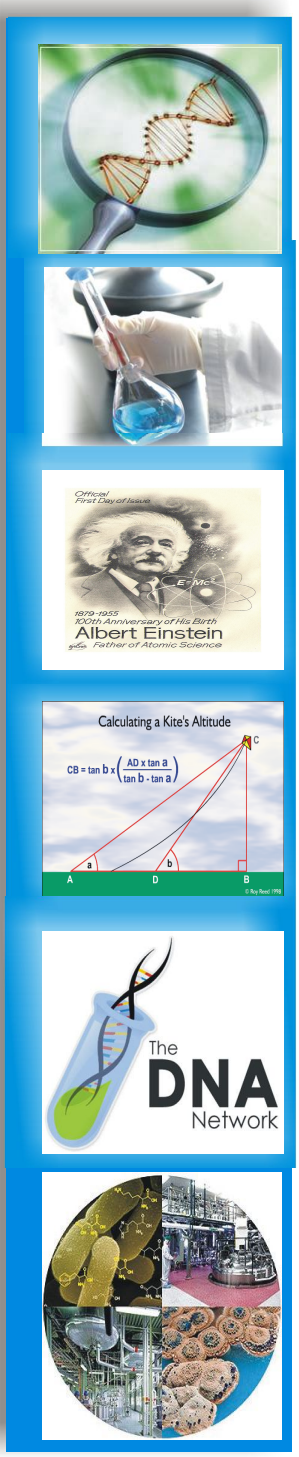

GATE - COMPUTER SCIENCE

SAMPLE THEORY

- **COMPLEXITY THEORY**
- **PAGE REPLACEMENT**

VPM CLASSES

For IIT-JAM, JNU, GATE, NET, NIMCET and Other Entrance Exams

1-C-8, Sheela Chowdhary Road, Talwandi, Kota (Raj.) Tel No. 0744-2429714

Web Site www.vpmclasses.com E-mail-vpmclasses@yahoo.com

COMPLEXITY THEORY

(a) Complexity theory focuses on classifying computational problems according to their inherent difficulty.

(b) Complexity of any algorithm tells us about its difficulty level, in terms of a function of size of input. Now, we just need to see if this function is within computability limits or not.

(c) "A complexity class is a set of problems of related complexity." Several complexity classes exist, but we will consider only the broadest classification of problems into P and NP.

Classification Based on Complexity

1. Class P

(a) We say that a decision problem belongs to the class P if there is an algorithm A and a number c such that for every instance I of the problem the algorithm will produce a solution in time O(B c), where B is the number of bits in the input string that represents I.

(b) To put it more briefly, P is the set of easy decision problems . P is the class of algorithms whose complexity is a polynomial function of the problems size.

2. Class NP

(a) The class NP is a little more subtle. A decision problem Q belongs to NP if there is an algorithm A that does the following.

Associated with each word of the language Q (i.e., with each instance I for which the answer is 'Yes') there is a certificate C(I) such that when the pari (I, C(I)) are input to algorithm A it recognizes that I belongs to the language Q.

If I is some word that does no belong to the language Q then there is no choice of certificate C(I) that will cause A to recognize I as a member of Q.

Algorithm A operates in polynomial time.

(b) NP means Non-deterministic Polynomial. Suppose a computer program could "guess" a solution to a problem and then could check if the guessed "solution" actually solved the problem and this check could be done n polynomial time, then the program is said to be in the class NP.

(c) Non-deterministic is another word for guessing. An NP problem is the one which is not solvable within polynomial time by computational means, yet it is easy to verify a given solution within polynomial time.

(d) Most [problems are in this class. Some well known problems in NP are:

Find a Hamiltonian circuit through a graph

Find all subsets of a set

Travelling salesman problem

(e) **Partition Problem:** Given n positive integers, determine if it is possible to partition them into two disjoint subsets which have equal sum.

(f) **Graph Coloring:** For a given graph find the smallest number of colors that need to be assigned to the graph vertices so that no two adjacent vertices share the same colour.

(g) NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information. So we are not asking for a way to find a solution, but only to verify that an alleged solution really is correct.

(h) NP includes P or P $\subset$ NP. Indeed if Q $\in$ P is some decision problem then we can verify membership in the language Q with the empty certificate.

(i) It seems natural to suppose that NP is larger than P. That is, one might presume that there are problems whose solutions can be quickly checked with the aid of a certificate even through they can not be quickly found in the first place.

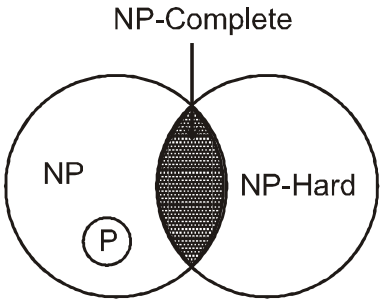

Fig.1 : Relationship between Various Complexity Classes

(j) The class NP was not of much use in itself, until definition of NP – completeness and Cook's theorem evolved. Before discussing NP-complete class of problem, we need to develop a concept called - reducibility.

3. Both P and NP

P is the set of all decision problems solvable by deterministic algorithms in polynomial time. NP is the set of all decision problems solvable by nondeterminstic algorithms in polynomial time.

(a) Since deterministic algorithms are just a special case of nondeterministic ones, we conclude that P $\subseteq$ NP. What we do not know, and what has become perhaps the most famous unsolved problem in computer science, is whether P = NP or P $\neq$ NP.

(b) A proof that p $\neq$ NP is just as elusive an seems to require as yet undiscovered techniques.

(c) But as with many famous unsolved problem, they serve to generate other useful results, and the question of whether NP $\subseteq$ P is no exception

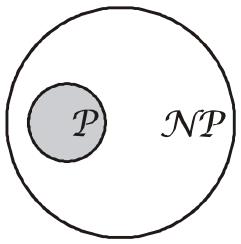

Fig. 2 Commonly believed relations hip between P and NP

4. NP - Completeness

(a) A decision problem is NP - complete if it belongs to NP and every problem in NP is quickly reducible to it. The implications of NP-completeness are numerous.

(b) Suppose we could prove that a certain decision problem Q is NP–complete. The we could concentrate our efforts to find polynomial time algorithms on just that one problem Q. Indeed if we were to succeed in finding a polynomial time algorithm to solve instance of Q then we would automatically have found a fast algorithm for solving every problem in NP.

(c) Take an instance I, of some problem Q' in NP. Since Q' is quickly reducible to Q we could transform the instance I' into an instance I of Q. Then use the super algorithm that we found for transform the instance I' into an instance I of Q. The use the super algorithm that we found for problem in Q to decide I. Altogether only a polynomial amount of time will have been used form start to finish.

(d) The properties of the NP- complete problems are:

The problems all seem to be computationally very difficult and no polynomial time algorithm have been found for any of them.

If has not been -proved that polynomial time algorithms for these problems do not exist.

5. NP-Hard

(a) The term NP-hard refers to a problem as hard as any NP problem. Formally, problem if a called NP-hard if it cannot be decided, or does not belong to NP class, but all NP problems are reducible to it in polynomial time.

(b) all NP - complete problems are NP-hard, but all NP-hard problems are not NP-complete. Also, an NP-hard problem need not be a decision problem, which explains why we cannot decide whether the problem belongs to NP or not.

(c) Consider a problem Q which may not belong to NP, because there is no certificate available. But, since an NP-complete problem (L) is the hardest problem in NP class, and L can be reduce to Q, we say Q is as hard as L. Thus, Q is NP -hard.

6. NP complete and NP hard

(a) A problem L is NP -hard if and only if satisfiability reduces to L (satisfiability $\propto$ L). A problem L is NP- complete if and only if L is NP-hard and L $\in$ NP.

(b) It is easy to see that there are NP -hard problems that are not NP-complete. Only a decision problem can be NP -complete,. However, an optimization problem may be NP-hard.

(c) Furthermore if L_1 is a decision problem and L_2 an optimization problem, it is quite possible that $L_1 \propto L_2$. One can trivially show that the knapsack decision problem reduces to the knapsack optimization problem.

(d) One can also show that these optimization problems reduce to their corresponding decision problem (see the exercise). Yet, optimization problems cannot be NP-complete whereas decision problem can. There also exist NP-hard decision problems that are not NP–complete.

Two problems L_1 and L_2 are said to be polynomially equivalent if and only if $L_1 \propto L_2$ and $L_2 \propto L_1$.

To show that a problem L_2 is NP-hard, it is adequate to show $L_1 \propto L_2$ where L_1 is some problem already known to be NP-hard. Since $\propto$ is a transitive relation , it follows that is satisfiability $\propto L_1$ and $L_1 \propto L_2$, then satisfiability $\propto L_2$. To shown that an NP- hard decision problem is NP-complete, we have just to exhibit a polynomial time nondeterministic algorithm for it.

PAGE REPLACEMENT

(a) Page replacement takes the following approach. If no frame is free, we find once that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table to indicate that the page is not longer in memory.

(b) We modify the page-fault service routine to include page replacement

1. Find the location of the desired page on the disk.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. It there is not free frame , use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

(c) We can reduce this overhead by using a modify bit (or dirty bit). When this hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.

(d) When we select a page for replacement, we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk

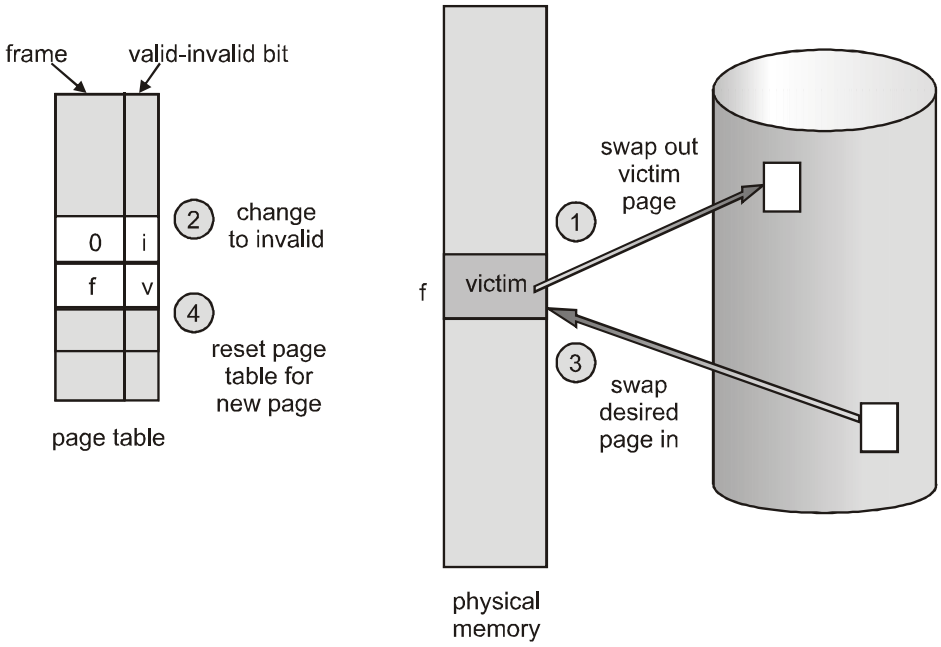

Fig.3. Page replacement

Page replacement is basic to demand paging.

(e) We must solve two major problems to implement demand paging: we must develop a frame-allocation algorithm and a page-replacement algorithm.

1. FIFO Page Replacement

(a) A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

(b) For our example reference string, our three frames are initially empty. the first three references (7, 0, 1) cause page faults and are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference and 0 is already in memory, we have no fault for this reference.

(c) The first reference to 3 results in replacement of page 0, since reference string

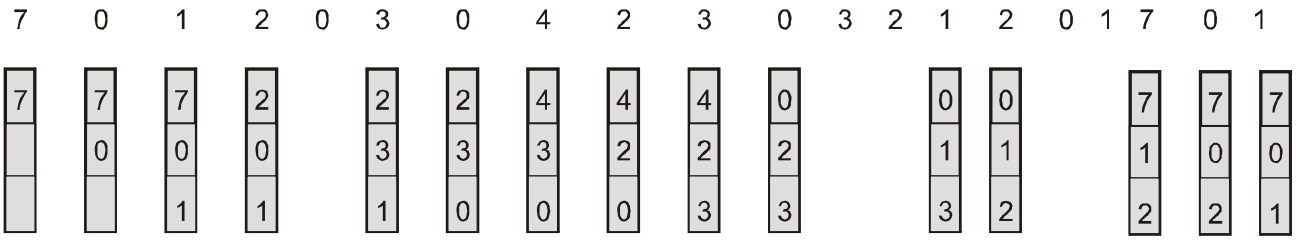

Fig.4. FIFO page-replacement algorithm

It is now first in line. Because of this replacement, the next reference, to 0, will fault. Page 1 is then replacement by page 0. Every time occurs , We sow which pages are in our three frames. There are fifteen faults altogether.

2. Optimal Page Replacement

(a) One result of the discovery of Belady's anomaly was the search for an optimal page-replacement algorithm, which has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. Such an algorithm does exist and has been called OPT or MIN

(b) Replace the page that will not be used for the longest period of time.

(c) For example, on our sample reference string, the optimal page replacement algorithm would yield nine page faults.

(d) The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page

0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1 as page 1 will be the last of the three pages in memory to be referenced again.

(e) With only nine page faults, optimal replacement is much better that a FIFO algorithm, which results in fifteen faults.

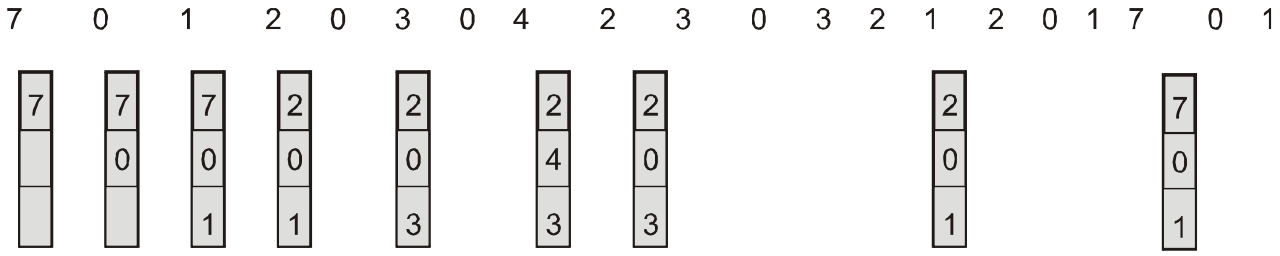

Fig.4. Optimal page-replacement algorithm.

3. LRU page Replacement

(a) If we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time. This approach is the least-recently-used (LRU algorithm.

(b) The result of applying LRU replacement to our example reference string is shown in Fig.5. The LRU algorithm produces twelve faults. Notice that the first five faults are the same as those for optimal replacement.

(c) When the reference to page 4 occurs, however, LRU replacement sees that, of the three frames in memory, page 2 was used least recently. Thus, the LRU algorithm replaces page 3, since it is now the least recently used of the three pages in memory. Despite these problems, LRU replacement with twelve faults is much better than FIFO replacement with fifteen.

(d) Two implementations are feasible:

Centers. In the simplest case, we associate with each page-table entry a time-of use field and add to the CPU a logical clock or counter.

reference string

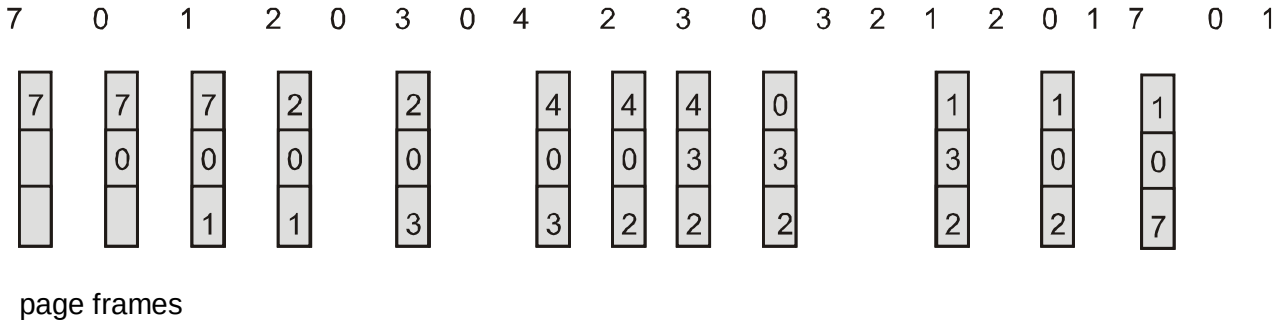

Fig.5: LRU page-replacment algorithm

Stack. Another approach to implementing LRU replacement is to keep a stack of page numbers

4. Counting-Based Page Replacement

(a) There are many other algorithms that can be used for page replacement

- The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced.
- The most frequently used (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.