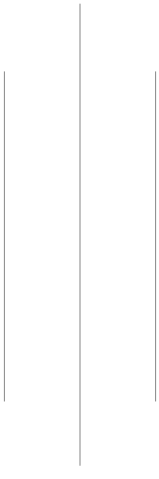
Notes On

Algorithmic Complexity



Central Department of MscCSIT,
Tu, Kirtipur

Table of Contents

Unit 1

Advance Algorithm Analysis and Design

Advance Algorithm Analysis and Design

1.0 Questions:

i. Explain accounting method of amortized analysis with example. [10 marks, 2074]
ii. Compare divide-and-conquer and dynamic approaches of algorithm design with example. [6 marks, 2071]
iii. Compare greedy method and backtracking method of algorithm design with example. [2074]
iv. what do you mean by randomized algorithm ? Explain with examples.
v. Justify the applicability of randomized algorithm with suitable example. [5 Marks, 2073]
vi. How can you implement randomized algorithm in identifying repeated elements.
vii. How dynamic programming approach can be used to solve optimal BST.

1.1 Advance Algorithm analysis technique

Amortized analysis:

- the time required to perform a sequence of data structure operations is averaged over all the operations performed.
- Amortized analysis can be used to show that the average cost of an operation is small, even though some operations within the sequence is expensive.
- Amortized analysis differs from average-case analysis in that probability is not involved;
- it guarantees the average performance of each operation in the worst case.

Three Methods of Amortized Analysis:

i. Aggregate analysis
ii. The accounting method
iii. The potential method

Aggregate Analysis:

- We show that a sequence of n operations take worstcase time T(n) in total. In the worst case, the ave. cost, or amortized cost, per operation is therefore T(n) / n.
- In the aggregate method, all operations have the same amortized cost.

Amortized Analysis of algorithm:

Example: Stack operations:

Push(S,x): pushes object x onto stack S

Pop(S): pops the top of the stack S and returns the poped object

Multipop(S,k): Removes the k top objects of stack S

The action of Multipop on a stack S is as follows:

```
Multipop (S,k)
    while not STACK-EMPTY(S) and k ≠ 0
        do POP(s)
            k ← k –1
```

The top 4 objects are popped by Multipop(S,4).

- In fact, although a single Multipop operation can be expensive, any sequence of n Push, Pop, and Multipop operations on an initially empty stack can cost at most $O(n)$. Why?
- Because each object can be poped at most once for each time it is pushed. Therefore, the number of times that Pop can be called on a nonempty stack, including calls within Multipop, is at most the number of Push, which is at most n. For any value of n, any sequence of n Push, Pop, and Multipop operations takes a total of $O(n)$ time.
- The amortized cost of an operation is the average: $O(n)/n = O(1)$.
- Amortized cost in aggregate analysis is defined to be average cost.

Accounting Method:

- In the accounting method of amortized analysis, different charges to different operations are assigned. Some operations are charged more or less than they actually cost.
- The amount we charge an operation is called its amortized cost.
- When an operation's amortized cost exceeds its actual cost, the difference is assigned to specific objects in the data structure as credit.
- Credit can be used later on to help pay for operations whose amortized cost is less than their actual cost.
- One must choose the amortized costs of operations carefully. The total credit in the data structure should never become negative, otherwise the total amortized cost would not be an upper bound on the total actual cost.

Example: Amortized analysis of Stack Operations using accounting method.

- Actual costs:
  – PUSH :1, POP :1, MULTIPOP: min(s,k).
- Let assign the following amortized costs:
  – PUSH:2, POP: 0, MULTIPOP: 0.
- Similar to a stack of plates in a cafeteria.
  - Suppose $1 represents a unit cost.
  - When pushing a plate, use one dollar to pay the actual cost of the push and leave one dollar on the plate as credit.
  - Whenever POPing a plate, the one dollar on the plate is used to pay the actual cost of the POP. (same for MULTIPOP).
  - By charging PUSH a little more, do not charge POP or MULTIPOP.
- The total amortized cost for n PUSH, POP, MULTIPOP is $O(n)$, thus $O(1)$ for average amortized cost for each operation.
- Conditions hold: total amortized cost ≥total actual cost, and amount of credits never becomes negative.

The Potential Method:

- The potential is associated with the data structure as a whole rather than with specific objects within the data structure.
- The potential method works as follows:
  ➔ We start with an initial data structure D0 on which n operations are performed.
  ➔ For each i = 1,2,...,n, we let c_i be the actual cost of the ith operation and D_i be the data structure that results after applying the ith operation on data structure D_{i-1}.
- A potential function Φ maps each data structure Di to a real number.
- Φ(Di) is potential associated with data structure Di .

- The amortized cost ac_i of the ith operation with respect to potential function Φ is defined by $ac_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.
- The amortized cost of each operation is therefore its actual cost(ci) plus the increase in potential (Φ(Di) - Φ(Di-1)) caused by ith operation.
- So, the total amortized cost of the n operations is $\sum 1 \leq i \leq n$ aci = $\sum 1 \leq i \leq n$ (ci + Φ(Di) - Φ(Di-1)) = $\sum 1 \leq i \leq n$ ci + Φ(Dn) - Φ(D0).

Randomized Algorithm:

- A probabilistic algorithm is an algorithm where the result and/or the way the result is obtained depend on chance. These algorithms are also sometimes called randomized algorithms.
- It makes use of randomizer(random number generator) to make some decision in a algorithm.
- The output and/or execution time of a randomized algorithm could also differ from run to run for the same input.
- It can be categorized into:
  i. Las vegas algorithm
  ii. Monte carlo algorithm

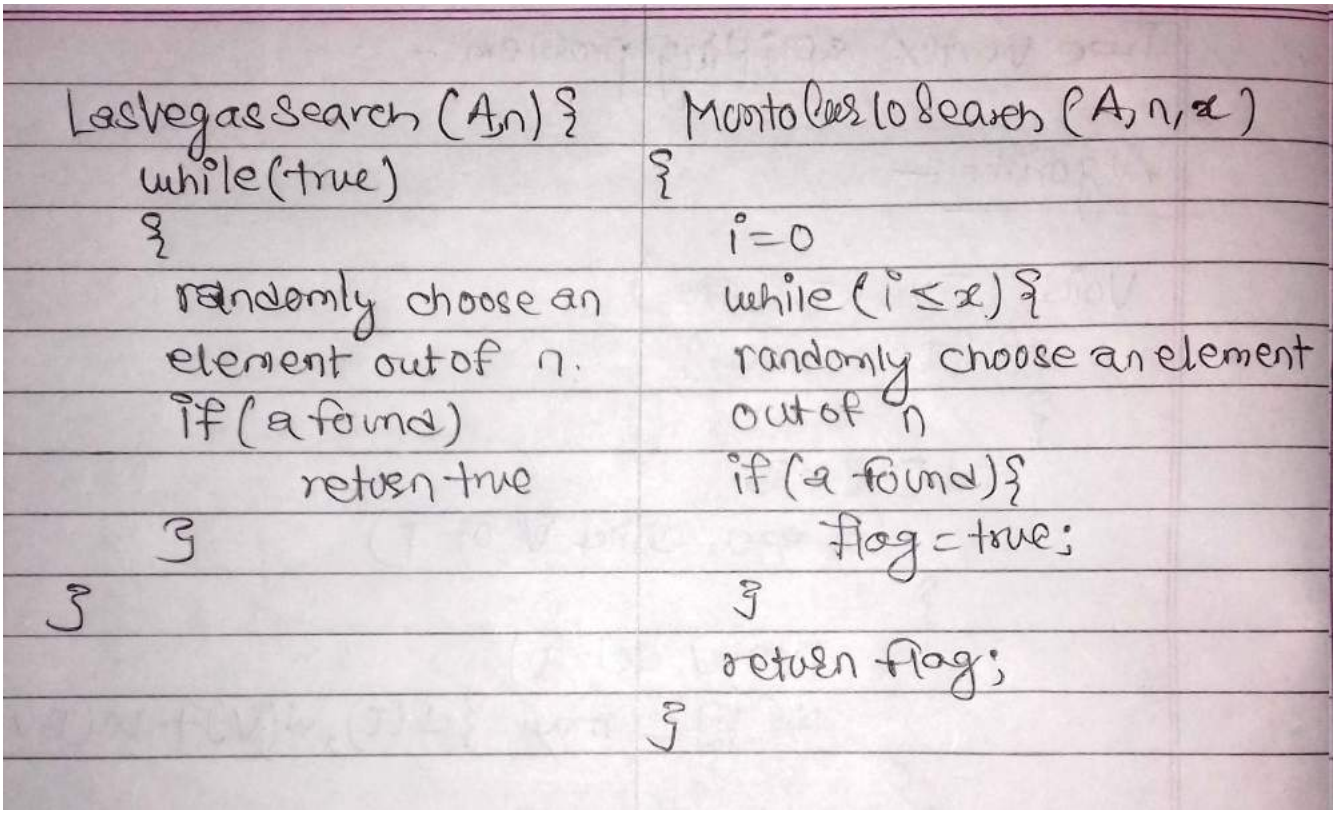
Las vegas Algorithm

- this algorithm always same (correct) output for the same input.
- Execution time of this algorithm is characterized as a random variable.
- The execution time depends on the output of randomizer.

Monte Carlo Algorithm

- Monte carlo is the algorithm whose output might differ from run to run for the same input.
- It may generate the wrong answer sometimes that depends on the output of the randomizer.

- The execution time is fixed.



1.2 Advance Algorithm Design Technique

Tree Vertex Spitting Problems:

Let T= (V, E, W) be a directed tree. A weighted tree can be used to model a distribution network in which electrical signals are transmitted. Nodes in the tree correspond to receiving stations & edges correspond to transmission lines. In the process of transmission some loss is occurred. Each edge in the tree is labeled with the loss that occurs in traversing that edge. The network model may not able tolerate losses beyond a certain level. In places where the loss exceeds the tolerance value boosters have to be placed. Given a networks and tolerance value, the TVSP problem is to

determine an optimal placement of boosters. The boosters can only placed at the nodes of the tree.

In TVSP,

v→ set of vertices

E→ set of edges

w(i,j)→ wight of edges <i,j> € E

& source vertex has in degree 0 and out vertex has out degree 0.

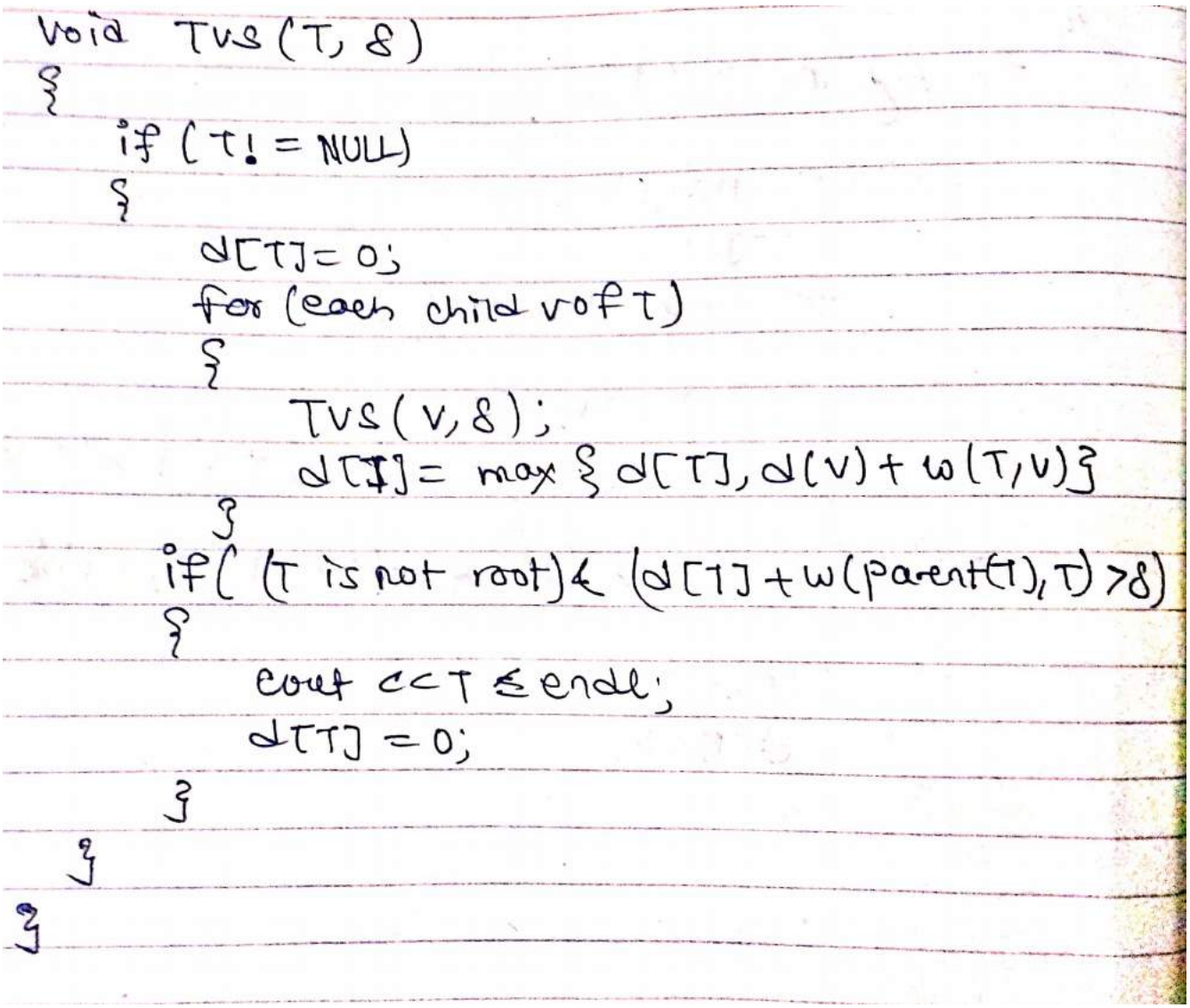
we define delay of node as:

d (u) = Max { d(v) + w(Parent(u), u)},

where d(u) → delay of node, v-set of all edges & v belongs to child(u)

δ tolerance value

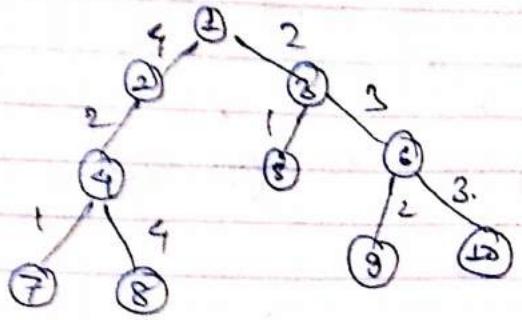
& if d (u) >= δ than place the booster.



```
void TVS(T, δ)
{
    if (T != NULL)
    {
        d[T] = 0;
        for (each child v of T)
        {
            TVS(v, δ);
            d[T] = max{d[T], d(v) + w(T, v)}
        }
        if( (T is not root) && (d[T] + w(parent(T), T) > δ)
        {
            cout << T << endl;
            d[T] = 0;
        }
    }
}
```

Fig: Algorithm for TVS problem

Tree vertex splitting.



Soln.

tolerance $\delta = 7$

& for each leaf node 7, 8, 5, 9, 10, the delay is 0.

delay of node (u) (any node) is given by

$$d(u) = \max\{d(v) + w(\text{parent}(u), u)\}$$

So calculating delay from child,

$d(7) = \max\{0 + d(4,7)\} = 1$.

Since, $d(7) < \delta$ so no booster is required.

$d(8) = \max\{0 + \delta(4,8)\} = 4$

Since, $d(7) < \delta$, so no booster is required.

$d(4) = \max\{1 + w(2,4), 4 + w(2,4)\}$

$= \max\{3, 4+3\}$

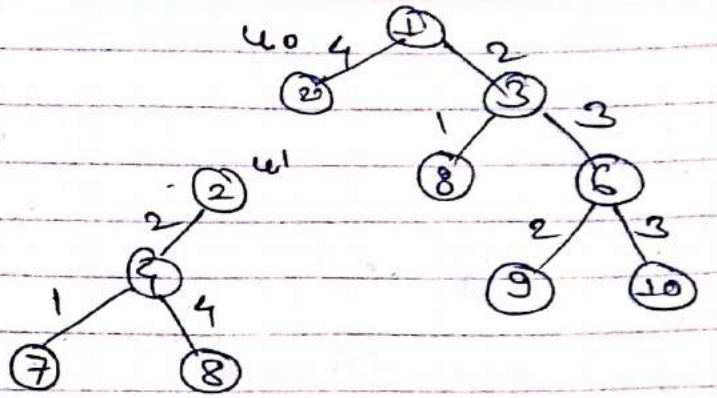
$= 7$.

Since $\delta(7) = \delta$, so no booster is required.

[note $d(T) \geq \delta$ to have booster]

$d(2) = \max\{7 + w(1,2)\}$

$= 11 > \delta$ so booster is required.



$d(9) = \max\{0+2\} = 2$

Since $d(9) < 8$ no booster is required.

$d(10) = \max\{0 + w(8,10)\} =$

$= 3$

Since $d(10) < 8$, no booster is required.

$d(6) = \max\{(2 + w(3,6)), (3 + w(3,6))\}$

$= \max\{2+3, 3+3\}$

Since $6 < 8$; no booster is required.

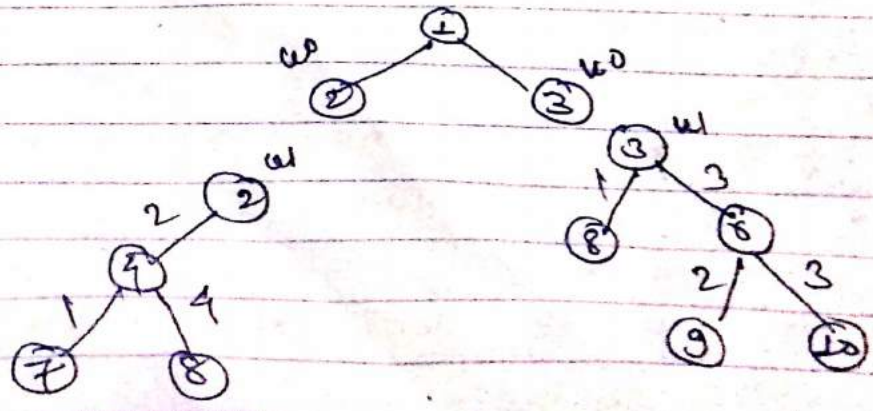
$d(8) = \max\{0 + w(3,8)\}$

$= 0 + 1 = 1$

Since $d(8) < 8$ so no booster is required.

$d(3) = \max\{1 + w(2,3), 8 + w(2,3)\}$

$= 8 > 8$, so booster is required.

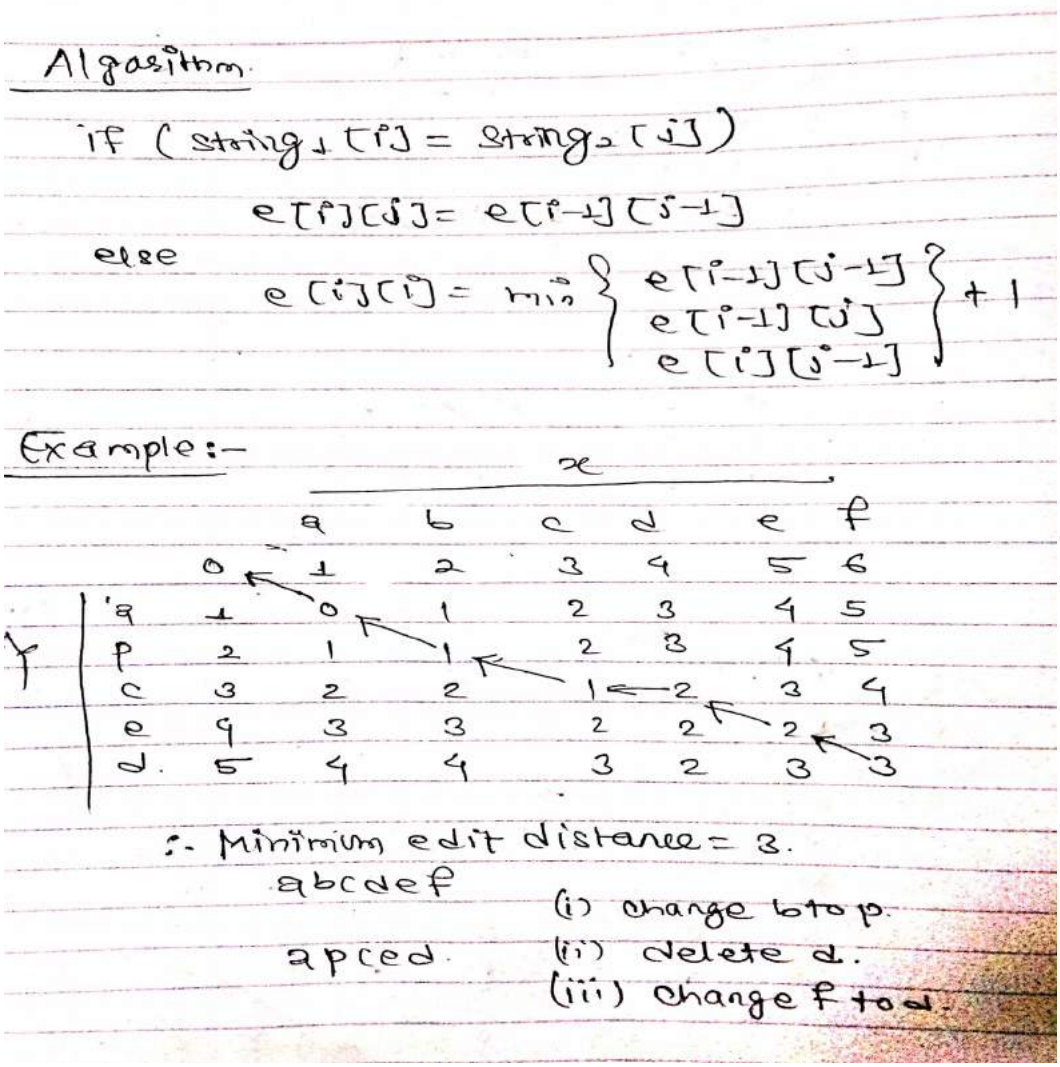


String Edition(Minimum Edit Distance)

- **Problems:** what is the minimum number of edit operations (i.e. change, delete, insert) to change one string to another. ?
- The problem can be solved by using dynamic programming.
- Given two strings:

  x= abcdef

  y= apced



Optimal BST

-

Greedy Method:

- the simplest and straightforward approach
- It takes decision on the basis of current available information without worrying about the effect of the current decision in future.
- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a local optimum at each step,
  - you will end up at a global optimum

Example: Counting Money

Suppose you want to count out a certain amount of money, using the fewest possible bills and coins.

A greedy algorithm would do this would be:

- At each step, take the largest possible bill or coin that does not overshoot
- Example: To make $6.39, you can choose:
  - a $5 bill
  - a $1 bill, to make $6
  - a 25¢ coin, to make $6.25

- A 10¢ coin, to make $6.35
- four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution

Areas of Application:

Greedy approach is used to solve many problems, such as:

- Finding the shortest path between two vertices using Dijkstra's algorithm.
- Finding the minimal spanning tree in a graph using Prim's /Kruskal's algorithm, etc.

Where Greedy Approach Fails:

In many problems, Greedy algorithm fails to find an optimal solution, moreover it may produce a worst solution. Problems like Travelling Salesman and Knapsack cannot be solved using this approach.

Backtracking:

Suppose you have to make a series of decisions, among various choices, where,

- You don't have enough information to know what to choose
- Each decision leads to a new set of choices

- Some sequence of choices (possibly more than one) may be a solution to your problem

Backtracking is a methodical way of trying out various sequences of decisions, until you find one that "works"

Backtracking Algorithm for Subset Sum

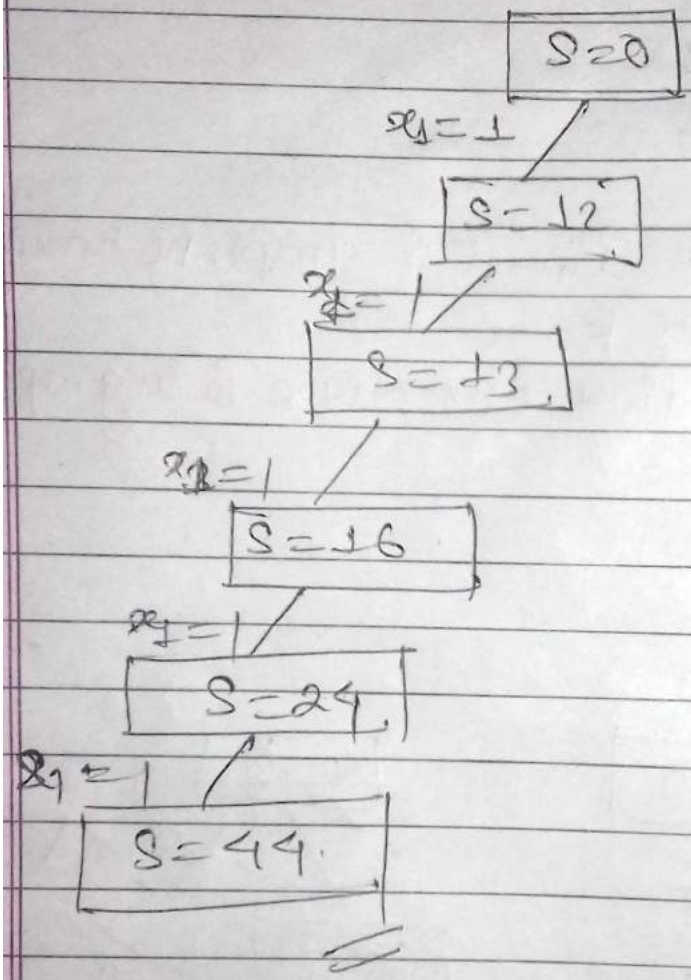
- **Problem:** Given a lists of positive integers a[1...n] and an integer t, is there some subset of a that sum exactly t ?
- Example: a = [12,1,3,8,20,50] and t = 44

Algorithm:

```
Begin
   if total = sum, then
      display the subset
      //go for finding next subset
      subsetSum(set, subset, , subSize-1, total-set[node], node+1, sum)
      return
   else
      for all element i in the set, do
         subset[subSize] := set[i]
         subSetSum(set, subset, n, subSize+1, total+set[i], i+1, sum)
      done
End
```

In the above example if target is 44 then we can implement it by

target sum = 44.



∴ we got our solution for target sum for 44.

Job Sequencing with Deadline:

In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

Solution:

Let us consider, a set of **n** given jobs which are associated with deadlines and profit is earned, iff a job is completed by its deadline. These jobs need to be ordered in such a way that there is maximum profit.

It may happen that all of the given jobs may not be completed within their deadlines.

Assume, deadline of **ith** job **Ji** is **di** and the profit received from this job is **pi**. Hence, the **optimal solution** of this algorithm is a feasible solution with maximum profit.

Feasible Solution is subset of jobs such that each job in the subset can be completed by its deadline.

Thus, D(i)>0 for 1⩽i⩽n.

**

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_job_sequencing_with_deadline.htm

**

Divide and Conquer:

- Many algorithms are recursive in nature
- In divide and conquer approach, a problem is divided into smaller problems then the smaller problems are solved independently and finally the solutions of smaller problems are combined into a solution for the large problem.

Generally, divide-and-conquer algorithms have three parts –

i. **Divide the problem** into a number of sub-problems that are smaller instances of the same problem.

ii. **Conquer the sub-problems** by solving them recursively. If they are small enough, solve the sub-problems as base cases.

iii. **Combine the solutions** to the sub-problems into the solution for the original problem.

Pros and cons of Divide and Conquer Approach:

Divide and conquer approach supports parallelism as sub-problems are independent. Hence, an algorithm, which is designed using this technique, can run on the multiprocessor system or in different machines simultaneously.

In this approach, most of the algorithms are designed using recursion, hence memory management is very high. For recursive function stack is used, where function state needs to be stored.

Application of Divide and Conquer Approach

Following are some problems, which are solved using divide and conquer approach.

- Finding the maximum and minimum of a sequence of number
- Strassen's matrix multiplication
- Merge sort
- Binary search

Q. Compare Divide and Conquer and dynamic Programming approach of algorithm design with Example.[2072, MscCsit TU]

Ans. Divide and Conquer discussed above. Dynamic programming here you go.

Dynamic Programming:

- Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of sub-problems.
- Moreover, Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.

- To solve the given problem with dynamic programming, the problem should have main two properties,
  i. **overlapping sub-problems**
  ii. **optimal substructure.**

I. Overlapping Sub-Problem:

- It means the given problem requires the solution of one sub-problem repeatedly.
- For example: Binary search doesn't have overlapping sub-problem while as recursive program of Fibonacci numbers have many overlapping sub-problems.

II. Optimal sub-structure:

- It means, the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.
- Example: to find the shortest path from source node **u**, to destination node **v**.

Steps Of Dynamic Programming:

i. Characterize the structure of an optimal solution.

ii. Recursively define the value of an optimal solution.

iii. Compute the value of an optimal solution, typically in a bottom-up fashion.

iv. Construct an optimal solution from the computed information.

Applications of Dynamic Programming Approach

i. Matrix Chain Multiplication

ii. Longest Common Sub sequence

iii. Travelling Salesman Problem

Optimal Binary Search Tree:

- **Given**: A list of keys with frequency
- **Goal:** to provide smallest possible search time in binary search tree.

Knapsack:

- Greedy approach solves Fractional Knapsack problem reasonably in a good time.

Knapsack Problem:

- Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- In this context, the item should be selected in such a way so that the thief will carry those items for which he gain maximum profit. Hence, the objective of thief is to carry maximize the profit.
- It can be categorized as fractional knapsack
  i. Fractional
  ii. 0-1 Knapsack

Fractional Knapsack:

- A thief is robbing a store and can carry a maximal weight of W into his knapsack.
- There are n items available in the store and weight of i_{th} item is w_i and its profit is p_i. What items should the thief take ?

Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)

```
for i = 1 to n
   do x[i] = 0
weight = 0
for i = 1 to n
   if weight + w[i] ≤ W then
```

```
        x[i] = 1
        weight = weight + w[i]
    else
        x[i] = (W - weight) / w[i]
        weight = W
        break
return x
```

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_fractional_knapsack.htm

0-1 Knapsack:

- In 0-1 knapsack, items can not be broken which means the thief should take item as a whole or should leave it. This is reason behind calling it as 0-1 Knapsack.

Dynamic-0-1-knapsack (v, w, n, W)

```
for w = 0 to W do
    c[0, w] = 0
for i = 1 to n do
    c[i, 0] = 0
    for w = 1 to W do
        if wi ≤ w then
            if vi + c[i-1, w-wi] then
                c[i, w] = vi + c[i-1, w-wi]
            else c[i, w] = c[i-1, w]
        else
            c[i, w] = c[i-1, w]
```

Q Illustrate job scheduling with deadline. [5 marks, 2074]

Ans:- Given set of **n** jobs.

- D_i and P_i are deadline and profit associated with the job J_i where `i` is an integer.
- For any job J_i, profit is earned iff J_i is finished by it's deadline.
- J_i takes one unit time.
- Single machine is available for processing.
- Feasible solution: Subset of jobs such that each jobs in the subset can be completed by its deadline.
- Value of feasible solution: Sum of profit of jobs in feasible solution
- optimal feasible solution: feasible solution with maximum value.
- Example:

  Let us consider a set of given jobs as shown in the following table. We have to find a sequence of jobs, which will be completed within their deadlines and will give maximum profit. Each job is associated with a deadline and profit.

Job	J_1	J_2	J_3	J_4	J_5
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

Solution

- To solve this problem, the given jobs are sorted according to their profit in a descending order. Hence, after sorting, the jobs are ordered as shown in the following table.

Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

- From this set of jobs, first we select ***J2***, as it can be completed within its deadline and contributes maximum profit.
- Next,*J1* is selected as it gives more profit compared to *J4*.
- In the next clock,*J4* cannot be selected as its deadline is over, hence*J3* is selected as it executes within its deadline.
- The job ***J5*** is discarded as it cannot be executed within its deadline.
- Thus, the solution is the sequence of jobs (***J2, J1, J3***), which are being executed within their deadline and gives maximum profit.
- Total profit of this sequence is **100 + 60 + 20 = 180**.

Unit 2:
Computational Complexity

2.0 Questions:

i. Differentiate between P, NP and Np hard problem with necessary examples. [5Marks, 2073]

ii. State and prove Cook's theorem. [5 Marks, 2073]

iii. Proof that formal satisfiability is is a NP class problem.

- Hint: First Part of Cook's Theorem

iv. Design and analyze approximation algorithm to solve "Vertex cover problem".

v. Write short notes on Subset Sum problem.

vi. Prove the satisfiability of Boolean formula (formula SAT is NP-Complete).

- Hint: Second part of Cook's theorem.

vii. What is vertex cover problem ? Illustrate with suitable example.

2.1 Computational Complexity:

- Computational complexity theory classifies computational problems according to computational complexity. i.e., the amount of resources needed to solve them, such as time and storage.

Complexity classes:

i. **P class:**

- The complexity class P is the set of all decision problems that can be solved with worst-case polynomial time-complexity i.e. can be solved by deterministic Turing machine in polynomial time.

- In other words, a problem is in the class P if it is a decision problem and there exists an algorithm that solves any instance of size n in $o(n^k)$ time, for some integer k
- So P is just the set of tractable decision problems: the decision problems for which we have polynomial-time algorithms.
- An example would be basic multiplication of two numbers, Even just using the typical multiplication algorithm we learn in school to multiply two *n* digit numbers will only require n^2 single-digit multiplications, which is a polynomial in *n*.

ii. **NP class:**

- The complexity class NP is the set of all decision problems that can be non-deterministically accepted in worst-case polynomial time. i.e. solved by non-deterministic Turing machine in non polynomial time.
- P is subset of NP($P \subseteq NP$), so any problem that can be solved by deterministic Turing machine in polynomial time also that cab be solved by Non-deterministic Turing machine in polynomial time.
- Example: Integer factorization is in NP

Source: **http://www.cs.ucc.ie/~dgb/courses/toc/handout32.pdf**

iii. **NP Hard**

- We say that a decision problem Pi is NP-hard if every problem in NP is polynomial time reducible to Pi.
- Note that NP-hard problems do not have to be in NP, and they do not have to be decision problems.
- Mathematically,

- Pi is NP-hard if, for every Pj ∈ NP, $Pj \xrightarrow{poly} Pi$.

- The halting problem is an NP-hard problem.

iv. **Np Complete**

- We say that a decision problem Pi is NP-complete if
  - it is NP-hard and
  - it is also in the class NP itself.
- In symbols, Pi is NP-complete if Pi is NP-hard and Pi ∈ NP
- Example:
  - 3SAT
  - k-Clique
  - Vertex Cover
  - Independent Set

Source: http://www.cs.ucc.ie/~dgb/courses/toc/handout33.pdf

2.2 Cook's Theorem:

- **Cook's Theorem** states that the Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable.
- Corollary: $SAT \in P$ if and only if P = NP
- **Proof:**

There are two parts to proving that the Boolean satisfiability problem (SAT) is NP-complete.

i. One is to show that SAT is an NP problem.

   → SAT is in NP because any assignment of Boolean values to Boolean variables that is claimed to satisfy the given expression can be verified in polynomial time by a deterministic Turing machine.

ii. The other is to show that every NP problem can be reduced to an instance of a SAT problem by a polynomial-time many-one reduction.

   → suppose that a given problem in NP can be solved by the nondeterministic Turing machine $M = (Q, \Sigma, s, F, \delta)$ where,

   - Q is the set of states
   - Σ is the alphabet of tape symbols,
   - $S \in Q$ is the initial state
   - $F \subseteq Q$ is the set of accepting states
   - and $\delta \subseteq ((Q \setminus F) \times \Sigma) \times (Q \times \Sigma \times \{-1, +1\})$ is the transition relation.

   And, M accepts or rejects an instance of the problem in time p(n) where n is the size of the instance and p is a polynomial function.

   For each input, I, we specify a Boolean expression which is satisfiable if and only if the machine M accepts I.

   The Boolean expression uses the variables set such that, $q \in Q$, $-p(n) \leq i \leq p(n)$, $j \in \Sigma$, and $0 \leq k \leq p(n)$.

If there is an accepting computation for M on input I, then B is satisfiable by assigning their intended interpretations. On the other hand, if B is satisfiable, then there is an accepting computation for M on input I that follows the steps indicated by the assignments to the variables.

There are O(p(n)2) Boolean variables, each encodeable in space O(log p(n)). The number of clauses is O(p(n)3) so the size of B is O(log(p(n))p(n)3). Thus the transformation is certainly a polynomial-time many-one reduction, as required.

SAT:

- SAT is also called Boolean satisfiability problem.
- A propositional logic formula φ is called satisfiable if there is some assignment to its variable that makes it evaluate to true.
- $p \wedge q$ is satisfiable .i.e. for p=> 1 & q=> 1 $p \wedge q$ is true, i.e. satisfiable.
- $p \wedge \neg p$ is not satisfiable, But no value of p could make boolean function ($p \wedge \neg p$) true, so not satisfiable.
- SAT is NP-complete.

Q. Prove that formula satisfiability is a NP class.

- A clause in the Propositional Calculus is a formula of the form ¬A1 v ¬A2 v · · · v ¬Am v B1 v B2 v · · · v Bn,

where $m \geq 0$, $n \geq 0$, and each A_i and B_j is a single schematic letter. Schematic letters and their negations are collectively known as literals; so a clause is simply a disjunction of literals.

- The clause above was written in disjunctive form. It can also be written in conditional form, without negation signs, as $A1 \wedge A2 \wedge \ldots A_m \rightarrow B1 \vee B2 \vee \cdots \vee Bn$.
- Note that if $m = 0$ this is simply $B1 \vee B2 \vee \cdots \vee Bn$,
- while if $n = 0$ it may be written as $\neg(A1 \wedge A2 \wedge \ldots Am)$.
- We shall use the conditional notation for clauses from now on. Note that any Propositional Calculus formula is equivalent to the conjunction of one or more clauses, for example:

  i. $A \leftrightarrow B \sim= (A \rightarrow B) \wedge (B \rightarrow A)$

  ii. $A \vee (B \wedge C) \sim= (A \vee B) \wedge (A \vee C)$

  iii. $A \wedge B \rightarrow \neg C \sim= \neg(A \wedge B \wedge C)$

- The satisfiability problem (SAT) may be stated as follows:

  → Given a finite set $\{C1, C2, \ldots, Cn\}$ of clauses, determine whether there is an assignment of truth-values to the schematic letters appearing in the clauses which makes all the clauses true.

- The size of an instance of SAT is the total number of schematic letters appearing in all the clauses; e.g., the instance

  $\{A \vee B, A \rightarrow C \vee D, \neg(B \wedge C \wedge E), C \rightarrow B \vee E, \neg(A \wedge E)\}$ is of size 13.

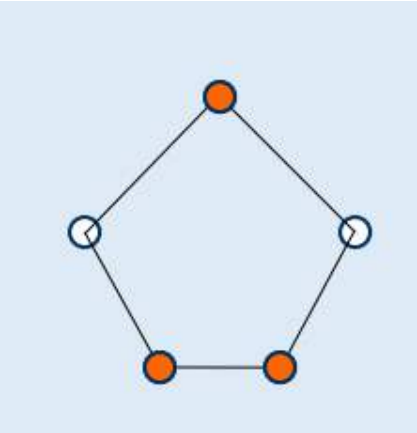
- SAT is easily seen to be NP. Given a candidate certificate, say A = true, B = f alse, C = f alse, D = true, E = f alse for the instance above, the time it takes to check whether it really is a certificate is in fact linear in the size of the instance.

3 SAT

- **Problem:** Given a CNF where each clause has 3 variables, decide whether it is satisfiable or not.
- $(x1 \vee x1 \vee X2) \wedge (\neg x1 \vee \neg x2 \vee \neg x2) \wedge (\neg x1 \vee x2 \vee x2)$
- A language 3 SAT is {φ | φ is satisfiable 3 CNF formula }
- 3SAT is NP Complete.

Vertex Cover:

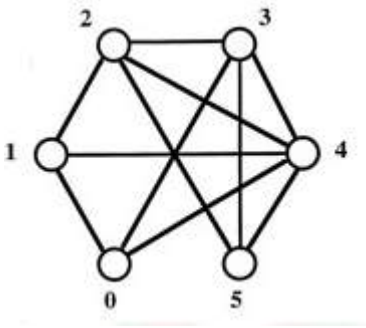
- In the mathematical discipline of graph theory, "A vertex cover (sometimes node cover) of a graph is a subset of vertices which "covers" every edge.
- An edge is covered if one of its endpoint is chosen.
- In other words "A vertex cover for a graph G is a set of vertices incident to every edge in G."
- **The vertex cover problem:** What is the minimum size vertex cover in G?
- **Problem:** Given graph G = (V, E), find smallest $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both.



- This problem can be solved by:
  i. **greedy Approach:**
  ii. **Approx optimal Solution:**

Maximum CLIQUE problem

- In a Graph G, a subset of vertices fully connected to each other, i.e. a complete subgraph of G is called clique.
- **The maximum clique problem:** how large is the maximum-size clique in a graph ?
- In another words, given a group of vertices some of which have edges in between them, the maximal clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset.



Given a graph with vertices 0,1,2,3,4,5. Here, the maximum clique problem is the largest subset of graph which is complete.

→ And the maximum clique size is 4, and the maximum clique contains the nodes 2,3,4,5.

Traveling salesman problem

→ The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one (e.g. the hometown) and returning to the same city. The challenge of the

problem is that the traveling salesman wants to minimize the total length of the trip.

- Travelling salesman problem is one of the most extensively studied optimization problem that is used to find the shortest possible route.
- TSP has many applications including following:
  i. The delivery of meals to office persons.
  ii. Manufacture of microchips.
  iii. The routing of courier trucks.
  iv. The routing of any salesman.

Solution of TSP:

1) **brute-force approach:** In this method every possible tour is evaluate and selects the best one. For n number of vertices in a graph, there are $(n - 1)!$ number of possibilities.
2) **Dynamic approach:** In this methods the solution can be obtained in lesser time. Algorithm is given below:

```
C ({1}, 1) = 0
for s = 2 to n do
   for all subsets S Є {1, 2, 3, … , n} of size s and containing 1
      C (S, 1) = ∞
   for all j Є S and j ≠ 1
      C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}
 Return minj C ({1, 2, 3, …, n}, j) + d(j, i)
```

Subset Sum problem

- In the subset-sum problem, we are given a finite set S of positive integers and an integer target t > 0.
- Subset sum problem is to find subset of elements that are selected from a given set S whose sum adds up to a given number K.
- Mathematically, Subset Sum: {(s,t): there exists a subset S' $S' \subseteq S$ such that $\sum_{s \in S'} S$
- **Problem:** Given a lists of positive integers a[1...n] and an integer t, is there some subset of a that sum exactly t ?
- Example: a = [12,1,3,8,20,50]
  i. t =44 → True
  ii. t=12 → False
- It can be solved by
  i. **Brute force methods:**
     - Making all possible subset-sum
     - $O(2^n N)$ complexity to sum them
     - So, it's not efficient algorithm for solving this problems
  ii. Sometimes, the divide and conquer approach seems appropriate but fails to produce an efficient algorithm.
  iii. One of the reasons is that D&Q produces overlapping sub problems.
  iv. Dynamic Approach:

- Solves a sub-problem by making use of previously stored solutions for all other sub problems.

Hamilton Cycle:

- Hamilton cycle, or Hamilton circuit, is a graph cycle (i.e. closed loop) through a graph that visits each node exactly once (except for the vertex that is both the start and end, which is visited twice).
- A graph possessing a Hamiltonian cycle is said to be a Hamilton graph.
- By convention, the singleton graph $\mathbf{K_1}$ is considered to be Hamilton even though it does not posses a Hamilton cycle, while the connected graph on two nodes $\mathbf{K_2}$ is not.
- **Hamiltonian path problem** are problems of determining whether a Hamiltonian path or a Hamiltonian cycle exists in a given graph.

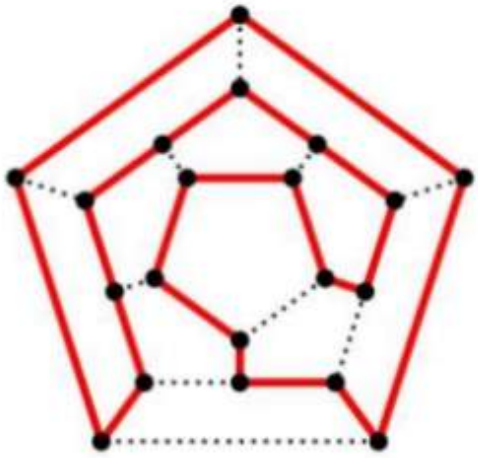


Fig: Hamilton Path

- It can be solved by
  - Brute Force method
  - Frank Rubin method
  - Dynamic Programming algorithm
  - Monte Carlo Algorithm

Unit 3
Online Algorithm & PRAM Algorithm

3.0 Questions

i. Compare online Algorithm with offline algorithm. Explain an algorithm for page replacement policy with examples.

ii. Compare and contrast different strategies that can be implemented in online algorithms.[10 marks, 2074]

iii. What are the advantages and disadvantages of of online algorithm ?

iv. State and give an online algorithm to solve rental problem. [5 marks, 2073]

v. Define speed up, asymptotic speed, total work-done and efficiency of PRAM algorithm with illustrative examples. [10 marks, 2071]

vi. Explain work optimal PRAM algorithm to solve prefix computation problem with example.[10 Marks, 2071]

vii. Where can the concept of binary tree can be implemented ? What is prefix computation ? Explain.[10 marks, 2073]

viii. How can you compute rank in a linear array ? Explain.[5 Marks, 2075]

ix. Demonstrate odd-even sort in Butterfly networks. Calculate its time-complexity. [10 Mars, 2075]

3.1 Online Algorithm

Online Vs Offline Algorithm

SN	Online	Offline
1.	Online Algorithms are algorithms that need to make decisions without full knowledge of the input.	Offline algorithm are algorithm which solves the problem with whole problem data from the beginning
2	Online algorithms may produce the results that are not optimal as it doesn't have the complete input.	Offline algorithms can produce the optimal solution as it is given the complete input.
3	It process its input piece-by-piece in a serial fashion.	It processes all the input at same time, as all data are initially available.
4.	Application: Stock Market, Secretary problem,	Application: Sorting in the list, Search in array

Q. state and give an online algorithm to solve ski rental problem. [5marks, 2073]

- Online Algorithms are algorithms that need to make decisions without full knowledge of the input. They have full knowledge of the past but no (or partial) knowledge of the future.

Ski rental problem:

- This problem explores the trade-off between renting and buying an item.
- Suppose there is an item (say a ski) which costs B units to buy and 1 unit per day for rent.
- You will need the item for T consecutive days, starting from day 1, where T is a non-negative integer.
- The online aspect arises from the fact that we do not know the value of T in advance.
- **Problem:** should we buy or rent that item?

There are few algorithm to solve the ski rental problems:

i. Offline algorithms
ii. Deterministic algorithm
iii. Randomized algorithms

Offline algorithm:

- If the value of T is given to us, the optimal solution is easy,
- buy the ski if $T \geq B$
- else rent it for the first T days
- Clearly, the cost of this solution is min(B, T)

Deterministic algorithm:

- problem is we don't know the value of T, and the algorithm says:
- rent till the first B – 1 days
- If we still need the item on day B, then buy it on day B

- If T ≥ B, the algorithm pays B – 1 + B = 2B – 1
- If T < B, then the algorithm pays T, which is also the optimal cost.
- Optimal Cost

  = (B - 1) + B

  = 2B – 1 < 2B

  = 2 * min(T, B)
- Competitive ratio is

  = (2B - 1)/B

  = 2 – 1/B

  < 2 (which is nearly equal to 2 for large B)

Randomized algorithm:

- it turns out to exist randomized algorithm with competitive cost ratio of (e/e-1) which is nearly 1.58
- This algorithm uses integer and linear programming.
- Conclusion: Both deterministic and Randomized algorithm are optimal.

Ref: http://www.cse.iitd.ernet.in/~amitk/SemII-2015/onlinealg.pdf

Load Balancing

- A computer methodology to distribute workload across multiple computing resources like computer cluster , network links, central processing units ,disk drives etc.
- Load balancing is required for

  i. High Availability,

ii. High Reliability,

iii. High Scalability

iv. Ease of maintenance

v. Resource sharing

- Load Balancing attempts to keep the workload evenly distributed across all processors in an SMP(symmetric multi-procesor system) system
- It is necessary where each processor has its own private queue
- Goals of load balancing is

i. Achieve optimal resource utilization

ii. Maximize throughput

iii. Minimize response time

iv. Avoid overload

v. Avoid Crashing

- Following algorithm are commonly used:

i. Round robin

ii. response time

iii. random

  - weighted
  - dynamic

3.2 PRAM Algorithm

Q. Explain work done and its efficiency with a suitable example of your own for PRAM. When do you confirm that the algorithm is work optimal?

- → **Speed Up:** The speedup is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors.
- Let Π be a given problem for which the best known sequential algorithm has a run time of S'(n) and the same problem with parallel algorithm on a p-processors machines runs in time T'(n,p), then speed up is given by
- Speed up = S'(n)/T'(n,p)
- → the ratio of asymptotic run time S(n) for sequential algorithm for problem Π to asymptotic run time T(n,p) for parallel algorithm for same problem then asymptotic speed up is
- Asymptotic Speed up = S(n)/T(n,p)
- **Work Done:** If a p-processors parallel algorithm for a given problem runs in time T(n,p), then the total work done is defined to be pT(n,p)
- **Efficiency:** Efficiency is the ration of asymptotic run time of best sequential machines to solve a problem Π to work done by p-processors parallel algorithm to solve the same problem.
- Example:
- we have to add upto 100 then a sequential algorithm takes 99 times,

- when two processors A and B are used, A can add upto 1 to 50 and B can add from 51 to 100
- So, total time for two processors= 50 time
- Speedup = S(n)/T(n,2)=99/50 = 1.98
- Efficiency = S(n)/ pT(n,p) = 99/ (2* 50) = 0.99

Computational Model

- Sequential computing model we have employed so far is the RAM(random access machine).
- In RAM model, any operation like addition, subtraction is performed in 1 unit of time.
- In parallel computing, the processors has to communicate to each other to synchronize their work which is called inter-process communication, which was absent in sequential model.
- Parallel computing model can be categorized into two:
  i. fixed communication
  ii. shared memory
- A fixed connection networks is a graph G(V,E) whose nodes represents processors and whose edges represents communication links between processors.
- Example of fixed connection: mesh, hypercube etc

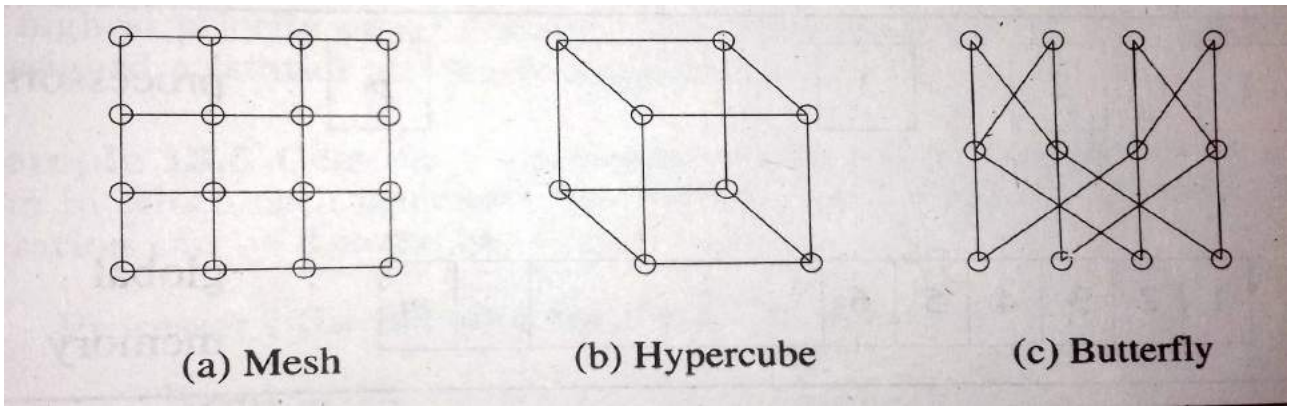


fig: example of fixed connection

- In shared memory, a number of processors work synchronously.
- They communicate with each other by reading and writing to/from common block of global memory that is accessible by all.

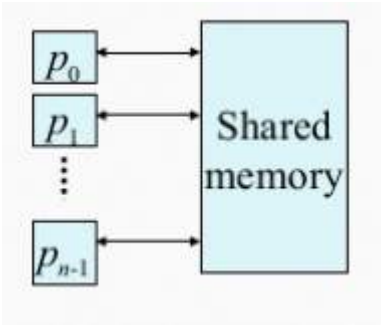


Fig: Shared Memory Model

- Communication by two processors i and j is performed in two steps:
  i. processor i writes its message into memory cell j.
  ii. processor j reads from its cell

Prefix computing:

- We are given an ordered set A of n elements

  $A = a_0, a_1, a_2 \ldots . a_{n-1}$

  and a binary associative operator $\oplus$
- We have to compute the ordered set

  $a_0, a_1 \oplus a_2 , \ldots . a_0 \oplus a_1 \oplus a_{n-1}$
- For example, if $\oplus$ is + and the input is the ordered set

{5, 3, -6, 2, 7, 10, -2, 8}

- then the output is

  {5, 8, 2, 4, 11, 21, 19, 27}

- Prefix sum can be computed in O (n) time sequentially

The work optimal can be achieved by using $n/logn$ processors where n is the input size

Algorithm

Step 1: processors I (i= 1, 2, 3, ….. n/logn) in parallel

Compute the prefixes of its logn assigned elements.

Step 2: All n/logn processors compute the prefixes of the n/logn last elements of each group using **pram algorithm for prefix computation** .

Step 3: Each processors update the prefixes it computed in step 1

Example: 5,12, ,8, 6, 3, 9, 11, 12, 1, 5, 6, 7, 10, 4, 3, 5

⊕: addition

n=16

Steps\ Processor	Processor 1	Processor 2	Processor 3	Processor 4
Initialize with	5,12,8,6	3,9,11,12	1,5,6,7	10,4,3,5

Steps\ Processor	Processor 1	Processor 2	Processor 3	Processor 4
n/logn last elements				
Step 1:	5,17,25,31	3,12,23,35	1,6,12,9	10,14,17,22

Step 2: each processors update the prefixes computed in step 1:

→ 31, 66, 85 107

Step 3:	5,17,25,31	34,43,54,66	67,72,78,85	95,99,102,107

Analysis:

- Step 1 takes O(logn)
- Step 2 takes O(logn)
- Step 3 takes O(logn)

The algorithm has complexity of O(logn)

Speed up = $\frac{O(n)}{O(logn)}$

efficiency = $\frac{o(n)}{\frac{n}{logn}0(logn)}$ = 1

Which is work optimal.

Maximum selection with n^2 processors

- Input: let k1, k2, k3 … kn be the input.
- Output: find maximum of n given numbers.
- **Algorithm:** This can be done in O(1) time using an n^2 processor CRCW PRAM. The steps are given below.
- Step 0: if n==1 output the key
- Step 1: If processors p_{ij} (for each $1 < i,j < n$ in parallel) computes m_{ij} such that $m_j = 1$ for $xi < xj$ otherwise 0
- Step 2: The n^2 processors are grouped into n groups G1, G2, G3 …. Gn, where each group consists of processors pi1, pi2 … pin. Each group computes the boolean OR of xi1, xi2 …. xin.
- step 3: If Gi computes a zero in step 2, then the processor Pi1 outputs ki as the answer

Unit 4:
Mesh and Hypercube Algorithm

4.0 Questions

1. Explain about hypercube. Discuss the process of drawing butterfly networks. [6 Marks, 2071]
2. Explain the broadcasting problem for mesh with example.
3. Design and algorithm to solve broadcasting problem on Hypercube.
4. Design an algorithm to solve prefix computation on Hypercube.
5. Explain data concentration problem on linear array with examples. [5 Marks,2074]
6. **Design an algorithm to solve prefix computation problem on Hypercube. [6 Marks, 2070]**
7. **Discuss the feature of hypercube and explain the process of Embedding of binary tree with hypercube.[6 Marks, 2070]**

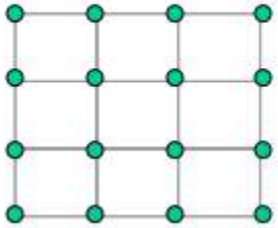
4.1 Mesh Algorithm

Computation Model

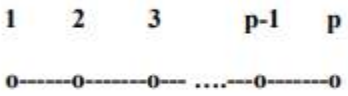
- A mesh is an a × b grid in which there is a processor at each grid point

- The edges are bi-directional communication links, i.e. two separated uni-directional links
- Each processor can be labeled with a tuple (i, j)
- Each processor has some local memory, and can perform basic operations
- There is a global clock which synchronize all processors
- We only consider square meshes here, i.e. a = b, and linear array

A 4 × 4 mesh (16 processors)



A Linear array of p processors



Fundamental Algorithm

Broadcasting

- **Problem:** to send a copy of message that originates from a particular processor to a specified subset of other processors, in a interconnection network.
- Subset is assumed to be every other processors, unless specified.
- Broadcasting is a primitive form of inter-processor communication and widely used in design of several algorithm.
- In the case of √p * √p mesh broadcasting can be done in two phases.

Phase 1. If (i, j) is the processor of message origin, message M could be broadcast to all processors in row i. It can be complete in P steps or less.

Phase 2. Broadcasting of message M is done in each column. This algorithm takes $\leq 2(\sqrt{p}-1)$ steps.

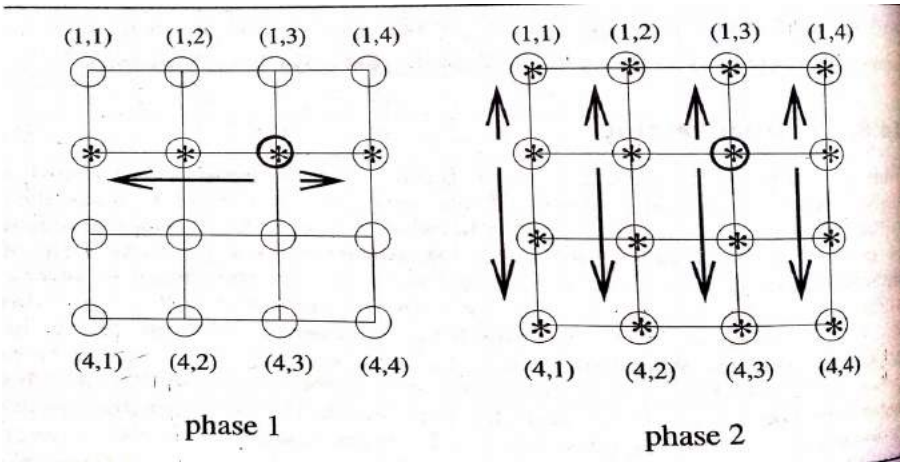
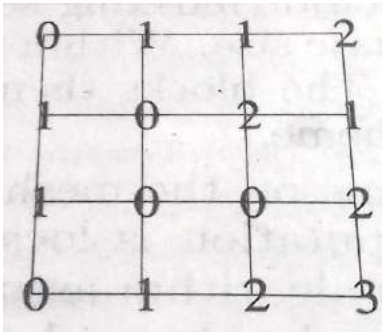


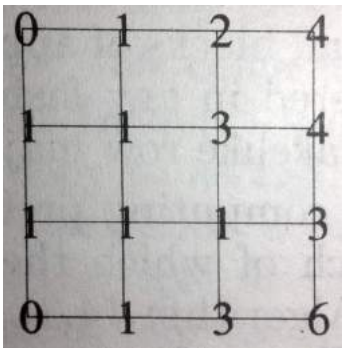
Fig: Broadcasting in a mesh

Prefix Computation

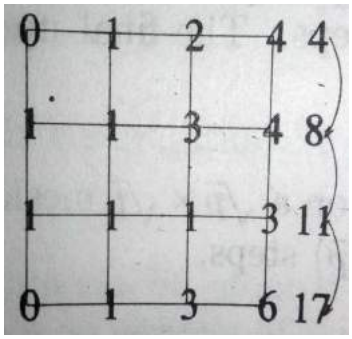
- Let, ∑ be any domain, has n input elements $x_1, x_2 \ldots\ldots, x_n$.
- ⊕ be the binary associated unit time computable operator defined in ∑.
- **Problems:** Compute n elements as $x_1, x_1 \oplus x_2, x_1 \oplus x2 \oplus x_3, \ldots\ldots x_1 \oplus x2 \oplus \ldots\ldots. xn$
- Prefix computation on a **√p * √p** mesh in a row major order can be performed in 3√p+2 = O(√p) steps.
- Prefix computation on a mesh can be done in three phases.
- Consider an example of 4 * 4 mesh in row major scheme, Given



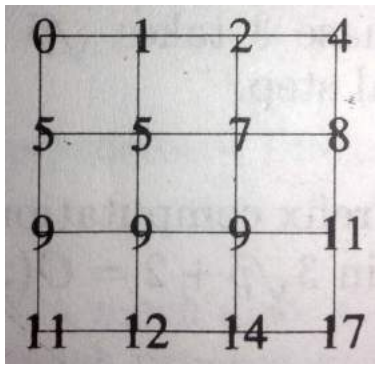
- **Phase 1:** Each row computes the prefix of its **√p** elements.



- **Phase 2:** Prefix sum is computed only in last column.

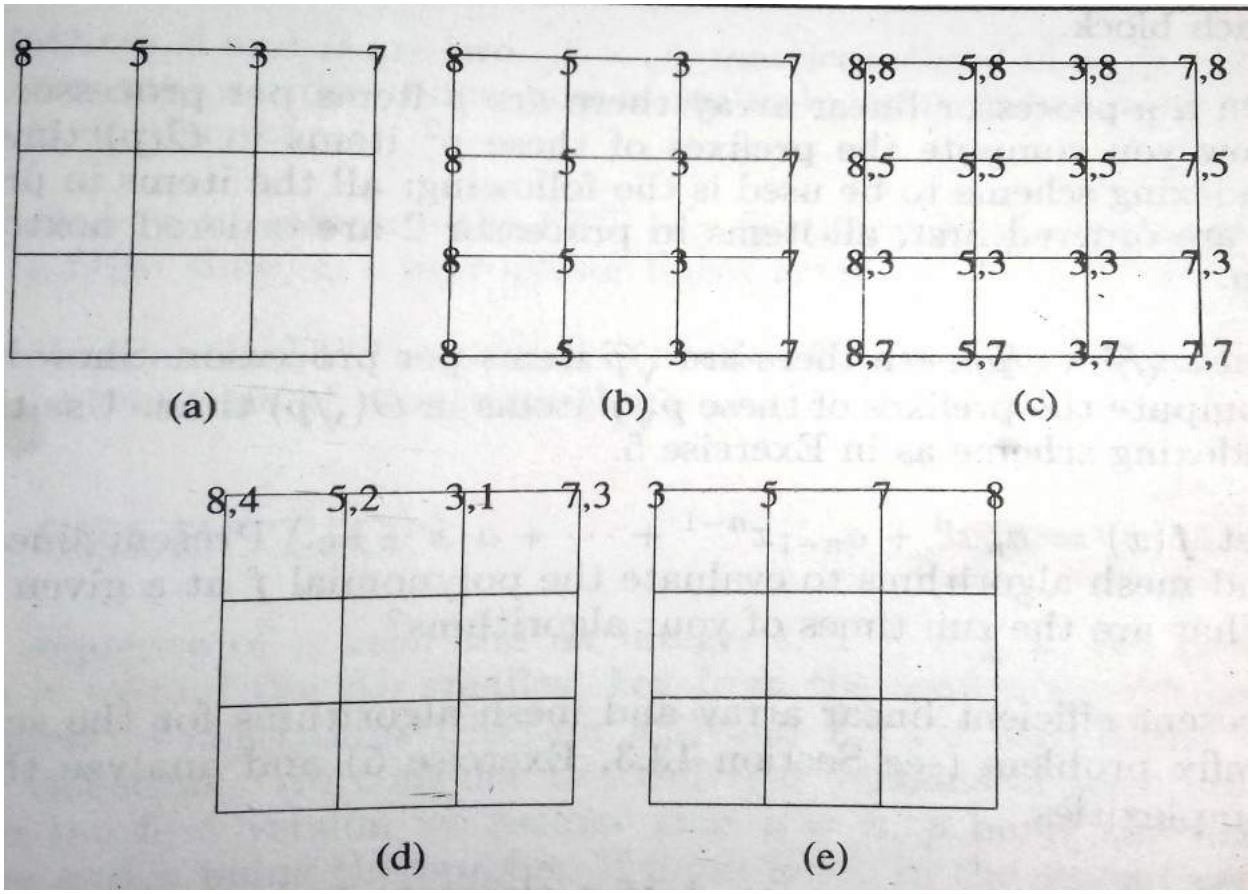


- **Phase 3:** Prefix sum is updated in each row



Sparse enumeration sort

- An instance of sorting in which the number of keys to be sorted is much less than the networks size is referred to as the sparse enumeration sort.
- On the mesh, sparse enumeration sort can be done by computing the rank of each key and routing the key to its correct position in sorted order.
- Consider the problem of sorting the four keys k1,k2,k3, k4=8,5,3,7 on a 4*4 mesh. Input to the mesh is given in the first row. (Fig a) and the output should also appear in the same row(Fig d)
- **Algorithm**
  i. In parallel, for 1<= j <= √p, broadcast kj along column j, Fig a.
  ii. In parallel, for 1<= i <= √p, broadcast ki along row i. Fig b.
  iii. In parallel, for 1<= i <= √p, compute the rank of ki along row i. Using prefix computation.
  iv. In parallel, for 1<= j <= √p, send rank of kj to (1,j).
  v. In parallel, for 1<= j <= √p, route the key whose rank is r to the node (1,r)


8 5 3 7
(a)
8 5 3 7
8 5 3 7
8 5 3 7
8 5 3 7
(b)
8,8 5,8 3,8 7,8
8,5 5,5 3,5 7,5
8,3 5,3 3,3 7,3
8,7 5,7 3,7 7,7
(c)
8,4 5,2 3,1 7,3
(d)
3 5 7 8
(e)

4.2 Hypercube Algorithm

Computational Model:

- Each node of a d-dimensional hypercube is numbered using d bits. Hence, there are 2^d processors in a d-dimensional hypercube.
- We define hypercube with following terms:
  i. Dimension
  ii. Coding of processors
  iii. Hamming Distance
  iv. Diameter
- Ex: for d=3; I.e dimension=3

  no. of vertex= $2^d = 2^3 = 8$

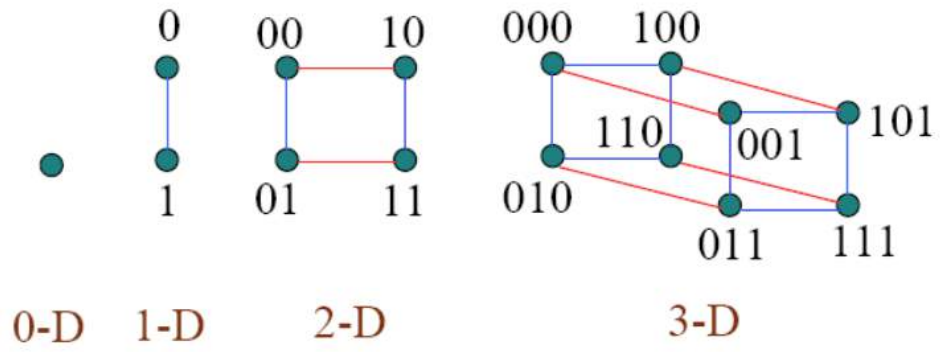
  no. of processors = 8

  Coding of processors = 000, 001, 011, 010, 110, 100, 101, 111

- Coding should be carried out in a such a way that the difference of code of two adjacent processors should be 1, which is called hamming distance.
- The diameter of a d-dimensional hypercube is d.
- The bisection width of a d-dimensional hypercube is 2^(d-1).
- The hypercube is a highly scalable architecture. Two d-dimensional hypercubes can be easily combined to form a d+1-dimensional hypercube.

- The hypercube has several variants like butterfly, shuffle-exchange network and cube-connected cycles.

Fig:



Computational Model Of Hypercube

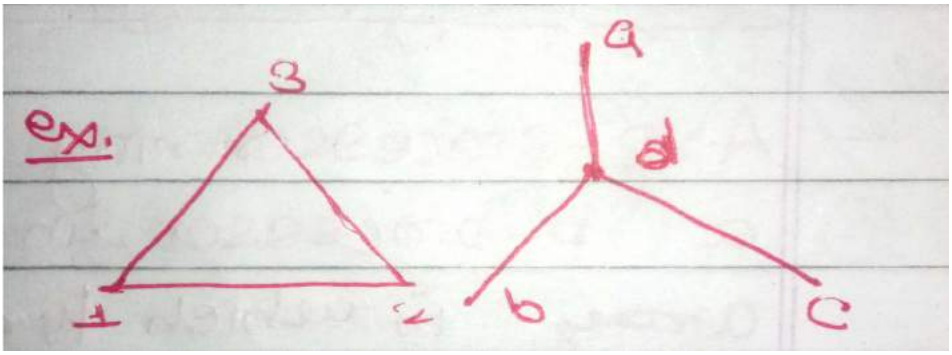
The Butterfly Network

- Algorithms for Hypercube can easily be adapted for Butterfly network and vice-versa.
- A d-dimensional butterfly (Bd) has (d+1)2^d processors and d2^(d+1) links.
- Processor represented as tuple <r,l>, r is row and l is level.
- Each processor u, is connected to two processors in level l+1: v = <r, l+1> and w= $\langle r^{(l+1)}, l+1 \rangle$
- (u,v) is called direct link and (u,w) is called cross link
- There exists a unique path of length d from u at level 0 and v at level d, called as greedy path.
- Hence, diameter of butterfly network is 2d.
- When each row of Bd is collapsed into a single processor, preserving all the links then resultant graph is a hypercube (Hd).
- Each step of Bd can be simulated in one step on parallel version of Hd.

- Normal butterfly algorithm – If at any given time, processors in only level participate, it is normal algorithm.
- A single step of any normal algorithm can be simulated on sequential Hd.

Embedding

- An embedding is a 1-1 function (also called a mapping) that specifies how the nodes of a domain network can be mapped into a range network.
- Mapping should take care of limit of the complexity parameters.
- **Dilation:** the length of longest path that any link of G is mapped to H.
- **Expansion:** the expansion of an embedding is defined as |v2|/|V1| where v1 and v2 are set of vertices
- **Congestion:** Congestion of any link of H is defined to be the no. of path(say in G) that if is on.



- Possible mapping of figure left to right is
  - 1 → b

- 2 → c
- 3 → a

- Dilation =2 and Expansion 4/3 & (1,2) is mapped with (b,d) & (d,c), (1,3) is mapped with (b,d) & (a,d) & so on.
- Congestion of (1,2) = (b,d) and (d,c)= 2 & congestion of path (1,3)= (b,d) & (a,d) = 2

Embedding of a binary tree

- A p-leaf (p=2^d) binary tree T can be embedded into Hd.
- More than one processor of T have to be mapped into same processor of Hd.
- If tree leaves are 0,1,2...p-1, then leaf i is mapped to ith processor of Hd.
- Each processor of T is mapped to the same processor of Hd as its leftmost descendant leaf

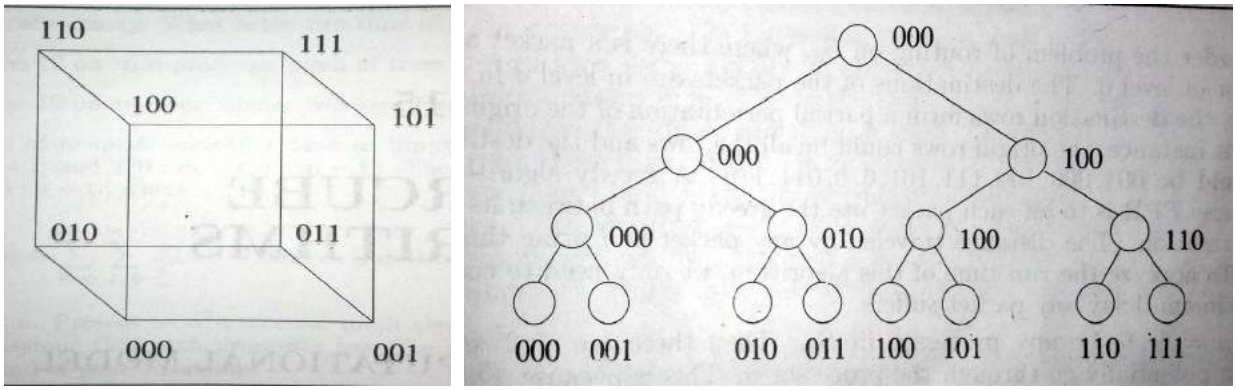


Fig: Embedding of hypercube

Embedding of ring Topology:

- A general mapping of one network G(V1, E1) into another H(v2, E2) is called embedding.

- A p-processors ring is a p-processors linear array in which processor 1 and p are connected by link.
- A ring with 2^d processors can be embedded in Hd.
- If 0, 1 ,2... 2d-1 are processors of a ring, processors 0 is mapped to processor 000 of the hypercube.
- Mapping is obtained using gray codes

PPR Routing:

A Greedy Algorithm

- In Bd, Origin of packet is at level 0 and destination is level d.
- Greedy algorithm for each packet is to choose a greedy path between its origin and destination.
- Distance travelled by any packet is d. Algorithm runs in $O(2^{d/2})$ time, the average queue length being $O(2^{d/2})$.

Broadcasting In hypercube

- The problem of broadcasting in an interconnection networks is to send a copy of message that originates from a particular processor to a subset of other processors. Broadcasting is quite useful since it is widely used in the design of several algorithm.
- To perform broadcasting on Hd, we employ the binary tree embedding.
- Assume a message M, at root of the tree is to be broadcast.
- The root makes two copies of message and sends one to left child and another to right child.

- On internal processors, on receipt of message from its parents, makes two copies of message and sends one to left child and another to right child.
- This proceed until all the node have a copy of M.

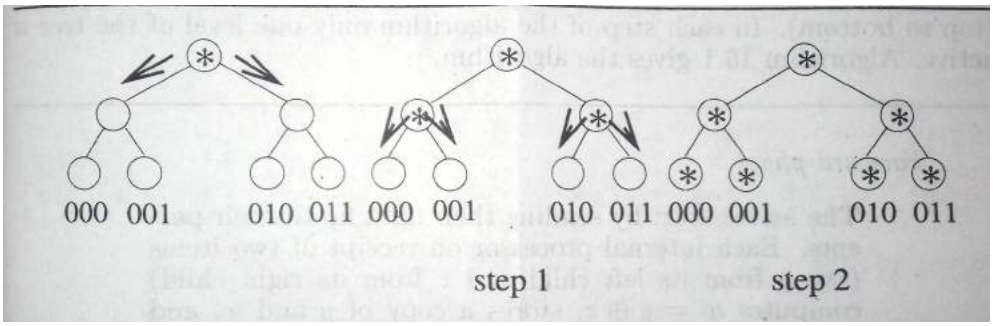


Fig: Broadcasting in hypercube

Prefix Computation on hypercube

- Binary tree embedding can be used.
- Two phases:
  i. forward phase &
  ii. reverse phase
- **Forward Phase:** The leaves start by sending their data up to their parents. Each internal processor on receipt of two items (y is left child and z is right child) computes w=y ⊕ z, stores a copy of y and w and sends w to its parent. At the end of d steps, each processor in the tree has stored in its memory the sum of all data items in the subtree rooted at this processor. The root has sum of all elements in tree.

- **Reverse Phase:** The root starts by sending zero to its left child and y to its right child. Each internal processor on receipt of a datum (say q) from its parent sends q to its left child and q+y to its right child. When ith leaf gets a datum q from its parent, it computes q+xi and stores it as final result.

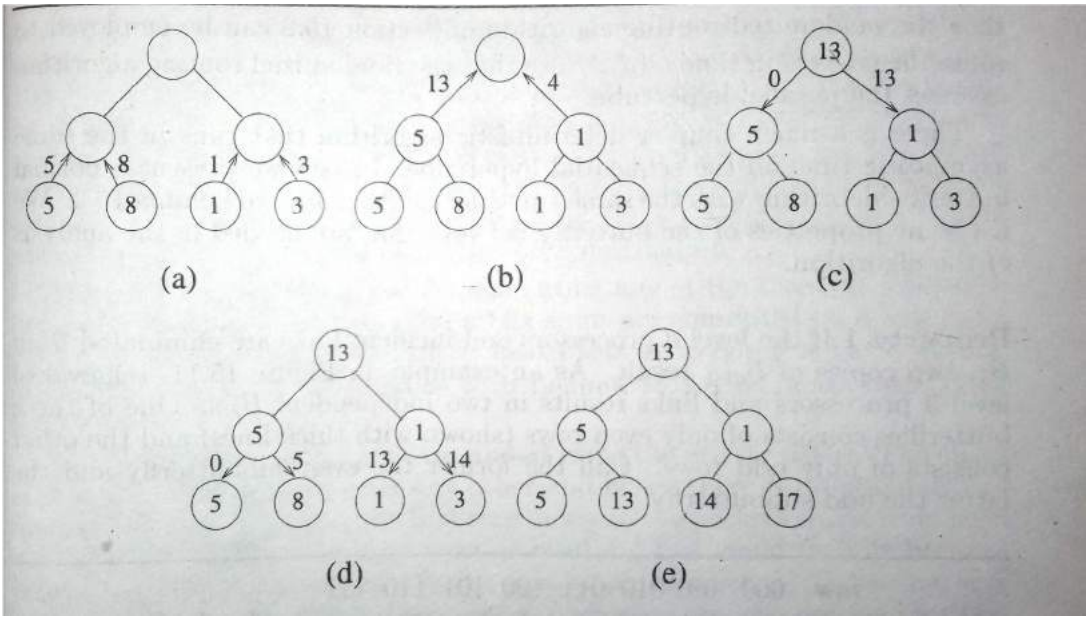


Fig: Prefix Computation on a binary tree

- prefix computation on 2^d-leaf binary tree as well as Hd can be done in O(d) steps.

Data Concentration On Hypercube Network

- On Hypercube $\mathbf{H_d}$ assume that there are k < p data items arbitrary distributed with at most one date per processor.
- The problem of data concentration is to move the data into processors 0,1,2,……k-1 of $\mathbf{H_d}$, one data per processor.
- We know, any butterfly network algorithm on B_d can be simulated in one steps on sequential Hypercube Network H_d.

- Assume that the k <= 2^d data items are arbitrary distributed in level d of B_d. At the end, these data items have to be moved to successive rows of level zero.
- There are two phases in this algorithm:
  - a prefix sum operation is performed to compute the destination address of each data item.
  - Packet is routed to its destination using the greedy path from its origin to destination
- In second phase, when packets are routed using greedy paths, we claim that no packet gets to meet any other in path, hence there is no contention
- Data concentration can be performed in Bd as well as sequential Hd in O(d) times.

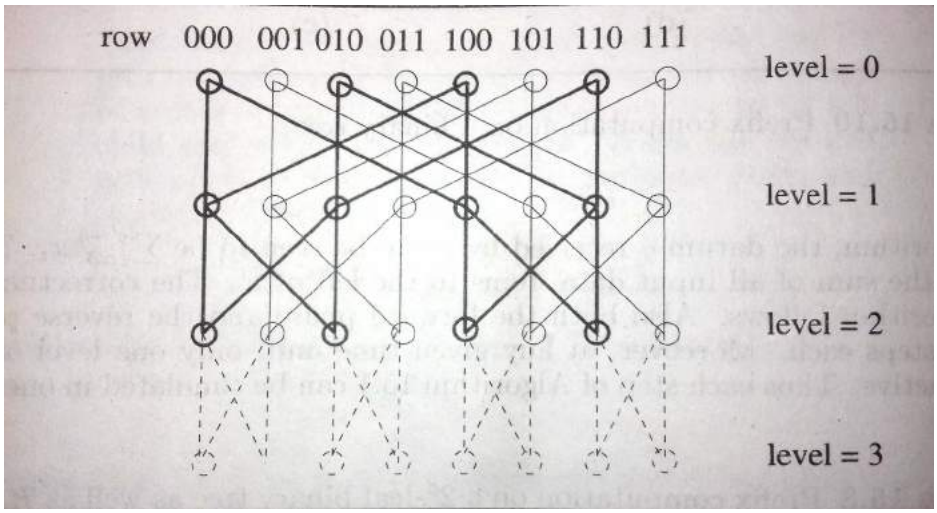


Fig: removal of level d processors and links

- For example: If there are 5 items in level 3 of B3, row 001 → (this notation means that the processor(001, 3) has the item a), row 001 → b, row 100 → c, row 101 → d, row 111 → e, then at the end these item

will be at level 0 and row 000 → a row 001 → b, row 010 →c, row 011 → d, row 100 → e. There are two phases in this algorithm as mentioned above. a prefix sum operation is performed to compute the destination address of each data item. Packet is routed to its destination using the greedy path from its origin to destination

Selection on Hypercube

- Given a sequence, problem is to find ith smallest key from sequence
- There are two different versions: a) p=n and b) n>p
- The work optimal algorithm for mesh can be adapted to run optimally on Hd as well.
- Selection from n=p keys can be performed in O(d) time on Hd.

Merging: Odd-even Merge On Hypercube/ Butterfly

Step 1: The problem of merging is to take two sorted sequences as input and produce a sorted sequence of all the elements.

Step 2: Given X= k_o, k_1, k_2 ….. k_{n-1} is the given sequence of n keys.

Step 3: ***Algorithm***

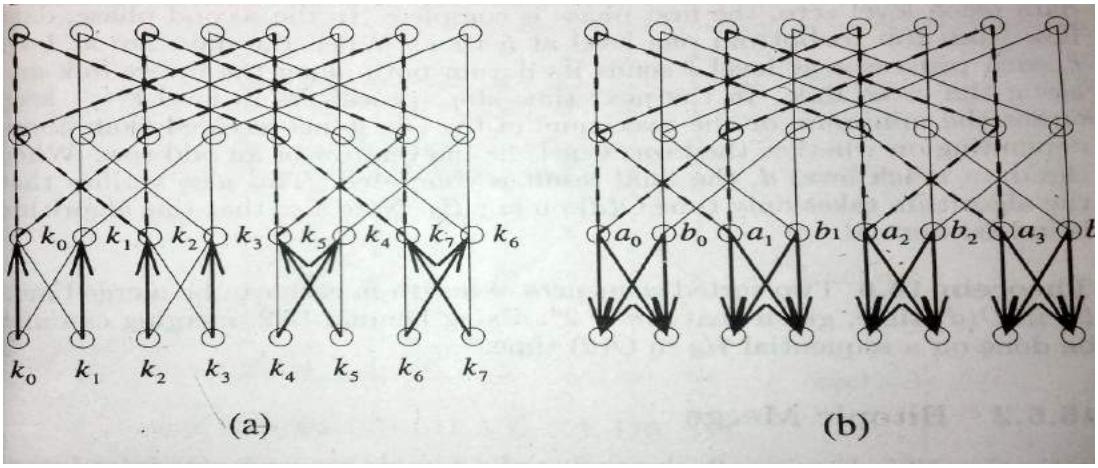
Step 4: Partition the sequence into two sub-sequences: X1'=k0, k1, k2... kn/2 and X2'= kn/2+1, kn/2+2... kn. X1 is in first half rows of butterfly network and X2 is in remaining rows.

Step 5: Sort the two sub-sequences recursively assigning n/2 processors to each. Sort first part using left sub-butterfly and second using right.

Step 6: Odd even Merge sort takes O(d^2) time on Bd, where p=2^d.

Merge two sorted sub-sequence using Odd-even Merge.

Fig:



Odd-even Merge on Hypercube Network

Sorting From Copy