MADC 201R03 Randomized Algorithms
Unit I

N. Sairam

sairam@cse.sastra.edu

School of Computing, SASTRA University, Thanjavur.

Outline of the Presentation

Overview of the Syllabus

Overview of the Syllabus

- Unit I
  - Introduction: RandQS, Mincut, LasVegas, MonteCarlo
  - Game Theoretic Techniques
  - Moments and Deviations: Occupancy problem, Markov and Chebyshev Inequalities, Coupon Collector Problem
- Unit II
  - Tail Inequalities: Chernoff bound, Wiring Problem, Martingales
  - Probabilistic Method: Max SAT, Expanders, Random Walks
- Unit III
  - Algebraic Techniques: Finger Printing, Freivald Tech., Perfect Matchings, Interactive Proof System
  - Data Structures: Random Treaps, Skiplists,Hashing
  - Geometric Algorithms: Convex Hull, Delauny Triangularization

Overview of the Syllabus Contd..

- Unit IV
  - Graph Algorithms: All Pairs Shortest Path, Minimum Spanning Tree
  - Approximate Counting: DNF Counting
- Unit V
  - Parallel and Distributed Algorithms:Byzantine Agreement, k-server problem
- References:

1. Rajeev Motwani, Prabhakar Raghavan, "Randomized Algorithms", Cambridge University Press, 2006
2. Devdatt P. Dubhashi , Alessandro Panconesi, "Concentration of Measure for the Analysis of Randomized Algorithms", Cambridge University Press; 1 edition, 2009

SASTRA UNIVERSITY

Introduction

Randomized Algorithm

Definition

An algorithm that makes random choices during execution

- Example: Randomized Quick Sort

Randomized Quick Sort

1. Input: A set of numbers S
2. Output: The elements of S sorted in increasing order
   - 2.1 Choose an element 'y' uniformly at random from S
   - 2.2 By comparing each element of S with 'y', determine the set s_1 of elements smaller than 'y' and the set s_2 of elements larger than 'y'
   - 2.3 Recursively sort s_1 and s_2. Output the sorted version of s_1, followed by y and the sorted version of s_2

Running time of RandQS

Theorem

The expected number of comparisons in an execution of RandQS is atmost $2nH_n$

Proof.

Refer BB □

Principal Advantages of RA

1. Runs faster than the best known deterministic algorithm
2. Many Randomized Algorithm are simpler to describe and implement

Las Vegas Algorithm

- Algorithm that always gives the correct solution and its running time may vary from one run to another
- It is said to be efficient Las Vegas if on any input its expected running time is bounded by polynomial function of the Input size

Min Cut Algorithm

Basic Definitions

Definition (Independent Events)

Two events ϵ_1 and ϵ_2 are said to be independent if the probability that they both occur is given by $Pr[\epsilon_1 \cap \epsilon_2]$=$Pr[\epsilon_1]XPr[\epsilon_2]$

Definition (General Case)

If ϵ_1 and ϵ_2 are not necessarily independent then $Pr[\epsilon_1 \cap \epsilon_2] =$ $Pr[\epsilon_1 \mid \epsilon_2]Pr[\epsilon_2]$=$Pr[\epsilon_2 \mid \epsilon_1]Pr[\epsilon_1]$

In general,

$Pr[\cap_{i=1}^{k}\epsilon_i]$=$Pr[\epsilon_2 \mid \epsilon_1]Pr[\epsilon_1]XPr[\epsilon_3 \mid \epsilon_1 \cap \epsilon_2]..Pr[\epsilon_k \mid \cap_{i=1}^{k-1}\ \epsilon_i]$

Graph Theoretic Example

Definition (Multigraph)

A graph containing multiple edges between any pair of vertices

Definition (Cut in a graph G)

Let G be a connected, undirected multigraph with 'n' vertices. A Cut in G is a set of edges whose removal results in G broken into two or more components

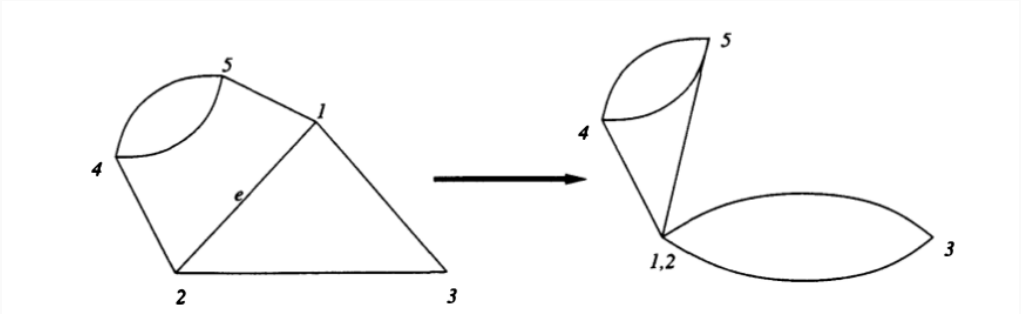
Definition (Mincut in a graph G)

A min cut is a cut of minimum cardinality

SASTRA UNIVERSITY

Simple Algorithm to find a min cut of a graph

- Input: A multi graph G
- Output: A Cut C in G
  1. Pick an edge uniformly at random and merge the two vertices at its end points as shown below.



  2. If as a result there are several edges between some pairs of(newly formed) vertices, retain them all. Edges between the vertices that are merged are removed so that there are never self loops$\rightarrow$ Contraction of that edge $\Longrightarrow$ number of vertices of G decreases by one

Min cut algorithm contd..

3. The algorithm continues the contraction process until only two vertices remain
4. The set of edges between these two vertices is a cut in G and is output as a candidate min cut

Theorem (Correctness of the min cut algorithm)

The algorithm is correct with probability atleast $\frac{2}{n^2}$

Proof.

Refer BB

□

Monte Carlo Technique

- Algorithm that may sometimes produce incorrect solution is called an Monte Carlo Algorithm
- Probability of such an incorrect solution may be bounded
- If both the running time and the quality of the solution are random variables, then sometimes it is also referred as Monte Carlo Algorithm

Two types of Monte Carlo Algorithm

- For Decision problems, there are two types of Monte Carlo Algorithm
  1. Two Sided Error
     - If there is a non zero probability that it errs when it output either YES or NO
  2. One Sided Error
     - If the probability that it errs is zero for atleast one of the possible output YES or NO that it produces

Which is better Las Vegas or Monte Carlo?

- Depends on the application
- Las Vegas Algorithm$\rightarrow$ A Monte Carlo Algorithm with error probability zero

Efficient Monte Carlo Algorithm

- If on any input the worst case running time of Monte Carlo Algorithm is bounded by a polynomial function of the input size then it is called an efficient MC algorithm

Binary Planar Partition

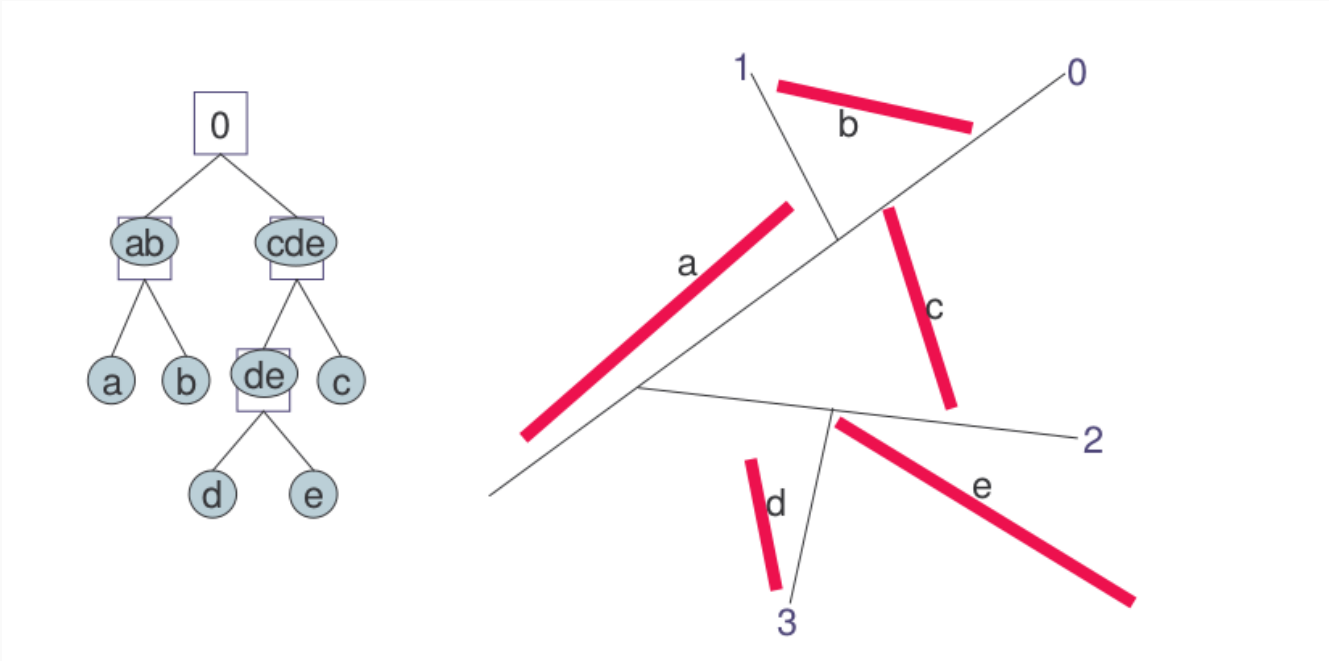
Binary Planar Partition

Definition

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

1. Every internal node of the tree has two children
2. Each node 'v' of a tree is associated with r(v) called the region of the plane
3. Each internal node 'v' of the tree is associated with a line l(v) that intersects r(v).
4. The region corresponding to the root is the entire plane
5. The region r(v) is partitioned by l(v) into two regions r1(v) and r2(v) which are regions associated with the children of

Example



Auto Partition

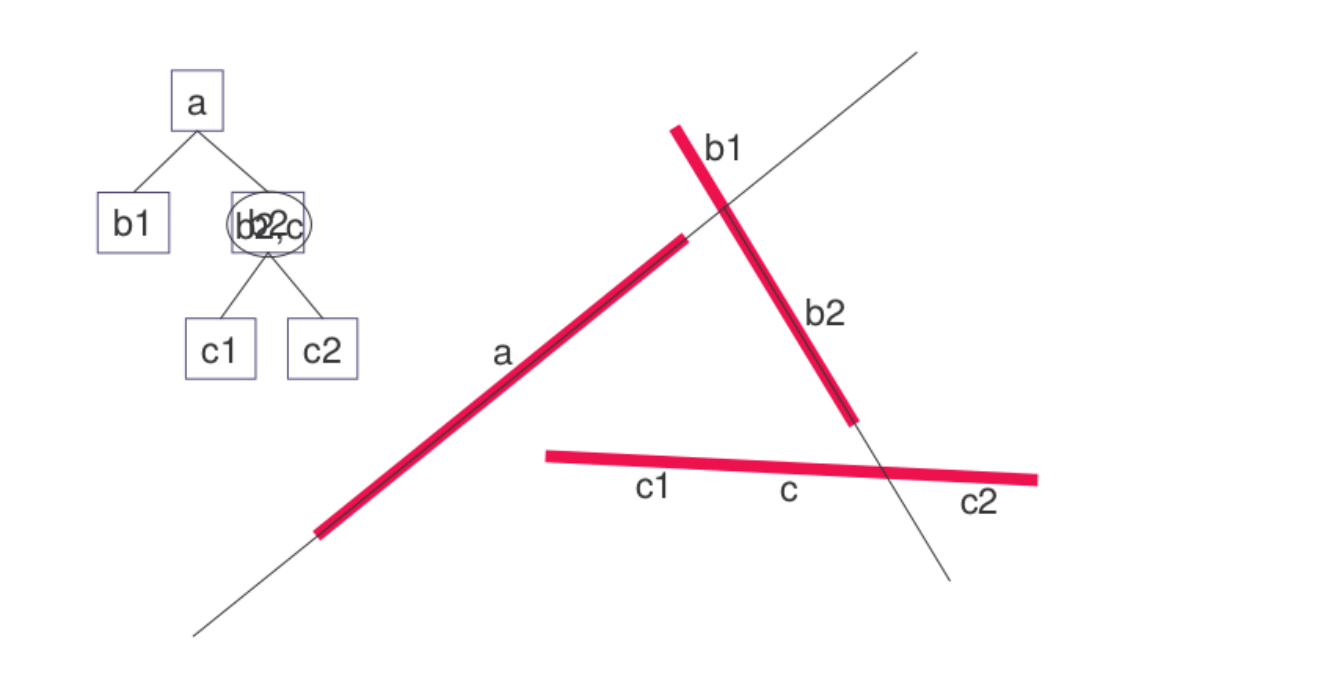
Definition

For a line segment s, l(s) denote the line obtained by extending s on both sides to ∞. For S=$\{s_1,s_2,\ldots,s_n\}$ of line segments, a simple and natural class of partitions is the set of auto partitions, which are formed by only using lines from the set $\{l(s_1), l(s_2),\ldots,l(s_n)\}$ in constructing the partition

Algorithm RandAuto

- Input: A set S=$\{s_1,s_2,\ldots,s_n\}$ of non intersecting line segments
- Output: A binary auto partition P_n of S
  1. Pick a permutation π of $\{1,2,\ldots,n\}$ uniformly at random from n! possible permutations
  2. While a region contains more than one segment cut it with $I(s_1)$ where '1' is the first in the ordering π such that s_i cuts that region
- Note: A segment may lie on the boundary between two regions of the partition, then that segment is declared to lie in one region or other in the convenient way

Example



Note

- A Segment may lie on the boundary between two regions of the partition, then that segment is declared to lie in one region or other in the convenient way

Theorem

The expected size of the autopartition produced by RandAuto is O(n log n)

Proof.

Refer BB □

Probability Recurrence

Probabilistic Recurrence

- Find Algorithm
- Selecting the kth smallest of a set S of 'n' elements works as follows
  1. Pick a random element y and partition S$\setminus$ {y} into two sets S_1 and S_2(Elements $<$ y are in S_1 and $>$ y in S_2)
  2. Suppose $|S_1| =$ k-1, then y is the desired element and hence skip
  3. Otherwise, if $|S_1| \geq$ k, we recursively find the kth smallest element of S_1 else we recursively find the (K-$|S_1|$ - 1)th smallest element in S_2

One Important Question

- What is the expected number of times we pick a random element in the algorithm?
- May not be especially important for the Find algorithm, it is the kind of question that arises in the analysis of a number of parallel and geometric algorithm
- Intuitively, we expect that the size of tha residual problem in the Find algorithm is divided by a constant factor at each recursive level, so that one can expect that the number recursive invocations is O(log n)

Generalized Setting of teh Intuition

- Let g(x) be a monotone non decreasing function from the positive reals to the positive reals
- Consider a particle whose position changes at discrete time steps and is always at a positive integer
- If the particle is currently at position m $>$ 1, it proceeds at the next step to the position m-X, where X is a random variable ranging over the integers 1,...m-1
- We know that X is that E(X) $\geq$ g(m) and that X is chosen independently of the past
- The particle will always reach position 1 and the process terminate in that state

Important Question

- Assuming that the particle starts at position 'n', what is the expected number of steps before it reaches position 1?
- The position of the particle is asscoiated with the size of the problem in a recurrence call of the Find algorithm

Theorem

Let T be the random variable denoting the number of steps in which the particle reaches the position 1. Then $E(T) \leq \int_1^n \frac{dx}{g(x)}$

Proof.

Refer BB □

Computation Model and Complexity Classes

Deterministic Turing Machine

Definition

It is a quadruple M=(S,$\sum$, δ,s) where S$\rightarrow$ Finite set of states, s$\in$S called the initial state, $\sum \rightarrow$ finite set of symbols used by M includes special symbols BLANK and FIRST, δ is the transition function from SX$\sum \rightarrow$ (S $\cup${HALT,YES,NO})X$\sum$X{$\leftarrow$,$\rightarrow$,STAY}

- The machine has three halting states HALT,YES and NO(Reject)
- The input to the TM is generally thought of as being written on the tape, unless otherwise specified, the machine may read fromand write on this tape
- The machine begins in the initial state s with its cursor at the first symbol of the input x (left end of the tape)
- This symbol is always FIRST

Deterministic TM

- The rest of the input is a string of finite length from $(\sum \setminus \{BLANK, FIRST\})^*$
- Leftmost BLANK in the input tape identifies the end of the input string
- The Transition function dictates the actions of the machine and may be thought of as its program
- In each step, the machine reads the symbol α of the input currently pointed to by the cursor
- Based on this symbol and the current stack of machine it chooses the next state, a symbol β to overwritten on α and a cursor motion direction from $\{\leftarrow, \rightarrow, \text{STAY}\}$

Deterministic TM

- The transition function is designed to ensure that the cursor never falls off the left end of the input identified by FIRST
- The machine may overwrite the BLANK symbol
- An algorithm corresponds to a TM that always halts

Probabilistic TM

Definition

A TM augmented with the ability to generate an unbiased coin flip in one step. It corresponds to a randomized algorithm.

- On any input x, a probabilistic TM accepts x with some probability
- An algorithm accepting or rejecting an input and similarly a Randomized Algorithm accepting or rejecting input with some probability

RAM(Random Access Memory Model)

- Model of computation when describing and analyzing algorithm
- Following types of operations involving registers and main memory can be performed
  1. I/O Operations
  2. Memory Register Transfers
  3. Indirect Addressing
  4. Branching
  5. Arithmetic Operations

RAM Models

- Each register or memory location may hold an integer that can be accessed as a unit, but an algorithm has no access to the representation of the number
- Arithmetic operations permitted are +,-,X,/
- In addition, an algorithm can compare two numbers and evaluate the square root of a positive number

Types of RAM Models

- Based on the cost used for measuring the running time of a running program
  1. Unit Cost RAM(Uniform RAM)
     - Each instruction can be performed in one time step
     - Believed to be too powerful since there is no known polynomial time simulation of this model by a TM
  2. Log Cost RAM
     - Realistic Version
     - Each instruction requires time proportional to the logarithm of the size of its operands

Complexity Classes

- Some basic complexity classes involve randomized algorithm
- Model of computation assumed: Turing Machines
- Can be substituted by a log cost RAM or Unit cost RAM
- Example: SAT problem
- SAT$\rightarrow$ An instance consists of a set of clauses in CNF
- Variables: May appear in either uncomplemented or complemented form in a clause
- Literals: The uncomplemented or complemented variables in a clause are known as literals(un negated and negated literals)
- Satisfied Clause: A clause is said to be satisfied if atleast one the literals in it is TRUE

Complexity Classes

- Solution: Either an assignment of Boolean Values to the variables that ensures that every clause is satisfied(truth assignment) or a negative answer that it is not possible to assign input so as to satisfy all the clauses simulataneously
- Decision version of the problem:
  - Abbreviated is SAT seeks YES or NO answer

Complexity Classes

- Language Recognition Problem
  - Any decision problem can be recognized as a language recognition problem
  - Let $\sum$ be a finite alphabet with $\sum=\{0,1\}$, s$\in \sum^*$, $|s|$ is the length of the string s and L$\subseteq \sum^*$ called the collection of strings over $\sum$
  - Language recognition problem is to decide whether x $\in \sum^*$ belongs to L
- An algorithm solves a language recognition problem for L by accepting any input string contained in L and rejecting any input not in L
- SAT problem can be designed in the form of language recognition problem by devising a suitable encoding of formula as bit strings

Complexity Classes

- Complexity class
  - A collection of languages all of whose recognition can be solved under prescribed bounds on the computational resources
  - Interested in various forms of efficient algorithms where efficient is defined as polynomial time
  - Polynomial running time $\rightarrow$ on any input of length 'n' halts within O(1) time

Complexity Classes

Definition (Class P)

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

Definition (Class NP)

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

Useful View of P and NP

- P - Consists of all languages L such that for any x$\in$L, a proof of the membership can be found and verified efficiently
- NP-Consist of all languages L such that for any x$\in$L, a proof the membership can be verified efficiently
- P$\subseteq$NP, but it is not known whether P = NP
- If P = NP, the existence of an efficiently verifiable proof $\Longrightarrow$ it is possible to actually find such a proof efficiently

co-C

- co-C $=\{L:\bar{L} \in C\}$ for any complexity class C
- It is obivious that P=co-P and P$\subseteq$NP$\cap$co-NP
- We do not know whether P = NP $\cap$ co-NP or NP=co-NP, although both statements are widely believed to be false

Deterministic and non deterministic complexity classes for different bounds on the running time

- Exponential time $\rightarrow$ Running time which is $2^{p(n)}$ for some polynomial p(n) in the input size
- Allowing exponential time instead of polynomial time in the previous definition gives us the complexity classes EXP and NEXP
- Clearly EXP $\subseteq$ NEXP but once again we do not know whether this inclusion is strict
- On the other hand, we do not know that if P=NP then EXP=NEXP

Space Complexity

- Bound on the space used by an algorithm
- In case of TM, the space used is determined by the number of distinct positions on the tape that are scanned during an execution
- RAM $\rightarrow$ No. of words of memory required by an algorithm
- PSPACE
  - A PSPACE algorithm may run for super polynomial time
  - These classes behave differently from the time complexity classes
  - For example, PSPACE=NPSPACE and PSPACE=co-PSPACE

Polynomial reduction for a Complexity Class

- A polynomial reduction from a language $L_1 \subseteq \sum^*$ to a language $L_2 \subseteq \sum^*$ is a function f:$\sum^* \rightarrow \sum$ such that
  1. There is a polynomial time algorithm that computes f
  2. $\forall\, x \in \sum^*$, $x \in L_1$ iff $f(x) \in L_2$

NP-Hard & NP-Complete

Definition

A language L is NP-Hard if for all L' $\in$ NP, there is a polynomial reduction L' to L

- If any NP-Hard decision problem can be solved in polynomial time then so can all problems in NP

Definition

A language L is NP-Complete if it is in NP and is NP-Hard

NP-Complete Problems

- The decision problems corresponding to NP Complete languages are the hardest problems in NP
- NP-Completeness applies only to decision problems
- The Optimization problem corresponding to an NP Complete decision problem is NP-Hard, but is not NP-complete because it is not in NP by definition

RP Class

Definition

The class RP(for randomized polynomial time) consists of all languages L that have randomized algorithm A running in worst case polynomial time such that for any input $x \in \sum^*$ $x \in L \implies Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$, $x \notin L \implies Pr[A(x) \text{ accepts}] = 0$

- The choice of bound on the error probability $\frac{1}{2}$ is arbitrary
- RP algorithm is a Monte Carlo algorithm that an error occurs only when $x \in L$. This is referred to as one sided error

co-RP

- Consists of languages that have polynomial time RA erring only in the case when $x \notin L$
- A problem belonging to both RP and co-RP can be solved by a RA with zero sided error (LasVegas)

ZPP

Definition

The class ZPP(Zero error Probabilistic Polynomial Time) is the class of languages that have Las Vegas algorithm running in expected Polynomial time

Problems that have randomized Monte Carlo algorithm making two sided errors

Definition

The class PP(Probabilistic Polynomial time)consists of all languages L that have a RA running in worst case polynomial time such that for any input $x \in \sum^*$, $x \in L \implies \Pr[A(x) \text{ accepts}] > \frac{1}{2}$ and $x \notin L \implies \Pr[A(x) \text{ accepts}] < \frac{1}{2}$

BPP class

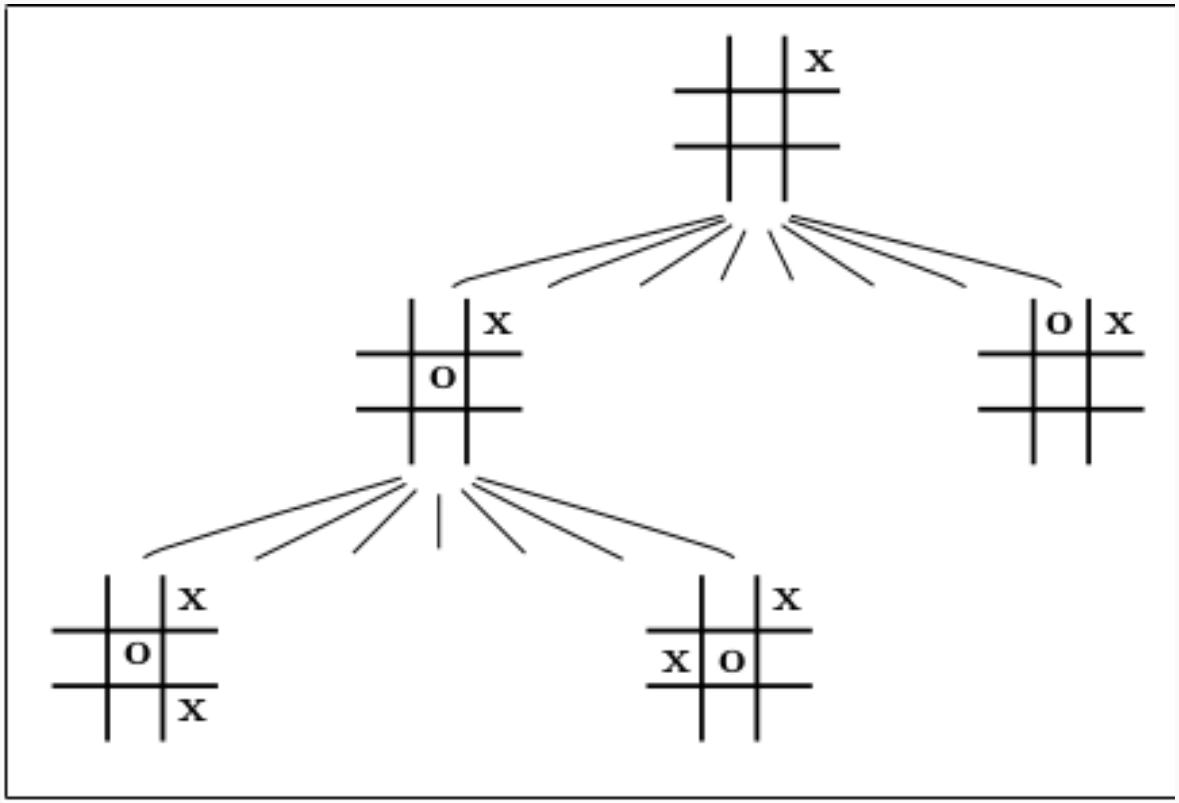
Definition

The class BPP(Bounded error Probabilistic Polynomial time)consists of all languages L that have a RA 'A' running in worst case polynomial time such that for any input $x \in \sum^*$, $x \in L \implies \Pr[A(x) \text{ accepts}] \geq \frac{3}{4}$, $x \notin L \implies \Pr[A(x) \text{ accepts}] \leq \frac{1}{4}$

- RandQS$\rightarrow$ ZPP
- Mincut $\rightarrow$ BPP

Game Theoretic Techniques

Game Tree Evaluation



Game Tree Evaluation

- Game trees are extensively used in the artificial intelligence field, especially in game-playing problems
- Consider a computer that plays a game of tic-tac-toe against a user
- At the root of the game tree is an tic-tac-toe grid with the computers first move marked in (we assume the computer always goes first with the same first move)
- This root has eight children, corresponding to the eight different answering moves the user can make
- Each of these children has more children in turn, corresponding to the subsequent moves that can be made

Game Tree Evaluation

- The leaves of the tree represent the value of the game
- One player tries to maximize this value, while the other tries to minimize it
- In the case of tic-tac-toe the tree will be finite, but this need not always be the case
- The game of chess, for instance, could have an infinite game tree
- Deciding who will win a game given a starting configuration is known as the game tree problem

MIN-MAX and AND-OR Game Trees

Definition

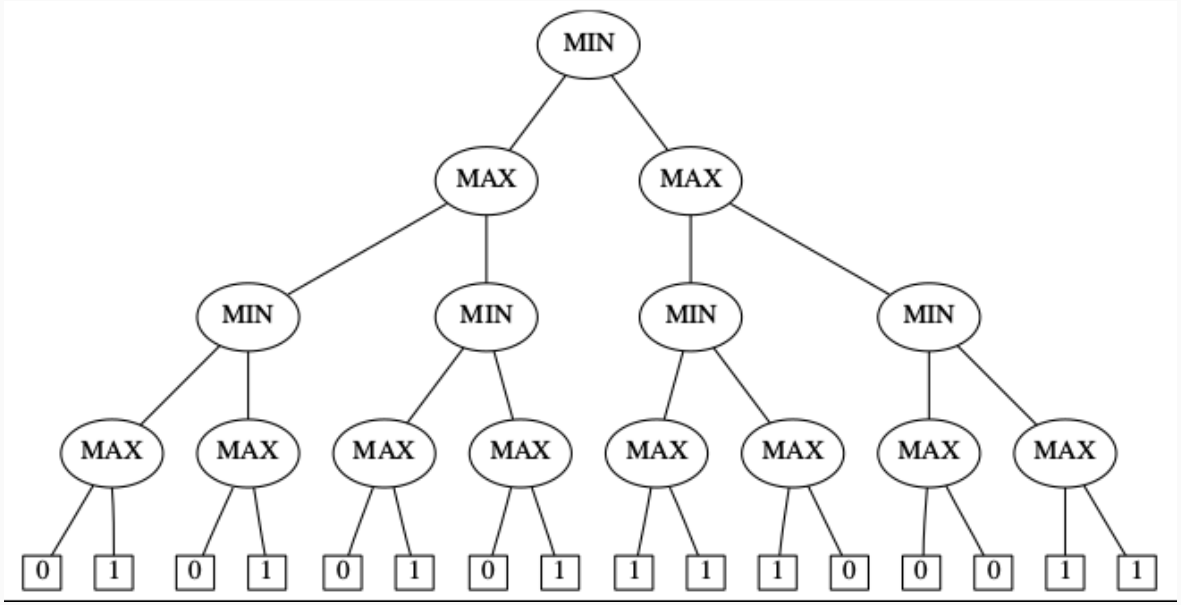
A game tree is a full, balanced, binary tree in which internal nodes at an even distance from the root are labeled MIN and internal nodes at an odd distance (including the root) are labeled MAX. Each node of the tree is a record containing four fields: "type", "l-child", "r-child" and "value". "Type" is MIN, MAX, or LEAF, "l-child" and "r-child" are pointers to the nodes left and right children respectively (which will be null in the case of a leaf), and "value" is the value returned by the node - either 0 or 1. The value returned by a MIN node is the lesser of the values of its two children, and the value returned by a MAX node is the greater of the values of its two children. Initially only the leaves have values

Some Observations

1. Every root-to-leaf path goes through the same number of MIN and MAX nodes (including the root)
2. If the number in observation is k, then the depth of the tree is 2k and the number of nodes in the tree is 2^{2k} - 1 = 4^k-1 (excluding the leaves)
3. The number of leaves is $2^{2k} = 4^k$

Example

- Let us consider the following Min Max tree



- A min-max tree of depth 2k = 2 2 = 4, with 2^{2k}-1 = 4^2-1 = 15 internal nodes and $2^{2k} = 4^2 = 16$ leaves

Example

- A game tree in which each node has d children and there are k MAX and k MIN nodes is signified by $T_{d,k}$. We will work exclusively with binary game trees, denoted by $T_{2,k}$. In binary computer science terms, a MIN node can be thought of as a logical AND and a MAX node as a logical OR
- Input and Output within a game tree

Inp	Out Min	Out Max
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1

- Our interest is evaluation
- Given $T_{2,k}$ and leaf values, what is the value returned by the root?

Deterministic Evaluation

```
1: Evaluate(node)
2: if (node.type = "LEAF") then
3:    return(node.value)
4: end if
5: if (node.type = "OR") then
6:    p1 = Evaluate(node.right)
7:    if (p1 = 1) then
8:       return p1
9:    else
10:      return (Evaluate(node.left))
11:   end if
12: end if
13: if (node.type = "AND") then
14:   p1 = Evaluate(node.left)
15:   if (p1 = 0) then
16:      return p1
17:   else
18:      return (Evaluate(node.right)
19:   end if
20: end if
```

Randomized Evaluation

- Randomized evaluation algorithm that uses coin tosses
- Whether the algorithm will evaluate the right child first or the left child first is dependent on the coin toss and is impossible for an adversary to know in advance
- It is a Las Vegas algorithm in that it always returns the correct result, and in terms of deterministic efficiency is no worse than a deterministic algorithm

Randomized Evaluation

```
1: RandEvaluate(node)
2: if (node.type = "LEAF") then
3:    return(node.value)
4: end if
5: if (node.type = "OR") then
6:    Toss a coin
7:    if (Heads) then
8:       p1 = RandEvaluate(node.left)
9:       if (p1 = 1) then
10:         return p1
11:      else
12:         return(RandEvaluate(node.right))
13:      end if
14:   end if
15:   if (Tails) then
16:      p2 = RandEvaluate(node.right)
17:      if (p2 = 1) then
18:         return p2
19:      else
20:         return(RandEvaluate(node.left))
21:      end if
22:   end if
23: end if
```

Randomized Evaluation

```
24: if (node.type = "AND") then
25:    Toss a coin
26:    if (Heads) then
27:       p1 = RandEvaluate(node.left)
28:       if (p1 = 0) then
29:          return p1
30:       else
31:          return(RandEvaluate(node.right))
32:       end if
33:    end if
34:    if (Tails) then
35:       p2 = RandEvaluate(node.right)
36:       if (p2 = 0) then
37:          return p2
38:       else
39:          return(RandEvaluate(node.left))
40:       end if
41:    end if
42: end if
```

Cost Analysis of Randomized AND-OR Game Tree Evaluation

Lemma

The expected cost of the randomized algorithm for evaluating $T_{2,k}$ is atmost 3^k

Proof.

Refer BB □

Game Theory

- Mini Max Principle:
  - Standard technique for proving a lower bound on RA
- Two Person Games
  - Consider the game Paper-Rock-Scissors
  - Played between two people who simultaneously make a sign signifying paper, rock or scissors
  - Paper beat rock by covering it, rock beats scissors by denting it and scissors beat paper by cutting it
  - When the same sign is chosen, the outcome is a draw
  - It can be represented by a pay off matrix M

$$\begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

Two Person Games

- Amount paid by column player to row player for different scenarios
    - If the number is negative then c receives money
- The game is called a Zero Sum game because the net amount won by two particpants is zero
- One of them may or may not walk away with more money at the end of the contest, but the total amount between will not have decreased or increased
- In general, we may represent any two person zero sum game with an mxn payoff matrix

Two Person Games

- m and n do not have to be equal-May be one competitor has any strategies to choose from, while the other has only a few
- The guiding force in picking a strategy is the minimization of loss
- Assume this is a zero information game, meaning that neither player has any information about the opponent's strategy
- Suppose R chooses strategy i. $\therefore$ R is guaranteed a pay off $\min_j M_{ij}$ regardless of which strategy C chooses

Two Person Games

- The optimal strategy V is one which maximizes the minimum payoff
- $V_R = \max_i \min_j M_{ij}$ is the lower bound on the payoff to R when optimal strategy is used by R
- An optimal strategy for C is one that minimizes the maximum that C will have to pay to R
- $V_C = \min_j \max_i M_{ij}$
- The following inequality holds for all payoff matrices

$$max_i min_j \ M_{ij} \ \leq \ min_j max_i \ M_{ij}$$

Two Person Games

- When V_R=V_C the game is said to have a solution and V is called the saddle point
- In the given pay off matrix(earlier slide) V_R=-1 and V_C=1. $\therefore$ the game does not have a solution
- If we consider the modified pay off matrix

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

- Here V_R=V_C=0
- But the original version has no solution

Randomized Gaming Strategies

- In game theory there are two strategies namely, a deterministic strategy known as a pure strategy and a randomized strategy is known as a mixed strategy
- A mixed strategy is a probability distribution on the set of possible strategies
- Consider a probability distribution vector, $\vec{p}$, on the rows, and a probability distribution vector, $\vec{q}$, on the columns

About $\vec{p}$ and $\vec{q}$

1. Every element in $\vec{p}$ and $\vec{q}$ is between 0 and 1
2. All the elements in $\vec{p}$ sum to 1, as do all the elements in $\vec{q}$
3. R will pick row i with probability p_i and C will pick column j with probability q_j

Expected Pay off

$$E[payoff] = \sum_{i=1}^{n}\sum_{j=1}^{m} p_i M_{ij} q_j = \vec{p}^T M \vec{q}$$

- In the randomized technique, choosing a strategy amounts to choosing a probability distribution vector from the set of all probability distributions
- The optimal strategies for R and C are

$$V_R = max_{\vec{p}} min_{\vec{q}} \vec{p}^T M \vec{q}, \;\; V_C = min_{\vec{q}} max_{\vec{p}} \vec{p}^T M \vec{q}$$

- where the min and max range over all possible probability distributions

von Neumanns Minimax Theorem

Theorem

For any two-person game, specified by M, $V_R = V_C$, i.e. all games have solutions when randomized strategies are used

Game Theory Application to Algorithm Design

- Suppose C is an algorithm designer
- R is the adversary and is responsible for designing inputs that will thwart C's algorithms
- In the payoff matrix,
  - columns = {all possible correct, deterministic, terminating algorithms for input of fixed size}
  - rows = {all possible inputs of a fixed size} and
  - M_{ij} = time taken by algorithm A_j on input I_i
  - Both sets are taken to be finite
- A pure strategy for C is the selection of one of the algorithms, and an optimal pure strategy for C is the choice of the best worst-case deterministic algorithm for solving the problem
- A pure strategy for R is the selection of one of the inputs

Complexity of Algorithm

Definition (Deterministic Complexity)

The Deterministic Complexity of a problem is the worst case running time of any deterministic algorithm for the problem

Definition (Distributional Complexity)

The Distributional Complexity of a problem is the expected running time of the best deterministic algorithm for the worst distribution on the inputs

SASTRA UNIVERSITY

Yao's Min Max Principle

Theorem

For all distributions $\vec{p}$ over I and $\vec{q}$ over A $min_{A \in A}$ $E[C(I_{\vec{p}}, A)] \leq$ max $E[C(I, A_{\vec{q}})]$

Moments and Deviations

Probability

Inclusion-Exclusion: For arbitrary events $A_1, A_2, \ldots, A_n$,

$$\mathbb{P}\left[\cup_{i=1}^{n} A_i\right] = \sum_{i=1}^{n} \mathbb{P}\left[A_i\right] - \sum_{i<j}^{n} \mathbb{P}\left[A_i \cap A_j\right] + \sum_{i<j<k}^{n} \mathbb{P}\left[A_i \cap A_j \cap A_k\right] - \ldots$$

Truncating yields upper (or lower) bound if the last term is positive (or negative). Union bound, $\mathbb{P}\left[\cup_{i=1}^{n} A_i\right] \leq \sum_{i=1}^{n} \mathbb{P}\left[A_i\right]$

Conditional Probability: For arbitrary events A and B,

$$\mathbb{P}\left[A|B\right] = \mathbb{P}\left[A \cap B\right] / \mathbb{P}\left[B\right]$$

and $\Pr(\cap_{i=1}^{n} A_i) = \Pr(A_1) \Pr(A_2|A_1) \ldots \Pr(A_n| \cap_{i=1}^{n-1} A_i)$

Independence: A and B are independent is $\mathbb{P}\left[A|B\right] = \mathbb{P}\left[A\right]$ (or equivalently $\mathbb{P}\left[A \cap B\right] = \mathbb{P}\left[A\right] \mathbb{P}\left[B\right]$.)

Random Variables

Expectation: $\mathbb{E}[X] = \sum_r r\mathbb{P}[X = r]$

Variance: $\mathbb{V}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$

Standard deviation: $\sigma_X = \sqrt{\mathbb{V}[X]}$

Theorem

$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

$\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$ *if* X *and* Y *independent.*

$\mathbb{V}[X + Y] = \mathbb{V}[X] + \mathbb{V}[Y]$ *if* X *and* Y *independent.*

SASTRA UNIVERSITY

Moment Generating Functions

Let X be a non-negative integer-valued random variable. The *probability generating function* of X is

$$G_X(z) = \sum_{i=0}^{\infty} z^i \mathbb{P}[X = i]$$

Lemma

$$\mathbb{E}[X] = G'(1).$$

$$\mathbb{V}[X] = G'' + G'(1) - G'(1)^2.$$

Markov Inequality

Theorem

Let Y be a random variable assuming only non-negative values. Then, for all $t > 0$, $P[Y \geq tE[Y]] \leq \frac{1}{t}$

Proof.

- Define f(y) = 1 if y $\geq$tE[Y] and 0 otherwise.
- Note that f(y) $\leq$ y/(tE[Y])
- P[Y$\geq$tE[Y]] = E[f(Y)]
- Then, E[f(Y)]$\leq$E[Y/(tE[Y])] = 1/t

Chebyshev Inequality

Theorem

Let X be a random variable with expectation $_X$ and standard deviation σ_X . Then for t > 0, P[$|X - \mu_X| \geq t\sigma_X$] $\leq \frac{1}{t^2}$

Proof.

- Note that P[$|X - \mu_X| \geq t\sigma_X$] = P[(X - μ_X)$^2 \geq t^2\sigma_X^2$]
- Let Y = (X - μ_X)^2 and note E[Y] = σ_X^2
- Use Markovs inequality to show P[Y $\geq t^2$E[Y]]$\leq \frac{1}{t^2}$

□