Randomized Algorithms

Two Types of Randomized Algorithms

and

Some Complexity Classes

Speaker: Chuang-Chieh Lin

Advisor: Professor Maw-Shang Chang

National Chung Cheng University

2006/9/20

References

- Professor Hsueh-I Lu’s slides.
- *Randomized Algorithms*, Rajeev Motwani and Prabhakar Raghavan.
- *Probability and Computing - Randomized Algorithms and Probabilistic Analysis*, Michael Mitzenmacher and Eli Upfal.

Outline

- Las Vegas algorithms and Monte Carlo algorithms
- RAMs and Turing machines
- Complexity classes
  - P, NP, RP, ZPP, BPP and their complementary classes
  - Open problems

Las Vegas vs. Monte Carlo

- Las Vegas algorithms
  - Always produces a (correct/optimal) solution.
  - Like RandQS.

- Monte Carlo algorithms
  - Allow a small probability for outputting an incorrect/non-optimal solution.
  - Like RandMC.
  - The name is by von Neumann.

Las Vegas Algorithms

- For example, RandQS is a Las Vegas algorithm.
- A Las Vegas always gives the correct solution
- The only variation from one run to another is its running time, whose distribution we study.

Randomized quicksort

algorithm RandQS(X) {

  if X is empty then

    return;

randomization

  select x uniformly at random from X;

  let $Y = \{y \in X \mid y < x\}$;

  let $Z = \{z \in X \mid z > x\}$;

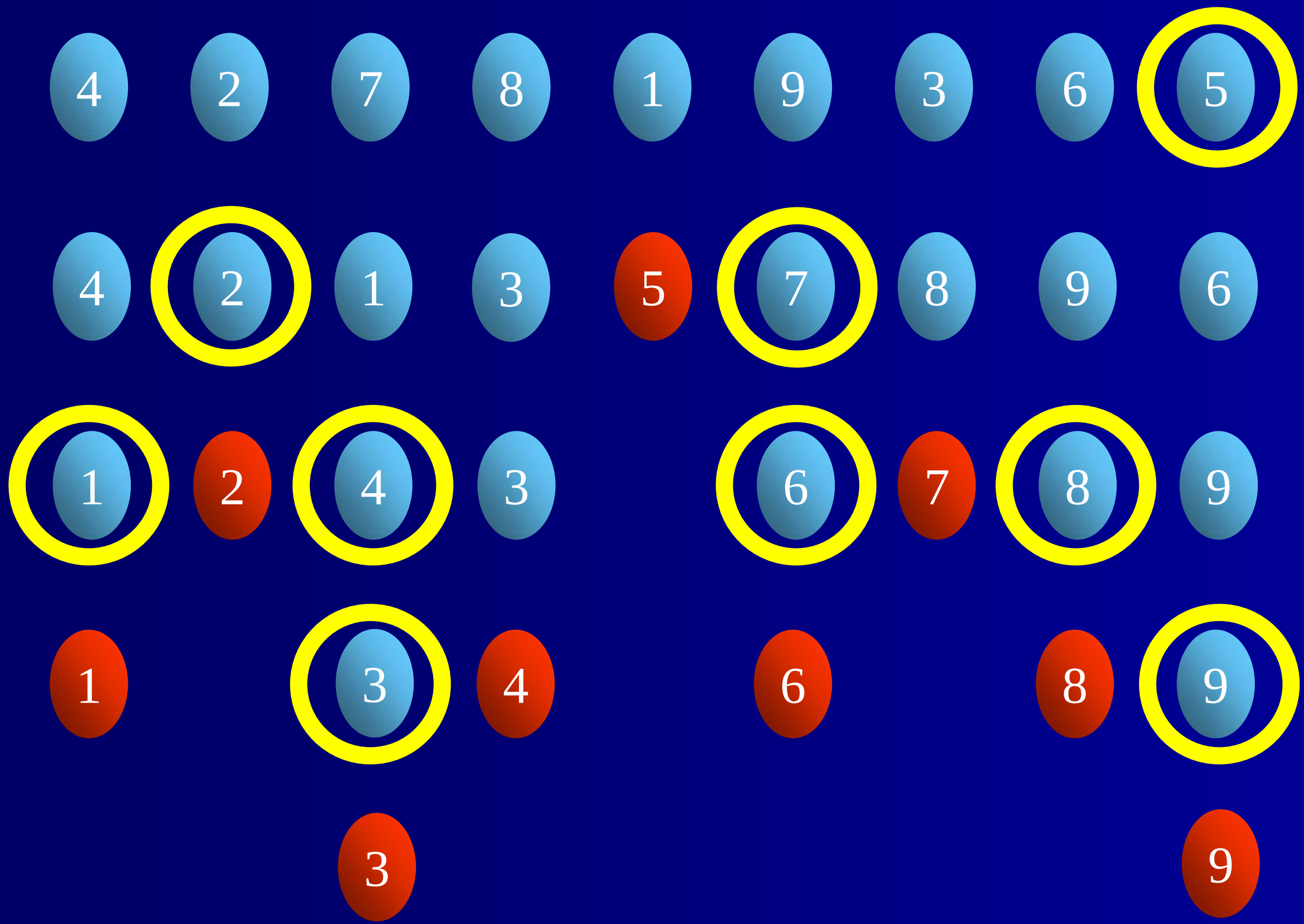
  call RandQS(Y);

  print x;

  call RandQS(Z);

}

An illustration



2 Questions for RandQS

- Is RandQS correct?
  - That is, does RandQS "always" output a sorted list of X?
- What is the time complexity of RandQS?
  - Due to the randomization for selecting x, the running time for RandQS becomes a random variable.
  - We are interested in the expected time complexity for RandQS.

Monte Carlo algorithms

- For example, RandEC (the randomized minimum-cut algorithm we have discussed) is a Monte Carlo algorithm.
- A Monte Carlo algorithm may sometimes produce a solution that is incorrect.
- For decision problems, there are two kinds of Monte Carlo algorithms:
  - those with one-sided error
  - those with two-sided error

Which is better?

- The answer depends on the application.
- A Las Vegas algorithm is by definition a Monte Carlo algorithm with error probability 0.
- Actually, we can derive a Las Vegas algorithm ***A*** from a Monte Carlo algorithm ***B*** by repeated running ***B*** until we get a correct answer.

Computation model

- Throughout this talk, we use the *Turing machine* model to discuss complexity theory issues.

- As is common, we switch to the ***RAM*** (random access machine) as the model of computation when describing and analyzing algorithms.

Computation model (cont'd)

- For simplicity, we will work with the general *unit-cost* **RAM** model.
- In unit-cost RAM model, each instruction can be performed in one time step.

Determinist

You can refer to any computation theory textbook to for more details here.

- A deterministic Turing machine is a quadruple $M = (S, \Sigma, \delta, s)$.
  - Here S is a finite set of states, of which $s \in S$ is the machine's initial state.
  - Σ : a finite set of symbols (this set includes special symbols BLANK and FIRST).
  - δ : the transition function of the Turing machine, mapping $S \times \Sigma$ to $(S \cup \{\text{HALT}, \text{YES}, \text{NO}\}) \times \Sigma \times \{\leftarrow, \rightarrow, \text{STAY}\}$.
- The machine has three states: HALT (the halting state), YES (the accepting state), and NO (the rejecting state) (these are states, but formally not in S.)

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

$\triangleright$	0	1	0	0			

Q: What does this Turing machine do?

A Turing machine with one tape

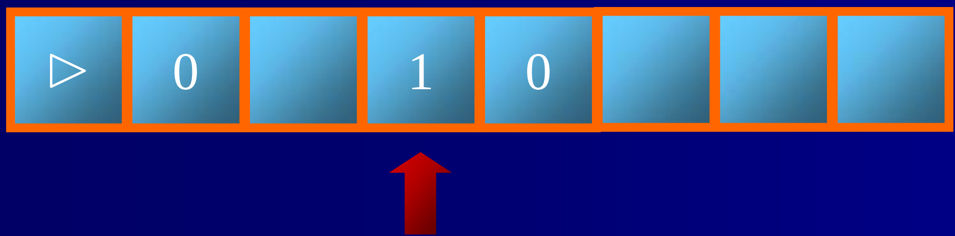
state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

$\triangleright$	0	1		0			

Q: What does this Turing machine do?

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$



Q: What does this Turing machine do?

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

Q: What does this Turing machine do?

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

Q: What does this Turing machine do?

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

$\triangleright$		0	1	0			

Q: What does this Turing machine do?

A Turing machine with one tape

state \ symbol	0	1	$\sqcup$	$\triangleright$
s	$(s, 0, \rightarrow)$	$(s, 1, \rightarrow)$	$(q, \sqcup, \leftarrow)$	$(s, \triangleright, \rightarrow)$
q	$(q_0, \sqcup, \rightarrow)$	$(q_1, \sqcup, \rightarrow)$	$(q, \sqcup, -)$	$(h, \triangleright, \rightarrow)$
q_0	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(s, 0, \leftarrow)$	$(h, \triangleright, \rightarrow)$
q_1	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(s, 1, \leftarrow)$	$(h, \triangleright, \rightarrow)$
h	$\times$	$\times$	$\times$	$\times$

Q: What does this Turing machine do?

A probabilistic TM

- A *probabilistic Turing machine* is a (nondeterministic) Turing machine augmented with the ability to generate an unbiased coin flip in one step.

- It corresponds to a randomized algorithm.

A probabilistic TM (cont'd)

- On any input *x*, a probabilistic Turing machine accepts *x* with some probability, and we study this probability.

Language recognition problem

- Any decision problem can be treated as a language recognition problem.
- Let Σ^* be the set of all possible strings over Σ.
- A language $L \subseteq \Sigma^*$ is a collection of strings over Σ.

Language recognition problem (cont'd)

- The corresponding language recognition problem is to decide whether a given string $x \in \Sigma^*$ belongs to L.
- An algorithm solves a language recognition problem for a specific language L by *accepting* (output YES) any input string contained in L, and *rejecting* (output NO) any input string not contained in L.

Complexity Classes

- A complexity class is a collection of languages all of whose recognition problem can be solved under prescribed bounds on the computational resources.
- We are primarily interested the classes in which algorithms is *polynomial-time* bounded.

The Class: P

- The class **P** consists of all languages L that have a polynomial time algorithm A such that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow A(x)$ accepts
  - $x \notin L \Rightarrow A(x)$ rejects

The Class: NP

Here y can be regarded as a "certificate"

- The class **NP** consists of all languages L that have a polynomial time algorithm A such that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow \exists\, y \in \Sigma^*,\ A(x, y)$ accepts, where $|y|$ is bounded by a polynomial in $|x|$.
  - $x \notin L \Rightarrow \forall\, y \in \Sigma^*,\ A(x, y)$ rejects

A useful view of P and NP

- The class **P** consists of all languages *L* such that for any *x* in *L*, a proof (certificate) of the membership *x* in *L* (represented by the string *y*) can be *found* and *verified* efficiently.
- The class **NP** consists of all languages *L* such that for any *x* in *L*, a proof (certificate) of the membership of *x* in *L* can be *verified* efficiently.

A useful view of **P** and **NP** (cont'd)

- Obviously $\mathbf{P} \subseteq \mathbf{NP}$, but it is not known whether $\mathbf{P} = \mathrm{N}\mathbf{P}$.
- If $\mathbf{P} = \mathrm{N}\mathbf{P}$, the existence of an efficiently verifiable proof (certificate) implies that it is possible to actually find such a proof (certificate) efficiently.

- When randomized algorithms are allowed, we have some basic classes as follows.

The Class: RP

Actually, the choice of the bound on the error probability ½ can be arbitrary.

- The class **RP** (for **R**andomized **P**olynomial time) consists of all languages L that have a *randomized* algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \geq ½$.
  - $x \notin L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] = 0$.

One-sided error

One-sided error vs. two-sided error

- A randomized algorithm A for recognizing a language L is of *one-sided error* if for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \neq 1$
  - $x \notin L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] = 0.$

  or

  - $x \in L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] = 1.$
  - $x \notin L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \neq 0.$

One-sided error vs. two-sided error (cont'd)

- A randomized algorithm A for recognizing a language L is of *two-sided error* if for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \neq 1.$
  - $x \notin L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \neq 0.$

The Class: ZPP

- The class **ZPP** (for zero-error Probabilistic Polynomial time) is the class of languages that has Las Vegas algorithms running in expected polynomial time.

The Class: ZPP (cont'd)

- For example,
  RandQS is a **ZPP** algorithm.

The Class: PP

- The class **PP** (for Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow$ **Pr**$[A(x)$ accepts$] > ½$.
  - $x \notin L \Rightarrow$ **Pr**$[A(x)$ accepts$] < ½$.

Exercise 1.10

- Consider a randomized algorithm with two-sided error probabilities as in the definition of **PP**. Show that a polynomial number of independent repetitions of this algorithm need not suffice to reduce the error probability to ¼ .
  - Consider the case where the error probability is $\frac{1}{2}+\frac{1}{2^n}$.

The Class: PP (cont'd)

- The definition of **PP** is weak.
  - It can be proved that it may not be possible to use a small number of repetitions of an algorithm *A* with such two-sided error probability to obtain an algorithm with "significantly smaller" error probability. (proved by using the Chernoff bound)
- Compared to the class **BPP**!

The Class: PP (cont'd)

- Note:
  - To reduce the error probability of a two-sided error algorithm, we can perform several independent iterations on the same input and produce the output that occurs in the majority of these iterations.
  - This can be done by using the Chernoff bound.

The Class: BPP

Actually, we only have to make sure that the difference between the "green one" and the "red one" is only polynomially small.

- The class BPP (for Bounded-error Probabilistic Polynomial time) consists of all languages L that have a randomized algorithm A running in worst-case polynomial time that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \geq ¾$.
  - $x \notin L \Rightarrow \mathbf{Pr}[A(x) \text{ accepts}] \leq ¼$.

Note

- Exponentially small
  - $1/2^n$, $1/3^n$, …
- Polynomially small
  - $1/n^2$, $1 / \log n$, ….

The Class: BPP (cont'd)

- One can show that for this class of algorithms, the error probability can be reduced to $1/2^n$ with only a polynomial number of iterations.

Problem 4.8: Consider a **BPP** algorithm that has an error probability of $\frac{1}{2} - \frac{1}{p(n)}$, for some polynomially bounded function $p(n)$ of the input size n. Using the Chernoff bound on the tail of the binomial distribution, show that a polynomial number of independent repetitions of this algorithm su± ce to reduce the error probability to $\frac{1}{2^n}$.

The Class: BPP (cont'd)

- Consider the decision version of the min-cut problem:
  - Given a graph *G* and an integer *K*, verify that the min-cut size in *G* equals *K*.
- Assume that we have modified the Monte Carlo algorithm RandEC to reduce its error probability to be less than ¼ (by sufficiently many repetitions).
  - We then get a **BPP** algorithm.

The Class: BPP (cont'd)

- In the case where K is indeed the min-cut value, the algorithm may not come up with the right value and, hence, may reject the input .
- If the min-cut value is smaller than K, the algorithm may only find cuts of size K, and hence, accept the input.
- If the min-cut value is larger than K, the algorithm will never find any cut of size K, and hence, reject the input.

Note

- Consider another decision version of the min-cut problem:
  - Given a graph *G* and an integer *K*, verify that the min-cut size in *G* is at most *K*.
- Assume again that we have modified the Monte Carlo algorithm RandEC to reduce its error probability to be less than ¼ (by sufficiently many repetitions).
  - We then get a **RP** algorithm for this problem.

Note (cont'd)

- In the case where the actual min-cut size C is larger than K, the algorithm will never accept the input.
- If the min-cut value is at most K, the algorithm may find cuts of size at most K, and hence, accept the input.

One-sided error!

Complement Classes

- For any complexity class ***C***, we define the complementary class co-***C*** as the set of languages whose complement is in the class ***C***.
  - That is,

$$\text{co-}C = \{L \mid \bar{L} \in C\}$$

Complement Classes (cont'd)

- Then we have co-**P**, co-**NP**, co-**RP**, co-**PP**, co-**ZPP**, co-**BPP**, …

- For example,

The Class: co-RP

- The class co-**RP** consists of all languages L that have a *randomized* algorithm A running in worst-case polynomial time such that for any input $x \in \Sigma^*$,
  - $x \in L \Rightarrow$ **Pr**[$A(x)$ accepts] = 1.
  - $x \notin L \Rightarrow$ **Pr**[$A(x)$ accepts] $\leq$ ½.

Exercise

- Show that **ZPP** = **RP** ∩ co-**RP**.

Open problems

- Is **NP** = **P**?
- Is **RP =** co- **RP**?
- Is **RP** $\subseteq$ **NP** $\cap$ co-**NP**?
- **Is BPP** $\subseteq$ **NP?**
- Is **BPP** = **P**?
- ……

Thank you.