Exercise

1. Suppose that a complete undirected graph $G = (V, E)$ with at least 3 vertices has a cost function c that satisfies the triangle inequality. Prove that $c(u,v) \geq 0$ for all $u,v \in V$.
2. Show that the decision version of the set-covering problem is NP-complete by reduction from the vertex-cover problem.
3. Write the short note on approximation algorithm.
4. Explain approximation algorithm with an appropriate example.

CHAPTER

40

Randomized Algorithm

40.1 Introduction

Historically, the study of randomized algorithms was spurred by the discovery by **Miller** and **Robin** in 1976 that the problem of determining the primality of a number can be solved efficiently by a randomized algorithm. At that time, no practical deterministic algorithm for primality was known. Given an integer 'n', the problem of deciding whether 'n' is a prime is known as primality testing.

A randomized algorithm is defined as an algorithm that is allowed to access a source of independent, unbiased bits, and it is then allowed to use these random bits to influence its computation. An algorithm is randomized if its output is determined by the input as well as the values produced by a random-number generator.

A randomized algorithm is one that makes use of a randomizer such as a random number generator. Some of the decisions made in the algorithm depend on the output of the randomizer. Since the output of any randomizer might differ in an unpredictable way from run to run, the output of a randomized algorithm could also differ from run to run for the same input. So, the execution time of a randomized algorithm could also vary from run to run for the same input.

In common practice, this means that the machine implementing the algorithm has access to a pseudo-random number generator. The algorithm typically uses the random bits as an auxiliary input to guide its behaviour, in the hope of achieving good performance in the "average case". Formally, the algorithm's performance will be a random variable determined by the random bits, with (hopefully) good expected value ; this expected value is called the *expected runtime*. The "worst case" is typically so unlikely to occur that it can be ignored.

Randomized algorithms are particularly useful when faced with a malicious "adversary" or attacker who deliberately tries to feed a bad input to the algorithm. It is for this reason that randomness is ubiquitous in cryptography. In cryptographic applications, pseudo-random numbers cannot be used, since the adversary can predict them, making the algorithm effectively deterministic. Therefore either a source of truly random numbers or a cryptographically secure pseudo-rando number generator is required. Another area in which randomness is inherent is quantum computing.

Randomized algorithms can be categorized into *two* classes :

1. Las Vegas algorithms
2. Monte Carlo algorithms

Las Vegas algorithm always produces the same (correct) output for the same input. The execution time of a Las Vegas algorithm depends on the output of the randomizer. If we are lucky, the algorithm might terminate fast, and if not, it might run for a longer period of time. In general, its runtime for each input is a random variable whose expectation is bounded.

The second is algorithms whose output might differ from run to run for the same input. These are called **Monte Carlo algorithms.** Consider any problem for which there are only two possible answers, say yes and no. If a Monte Carlo algorithm is used to solve such a problem, then the algorithm might give incorrect answers depending on the output of the randomizer. We require that the probability of an incorrect answer from a Monte Carlo algorithm be low.

Typically, for a fixed input, a Monte Carlo algorithm does not display much variation in execution time between runs, whereas in the case of a Las Vegas algorithm this variation is significant.

40.2 Applications

40.2.1 Randomized Quick Sort Algorithm

- In randomized quick sort, we will pick randomly an element as the pivot for partitioning.
- The expected runtime of any input is $O(n \log n)$.

Analysis of Randomized Quick Sort

- Let $s(i)$ be the ith smallest in the input list S.
- X_{ij} is a random variable such that $X_{ij} = 1$ if $s(i)$ is compared with $s(j)$; $X_{ij} = 0$ otherwise.

- Expected runtime t of randomized QS is :

$$t = E\left[\sum_{i=1}^{n} \sum_{j \geq 1} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j \geq 1} E[X_{ij}]$$

- $E[X_{ij}]$ is the expected value of X_{ij} over the set of all random choices of the pivots, which is equal to the probability p_{ij} that $s(i)$ will be compared with $s(j)$.

The randomized quick sort algorithm uses $i \leftarrow$ Random (p, r) to be the new pivot element.

Randomized-Partition (A, p, r)

1. $i \leftarrow$ Random (p, r)
2. exchange $A[r] \leftrightarrow A[i]$
3. return Partition (A, p, r)

Randomized-Quicksort (A, p, r)

1. If $p < r$
2. then $q \leftarrow$ Randomized-Partition (A, p, r)
3. Randomized-Quicksort $(A, p, q-1)$
4. Randomized-Quicksort $(A, q+1, r)$

40.2.2 Randomized Minimum-cut Algorithm

A more complex example, representative of the use of randomized algorithms to solve graph theoretic problems, is the following **randomized minimum cut algorithm :**

```
Find-min-cut(undirected graph G)
{
  while there are more than 2 nodes in G do
  {
      pick an edge (u,v) at random in G
      contract the edge, while preserving multi-edges
      remove all loops
  }
  output the remaining edges
}
```

Here, contracting an edge (u, v) means adding a new vertex w, replacing any edge (u, x) or (v, x) with (w, x), and then deleting u and v from G.

- Let k be the min-cut of the given $G(E, V)$ where $|V| = n$.
- Then $|E| \geq kn/2$.

- The probability q_1 of picking one of those k edges in the first merging step $\leq 2/n$.
- The probability p_1 of not picking any of those k edges in the first merging step $\geq (1-(2/n))$
- Repeat the same argument for the first $n-2$ merging steps.
- Probability p of not picking any of those k edges in all the merging steps $\geq (1-2/n)(1-2/(n-1)(1-2/(n-2))...(1-2/3)$.
- Therefore, the probability of finding the min-cut :

$$p \geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)...\left(1-\frac{2}{3}\right)$$

$$= \frac{(n-2)(n-3)...(1)}{n(n-1)...3}$$

$$= \frac{2}{n(n-1)}$$

- If we repeat the whole procedure $n^2/2$ times, the probability of not finding the min-cut is at most

$$(1-2/n^2)^{n^2/2} \cong 1/e$$

Randomized Min-cut is a **Monte Carlo Algorithm.**

40.2.3 There are some interesting complexity classes involving randomized algorithms

- Randomized Polynomial time (*RP*)
- Zero-error Probabilistic Polynomial time (*ZPP*)
- Probabilistic Polynomial time (*PP*)
- Bounded-error Probabilistic Polynomial time (*BPP*)

1. *RP Definition.* The class *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 Σ^*.

$$x \in L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] \geq \frac{1}{2} \qquad \Pr \text{ means probability.}$$

$$x \notin L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] = 0$$

Independent repetitions of the algorithms can be used to reduce the probability of error to exponentially small.

Notice that the success probability can be changed to an inverse polynomial function of the input size without affecting the definition of *RP*.

2. *ZPP Definition.* The class ZPP is the class of languages, which have Las Vegas algorithms running in expected polynomial time.

$$\text{ZPP} = \text{RP} \cap \text{co-RP}$$

(Note that a language L is in co-X where X a comlexity class if and only if it's complement $\Sigma^* - L$ is in X.)

3. PP Definition. The class *PP* consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x in Σ^*.

$$x \in L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$$

$$x \notin L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] < \frac{1}{2}$$

To reduce the error probability, we can repeat the algorithm several times on the same input and produce the output which occurs in the majority of those trials.

However, the definition of *PP* is quite weak since we have no bound on how far from 1/2 the probabilities are. It may not be possible to use a small number (*e.g.*, polynomial number) of repetitions to obtain a significantly small error proability.

4. *BPP Definition.* The class BPP consists of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x in Σ^*.

$$x \in L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] \geq \frac{3}{4}$$

$$x \notin L \quad \Rightarrow \quad \Pr[A(x) \text{ accepts}] \leq \frac{1}{4}$$

For this class of algorithms, the error probability can be reduced to $1/2n$ with only a polynomial number of iterations.

In fact, the probability bounds 3/4 and 1/4 can be changed to $\frac{1}{2}+\frac{1}{p(n)}$ and $\frac{1}{2}-\frac{1}{p(n)}$ respectively where $p(n)$ is a polynomial function of the input size n without affecting the definition of BPP.

40.3 Advantages and Disadvantages

Two of the most important advantages of using randomized algorithms are their **simplicity** and **efficiency**. A majority of the randomized algorithms found in the literature are simpler than the best deterministic algorithms for the same problems. Randomized algorithms have also been shown to yield better complexity bounds. Not only do randomized algorithms yield superior asymptotic run-time bounds, but they also have demonstrated to be competitive in practice.

The randomized algorithm performs badly only if the random-number generator produces an unlucky permutation to be sorted.

A randomized strategy is typically useful when there are many ways in which an algorithm can proceed but it is difficult to determine a way that guaranteed to be good. If many of the alternatives are good, simply choosing one randomly can yield a good strategy.

Often, an algorithm must make many choices during its execution. If the benefits of good choice outweigh the costs of bad choices, a random selection of good and bad choices can yield an efficient algorithm.

Main Disadvantages of Randomized Algorithm

1. Computer architecture tries to predict the future behaviour of algorithms to optimize its execution.
2. Randomization is useful only for very hard problems.

Randomized algorithms can be used to solve a wide variety of real-world problems like approximation algorithms, they can be used to more quickly solve tough *NP*-complete problem. An advantage over the approximation algorithms, however, is that a randomized algorithm will eventually yield an exact answer if executed enough times.

BIBLIOGRAPHY

Cormen, Thomas H. "*Introduction to algorithms*", Cambridge, Mass : MIT Press ; New York : McGraw-Hill, c1990

Aho, Alfred V., "*Data structures and algorithms*", Reading, Mass : Addison-Wesley, c1983.

Aho, Alfred V., "*The design and analysis of computer algorithm*". edited by Attallah, Mikhail J. "*Algorithms and theory of computation handbook*", CRC Press, 1998 (QA76.9.A43 1999).

Baase, Sara., "*Computer algorithms : introduction to design and analysis*", Reading, Mass : Addison-Wesley Pub. Co., c1978.

Even, Shimon., "*Graph Algorithms*",Computer Science Press, 1979.

Garey, Michael R.,"*Computers and intractability : a guide to the theory of NP*", San Francisco : W. H. Freeman, c1979.

Gibbons, Alan M., "*Efficient Parallel Algorithms*", Cambridge University Press, 1988.

Greene, Daniel H., "*Mathematics for the analysis of algorithms*", Boston : Birkhauser, c1981.

Goodrich T. Michael, "*Algorithm Design - Foundations, Analysis & Internet Examples* "

Gonnet, Gaston H., "*Handbook of algorithms and data structures : in Pascal and C*", Wokingham, England ; Reading, Mass : Addison-Wesley Pub. Co.,

Gonnet, Gaston H. and Baeza-Yates, Ricardo editors, "*Handbook of Algorithms and Data Structures in Pascal and C*", Addison-Wesley, 1991.

Harel, David., "*Algorithmics : the spirit of computing*", second edition Addison-Wesley, 1992.

Horowitz, Ellis., "*Fundamentals of computer algorithms*", Potomac, Md : Computer Science Press, c1978.

Smith, Jeffrey D., "*Design and Analysis of Algorithms*", PWS-Kent, 1989.

Kingston, Jeffrey Howard., "*Algorithms and data structures : design, correctness, analysis*", Sydney : Addison-Wesley, 1990.

Koren, Israel., "*Computer Arithmetic Algorithms*", Prentice-Hall, 1993.

Kozen, Dexter C. "*The design and analysis of algorithms*", 1992.(QA76.9.A43 K69 1992).

Kreher, Donald L. and Stinson, Douglas Combinatorial Algorithms : Generation, Enumeration and Search, (CRC Press, 1998)

Kruse, Robert L., "*Data Structures and Program Design*", second edition Prentice-Hall, 1996.