Design and Analysis of Algorithms (XII)

An Introduction to Randomized Algorithms

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Randomized Algorithms

"algorithms which employ a degree of randomness as part of its logic"

—Wikipedia

algorithms that flip coins

Begin with Median

Median

The median of a list of numbers is its $50th$ percentile: half the number are bigger than it, and half are smaller.

If the list has even length, we pick the smaller one of the two.

The purpose of the median is to summarize a set of numbers by a single typical value.

Computing the median of n numbers is easy, just sort them. ($O(n \log n)$).

Q: Can we do better?

Selection

Input: A list of number S; an integer k.

Output: The $k\,th$ smallest element of S.

A Randomized Selection

For any number v, imagine splitting list S into three categories:

- elements smaller than v, i.e., S_L;
- those equal to v, i.e., S_v (there might be duplicates);
- and those greater than v, i.e., S_R; respectively.

$$selection(S,k) = \begin{cases} selection(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ selection(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v| \end{cases}$$

How to Choose v?

It should be picked quickly, and it should shrink the array substantially, the ideal situation being

$$|\, S_L \,|, |\, S_R \,| \approx \frac{|\, S \,|}{2}$$

If we could always guarantee this situation, we would get a running time of

$$T(n) = T(n/2) + O(n) = O(n)$$

This requires picking v to be the median, which is our ultimate goal!

Instead, we pick v randomly from S!

How to Choose v?

Worst-case scenario would force our selection algorithm to perform

$$n + (n-1) + (n-2) + \ldots + \frac{n}{2} = \Theta(n^2)$$

Best-case scenario $O(n)$

The Efficiency Analysis

v is good if it lies within the $25th$ to $75th$ percentile of the array that it is chosen from.

A randomly chosen v has a 50% chance of being good.

Lemma:
On average a fair coin needs to be tossed two times before a heads is seen.

Proof:
Let E be the expected number of tosses before heads is seen.

$$E = 1 + \frac{1}{2}E$$

Therefore, $E = 2$.

The Efficiency Analysis

Let $T(n)$ be the expected running time on the array of size n, we get

$$T(n) \leq T(3n/4) + O(n) = O(n)$$

Randomized Algorithms

Basis

Randomized algorithms are a basis of

- Online algorithms
- Approximation algorithms
- Quantum algorithms
- Massive data algorithms

Pro and Con of Randomization

Cost of Randomness

- chance the answer is wrong.
- chance the algorithm takes a long time.

Benefits of Randomness

- on average, with high probability gives a faster or simpler algorithm.
- some problems require randomness.

Types of Randomized Algorithms

Las Vegas Algorithm (LV):

- Always correct
- Runtime is random (small time with good probability)
- Examples: Quicksort, Hashing

Monte Carlo Algorithm (MC):

- Always bounded in runtime
- Correctness is random
- Examples: Karger's min-cut algorithm

Las Vegas VS. Monte Carlo

LV implies MC:

Fix a time T and let the algorithm run for T steps. If the algorithm terminates before T, we output the answer, otherwise we output 0.

MC does not always imply LV:

The implication holds when verifying a solution can be done much faster than finding one.

Test the output of MC algorithm and stop only when a correct solution is found.

Las Vegas Algorithm: Quicksort

Quick Sort

`QuickSort(`x`);`

if $x == []$ **then** `return` $[]$;
Choose pivot $t \in [n]$;
`return`
 `QuickSort(`$[x_i | x_i < x_t]$`)` $+ [x_t] +$ `QuickSort(`$[x_i | x_i \geq x_t]$`);`

Why Random?

If $t = 1$ always, then the complexity is $\Theta(n^2)$.

Note that, we do not need an adversary for bad inputs.

In practice, lists that need to be sorted will consist of a mostly sorted list with a few new entries.

Complexity Analysis

Z_{ij} := event that ith largest element is compared to the jth largest element at any time during the algorithm.

Each comparison can happen at most once. Time is proportional to the number of comparisons $= \sum_{i<j} Z_{ij}$

$Z_{ij} = 1$ iff the first pivot in $\{i, i+1, \ldots, j\}$ is i or j.

- if the pivot is $< i$ or $> j$, then i and j are not compared;
- if the pivot is $> i$ and $< j$, then the pivot splits i and j into two different recursive branches so they will never be compared.

Thus, we have

$$Pr[Z_{ij}] = \frac{2}{j-i+1}$$

.

Complexity Analysis

$$
\begin{aligned}
E[Time] &\leq E\left[\sum_{i<j} Z_{ij}\right] \\
&= \sum_{i<j} E\left[Z_{ij}\right] \\
&= \sum_{i} \sum_{i<j} \left(\frac{2}{j-i+1}\right) \\
&= 2 \cdot \sum_{i} \left(\frac{1}{2} + \cdots + \frac{1}{n-i+1}\right) \\
&\leq 2 \cdot n \cdot (H_n - 1) \\
&\leq 2 \cdot n \cdot \log n
\end{aligned}
$$

Monte Carlo: Min Cuts

Min Cuts

The (s,t) cut is the set $S \subseteq V$ with $s \in S$, $t \notin S$.

The cost of a cut is the number of edges e with one vertex in S and the other vertex not in S.

The min (s,t) cut is the (s,t) cut of minimum cost.

The global min cut is the min (s,t) cut over all $s,t \in V$.

Complexity

Iterate over all choices of s and t and pick the smallest (s,t) cut, by simply running $O(n^2)$ the Ford-Fulkersons algorithms over all pairs.

The complexity can be reduced by a factor of n by noting that each node must be in one of the two partitioned subsets.

Select any node s and compute a $\min(s,t)$ cut for all other vertices and return the smallest cut. This results in $O(n)$ the Ford-Fulkersons algorithm, resulting in complexity $O(n \cdot nm) = O(n^2 m)$.

The First Randomized Algorithm

`MinCut1(`G`);`

while $n > 1$ **do**
    Choose a random edge;
    Contract into a single vertex;
end

Contracting an edge means that removing the edge and combining the two vertices into a super-node.

Analysis

Lemma

The chance the algorithm fails in step 1 is $\leq \frac{2}{n}$

Proof

The chance of failure in the first step is $\frac{OPT}{m}$ where
OPT = cost of true min cut $=\mid E(S,\bar{S}) \mid$ and m is the number of edge.

For all $u \in V$, let $d(u)$ be the degree of u.

$$OPT =\mid E(S,\bar{S}) \mid \leq \min_u d(u) \leq \frac{1}{n}\sum_{u\in V} d(u) \leq \frac{2\cdot m}{n}$$

Dividing both sides by m we have

$$\frac{\mid E(S,\bar{S}) \mid}{m} \leq \frac{2}{n}$$

Analysis

Continue the analysis in each subsequent round, we have

$$Pr(\text{fail in } 1^{st} \text{ step}) \leq \frac{2}{n}$$

$$Pr(\text{fail in } 2^{nd} \text{ step} \mid \text{success in } 1^{st} \text{ step}) \leq \frac{2}{n-1}$$

$$\vdots$$

$$Pr(\text{fail in } i^{th} \text{ step} \mid \text{success till } (i-1)^{th} \text{ step}) \leq \frac{2}{n+1-i}$$

Analysis

Lemma `MinCut1` succeeds with $\geq \frac{2}{n^2}$ probability.

Proof

$Z_i :=$ the event that an edge from the cut set is picked in round i.

$$Pr[Z_i | \bar{Z}_1 \cap \bar{Z}_2 \cap \cdots \cap \bar{Z}_{i-1}] \leq \frac{2}{n+1-i}$$

Thus the probability of success is given by

$$\begin{aligned} Pr(Succ) = Pr[\bar{Z}_1 \cap \bar{Z}_2 \cap \cdots \cap \bar{Z}_{n-2}] &\geq (1-\frac{2}{n})(1-\frac{2}{n-1})\ldots(1-\frac{2}{3}) \\ &\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{n-5}{n-3} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{1}{n} \cdot \frac{1}{n-1} \cdot \frac{2}{1} \cdot \frac{1}{1} \\ &= \frac{2}{n(n-1)} \geq \frac{2}{n^2} \end{aligned}$$

A Lemma

Suppose that the `MinCut1` is terminated when the number of vertices remaining in the contracted graph is exactly t. Then any specific min cut survives in the resulting contracted graph with probability at least

$$\frac{\binom{t}{2}}{\binom{n}{2}} = \Omega(\frac{t}{n})^2$$

The Second Randomized Algorithm

`MinCut2(`G, k`);`

$i = 0$;
while $i > k$ **do**
    `MinCut1(`G`);`
    $i++$;
end
pick the optimization;

If we run `MinCut1` k times and pick the set of min cost, then

$$Pr(failure) \leq (1 - \frac{2}{n^2})^k \leq e^{\frac{-2k}{n^2}}$$

A Useful Bound

$$(1 - a) \leq e^{-a} \qquad \forall a > 0$$

How to Choose k

Set $k = \frac{n^2}{2} \log(\frac{1}{\delta})$ to get $1 - \delta$ success probability.

Observation

Initial stages of the algorithm are very likely to be correct. In particular, the first step is wrong with probability at most $2/n$.

As contracting more edges, failure probability goes up.

Since earlier ones are more accurate and slower, why not do less of them at the beginning, and more as the number of edges decreases?

The Third Randomized Algorithm

`MinCut3(`G`);`

Repeat twice{

  take $n - \frac{n}{\sqrt{2}}$ steps of contraction;

  recursively apply this algorithm;

}

take better result;

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2) = n^2 \log n$$

Success Probability

$$
\begin{aligned}
pr(n) &= 1 - (\text{failure probability of one branch})^2 \\
&= 1 - (1 - \text{success in one branch})^2 \\
&\geq 1 - (1 - (\frac{\frac{n}{\sqrt{2}}}{n})^2 \cdot pr(\frac{n}{\sqrt{2}}))^2 \\
&= 1 - \left(1 - \frac{1}{2}\, pr\left(\frac{n}{\sqrt{2}}\right)\right)^2 \\
&= pr\left(\frac{n}{\sqrt{2}}\right) - \frac{1}{4}\, pr\left(\frac{n}{\sqrt{2}}\right)^2
\end{aligned}
$$

Success Probability

Let $x = \log_{\sqrt{2}} n$, by setting $f(x) = pr(n)$, we get

$$f(x) = f(x-1) - f(x-1)^2$$

$f(x) = \frac{1}{x}$ gives:

$$f(x-1) - f(x) = \frac{1}{x-1} - \frac{1}{x} = \frac{1}{x(x-1)} \approx \frac{1}{(x-1)^2} = f(x-1)^2$$

Thus, $pr(n) = O\left(\frac{1}{\log n}\right)$.

The Fourth Randomized Algorithm

Repeat `MinCut3` $O(\log n \log \frac{1}{\delta})$ times.

Complexity & Success Analysis: DIY!

Linearity of Expectation

Linearity of Expectation

$$E[\sum_i X_i] = \sum_i E[X_i]$$

Sailors Problem

A ship arrives at a port, and the 40 sailors on board go ashore for revelry. Later at night, the 40 sailors return to the ship and, in their state of inebriation, each chooses a random cabin to sleep in.

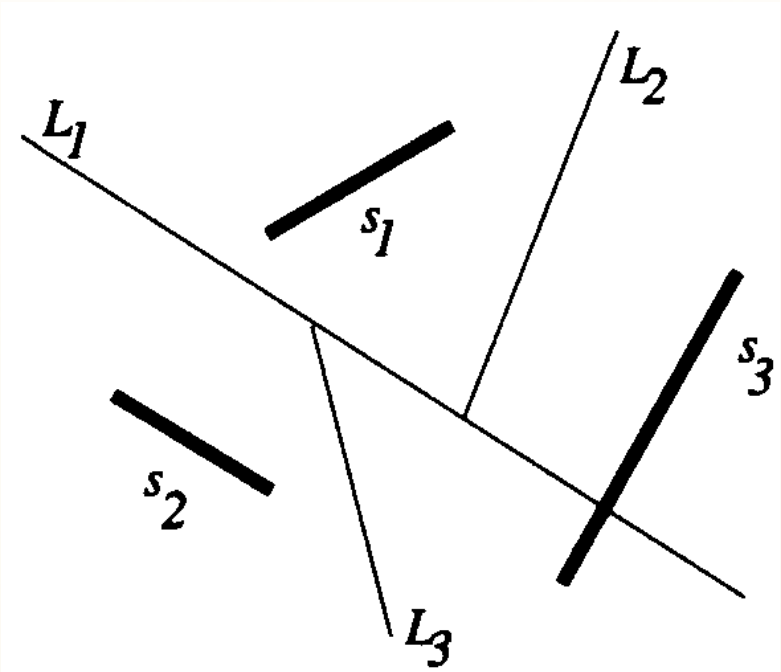
What is the expected number of sailors sleeping in their own cabins.

$$E[\sum_{i=1}^{40} X_i] = \sum_{i=1}^{40} E[X_i] = 1$$

Binary Planar Partition

Given a set $S = \{S_1, S_2, \ldots, S_n\}$ of non-intersecting line segments in the plane, we wish to find a binary planar partition such that every region in the partition contains at most one line segment (or a portion of one line segment).

An Example



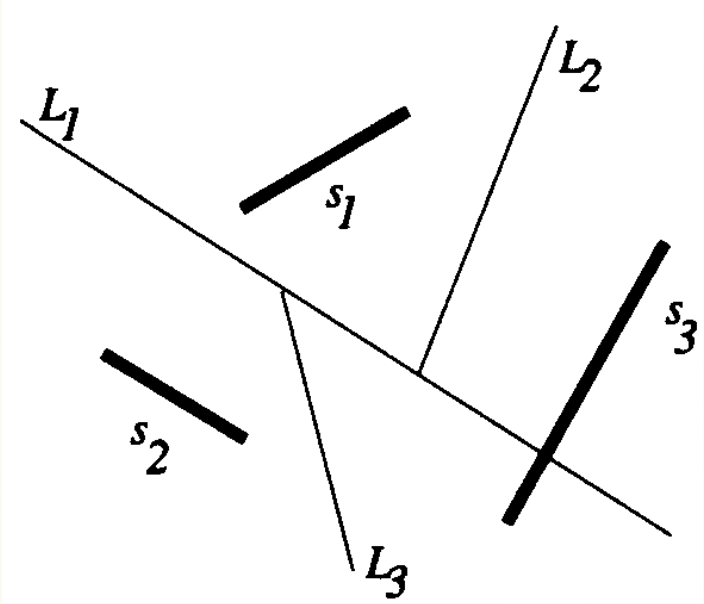
Binary Partition Tree

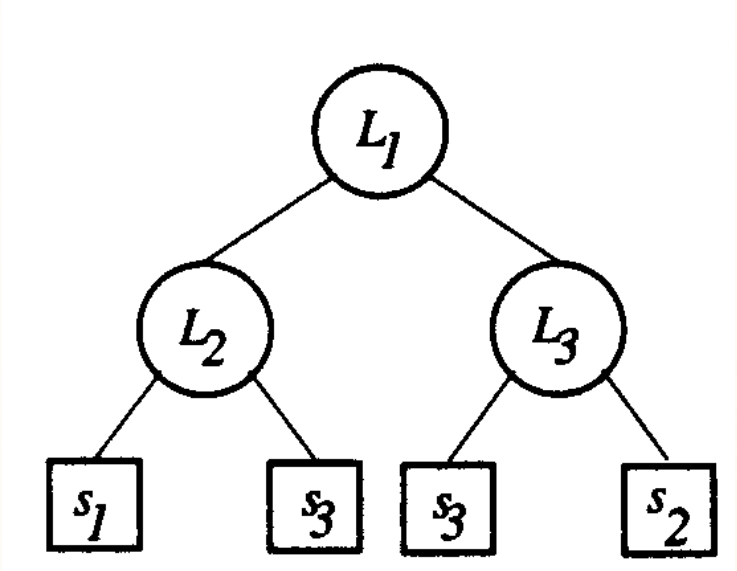
A binary planar partition consists of a binary tree together with some additional information.

Associated with each node v is a region $r(v)$ of the plane, and with each internal node v is a line $l(v)$ that intersects $r(v)$.

The region corresponding to the root is the entire plane. The region $r(v)$ is partitioned by $l(v)$ into two regions $r_1(v)$ and $r_2(v)$, which are the regions associated with the two children of v.

The Example





Remark

Because the construction of the partition can break some of the input segments S_i into smaller pieces, the size of the partition need not be n.

it is not clear that a partition of size $O(n)$ always exists.

Autopartition

For a line segment s, let $l(s)$ denote the line obtained by extending s on both sides to infinity.

For the set $S = \{s_1, s_2, \ldots, s_n\}$ of line segments, a simple and natural class of partitions is the set of autopartitions, which are formed by only using lines from the set $\{l(s_1), l(s_2), \ldots, l(s_n)\}$

An Algorithm

`AutoPartition(`$\{s_1, s_2, \ldots, s_n\}$`)`

Pick a permutation π of $\{1, 2, \ldots, n\}$ uniformly at random from the $n!$ possible permutations;

while *a region contains more than one segment* **do**

  cut it with $l(s_i)$ where i is first in the ordering π such that s_i cuts that region;

end

Analysis

Theorem

The expected size of the autopartition produced by `AutoPartition` is $O(n \log n)$.

Proofs

$index(u, v) = i$ if $l(u)$ intersects $i - 1$ other segments before hitting v.

$u \dashv v$ denotes the event that $l(u)$ cuts v in the constructed partition.

The probability of $index(u, v) = i$ and $u \dashv v$ is $1/(i + 1)$.

Let $C_{uv} = 1$ if $u \dashv v$ and 0 otherwise,

$$E[C_{uv}] = Pr[u \dashv v] \leq \frac{1}{index(u, v) + 1}$$

Proofs

$$
\begin{aligned}
Pr(\text{partition numbers}) &= n + E[\sum_u \sum_v C_{uv}] \\
&= n + \sum_u \sum_v E[C_{uv}] \\
&= n + \sum_u \sum_{v \neq u} Pr[u \dashv v] \\
&\leq n + \sum_u \sum_{v \neq u} \frac{1}{index(u,v)+1} \\
&\leq n + \sum_u \sum_{i=1}^{n-1} \frac{2}{i+1} \\
&\leq n + 2nH_n
\end{aligned}
$$

Randomized Complexity Classes

Common Complexity Class

P: Deterministic Polynomial Time. We have $L \in \mathbf{P}$ iff there exists a polynomial-time algorithm that decides L.

NP: Nondeterministic Polynomial-Time. We have $L \in \mathbf{NP}$ iff for every input $x \in L$ there exists some solution string y such that a polynomial-time algorithm can accept x if $x \in L$ given the solution y.

Randomized Complexity Classes

ZPP: Zero-error probabilistic Polynomial-Time. $L \in$ **ZPP** iff there exists an algorithm that decides L, and the expected value of its running time is polynomial. Note that this class describes the Las Vegas algorithms.

RP: Randomized polynomial time. This class has tolerance for one-sided error, that is, $L \in$ **RP** iff there exists a polynomial-time algorithm A such that:

- if $x \in L$ then A accepts with probability $\geq 1/2$.
- if $x \notin L$ then A rejects.

Randomized Complexity Classes

PP: Probabilistic Polynomial. $L \in \mathbf{PP}$ iff there exists a polynomial-time algorithm A s.t.

- if $x \in L$ then A accepts with probability $> 1/2$.
- if $x \notin L$ then A rejects with probability $\geq 1/2$ (or accepts $< 1/2$).

BPP: Bounded Probabilistic Polynomial. $L \in \mathbf{BPP}$ iff there exists a poly-time algorithm A s.t.

- if $x \in L$ then A accepts with probability $\geq 2/3$
- if $x \notin L$ then A rejects with probability $> 2/3$ (or accepts $\leq 1/3$).

Relations

$$\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{NP} \subseteq \mathbf{PP}$$

$$\mathbf{RP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$$

Conjecture

$$\mathbf{P} = \mathbf{BPP} \subseteq \mathbf{NP}$$

Referred Materials

Content of this lecture comes from Section 2.4 in [DPY07], and Section 1.1, 1.2, 1.3, 1.5, in [MR95].