Course Code : MCS-021
Course Title : Data and File Structures
Assignment Number : MCA(2)/021/Assign/13
Maximum Marks : 100
Weightage : 25%

Last Dates for Submission : 15th October, 2013 (For July 2013 Session)
15th April, 2014 (For January 2014 Session)

This assignment has four questions which carry 80 marks. Answer all the questions. Each question carries 20 marks. You may use illustrations and diagrams to enhance the explanations. Please go through the guidelines regarding assignments given in the Programme Guide. Ensure that you don't copy the program from course material or any other source. All the implementations should be in C language.

Question 1:
Write an algorithm for the implementation of Doubly Linked Lists.
Answer 1:

```
#include <stdlib.h>

typedef struct dnode {
 int data;
 struct dnode *next, *prev;
} DNODE;

DNODE *InsFront(DNODE *, int);
DNODE *InsBefore(DNODE *, int, int);
DNODE *DelNode(DNODE *, int);
void Display(DNODE *);

main() {
 DNODE *start = NULL; /* Main Program */
 int opn, elem, info, n, i;
 do {
  clrscr();
  printf("\n ### Doubly Linked List Operations ### \n\n");
  printf("\n Press 1-Creation with front Insertion");
  printf("\n       2-Insert Before a Given Node");
  printf("\n       3-Delete a Given Node");
  printf("\n       4-Display");
  printf("\n       5-Exit\n");
  printf("\n       Your option ? ");
  scanf("%d", &opn);
  switch (opn) {
  case 1:
   printf("\n\nHow Many Nodes ?");
   scanf("%d", &n);
```

```
   for (i = 1; i <= n; i++) {
    printf("\nRead the Data for Node %d ?", i);
    scanf("%d", &elem);
    start = InsFront(start, elem);
   }
   printf("\nDoubly Linked list with %d nodes is ready toUse!!\n", n);
   break;
  case 3:
   printf(" Read the Info of the Node to be deleted ? ");
   scanf("%d", &info);
   start = DelNode(start, info);
   break;
  case 2:
   printf(" Read the Data for New node\n");
   scanf("%d", &elem);
   printf(
     " Read the Info of the Node(to the left of which new node tobe inserted
? ");
   scanf("%d", &info);
   start = InsBefore(start, elem, info);
   break;
  case 4:
   printf(" Doubly Linked List is \n");
   Display(start);
   break;
  case 5:
   printf("\n\n Terminating \n\n");
   break;
  default:
   printf("\n\nInvalid Option !!! Try Again !! \n\n");
   break;
  }
  printf("\n\n\n\n  Press a Key to Continue . . . ");
  getch();
 } while (opn != 5);
}

DNODE *InsFront(DNODE *start, int elem) {
 DNODE *temp;
 temp = (DNODE *) malloc(sizeof(DNODE));
 if (temp == NULL) {
  printf(" Out of Memory !! Overflow !!!");
  return (start);
 } else {
  temp->data = elem;
  temp->prev = NULL;
  temp->next = start;
  start->prev = temp;
  printf(" New Node has been inserted at Front Successfully !!");
  return (temp);
```

```
 }
}

DNODE *InsBefore(DNODE *start, int elem, int info) {
 DNODE *temp, *t;
 temp = (DNODE *) malloc(sizeof(DNODE));
 if (temp == NULL) {
  printf(" Out of Memory !! Overflow !!!");
  return (start);
 } else {
  temp->data = elem;
  temp->next = NULL;
  temp->prev = NULL;

  if (start->data == info) /* Front Insertion */
  {
   temp->next = start;
   start->prev = temp;
   return (temp);
  } else {
   t = start;
   while (t != NULL && t->data != info)
    t = t->next;
   if (t->data == info) /* Node found */
   {
    temp->next = t;
    temp->prev = t->prev;
    t->prev->next = temp;
    t->prev = temp;
   } else
    printf(" Node not found,Invalid Info !!!");
   return (start);
  }
 }
}

DNODE *DelNode(DNODE *start, int info) {
 DNODE *t;
 if (start == NULL) {
  printf(" Underflow!!!");
  return (start);
 } else {
  t = start;
  if (start->data == info) /* Front Deletion */
  {
   start = start->next;
   start->prev = NULL;
   t->next = NULL;
   free(t);
   return (start);
```

```
  } else {
   while (t != NULL && t->data != info)
    t = t->next;
   if (t->data == info) /* node to be deleted  found*/
   {
    t->prev->next = t->next;
    t->next->prev = t->prev;
    t->next = t->prev = NULL;
    free(t);
   } else
    printf("Node not found, Invalid Info !!");
   return (start);
  }
 }
}

void Display(DNODE *start) {
 DNODE *t;
 if (start == NULL)
  printf("Empty List\n");
 else {
  t = start;
  printf("Forward Traversal \n\n Start->");
  while (t) {
   printf("[%d]->", t->data);
   t = t->next;
  }
  printf("Null\n");
 }
}
```

Question 2:
Implement multiple queues in a single dimensional array. Write algorithms for various queue operations for them.

Answer 2:
Program 5.4 gives the program segment using arrays for the addition of an element to a queue in the multiqueue.

```
addmq(i,x) /* Add x to queue i */
{
int i,x;
++rear[i];
if ( rear[i] == front[i+1])
printf("Queue is full");
else
{
```

```
rear[i] = rear[i]+1;
mqueue[rear[i]] = x;
}
}
```

Question 3:
Write a note of not more than 5 pages summarizing the latest research in the area of "Searching Algorithms". Refer to various journals and other online resources. Indicate them in your assignment.

Answer 3:
Algorithms and theory research is motivated equally by the goals of having significant impact on the real world and by advancing the state of the art in pure research. To this end, we pursue a wide variety of projects over many diverse research areas, most recently including graph algorithms, database privacy, approximation algorithms, algorithmic mechanism design, cryptography, algorithms for large data sets and the theory of distributed computing, among others.

Algorithms for Very Large Data Sets
Data from search engine indices, search engine query logs, and instant messaging clients can be extremely useful in improving search quality and understanding social networks, but their sheer scale makes it difficult to process them using existing algorithms . We continue to design, implement, and test new algorithms for analyzing these Internet scale data sets with a view towards making search and instant messaging more effective and efficient.

Algebraic Computation
We also conduct research in some aspects of pure theory; one area of recent interest involves algebraic computation and the hardness of computing the determinant of a matrix in various restricted settings. It is well-known that determinant is efficiently computable over any commutative ring, but the question of whether this remains true in a non-commutative setting is generally open. Aside from the philosophical interest in the computational power of commutativity, an answer to this question would have direct bearing on applications in approximate counting.

Approximation Algorithms
Many natural optimization problems, such as the traveling salesman problem, are NP-hard. The area of approximation algorithms attempts to design efficient algorithms that are guaranteed to produce solutions that have cost within a small factor of the optimal. Closely related is the area of hardness of approximation: for some problems, we can prove that even getting good approximations efficiently is as hard as solving these (NP-hard) problems exactly. We are studying the design and analysis of approximation algorithms, often using tools from and contributing to the areas of metric embeddings and combinatorial optimization.

Algorithmic Game Theory
We study several problems related to game theory. These problems are motivated by e-commerce applications and applications of game theory to computer system and network design. In mechanism design, we aim to develop mechanisms with useful properties which optimize an objective function, such as seller's revenue or global welfare of the system, in the worst- or average-case. Our work shows that techniques from learning, on-line algorithms, and coding theory can be applied to mechanism design. We also study complexity of computational problems that come up in implementations of game-theoretic mechanisms. Finally, we are interested in research problems on the borderline of Economics and Computer Science, including Game Theory, Social Networks, and Electronic Markets.

Database Privacy
Statistical databases such as are produced by the US Census contain a large volume of illuminating and potentially useful data. They also run the risk of revealing a great deal of specific information about the participants, which participants generally dislike. There is an inherent tradeoff between the utility that databases can offer and the privacy they afford their constituents. We are studying this tradeoff formally, attempting to understand the relationship between privacy and utilily, and thereby find a comfortable position between the extremes of fully disclosed and completely withheld data.

Multi-Armed Bandits
An umbrella project for several independent projects that study various MAB formulations motivated by web search and ad placement. The (basic) MAB problem is a classical problem in Machine Learning in which an online algorithm chooses from a set of strategies in a sequence of n trials so as to maximize the total payoff of the chosen strategies.

Network and Graph Algorithms
We are exploring the structure of large and dynamic networks and devising algorithms for them. This research area includes various sub-topics, including but not limited to: routing and shortest paths; network flows; locality-sensitive methods such as caching and nearest-object location; overlay networks and peer-to-peer tools; scalable information dissemination and gossip; metric embeddings.

Sequoia
Sequoia aims to make distributed applications network-aware. That is, enable applications to take advantage of the characteristics of the underlying network such as proximity, bandwidth capacity, and topology. It intends to achieve this through the key concept of prediction trees, a virtual topology of the network, where virtual nodes representing routers connect real end hosts, and carefully computed edge weights model path properties such as latency and bandwidth.

SPA
Shortest path problems are among the most fundamental optimization problems with many applications. The project is devoted to theoretical and experimental study of algorithms for the shortest path and related problems, including single-source, feasibility, minimum mean cycle, and point-to-point shortest paths.

Algorithms Research is an international peer-reviewed journal which presents papers on algorithms that are inherently discrete and finite and that have some natural mathematical content, either in their objective or in their analysis. The journal features new algorithms and data structures, new analyses or comparisons of known algorithms, complexity studies, and sharply focused review articles of subject areas that are currently active.
Subject areas suitable for publication include, but are not limited to the following fields:

- Algorithms on Graphs
- Arithmetic Algorithms
- Combinatorial Searches and Objects
- Complexity Studies
- Complexity Theory
- Discrete Optimization
- Geometric Algorithms
- Methods of Algorithmic Analysis
- New Algorithms and Data Structures
- New Analyses or Comparisons of Known Algorithms

Question 4:
What are the applications of Tries?

Answer 4:

Trees are used enormously in computer programming. These can be used for improving database search times (binary search trees, 2-3 trees, AVL trees, red-black trees), Game programming (minimax trees, decision trees, pathfinding trees), 3D graphics programming (quadtrees, octrees), Arithmetic Scripting languages (arithmetic precedence trees), Data compression (Huffman trees), and file systems (Btrees, sparse indexed trees, tries).

The General tree (also known as Linked Trees) is a generic tree that has one root node, and every node in the tree can have an unlimited number of child nodes. One popular use of this kind of tree is in Family Tree programs. In game programming, many games use these types of trees for decision-making processes

The interesting thing about using a tree for decision-making is that the options are cut down for every level of the tree as we go down, greatly simplifying the subsequent moves and improving the speed at which the AI program makes a decision.