Chapter 7 Randomized ALGs

Key Points

- Understand Pseudo-Random Generation ALG (PRGA)
- Master the designs of:
    - Numerical Randomized ALG
    - Sherwood ALG
    - Las Vegas ALG
    - Monte Carlo ALG

Basics

During the execution of randomized ALGs

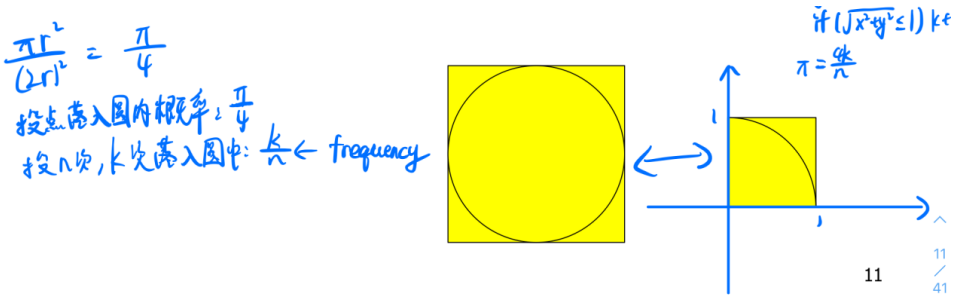
- the next step can be randomly selected
- stochastic choice
    - **time saving** as compared to optimal choice
    - largely reduced the complexity
    - approximate the optimal solution
    - may fail to yield a solution
    - also may lead to a wrong with small probablity
- The same randomized ALG applying to the same instance of a problem for two times may lead to entirely different execution time and solutions

Randomized ALGs

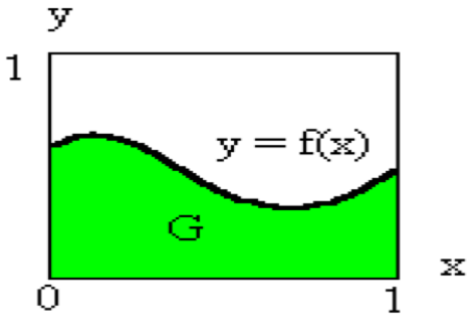
- Four categories
    - Numerical Randomized ALG
    - Sherwood ALG
    - Las Vegas ALG
    - Monte Carlo ALG

Numerical Randomized ALG

- Solve numerical problem
- Yield an approx. solution and its accuracy increases as the computation time increases
- A satisfactory solution can be given when computation of an exact solution is impossible or unnecessary
- Examples
    - Calculate π



- Calculate definite integral



  - $f(x)$, **a continuous function,** $0 \le x \le 1, 0 \le f(x) \le 1$
  - **Calculate** $I = \int_0^1 f(x)dx = G$
- Solve nonlinear equations

Sherwood ALG

- Always obtain a correct solution
- If the complexity of a deterministic ALG in the worst case differs much from that on the avg, randomness can be introduced to design a Sherwood
  - eliminate or reduce the difference btw. good and bad instances
- The essence of Sherwood ALG is not to avoid the worst case, but to eliminate the correlation btw. the worst case and a specific case
- cannot be directly reconstructed to a Sherwood ALG: **random preprocessing**
  - shuffle the input instance to achieve the same effect

Quick Sort

- performance depends on if the division is symmetric
- the worst case would be dividing the array into 2 subarrays with sizes of n-1 and 1 separately

- **If unsymmetrical**

  $T(n)= O(n^2)$

$$T(n)=\begin{cases} O(1) & n \le 1 \\ T(n-1)+O(n) & n > 1 \end{cases}$$

- **If symmetric**

  $T(n)= O(n\log n)$

$$T(n)=\begin{cases} O(1) & n \le 1 \\ 2T(n/2)+O(n) & n > 1 \end{cases}$$

Random Ver. of Quick Sort

- designed to improve the performance
- Before each partition, the pivot is randomly selected from a[p:r] so that the expected partition is symmetric

Las Vegas ALG

- LV improves the efficiency of deterministic ALGs
- LV either yields a correct solution or fails
- LV can run many times until a solution is found

```
void obstinate(Object x, Object y)
{  // repeatedly invoke LV(x, y) until a solution y is found
     bool success= false;
     while (!success) success=LV(x, y);
}
```

LV returns a bool variable, two parameters:
1) the input; 2) the solution if successful

- Once the solution is found, it's correct
- The prob. of obtaining a sol. is proportional to the computation time
- The prob. of failure can be reduced to arbitrarily small if increasing its running times

LV ALG Analysis

- prob that LV(x,y) succeeds in finding a solution for the input x: p(x)
    - p(x) > 0
- the avg. time that obstinate returns a solution for the instance x: t(x)
- avg. time obstinate succeeds: s(x)
- avg. time obstinate fails: e(x)

$$t(x) = p(x)s(x) + (1 - p(x))(e(x) + t(x))$$

$$t(x) = s(x) + \frac{1 - p(x)}{p(x)} e(x)$$

LV ALG for N Queen Problem

- Randomly put queens on the chessboard line by line ensuring that the newly placed queen doesn't attack the placed ones
- Until N queens are placed (return true)

  Or no queen can be placed (return false)

Implementation

- 对kth皇后找出所有feasible solution
    - 记下solution的个数
- count>0，随机选一个一个
- 循环结束，return (count>0)

LV + BT for N Queen

- $X=\langle x_1, x_2, \cdots, x_n \rangle$
- LV for the first k components
- BT for the next n-k components until a feasible solution is found or fail to find

- The more the queens are randomly placed, the less time is required for the BT
- But the more probability that the ALG fails

Why does the execute time increase as the number of queens randomly put increases?

- LV部分：增加
    - LV是Obstinacy的

Monte Carlo ALG

- both rdm & det ALGs can't guarantee to always find a correct solution
- MC ALGs generally find a correct solution with high prob.
    - usually difficult to determine if the solution is correct
- A MC ALG always give a solution, but the solution sometimes is wrong and difficult to prove its correctness

Application Scenario

- Solve an exact solution for problems where approx. solution is meaningless
- A solution is always given, but its correctness is not guaranteed
- The probability of a correct solution is proportional to the computation time
- The correctness of a solution cannot be effectively determined

A Consistent p-correct MC ALG

p-correct

- the probability that a MC ALG finds a correct solution for any instance is not smaller than p

  $\frac{1}{2} < p < 1$
- the MC ALG is p-correct and $p - \frac{1}{2}$ is the advantage of the MC ALG

Consistent

- A MC ALG would not give 2 different correct solutions for any instance 正确答案是唯一的

Increase the Probability that MC finds a Correct Solution

- For a consistent p-correct MC ALG, the probability that it finds a correct solution can be increased by executing it many times and choosing the solution that occurs the most.
- The prob. that a MC ALG gives a wrong solution can be arbitrarily small

例：用MC ALG对一个问题求解，给出正确答案的概率为$\frac{1}{2}-\epsilon$，则重复$2m-1$次，一个答案出现了k次

- 若为right answer，则 $k \geq m$
- 给出正确答案的概率

$$\sum_{k=m}^{2m-1} C_{2m-1}^{k}(\frac{1}{2}+\epsilon)^{k}(\frac{1}{2}-\epsilon)^{2m-1-k} = 1-\sum_{k=0}^{m} C_{n}^{k}(\frac{1}{2}+\epsilon)^{k}(\frac{1}{2}-\epsilon)^{2m-1-k}$$

True-based MC ALG

True-based MC ALG

- MC(x): the MC ALG for a specific decision problem
    - return true: always correct
    - return false: may be wrong

A consistent p-correct true-based MC ALG

- revoking MC(x) k times
    - the prob. that it find a correct solution can be increased to $(1-(1-p)^{k})$
        - 每次给出 True / False
        - 偏真的MC，所以只要给出一个 True，就找到了正确的解
        - k次全为False的概率为 $(1-p)^{k}$，即对应的没有找到正确解
        - 对k次没找到正确解取反，就是找到正确解的概率

$$1-(1-p)^{k}$$

Major Element

Traditional Method

- Make statistics of the occurrence of elements in T

MC ALG for Major Element

- Basic idea
    - Randomly pick T[i] and then compute its occurrence in T 用遍历方法
    - If its occurrence is larger than n/2, return true, else false

```
template<class Type>
bool Majority(Type *T, int n)
{ // determine if there is a major element
   int i=rnd.Random(n)+1;
   Type x=T[i];    // randomly pick an element
   int k=0;
   for (int j=1;j<=n;j++)
    if (T[j]==x) k++;
   return (k>n/2);
}
```

- Success ratio analysis

  A consistent ½-correct true-based MC ALG

  - If there is a major element, it is picked with a prob. larger than ½ and thus the success ratio is larger than ½
  - return true, then the answer must be correct
  - return false, its correctness is not sure
- If we run the MC ALG k times, the error probability is reduced to 2^{-k}

```
template<class Type>
bool MajorityMC(Type *T, int n, int k)
{// repeatedly invoke Majority
   for (int i=1;i<=k;i++)
    if (Majority(T,n)) return true;
   return false;}
```

For any ε>0, majorityMC repeatedly invokes majority $\lceil \log(1/\varepsilon) \rceil$ times, its error probability is smaller than ε and time complexity $O(n\log(1/\varepsilon))$

MC ALGs vs. LV ALGs

- MC
  - Give a solution as far as possible, but the correctness of the solution is not guaranteed
  - The correctness of a solution cannot be effectively verified
- LV
  - Give a solution as far as possible, but the solution is not guaranteed to obtain
  - The correctness of a solution is easy to verify
- Commonality: the probability of obtaining a correct solution is proportional to the computation time