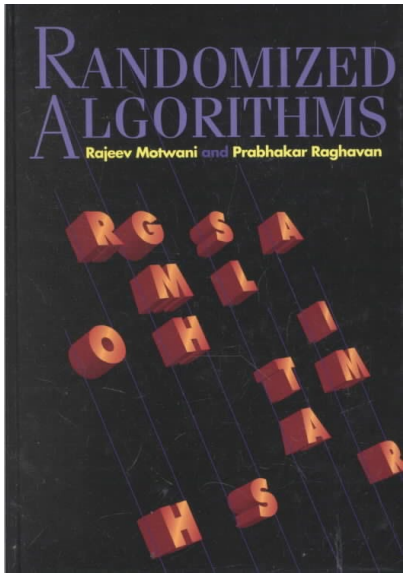
実践的アルゴリズム理論
Theory of Advanced Algorithms

10. 乱択アルゴリズム

担当:上原隆平



Theory of Advanced Algorithms
実践的アルゴリズム理論

10. Randomized Algorithms

Ryuhei Uehara

決定性アルゴリズムと乱択アルゴリズム

決定性アルゴリズム:

入力が同じならば同じ動作をする.

アルゴリズムにとって最悪の場合は入力だけで決まる.

乱択アルゴリズム:

入力が同じでも乱数によって動作が変わる.

アルゴリズムにとって最悪の場合は，乱数によって決まる.

⇒最悪の場合が発生しにくい

これが乱数導入の最大の効果

乱数を用いたアルゴリズム

モンテカルロ・アルゴリズム

間違っているかも知れないが，必ず何らかの答を返す.

ラスベガス・アルゴリズム

必ず答を返すとは限らないが，答を返す場合には必ず正しい答を返す

Deterministic and Randomized Algorithms

Deterministic Algorithm:

Its behavior is the same if input is the same.

The worst case for the algorithm is determined by input.

Randomized Algorithm:

Behavior may be different for the same input.

The worst case is determined by random numbers.

⇒worst case rarely happens

The largest effect of introducing random numbers.

Algorithms using random numbers

Monte-Carlo Algorithms

Its output may be wrong, but some output is always returned.

Las Vegas Algorithms

Although it may fail to find a solution, but if it finds any solution it is always a correct solution.

モンテカルロ・アルゴリズム

問題P30: 配列に蓄えられた多数の要素の中から，質問として与えられたデータに最も近い，あるいは近そうな要素を求めよ.

・n個のデータが配列a[]に蓄えられているとする.
・配列の要素がソートされていれば，2分探索によりO(log n)回の比較で十分.
・ソートされていなければ，質問データqに最も近い要素を見つけるにはn回の比較が必要になる.

最も近い要素を求めるには多数の比較が必要になる.
では，近そうな要素を求めるだけなら高速化可能か？
　　乱数を使ったアルゴリズム

Monte Carlo Algorithms

Problem P30: Given a query data q, find an element closest or seemingly closest to q among those elements stored in an array.

- Assume that n data are stored in an array a[].
- If they are sorted in the array, O(log n) comparisons suffice by binary search.
- If they are not sorted, n comparisons are required to find one closest to a query data q.

A number of comparisons are required to find the closest element.
Then, is any speedup possible to find only a seemingly closest one?

randomized algorithm

問題P30': 配列に蓄えられた多数の要素の中から，質問として与えられたデータとの近さにおいて全体の20%以内の任意の要素を求めよ.

アルゴリズムP30-A0:
入力：配列a[]に蓄えられたn個の要素と質問データq
配列a[]の先頭から(0.8n+1)番目までの要素と比較して，質問データに最も近い要素を答として返す.

上記のアルゴリズムでは全体の80%を超える要素を調べるから，その中で最も質問qに近い要素は問題の要求を満たす.
しかし，計算時間が0.8nに比例する．つまり，O(n).
もっと高速化できないか？

Problem P30': Among a number of data stored in an array, find any element that is within 20% in the order of closeness to a data given as a query.

Algorithm P30-A0:
Input : n elements stored in an array a[] and a query data q
Comparing q with the first through (0.8n+1)-th data in the array a[], return an element closest to the query data q as a solution.

Since the above algorithm examines more than 80% of the elements, the element closest to q among them satisfies the requirement of the problem.
However, it requires time proportional to 0.8n, that is O(n).
Is any speedup possible?

アルゴリズムP30-A1:
入力：配列a[]に蓄えられたn個の要素と質問データq
乱数によって配列a[]のランダムな要素を20個取り出し，
その中で質問データに最も近い要素を答として返す．

このアルゴリズムは必ず答を返すが，答が正しいとは限らない
　よって，モンテカルロ・アルゴリズム

このアルゴリズムの答が間違っている確率を解析しよう．
　求めたいのは，質問qとの近さにおいて全体の20%以内の任意の要素．　そのような要素の集合をA(q, 0.2)としよう．

一つの要素が集合A(q, 0.2)に入っている確率は1/5．
　逆に，集合A(q, 0.2)に入っていない（不正解の）確率は4/5．
アルゴリズムの答が不正解であるのは，不正解が20回続くとき．
その確率は，$(4/5)^{20}=0.0115292...$
つまり，100回に1回ぐらいしか間違えない！

Algorithm P30-A1:
Input: n elements stored in an array a[] and a query data q
Choose 20 elements randomly from the array a[] using random numbers and return the one closest to the query as a solution.

Although this algorithm always returns a solution, but it is not always correct. => Monte Carlo algorithm

Analysis of failure probability of the algorithm
What is required is an element within 20% in the closeness to q.
Let A(q, 0.2) be such a set of elements.

Probability that an element is in the set A(q, 0.2) is 1/5.
(failure) Probability that an element is not in the set is 4/5.
A solution of the algorithm is not correct when it fails to find a correct solution 20 times consecutively, with probability $(4/5)^{20}=0.0115292...$
That is, it fails only once in 100 trials.

演習問題E13-1: 間違いの確率を0.01以下にするには何個の要素をランダムに取り出すようにすればよいか？

演習問題E13-2: 100万個の要素の中から質問データとの近さにおいて全体の10%以内に入っている要素を求めたい.
(1) 決定性のアルゴリズムを用いて，必ず正しい答を求めるには何回の比較が必要か？
(2) 乱択アルゴリズムを用いて，間違い確率が0.01以下で答を求めるには何回の比較が必要か？

Exercise E13-1: How many elements should be taken randomly to make the failure probability less than 0.01?

Exercise E13-2: We want to find an element that is within 10% in the closeness to a query data among a million elements.
(1) How many comparisons are required to have a correct answer using a deterministic algorithm?
(2) How many comparisons are needed to find a solution with failure probability at most 0.01 using a randomized algorithm?

形式的な議論

データの集合Sが配列に蓄えられているとき，質問データqに十分近いSの要素a[i]を高い確率で見つけたい．

配列要素a[i]が質問qに十分近いという概念の形式的表現：

条件1：「ある小さな値εに対して，a[i]よりqに近い要素は高々ε|S|個しかない」

高い確率とは，

条件2：「ある小さな値δに対して，条件を満たす要素が見つかる確率が$1-\delta$以上」

ランダムに選んだ要素が条件1を満たさない確率は$(1-\varepsilon)$．
選んだk個の要素がすべて条件1を満たさない確率がδ未満となるのは，$(1-\varepsilon)^k<\delta$となるとき．よって，
$k=\log \delta/\log (1-\varepsilon)$個以上の要素をランダムに取り出せば
質問に十分近い要素を高い確率で取り出せる．

Formal argument

When a set S of data are stored in an array, we want to find an element a[i] of S that is sufficiently close to a query data q with high probability.

A formal expression of a notion that an array element a[i] is sufficiently close to a query q:

- **Condition 1**: "For a small value ε, at most ε|S| elements of S are closer to q than a[i]."

High probability:

- **Condition 2**: "**For a small value** δ, the probability that an element satisfying the condition is at least $1-\delta$."

The probability that a randomly chosen element does not satisfy the condition 1 is $1-\varepsilon$. The probability that none of k chosen ones satisfies the condition 1 is less than δwhen $(1-\varepsilon)^k<\delta$.

➔If we choose randomly $k=\log \delta/\log(1-\varepsilon)$ elements, we can find an element sufficiently close to a query with high probability.

ラスベガス・アルゴリズム

乱数を用いたアルゴリズム

モンテカルロ・アルゴリズム

間違っているかも知れないが，必ず何らかの答を返す．

ラスベガス・アルゴリズム

必ず答を返すとは限らないが，答を返す場合には必ず正しい答を返す

問題P31: 格子状に区切られた配線領域に配置された端子と，配線要求が与えられたとき，ブロック間を通る配線数の最大値が最小になるように配線を実現せよ．

配線は2端子を結ぶものに限定し，それぞれを1回の折れ曲がりだけで実現するものとする．
ブロック間の壁を通る配線数を混雑度と呼ぶ．
最大混雑度を最小にするように配線経路を決定したい．

Las Vegas Algorithms

Algorithms using random numbers

Monte-Carlo Algorithms

Its output may be wrong, but some output is always returned.

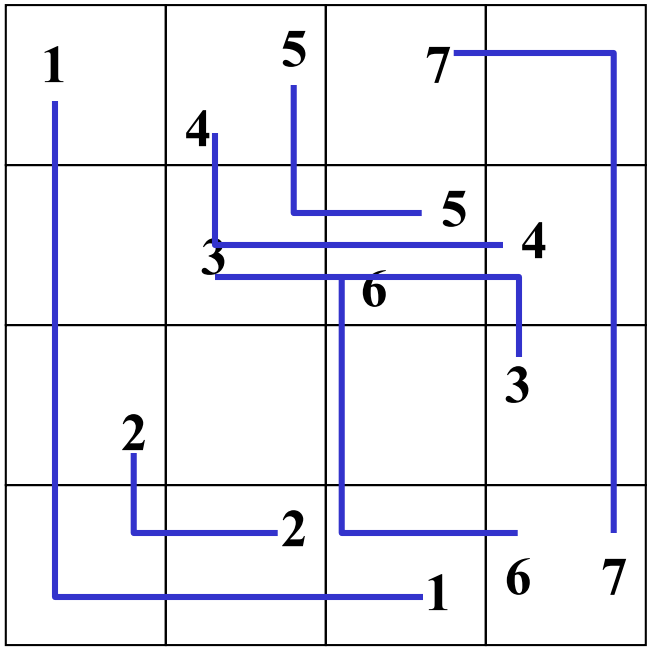
Las Vegas Algorithms

Although it may fail to find a solution, but if it finds any solution it is always a correct solution.

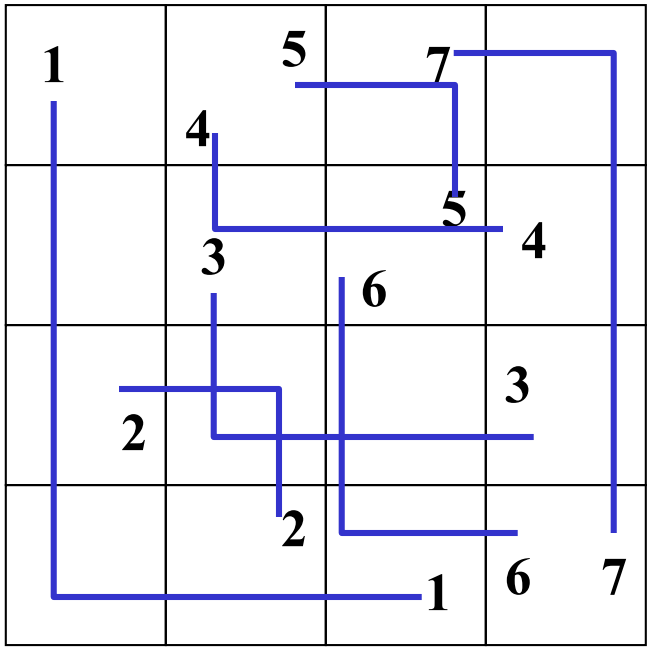
Problem P31: Given terminals arranged on the gridded wiring region and wiring requirements, find a wiring pattern so that the maximum number of wires passing between two blocks is minimum.

We restrict ourselves only to two-terminal nets and each net is realized using at most one bend.
The number of wires between two blocks is called the congestion.
Want to find wiring routes with minimum largest congestion.

例題:



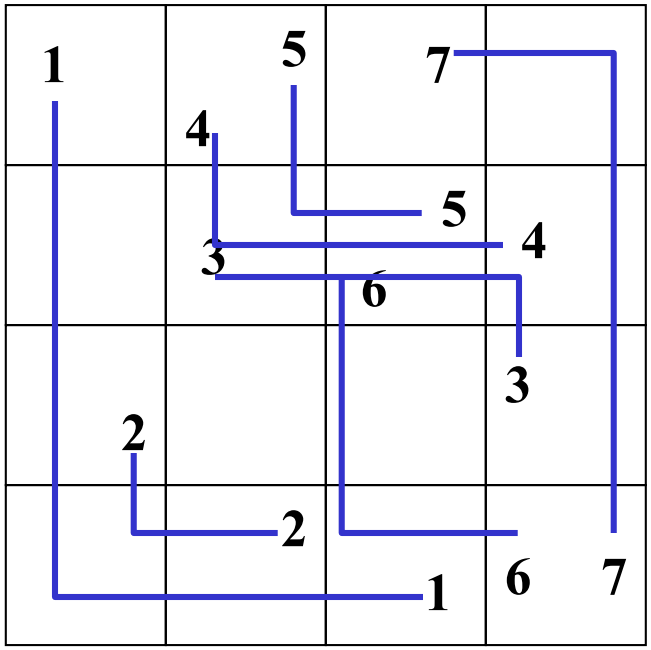
最大混雑度=3



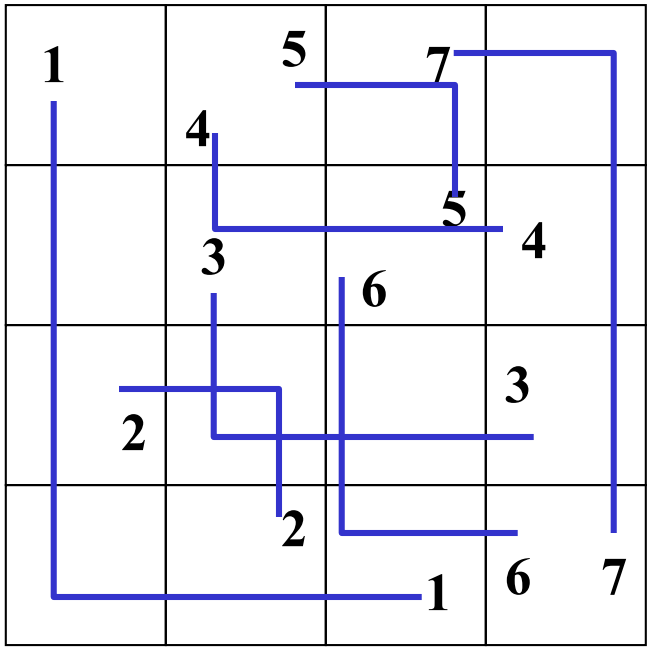
最大混雑度=1

どの配線についても2通りの配線経路あるから，全部で
2^n通りの配線パターンがある.
単純な方法では2^n以上の時間が必要.

Example:



Max congestion=3



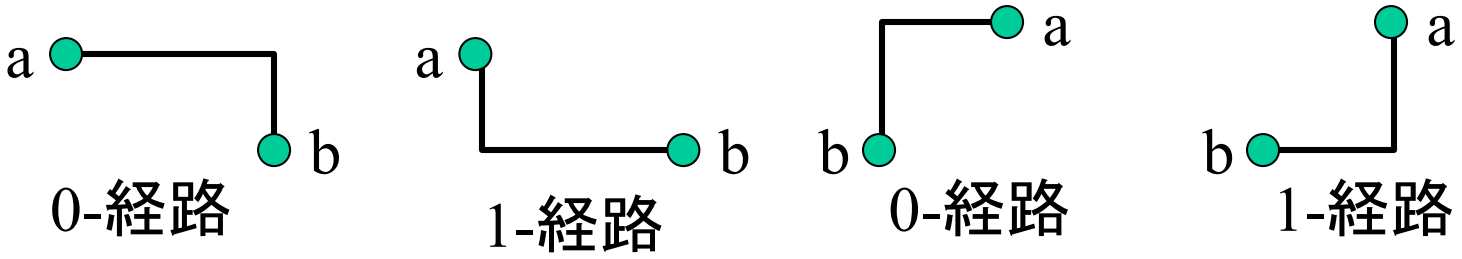
Max congestion=1

Since there are 2 routes for each net, there are 2^n different routes in total.

So, a naive algorithm takes at least 2^n time.

２点a, b間の２通りの配線経路

0-経路=aから水平線を延ばし，その後で垂直線でbと結ぶ経路
1-経路=aから垂直線を延ばし，その後で水平線でbと結ぶ経路



端子a, bの位置関係とは独立に0-経路と1-経路を区別する.

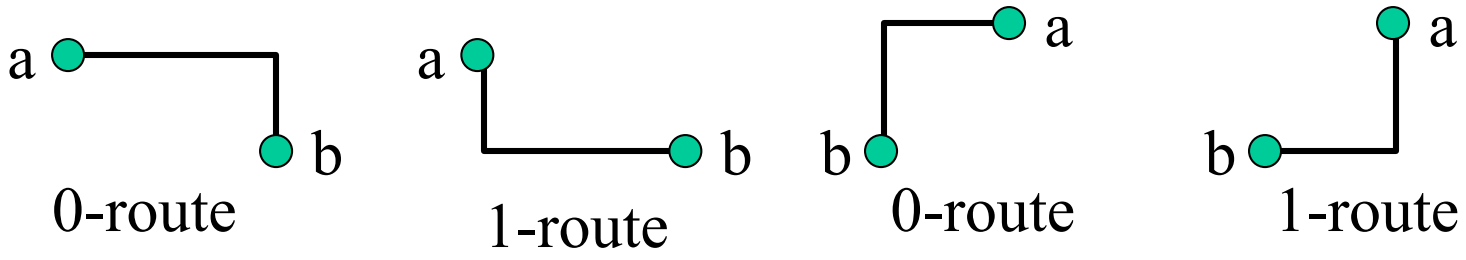
もし両端子が同じy座標をもつか，同じx座標をもつなら，
0-経路と1-経路は同じ経路となる.

また，各端子対に対して0-1の乱数を生成すると，どちらかの
経路をランダムに決めることができる.

Two different routes between two points a and b

0-route=horizontal line incident to a and a vertical line to b

1-route= vertical line incident to a and a horizontal line to b



Distinction between 0- and 1-routes is independent of relative location of terminals a and b.

If the two terminals have the same y- or x-coordinate, then 0-route is identical to 1-route.

Generating a 0-1 random number for each terminal pair, we can determine either route randomly.

アルゴリズムP31-A0: 経路決定アルゴリズム1:

入力:n本の配線を指定する2端子対の集合と整数d.
for i=1 to n{
　0か1の値をとる乱数rを生成する;
　i番目の配線経路をr-経路と定める;
}
それぞれの壁を通過する配線の本数の最大値maxを求める;
if max ≦d then この経路パターンを出力して終了;
else 失敗を報告して終了;

このアルゴリズムは，必ず答を返すとは限らないが，答を返すときには，その答は最大混雑度がd以内のものである.
問題は，正解が返される確率が十分に高いかどうか.

先の例では，最大混雑度1以内の解が2通りしかない．全部で2^7=128通りの配線パターンがあるから，正解が得られる確率は1/64でしかない．つまり，64回に1回しか正解が得られない.

Algorithm P31-A0: Route Finding Algorithm 1:

Input: A set of 2-terminal nets and an integer d.
for i=1 to n{
  generate a random number r taking 0 or 1;
  determine the i-th route as r-route;
}
find the maximum number max of wires passing through block boundary by checking all block pairs;
if max $\leqq$ d then stop after reporting this routing pattern;
else stop after reporting the failure;

Although this algorithm does not always report a solution, if it does, the solution is always the one with maximum congestion at most d. The problem is how high the probability of getting a solution is.

In the previous example, there are only two patterns with max congestion 1. Since there are 2^7=128 patterns, the probability to obtain a correct solution is only 1/64. That is, we find a correct solution once for 64 trials.

確率的丸めに基づく乱択アルゴリズム

i番目の配線の経路を2つの論理変数を用いて表現する:

$x_{i0}=1$ 配線iの経路が0-経路のとき,

$x_{i0}=0$ それ以外のとき(1-経路のとき).

$x_{i1}=1$ 配線iの経路が1-経路のとき,

$x_{i1}=0$ それ以外のとき(0-経路のとき).

各ブロック間の壁w_bについて, その0-経路がw_bを通過するような配線の番号の集合をT_{b0}とする. T_{b1}も同様に定める.

このとき, どの配線についてもどちらかの経路を選択するから,

$$\mathbf{x_{i0}+x_{i1}=1} \quad i = 1, 2, \ldots, n.$$

各ブロック間の壁w_bを通過する配線の本数がd以内であるという条件は,

$$\sum_{j\in Tb0} \mathbf{x_{j0}} + \sum_{j\in Tb1} \mathbf{x_{j1}} \leqq \mathbf{d}, \; j=1, 2, \ldots, m$$

と表現できる. ただし, mはブロック間の壁の個数である.

これは**整数線形計画問題**である.

Randomized algorithm based on probabilistic rounding

The i-th route is specified using two logic variables.

$x_{i0}=1$ if a route for a net i is 0-route,

$x_{i0}=0$ otherwise (when it is a 1-route).

$x_{i1}=1$ if a route for a net i is 2-route,

$x_{i1}=0$ otherwise (when it is a 0-route).

For each block boundary w_b, a set of wire numbers whose 0-route passes through it is denoted by T_{b0}. T_{b1} is also determined.

Then, we have to choose either route for each net, we have

$$\mathbf{x_{i0}+x_{i1}=1} \quad i = 1, 2, \ldots, n.$$

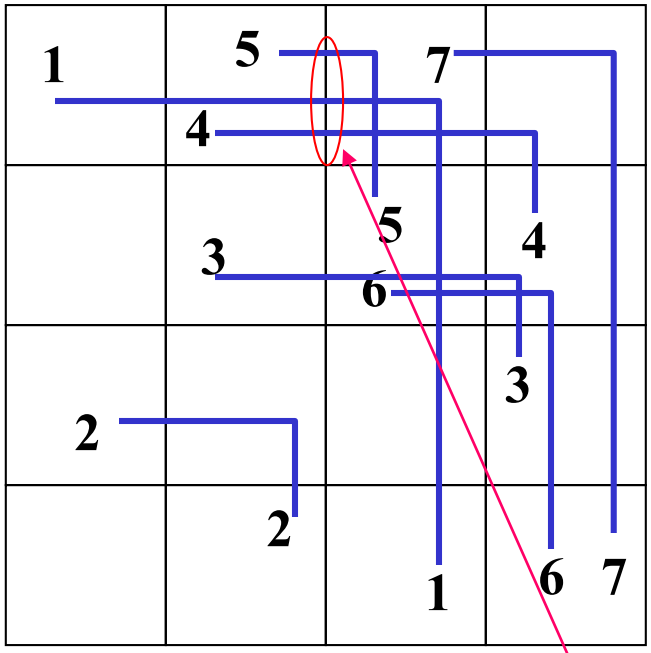
The condition that at most d wires pass through w_b is expressed as

$$\sum_{j \in Tb0} x_{j0} + \sum_{j \in Tb1} x_{j1} \leqq \mathbf{d}, \; j=1, 2, \ldots, m,$$

where m is the number of block boundaries.

This is a **Integer Linear Program.**

例題:



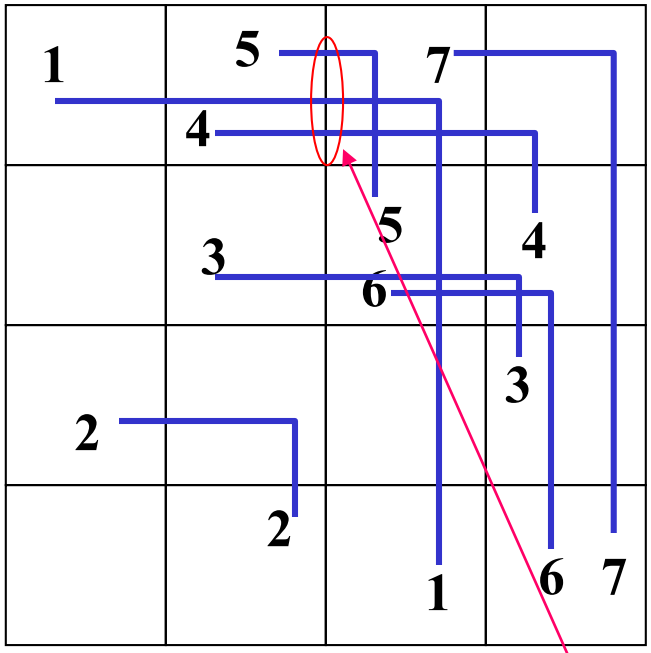
各端子対に対する
0-経路

$x_{10}+x_{40}+x_{50} \leqq d,$
$x_{41}+x_{51} \leqq d,$
$x_{40}+x_{70} \leqq d,$
$x_{10}+x_{50}+x_{71} \leqq d,$
$x_{30}+x_{41}+x_{51} \leqq d,$
$x_{30}+x_{41}+x_{60} \leqq d,$
$x_{10}+x_{61}+x_{71} \leqq d,$
$x_{11}+x_{21} \leqq d,$
$x_{30}+x_{60}+x_{70} \leqq d,$
$x_{61}+x_{71} \leqq d,$
$x_{11}+x_{21} \leqq d.$

この壁を通過するのは，配線1の0-経路，
配線4の0-経路，配線5の0-経路だから，
対応する不等式は

$$x_{10}+x_{40}+x_{50} \leqq d$$

Example:



0-route for each terminal pair

$x_{10}+x_{40}+x_{50} \leqq d,$
$x_{41}+x_{51} \leqq d,$
$x_{40}+x_{70} \leqq d,$
$x_{10}+x_{50}+x_{71} \leqq d,$
$x_{30}+x_{41}+x_{51} \leqq d,$
$x_{30}+x_{41}+x_{60} \leqq d,$
$x_{10}+x_{61}+x_{71} \leqq d,$
$x_{11}+x_{21} \leqq d,$
$x_{30}+x_{60}+x_{70} \leqq d,$
$x_{61}+x_{71} \leqq d,$
$x_{11}+x_{21} \leqq d.$

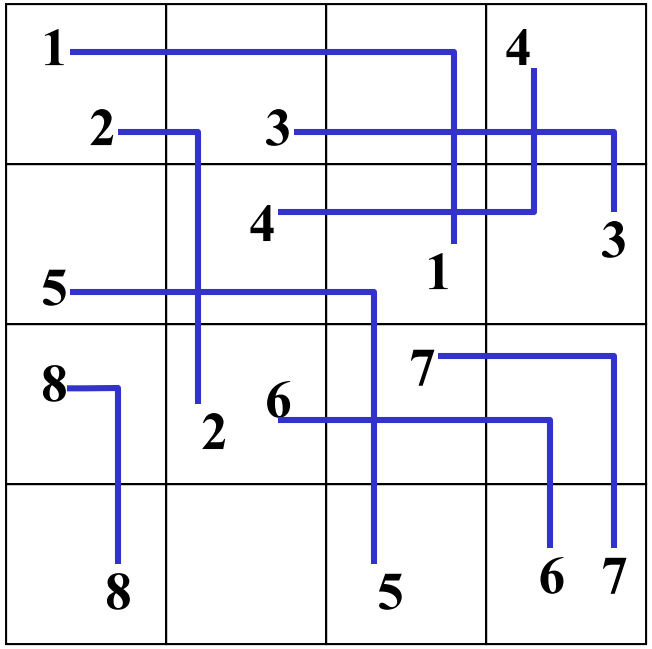
Since the 0-route for net 1, 0-route for net 4, and 0-route for net 5 pass through this block boundary, the corresponding inequality is $x_{10}+x_{40}+x_{50} \leqq d$

整数計画問題はNP完全だから，効率よい解決は難しい．

確率的丸めの考え方

(1)整数制約を無視して，整数計画問題を線形計画問題として解く．

(2)得られた解が整数解でなければ，近くの整数解で近似．
このとき，$0 \leqq p \leqq 1$の範囲の値pを確率pで1に丸める．
p=0.9なら1になる確率は高いが，p=0.2なら確率は低い．



この例題を線形計画問題として解くと

$x_{10}=0$, $x_{11}=1$,
$x_{20}=1$, $x_{21}=0$,
$x_{30}=1$, $x_{31}=0$,
$x_{40}=0.5$, $x_{41}=0.5$,
$x_{50}=0$, $x_{51}=1$,
$x_{60}=0.5$, $x_{61}=0.5$,
$x_{70}=1$, $x_{71}=0$,
$x_{80}=1$, $x_{81}=0$

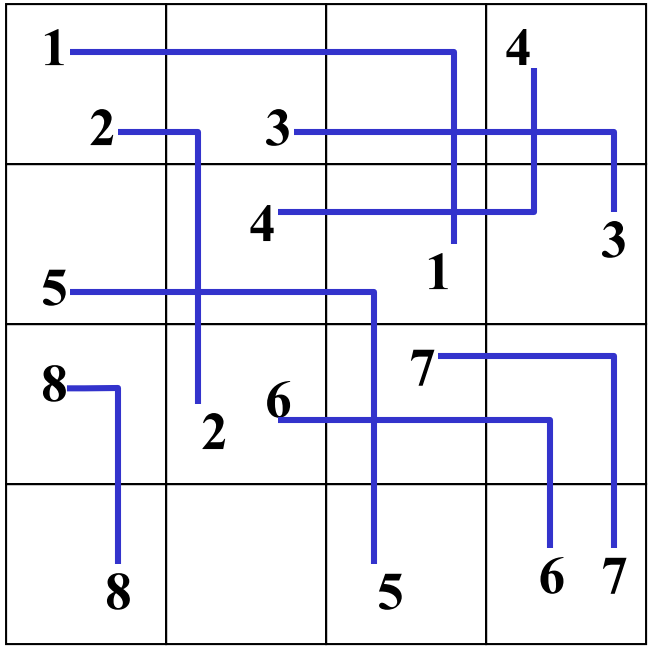
最後にx_{40}, x_{60}の値を乱数で決める．

Integer Linear Program is NP-complete. So, no efficient algorithm.

Introducing an idea of probabilistic rounding

(1) Solve the integer linear program as a linear program neglecting integer constraints.

(2) If the solution obtained is not an integer solution, then round it into an integer solution nearby by rounding a value p, $0 \leqq p \leqq 1$ into 1 with probability p.



Solving this problem as a linear program

$x_{10}=0$, $x_{11}=1$,
$x_{20}=1$, $x_{21}=0$,
$x_{30}=1$, $x_{31}=0$,
$x_{40}=0.5$, $x_{41}=0.5$,
$x_{50}=0$, $x_{51}=1$,
$x_{60}=0.5$, $x_{61}=0.5$,
$x_{70}=1$, $x_{71}=0$,
$x_{80}=1$, $x_{81}=0$

Finally, we determine x_{40} and x_{60} by random numbers.

定理13-1: εを$0 < \varepsilon < 1$の範囲にある任意の実数とする．n本の配線からなる配線経路決定問題が与えられたとき，対応する整数線形計画問題を線形計画問題に緩和した問題の解によって与えられる目的関数（最大混雑度）の値を$d\hat{}$とし，これに確率的丸めの考え方で求めた配線経路パターンの最大混雑度をd^pとする．さらに，最適解の最大混雑度をd^*で表す．
このとき，確率$1-\varepsilon$で次の関係が成り立つ．

$$d^p \leqq d^{(1+\delta)} \leqq d^*(1+\delta)$$

ただし，δは，混雑度が$d\hat{}$の$1+\delta$倍より大きくなる確率が$\varepsilon/2n$以下になる，すなわち，

$$\Pr\{d > (1+\delta)\, d\hat{}\} < \varepsilon/2n$$

を満たすような値である．

証明は省略．

Theorem 13-1: Let ε be an arbitrary number s.t. $0 < \varepsilon < 1$.
Given a problem of determining routes of n nets, let $d^{\wedge}$ be the value of an objective function of the linear program relaxed from the corresponding integer linear program, and d^{p} be the maximum congestion of a wiring pattern using the probabilistic rounding. Further, let d^{*} be the maximum congestion of an optimal solution. Then, with probability $1 - \varepsilon$ we have

$$d^{p} \leqq d^{(1+\delta)} \leqq d^{*}(1+\delta) ,$$

where δ is a value such that the probability that the congestion becomes $1+\delta$ times larger than $d^{\wedge}$ is below $\varepsilon/2n$, that is,

$$\Pr\{d > (1+\delta)\, d^{\wedge}\} < \varepsilon/2n.$$

Proof is omitted.

ランダム・サンプリング

与えられたデータ集合の中からランダムに一定個数のデータを取り出し，そこでの解を用いて元の問題を解く.

問題P32: 与えられたn個のデータの中央値を求めよ.

(1)n個のデータをソートして中央の値を答える.... O(n log n)時間
(2)クィックソートを変形したアルゴリズムを用いる
　最善はO(n)時間．最悪は$O(n^2)$時間．平均はO(n log n)時間.
(3)奇数個の要素からなるグループに分けて，各グループの中央値たちの中央値を用いる方法.....最悪でもO(n)時間.
漸近解析の立場からは(3)の方法が最良であるが，実際には時間がかかる.
もっと実用的な方法はないか？
　　ランダムサンプリングの考え方

Random Sampling

Extract a fixed number of data out of a given data set and solve an original problem using a solution to those samples.

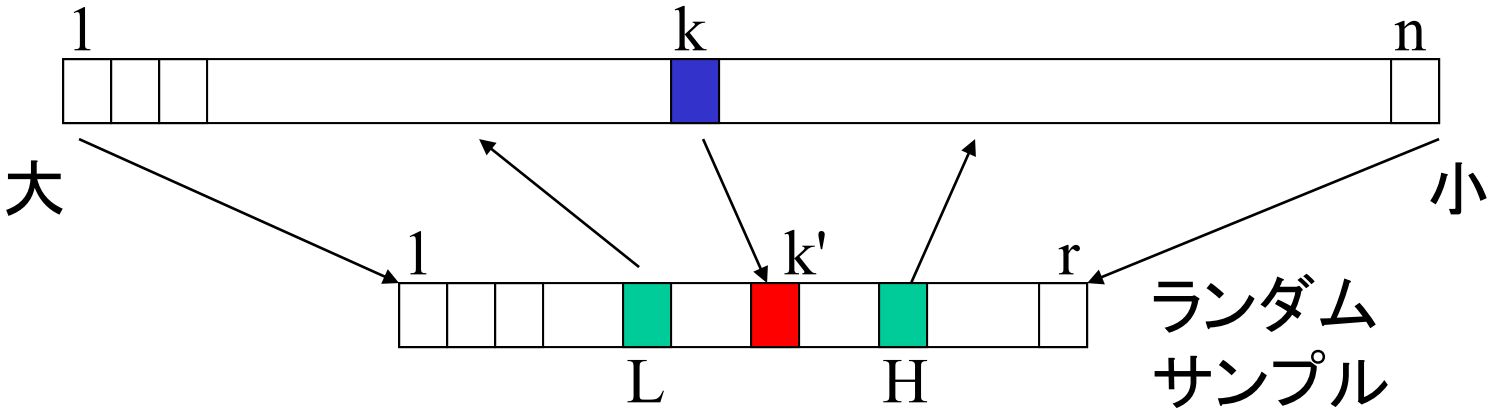
Problem P32: Find a median of n given data.

(1)Sort n data and output the median..... O(n log n) time
(2)Use a revised version of Quicksort.
Best case: O(n) time, worst case $O(n^2)$ time, average O(n log n).
(3) Partition them into groups of odd number of elements and use the median of their medians....worst case O(n) time.
In the asymptotic analysis the algorithm (3) is the best, but it takes more time in practice.
Is there any more practical algorithm?

How about randomized algorithm?

大きい方からk番目の要素を求める問題

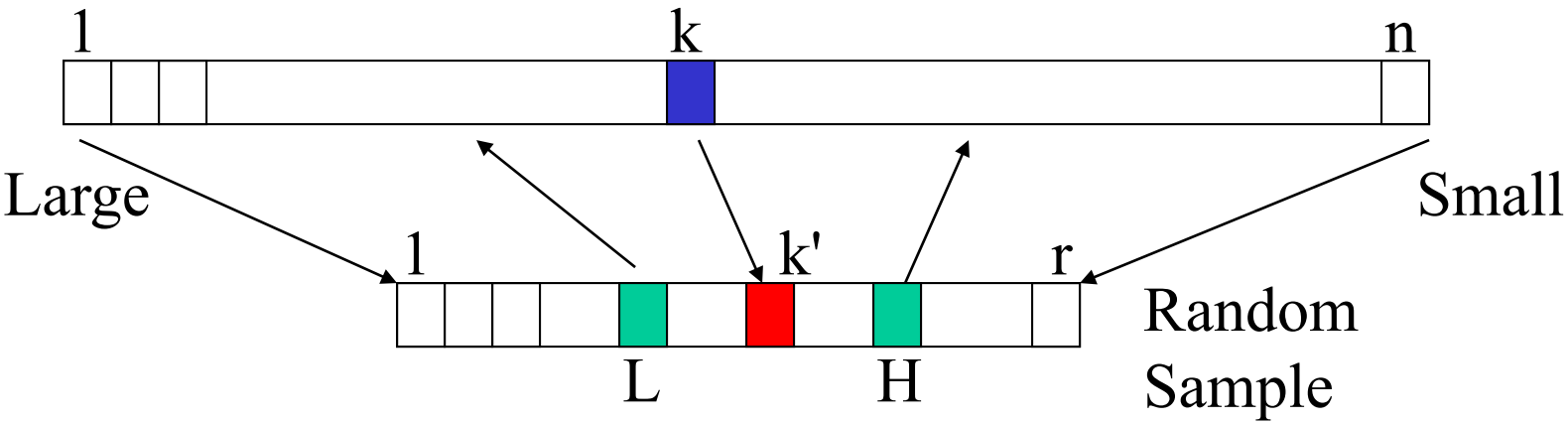
n個のデータS ⇒ サイズrのランダムサンプルR,
Rの中で元のデータ集合Sにおけるk番目に大きい要素に対応するものを求める.
その他に，目的の値より確実に小さいと思われる値Lと，確実に大きいと思われる値Hを求める



LとHに対応する値を元の集合Sにおいて求めると，
求めるk番目の要素の候補を絞り込むことができる.
また，r<<nならば，r個のデータをO(n)時間でソート可能.

Problem of finding the k-th largest element

set S of n data $\Rightarrow$ random sample R of size r,
Find an element in R corresponding to the k-th largest one in S.
In addition, find a value L which is seemingly smaller than the objective value and a value H larger than it.



By finding elements corresponding to L and H in the original set S, we can narrow candidates for our target value.
If r<<n, we can sort r data in O(n) time.

アルゴリズムP32-A0: (レイジーセレクト)
入力：n個のデータからなる集合Sと整数k, $1 < k < n$.
1. SからサイズrのランダムサンプルRを求める.
2. Rの要素をソートする.
3. Rにおいてkr/n+t番目に大きい要素Lを求める.
4. Rにおいてkr/n-t番目に大きい要素Hを求める.
5. Sを走査して，$S1=\{x|x<L\}$, $S2=\{x|L \leqq x \leqq H\}$,$S3=\{x|x>H\}$に分割する.
6. if $|S3| \geqq k$または$k > |S3|+|S2|$ then 失敗.
7. S2をソートし，S2においてk-|S3|番目に大きい要素を報告して終了.

ランダムサンプルのサイズrをどのように選ぶかが問題.
　r:大きい⇒推定は正確だが，時間がかかる.
　r:小さい⇒効率的だが，推定は不正確.
LとHの決め方も問題.
　LとHの幅tをどのように決めるか？

Algorithm P32-A0: (Lazyselect)

Input : A set S of n data and an integer k, $1 < k < n$.

1.Take a random sample R of size r from S.

2.Sort the elements of R.

3.Find an element L that is the (kr/n+t)-th largest element in R.

4. Find an element H that is the (kr/n-t)-th largest element in R.

5.Scanning S, partition S into $S1=\{x|x<L\}$, $S2=\{x|L \leqq x \leqq H\}$, and $S3=\{x|x>H\}$.

6.if $|S3| \geqq k$ or $k > |S3|+|S2|$ then failure.

7.Sort the set S2 and report the (k-|S3|)-th largest element in S2.

Problem is how to choose the size r of a random sample.

r: larger⇒estimation is exact, but more time.

r: smaller⇒efficient, but estimation is not exact.

How to determine L and H

How to determine the gap t between L and H

ランダムサンプルのサイズrとパラメータL, Hの決め方

$r = n^{3/4}$, $t=n^{1/2}$: LとHの幅を決めるパラメータ
このとき，ランダムサンプルのソートに要する時間は
$O(r \log r) = O(n^{3/4} \log n^{3/4}) = O(n)$
また，チェルノフの定理により，k番目に大きい要素がL以上H以下でない確率は，
$(n^{3/4}/n) \times (k/n) \times (1-(k/n)) < (k/n) \times (1/n^{1/4})$
で抑えられることがわかる．
これは，kが$1<k<n$の範囲のどんな値であってもnが大きくなれば失敗確率が無限に小さくなることを意味している．

計算時間

ランダムサンプルのソートは$O(n)$時間．
それ以外の操作も$O(n)$時間なので，全体でも$O(n)$時間．

演習問題E13-3: n個のデータの中からランダムにr個を重複なく取り出すアルゴリズムを考えよ．また，その計算時間を解析せよ．

Determining the size r of random sample and parameters L, H

$r = n^{3/4}$, $t=n^{1/2}$: parameter to determine the gap between L and H

➔time for sorting random sample is

$O(r \log r) = O(n^{3/4} \log n^{3/4}) = O(n)$.

By the Chernoff's theorem the probability that the k-th largest element is not between L and H is bounded by

$(n^{3/4}/n) \times (k/n) \times (1-(k/n)) < (k/n) \times (1/n^{1/4})$.

This implies that the failure probability becomes infinitely small as n becomes larger for any value of k in the range $1<k<n$.

Computation time

Sorting of a random sample is done in O(n) time.

The other operations are done in O(n) time. So, O(n) in total.

Exercise E13-3: Devise an algorithm for extracting r data out of n data randomly without any duplication. Also, analyze its computation time.

演習問題E13-4: 誕生日は366通りある。さてクラスにn人の学生がいるとして、同じ誕生日の2人がいる確率を求めよ。
また何組くらい同じ誕生日の組があるか，すなわち組の数の期待値を求めよ.

演習問題E13-5: Aをモンテカルロタイプの乱択アルゴリズムとする．Aは確率$p_A(n)$で正解を返すものとし，１回あたりの平均計算時間をT_A(n)とする．もしアルゴリズムAの出力が正解かどうかをT'_A(n)の時間で判定することが可能ならば，その判定アルゴリズムを用いて，モンテカルロタイプのアルゴリズムAをラスベガスタイプのアルゴリズムに変換できることを示せ．また，そのアルゴリズムの平均計算時間が$(T_A(n) + T'_A(n))/p_A(n)$で与えられることを示せ.

演習問題E13-6: チェルノフの定理について調べてみよ。

Exercise E13-4: There are 366 birthdays. Now, suppose that there are n persons in a class. Compute the probability that any two of them have the same birthday. Also, compute how many numbers of pairs have the same birthdays, that is, the expected number of pairs of the same birthdays.

Exercise E13-5: Let A be a randomized algorithm of Monte Carlo type. Suppose A returns a correct answer with probability $p_A(n)$ and average computing time be T_A(n). Show that we can transform the algorithm A into an algorithm of Las Vegas type if we can determine whether an output of algorithm A is correct or not in T'_A(n) time. Also, show that it is done in time $(T\_A(n) + T'\_A(n))/p_A(n)$ on average.

Exercise E13-6: Investigate Chernoff's theorem.

乱択アルゴリズムの計算時間解析

問題P33: クーポンコレクター問題:

- n種類のクーポンをランダムに集める
- すべてのクーポンが集まるまで試行する
- 手元に残るクーポンの枚数の期待値はいくらか?

…ラスベガスタイプのアルゴリズム解析で現れる

定理: クーポンコレクター問題の期待値は $O(n \log n)$

実際には
ほぼ $n \log n$
なので

例: 10種類のアイテムを集めるなら 23 個、
101種類のアイテムを集めるなら 460 個。

Running Time Analysis of Randomized Algorithms

Problem P33: Coupon Collectors Problem

- We want to collect all n kinds of coupons uniformly at random.
- We proceed until all kinds of coupons are collected
- How many coupons we will have on average?

…It appears when we analyze Las Vegas type algorithm.

Theorem: The expectation value of the number of coupons is $O(n \log n)$

In fact, it is approximately $n \log n$.

Ex: 23 items to collect 10 kinds of items,
460 items to collect 101 kinds of items.

乱択アルゴリズムの計算時間解析

定理: クーポンコレクター問題の期待値は $O(n \log n)$

証明

「状況 i」＝「クーポンを i 種類持っている」と定義する

すると試行は状態 0 から始まって状態 n で終わる

状態 i において状態 $i+1$ に変化する条件付確率は $(n-i)/n$

よって状態 i から状態 $i+1$ に変化するまでの試行の回数の期待値は $n/(n-i)$

期待値の線形性から求める期待値は

$$\sum_{i=0}^{n-1} \frac{n}{n-i} = n \sum_{i=0}^{n-1} \frac{1}{n-i} = n \sum_{i=1}^{n} \frac{1}{i} = nH_n$$

となる。ただしここで H_n は調和数であり $H_n = O(\log n)$ □

Running Time Analysis of Randomized Algorithms

Theorem: The expectation value of the number of coupons is O(n log n)

Proof

Define "state i"= "having i kinds of coupons".

Trials start at state 0 and finish at state n.

The conditional probability that it reaches to state i+1 from state i is (n-i)/n

Thus the expected number of trials from state i to state i+1 is n/(n-i)

By the linearity of expectation, we have

$$\sum_{i=0}^{n-1} \frac{n}{n-i} = n\sum_{i=0}^{n-1} \frac{1}{n-i} = n\sum_{i=1}^{n} \frac{1}{i} = nH_n$$

where H_n is the harmonic number and hence H_n=O(log n) . □

確率解析の例: QuickSortの解析

– ソーティング問題

Input: n 個のデータを記録した配列 a[n]

Output: 以下の条件を満たす配列 a[n]

a[1]<a[2]<…<a[n]

★話を単純にするため、a[i]=a[j], i≠j を満たすペアはないと仮定

– QuickSort は実用上、最速と言われることが多い
  - 典型的な分割統治法に基づくアルゴリズム
  - 都合良く分割されると $O(n \log n)$ 時間で計算が終わる
  - いつでも最悪の場合だと $O(n^2)$ かかる

…理論的な解析と速度保証はできるのか？

Example: Analysis of QuickSort

– Sorting Problem

Input: An array a[n] of n data

Output: The array a[n] such that

$a[1]<a[2]<\ldots<a[n]$

★To simplify, we assume that there are no pair $i \neq j$ with $a[i]=a[j]$

– In practical, QuickSort is said to be "the fastest sort"

- Representative algorithm based on divide-and-conquer
- If partition is well-done, it runs in $O(n \log n)$ time.
- If each partition is the worst case, it runs in $O(n^2)$ time.

…Can we analyze theoretically, and guarantee the running time?

確率解析の例: QuickSortの解析

- QuickSortのおさらい
  - qsort(a,1,n) を呼び出す
  - qsort(a, i, j) が呼び出されると、
    - pivot a[m] を(ランダムに)選ぶ
    - a[]を a[m] を基準に「前半」と「後半」に分ける。つまり
      $i \leqq i' < m$ なら $a[i'] < a[m]$
      $m < j' < j$ なら $a[j'] > a[m]$
      を満たすように並べ替える
    - qsort(a, i, i'), a[m], qsort(a, j', j) がソート結果

ラスベガスタイプのアルゴリズム

[余談]
毎回 O(*j*-*i*) 時間かければちょうど中央の値を見つけることもできる

- QuickSort は実用上、最速と言われることが多いが、、、
  - a[m] がいつでも a[i]..a[j] の中央の値だと
    $T(n) \leqq 2T(n/2) + (c+1)\,n$
    が成立するので、$T(n) = O(n \log n)$ を得る。
  - a[m] がいつでも a[i] や a[j] だと
    $T(n) \leqq T(1)+T(n-1)+(c+1)n$
    が成立するので、$T(n) = O(n^2)$ を得る。

平均的な場合はどうなのか？

Example: Analysis of QuickSort

- Review of QuickSort
  - Call qsort(a,1,n)
  - If qsort(a, i, j) is called,
    - (Randomly) choose a pivot a[m]
    - Divide a[] into "former" and "latter" by a[m]. I.e., sort as
      a[i']<a[m] for i≦ i' < m, and
      a[j']>a[m] for m< j'< j.
    - Return qsort(a, i, i'), a[m], qsort(a, j', j) as the result

Las Vegas Algorithm

[C.F.]
We can always find the center in O(*j*-*i*) time.

- Though they say that QuickSort is the fastest in a practical sense,,,
  - When a[m] is always the center of a[i]..a[j], we have
    $T(n) \leqq 2T(n/2) + (c+1)\, n$
    and hence $T(n) = O(n \log n)$.
  - When a[m] is always either a[i] or a[j], we have
    $T(n) \leqq T(1)+T(n-1)+(c+1)n$
    and hence $T(n) = O(n^2)$.

What about average case?

確率解析の例: QuickSortの解析

- QuickSort は実用上、最速と言われることが多いが、、、
  - 平均的には、a[i] ... a[j] の値が一様に選ばれると仮定する。
    - つまり k 番目に大きいデータを pivot にする確率は $1/(j-i+1)$

[定理]
上記の仮定のもとでの QuickSort の実行時間の期待値の上界は
$2n\,H(n) \sim 2n \log n$

オーバーヘッドの少なさから、確かに速そう

- 記法
  - » a[1]…a[n] の中で k 番目に来るべき要素を s_k と書く。
  - » 指示変数（indicator variable）X_{ij} を以下のように定義する

$$X_{ij} = \begin{cases} 0 & s_i \text{ と } s_j \text{ がアルゴリズム中で比較されないとき} \\ 1 & s_i \text{ と } s_j \text{ がアルゴリズム中で比較されるとき} \end{cases}$$

- QuickSort の実行時間~要素の比較回数$= \sum_{i=1}^{n} \sum_{j>i} X_{ij}$

Example: Analysis of QuickSort

- They say that QuickSort is the fastest in a practical sense,,,
  - Assumption: each item in a[i] ... a[j] is chosen uniformly at random.
    - Thus the *k*th largest value is chosen as the pivot with probability $1/(j-i+1)$

[Theorem] An upper bound of the expected value of the running time of QuickSort is $2n\,H(n) \sim 2n \log n$

It runs fast since few overhead.

- Notation
  - » s_k is the *k*th largest item in a[1]…a[n].
  - » Define indicator variable X_{ij} as follows

$$X_{ij} = \begin{cases} 0 & s_i \text{ and } s_j \text{ are not compared in the algorithm} \\ 1 & s_i \text{ and } s_j \text{ are compared in the algorithm} \end{cases}$$

- Running time of QuickSort

  ~ the number of comparisons $= \sum_{i=1}^{n} \sum_{j>i} X_{ij}$

確率解析の例: QuickSortの解析

[定理]
QuickSort の実行時間の期待値の上界は　$2n\,H(n) \sim 2n \log n$

（期待値の線形性）

– QuickSort の実行時間の期待値$= E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$

– 「p_{ij} ： s_i と s_j が比較される確率」と定義すると、

$$E[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

よって p_{ij} の値を考える

– s_i と s_j はどんなときに比較されるのか？

1. どちらかが pivot に選ばれている
2. それまでの計算過程で、別々の qsort にわけられていない
   $\Leftrightarrow$ s_i と s_j の間の要素が、まだ pivot として選ばれていない

Example: Analysis of QuickSort

[Theorem] An upper bound of the expected value of the running time of QuickSort is $2n\,H(n) \sim 2n \log n$

- The expected value of the running time of QuickSort=

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$$

(Linearity of expectation value)

- Define as "p_{ij} : probability that s_i and s_j are compared",

$$E[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

Thus consider the value of p_{ij}

- When s_i and s_j are compared??
  1. One of them is chosen as the pivot, and
  2. They are not yet separated by qsort up to there

     $\Leftrightarrow$ Any element between s_i and s_j are not yet chosen as a pivot

確率解析の例: QuickSortの解析

[定理]
QuickSort の実行時間の期待値の上界は　$2n\,H(n) \sim 2n \log n$

- s_i と s_j はどんなときに比較されるのか？
  1. どちらかが pivot に選ばれている
  2. それまでの計算過程で、別々の qsort にわけられていない
     $\Leftrightarrow$ s_i と s_j の間の要素が、まだ pivot として選ばれていない
     - $s_i, s_{i+1}, s_{i+2}, \ldots, s_{j-1}, s_j$ が pivot に選ばれる順番は、すべて等確率！
     - よってこれらの中で s_i か s_j が最初の pivot になる確率: $\dfrac{2}{j-i+1}$
       つまり QuickSort の実行時間の期待値=

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} p_{ij} = \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n}\sum_{k=2}^{n-i+1} \frac{2}{k} \le 2\sum_{i=1}^{n}\sum_{k=1}^{n} \frac{1}{k} = 2nH(n)$$

Example: Analysis of QuickSort

[Theorem] An upper bound of the expected value of the running time of QuickSort is $2n\,H(n) \sim 2n \log n$

- When s_i and s_j are compared?
  1. One of them is chosen as the pivot, and
  2. They are not yet separated by qsort up to there

     $\Leftrightarrow$ Any element between s_i and s_j is not yet chosen as a pivot
  - The ordering of pivots in $s_i, s_{i+1}, s_{i+2}, \ldots, s_{j-1}, s_j$ is uniformly at random!
  - Thus s_i or s_j is the first pivot with probability $\dfrac{2}{j-i+1}$

Therefore, the expected time of the running time of QuickSort

$$= E\left[\sum_{i=1}^{n}\sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} p_{ij} = \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n}\sum_{k=2}^{n-i+1} \frac{2}{k} \le 2\sum_{i=1}^{n}\sum_{k=1}^{n} \frac{1}{k} = 2nH(n)$$