Module 6 Advanced Trees Graphs

75. Code for search an element in Binary Threaded Tree  
bool search(int key) { Node \*tmp = root->left;  
for (;;) { if (tmp->key < key) { if (tmp->rightThread) return false; tmp = tmp->right; } else if (tmp->key > key) { if (tmp->leftThread) return false; tmp = tmp->left; } else { return true; } } }

76. Code to Print Binary Threaded Tree  
void printTree() { Node \*tmp = root, \*p; for (;;) { p = tmp; tmp = tmp->right; if (!p->rightThread) { while (!tmp->leftThread) { tmp = tmp->left; } } if (tmp == root) break; cout<<tmp->key<<" "; } cout<<endl; }

77. Code to traverse B+ tree  
void traverse(B+TreeNode \*p) { cout<<endl; int i; for (i = 0; i < p->n; i++) { if (p->leaf == false)  
traverse(p->child\_ptr[i]); cout << " " << p->data[i]; } if (p->leaf == false) traverse(p->child\_ptr[i]); cout<<endl; }

78. Code to Sort B+ Tree  
void sort(int \*p, int n) { int i, j, temp; for (i = 0; i < n; i++) { for (j = i; j <= n; j++) { if (p[i] > p[j]) { temp = p[i]; p[i] = p[j]; p[j] = temp; } } } }

79. Code to find the Height of AVL Tree  
int avlTree::height(avl\_node \*temp) { int h = 0; if (temp != NULL) { int l\_height = height (temp->left); int r\_height = height (temp->right); int max\_height = max (l\_height, r\_height); h = max\_height + 1; } return h; }

80. Code to find Height Difference in AVL Tree  
int avlTree::diff(avl\_node \*temp) { int l\_height = height (temp->left); int r\_height = height (temp->right); int b\_factor= l\_height - r\_height; return b\_factor; }

81. Code for balancing AVL Tree  
avl\_node \*avlTree::balance(avl\_node \*temp) { int bal\_factor = diff (temp); if (bal\_factor > 1) { if (diff (temp->left) > 0) temp = ll\_rotation (temp); else temp = lr\_rotation (temp); } else if (bal\_factor < -1) { if (diff (temp->right) > 0) temp = rl\_rotation (temp); else temp = rr\_rotation (temp); } return temp; }

82. Code to Insert an element in AVL Tree  
avl\_node \*avlTree::insert(avl\_node \*root, int value) { if (root == NULL) { root = new avl\_node; root->data = value; root->left = NULL; root->right = NULL; return root; } else if (value < root->data) { root->left = insert(root->left, value); root = balance (root); } else if (value >= root->data) { root->right = insert(root->right, value); root = balance (root); } return root; }

83. Code to display AVL Tree  
void avlTree::display(avl\_node \*ptr, int level) { int i; if (ptr!=NULL) { display(ptr->right, level + 1);  
printf("\n"); if (ptr == root) cout<<"Root -> "; for (i = 0; i < level && ptr != root; i++) cout<<" "; cout<<ptr->data; display(ptr->left, level + 1); } }

84. Code for InOrder Traversal of AVL Tree  
void avlTree::inorder(avl\_node \*tree) { if (tree == NULL) return; inorder (tree->left); cout<<tree->data<<" "; inorder (tree->right); }

85. Code for PreOrder Traversal of AVL Tree  
void avlTree::preorder(avl\_node \*tree) { if (tree == NULL) return; cout<<tree->data<<" "; preorder (tree->left); preorder (tree->right); }

86. Code for PostOrder Traversal of AVL Tree  
void avlTree::postorder(avl\_node \*tree) { if (tree == NULL) return; postorder ( tree ->left ); postorder ( tree ->right ); cout<<tree->data<<" "; }

87. Code for Right-Right Rotation (RR) of AVL Tree  
avl\_node \*avlTree::rr\_rotation(avl\_node \*parent) { avl\_node \*temp; temp = parent->right; parent->right = temp->left; temp->left = parent; return temp; }

88. Code for Left-Left Rotation (LL) of AVL Tree  
avl\_node \*avlTree::ll\_rotation(avl\_node \*parent) { avl\_node \*temp; temp = parent->left; parent->left = temp->right; temp->right = parent; return temp; }

89. Code for Left-Right Rotation (LR) of AVL Tree  
avl\_node \*avlTree::lr\_rotation(avl\_node \*parent) { avl\_node \*temp; temp = parent->left; parent->left = rr\_rotation (temp); return ll\_rotation (parent); }

90. Code for Right-Left Rotation (RL) of AVL Tree  
avl\_node \*avlTree::rl\_rotation(avl\_node \*parent) { avl\_node \*temp; temp = parent->right; parent->right = ll\_rotation (temp); return rr\_rotation (parent); }

98. Code for graph using Adjacency List  
void addReverseEdge(int src,int dest) { np1 = new adj\_list; np1->dest = src; np1->next = NULL; if (array[dest].ptr == NULL) { array[dest].ptr = np1; q = array[dest].ptr; q->next = NULL; } else { q = array[dest].ptr; while (q->next != NULL) { q = q->next; q->next = np1; } } void addEdge(int src,int dest) { np<!--more--> = new adj\_list; np->dest = dest; np->next = NULL; if (array[src].ptr == NULL) { array[src].ptr = np; p = array[src].ptr; p->next = NULL; } else { p = array[src].ptr; while (p->next != NULL) p = p->next; p->next = np; } addReverseEdge(src,dest); }

99. Program to Implement Adjacency Matrix #include <iostream> #include <cstdlib> using namespace std; #define MAX 20 class AdjacencyMatrix { private: int n; int \*\*adj; bool \*visited; public: AdjacencyMatrix(int n) { this->n = n; visited = new bool [n]; adj = new int\* [n]; for (int i = 0; i < n; i++) { adj[i] = new int [n]; for(int j = 0; j < n; j++) adj[i][j] = 0; } } void add\_edge(int origin, int destin) { if( origin > n || destin > n || origin < 0 || destin < 0) cout<<"Invalid edge!\n"; else adj[origin - 1][destin - 1] = 1; } void display() { int i,j; for(i = 0;i < n;i++) { for(j = 0; j < n; j++) cout<<adj[i][j]<<" "; cout<<endl; } } }; int main() { int nodes, max\_edges, origin, destin; cout<<"Enter number of nodes: "; cin>>nodes; AdjacencyMatrix am(nodes); max\_edges = nodes \* (nodes - 1); for (int i = 0; i < max\_edges; i++) {  
cout<<"Enter edge (-1 -1 to exit): "; cin>>origin>>destin; if((origin == -1) && (destin == -1)) break; am.add\_edge(origin, destin); } am.display(); return 0; }