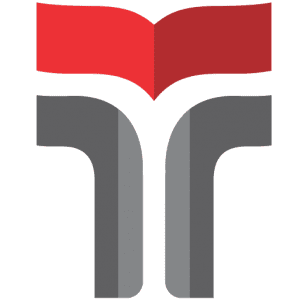
**LAPORAN PRAKTIKUM**

**MODUL 9**

**GRAPH DAN TREE**



**Disusun Oleh:**

RiyonAryono **: 2211102241**

**Dosen**

Muhamad Azrino Gustalika

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM**

**PURWOKERTO**

**2023**

**BAB I**

**TUJUAN PRAKTIKUM**

1. **Tujuan Praktikum**
2. Mahasiswa mampu menjelaskan definisi dan konsep dari Graph dan Tree
3. Mahasiswa mampu menerapkan Graph dan Tree kedalam pemograman

**BAB II**

**DASAR TEORI**

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai:

G = (V,E)

Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:

A picture containing line, circle, screenshot, diagram

Description automatically generated

Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan,dan pemodelan data.

A picture containing diagram, line, pattern

Description automatically generated

* 1. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
  2. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
  3. Weigth Graph : Graph yang mempunyai nilai pada tiap edgenya.

Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

A screenshot of a computer

Description automatically generated with low confidence

**BAB III**

**LATIHAN & TUGAS**

1. **Guided**

* Demo Graph

*Source Code*

|  |
| --- |
| #include <iostream>  #include <iomanip>  using namespace std;  string simpul[7] = {"Ciamis", "Bandung", "Bekasi","Tasikmalaya", "Cianjur", "Purwokerto", "Yogjakarta"};  int busur[7][7] = {{0, 7, 8, 0, 0, 0, 0}, {0, 0, 5, 0, 0, 15, 0}, {0, 6, 0, 0, 5, 0, 0}, {0, 5, 0, 0, 2, 4, 0},{23, 0, 0, 10, 0, 0, 8}, {0, 0, 0, 0, 7, 0, 3},{0, 0, 0, 0, 9, 4, 0}};  void tampilGraph()  {  for (int baris = 0; baris < 7; baris++)  {  cout << " " << setiosflags(ios::left) << setw(15)  << simpul[baris] << " : ";  for (int kolom = 0; kolom < 7; kolom++)  {  if (busur[baris][kolom] != 0)  {  cout << " " << simpul[kolom] << "("  << busur[baris][kolom] << ")";  }  }  cout << endl;  }  }  int main(){  tampilGraph();  return 0;  } |

*Output*

A screen shot of a computer screen

Description automatically generated with low confidence

Penjelesan

Kode di atas adalah contoh implementasi graf dalam bentuk adjacency matrix menggunakan bahasa pemrograman C++. Terdapat array simpul yang berisi simpul-simpul graf, serta array dua dimensi busur yang merepresentasikan matriks ketetanggaan (adjacency matrix). Fungsi tampilGraph() digunakan untuk menampilkan graf ke layar. Dalam fungsi main(), fungsi tampilGraph() dipanggil untuk menampilkan graf tersebut.

* Demo Tree

*Source Code*

|  |
| --- |
| #include <iostream>  using namespace std;  // Program Binary Tree  // Deklarasi Pohon  struct Pohon{  char data;  Pohon \*left, \*right, \*parent;  };  Pohon \*root, \*baru;  // Inisialisasi  void init(){  root = NULL;  }  // Cek Node  int isEmpty(){  if(root == NULL){  return 1;  }else{  return 0;  }  }  // Buat Node Baru  void buatNode(char data){  if(isEmpty() == 1){  root = new Pohon();  root->data = data;  root->left = NULL;  root->right = NULL;  root->parent = NULL;  cout << "\nNode " << data << " berhasil dibuat menjadi root."<<endl;  }else{  cout << "\nPohon sudah dibuat" << endl;  }  }  // Tambah Kiri  Pohon \*insertLeft(char data, Pohon \*node){  if(isEmpty() == 1){  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  }else{  // cek apakah child kiri ada atau tidak  if(node->left != NULL ){  //kalau ada  cout << "\nNode " << node->data << " sudah ada di child kiri!" << endl;  }else{  // Kalau tidak ada  baru = new Pohon();  baru->data = data;  baru->left = NULL;  baru->right = NULL;  baru->parent = node;  node->left = baru;  cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << baru->parent->data << endl;  return baru;  }  }  }  // Tambah kanan  Pohon \*insertRight(char data, Pohon \*node){  if(root == NULL){  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  }else{  // cek apakah child kanan ada atau tidak  if(node->right != NULL ){  //kalau ada  cout << "\nNode " << node->data << " sudah ada di child kanan!" << endl;  }else{  // Kalau tidak ada  baru = new Pohon();  baru->data = data;  baru->left = NULL;  baru->right = NULL;  baru->parent = node;  node->right = baru;  cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << baru->parent->data << endl;  return baru;  }  }  }  // Update Data  void update(char data, Pohon \*node){  if(isEmpty() == 1){  cout << "\nBuat tree terlebih dahulu!" << endl;  }else{  if( !node ){  cout << "\nNode yang ingin diganti tidak ada!!" << endl;  }else{  char temp = node->data;  node->data = data;  cout << "\n Node " << temp << " berhasil diubah menjadi " << data << endl;  }  }  }  // Lihat Isi Data Tree  void retrieve(Pohon \*node){  if (!root) {  cout << "\n Buat tree terlebih dahulu!" << endl;  }else{  if (!node){  cout << "\n Node yang ditunjuk tidak ada!" << endl;  }else{  cout << "\n Data node : " << node->data << endl;  }  }  }  void find(Pohon \*node){  if(!root){  cout << "\n Buat tree terlebih dahulu" << endl;  }else{  if(!node){  cout << "\n Node yang ditunujuk tidak ada!" << endl;  }else{  cout << "\n Data node: " << node->data << endl;  cout << "Root: " << root->data << endl;  if(!node->parent){  cout << "Parent : {tidak punya parent} " << endl;  }else{  cout << "Parent : " << node->parent->data << endl;  }  if(node->parent != NULL && node->parent->left != node && node->parent->right == node){  cout << " Sibling : " << node->parent->left->data << endl;  }else if(node->parent != NULL && node->parent->right != node && node->parent->left == node){  cout << " Sibling : " << node->parent->right->data <<endl;  }else{  cout << " Sibling : {tidak punya sibling}" << endl;  }  if(!node->left){  cout << " Child kiri : {tidak punya child kiri }" << endl;  }else{  cout << " Child kiri : " << node->left->data <<endl;  }  if(!node->right){  cout << " Child kanan : {tidak pynya child kanan}" << endl;  }else{  cout << "Child kanan : " << node->right->data << endl;  }  }  }  }  // Penelusuran (Traversal)  // PreOrder  void preOrder(Pohon \*node = root){  if(!root){  cout << "\n Buat tree terlebih dahulu"<<endl;  }else{  if(node != NULL){  cout << " " << node->data << ", ";  preOrder(node->left);  preOrder(node->right);  }  }  }  //in Order  void inOrder(Pohon \*node = root){  if(!root){  cout << "\n Buat tree terlebih dahulu"<<endl;  }else{  if(node != NULL){  inOrder(node->left);  cout << " " << node->data << ", ";  inOrder(node->right);  }  }  }  // Post Order  void postOrder(Pohon \*node = root){  if(!root){  cout << "\n Buat tree terlebih dahulu"<<endl;  }else{  if(node != NULL){  postOrder(node->left);  postOrder(node->right);  cout << " " << node->data << ", ";  }  }  }  // Hapus Node Tree  void deleteTree(Pohon \*node){  if(!root){  cout << "\n Buat tree terlebih dahulu" << endl;  }else{  if(node != NULL){  if(node != root){  node->parent->left = NULL;  node->parent->right = NULL;  }  deleteTree(node->left);  deleteTree(node->right);  if(node == root){  delete root;  root = NULL;  }else{  delete node;  }  }  }  }  // Hapus SubTree  void deleteSub(Pohon \*node){  if(!root){  cout << "\n Buat tree terlebih dahulu" << endl;  }else{  deleteTree(node->left);  deleteTree(node->right);  cout << "\n Node subtree" << node->data << " berhasil dihapus." << endl;  }  }  // Hapus Tree  void clear(){  if(!root){  cout << "\n Buat tree terebih dahulu!!"<<endl;  }else{  deleteTree(root);  cout << "\n Pohon berhasil dihapus."<<endl;  }  }  // Cek Size Tree  int size(Pohon \*node = root){  if(!root){  cout << "\n Buat tree terlebih dahulu!!"<<endl;  return 0;  }else{  if(!node){  return 0;  }else{  return 1 + size(node->left) + size(node->right);  }  }  }  // Cek Height Level Tree  int height(Pohon \*node = root){  if(!root){  cout << "\n Buat tree terlebih dahulu!"<<endl;  }else{  if(!node){  return 0;  }else{  int heightKiri = height(node->left);  int heightKanan = height(node->right);  if(heightKiri >= heightKanan){  return heightKiri + 1;  }else{  return heightKanan + 1;  }  }  }  }  // Karakteristik Tree  void charateristic(){  cout << "\n Size Tree : " << size() << endl;  cout << " Height Tree : " << height() << endl;  cout << " Average Node of Tree : " << size() / height() << endl;  }  int main(){  buatNode('A');  Pohon \*nodeB, \*nodeC, \*nodeD, \*nodeE, \*nodeF, \*nodeG, \*nodeH, \*nodeI, \*nodeJ;  nodeB = insertLeft('B',root);  nodeC = insertRight('C',root);  nodeD = insertLeft('D',nodeB);  nodeE = insertRight('E',nodeB);  nodeF = insertLeft('F',nodeC);  nodeG = insertLeft('G', nodeE);  nodeH = insertRight('H',nodeE);  nodeI = insertLeft('I',nodeG);  nodeJ = insertRight('J',nodeG);  retrieve(nodeC);  find(nodeC);  cout << "\n PreOrder :" << endl;  preOrder(root);  cout << "\n" << endl;  cout << " InOrder :" << endl;  inOrder(root);  cout << "\n"<< endl;  cout << " PostOrder :" << endl;  postOrder(root);  cout << "\n" << endl;  charateristic();  deleteSub(nodeE);  cout << "\n PreOrder :" << endl;  preOrder();  cout << "\n" << endl;  charateristic();  } |

Output

A screenshot of a computer

Description automatically generated

Penjelasan

Kode di atas adalah contoh implementasi pohon biner menggunakan bahasa pemrograman C++. Program ini memungkinkan pembuatan, pemrosesan, dan penghapusan pohon biner. Fungsi-fungsi yang disediakan termasuk pembuatan simpul, penambahan simpul anak, pemutakhiran data, penelusuran pohon, penghapusan pohon, dan perhitungan karakteristik pohon seperti ukuran dan tinggi. Pada main(), pohon dibuat dan beberapa simpul ditambahkan. Informasi dan hasil penelusuran pohon ditampilkan, serta dilakukan penghapusan subtree. Karakteristik pohon dicetak sebelum dan setelah penghapusan.

1. **Unguided**

* Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

*Source Code*

|  |
| --- |
| #include <iostream>  #include <vector>  #include <map>  #include <iomanip>  using namespace std;  void inputSimpul(vector<string> &namaSimpul)  {  int jumlahSimpul;  cout << "Silahkan masukkan jumlah simpul: ";  cin >> jumlahSimpul;  namaSimpul.resize(jumlahSimpul);  cout << "Silahkan masukkan nama simpul:\n";  for (int i = 0; i < jumlahSimpul; i++)  {  cout << "Simpul " << i + 1 << ": ";  cin >> namaSimpul[i];  }  }  void inputBobotAntarSimpul(vector<string> &namaSimpul, map<pair<string, string>, int> &bobotAntarSimpul)  {  cout << "Silahkan masukkan bobot antar simpul:\n";  for (int i = 0; i < namaSimpul.size(); i++)  {  for (int j = 0; j < namaSimpul.size(); j++)  {  string simpul1 = namaSimpul[i];  string simpul2 = namaSimpul[j];  int bobot;  cout << simpul1 << " --> " << simpul2 << " = ";  cin >> bobot;  bobotAntarSimpul[make\_pair(simpul1, simpul2)] = bobot;  }  }  }  void cetakGrafik(vector<string> &namaSimpul, map<pair<string, string>, int> &bobotAntarSimpul)  {  cout << "\n\t";  for (int i = 0; i < namaSimpul.size(); i++)  {  cout << setw(10) << namaSimpul[i] << "\t";  }  cout << endl;  for (int i = 0; i < namaSimpul.size(); i++)  {  cout << namaSimpul[i] << "\t";  for (int j = 0; j < namaSimpul.size(); j++)  {  string simpul1 = namaSimpul[i];  string simpul2 = namaSimpul[j];  int bobot = bobotAntarSimpul[make\_pair(simpul1, simpul2)];  cout << setw(10) << bobot << "\t";  }  cout << endl;  }  }  int main()  {  vector<string> namaSimpul;  map<pair<string, string>, int> bobotAntarSimpul;  inputSimpul(namaSimpul);  inputBobotAntarSimpul(namaSimpul, bobotAntarSimpul);  cetakGrafik(namaSimpul, bobotAntarSimpul);  return 0;  } |

*Output*

A picture containing text, screenshot, font

Description automatically generated

Penjelasan

Kode di atas adalah program yang mengimplementasikan representasi grafik menggunakan struktur data map dan vector. Program ini memungkinkan pengguna untuk memasukkan jumlah simpul dan bobot antar simpul dalam grafik. Setelah memasukkan informasi tersebut, program akan mencetak grafik berisi bobot antar simpul. Program menggunakan struktur data map untuk menyimpan bobot antar simpul berdasarkan pasangan simpul yang terhubung.

* Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!

*Source Code*

|  |
| --- |
| #include <iostream>  using namespace std;  struct Pohon  {  char data;  Pohon \*left, \*right, \*parent;  };  Pohon \*root, \*baru;  void init()  {  root = NULL;  }  int isEmpty()  {  if (root == NULL)  {  return 1;  }  else  {  return 0;  }  }  void buatNode(char data)  {  if (isEmpty() == 1)  {  root = new Pohon();  root->data = data;  root->left = NULL;  root->right = NULL;  root->parent = NULL;  cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;  }  else  {  cout << "\nPohon sudah dibuat" << endl;  }  }  Pohon \*insertLeft(char data, Pohon \*node)  {  if (isEmpty() == 1)  {  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  }  else  {  // cek apakah child kiri ada atau tidak  if (node->left != NULL)  {  // kalau ada  cout << "\nNode " << node->data << " sudah ada di child kiri!" << endl;  }  else  {  // Kalau tidak ada  baru = new Pohon();  baru->data = data;  baru->left = NULL;  baru->right = NULL;  baru->parent = node;  node->left = baru;  cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << baru->parent->data << endl;  return baru;  }  }  }  // Tambah kanan  Pohon \*insertRight(char data, Pohon \*node)  {  if (root == NULL)  {  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  }  else  {  // cek apakah child kanan ada atau tidak  if (node->right != NULL)  {  // kalau ada  cout << "\nNode " << node->data << " sudah ada di child kanan!" << endl;  }  else  {  // Kalau tidak ada  baru = new Pohon();  baru->data = data;  baru->left = NULL;  baru->right = NULL;  baru->parent = node;  node->right = baru;  cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << baru->parent->data << endl;  return baru;  }  }  }  // Update Data  void update(char data, Pohon \*node)  {  if (isEmpty() == 1)  {  cout << "\nBuat tree terlebih dahulu!" << endl;  }  else  {  if (!node)  {  cout << "\nNode yang ingin diganti tidak ada!!" << endl;  }  else  {  char temp = node->data;  node->data = data;  cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;  }  }  }  // Lihat Isi Data Tree  void retrieve(Pohon \*node)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu!" << endl;  }  else  {  if (!node)  {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  }  else  {  cout << "\nData node : " << node->data << endl;  }  }  }  void find(Pohon \*node)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  if (!node)  {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  }  else  {  cout << "\nData node: " << node->data << endl;  cout << "Root: " << root->data << endl;  if (!node->parent)  {  cout << "Parent : {tidak punya parent} " << endl;  }  else  {  cout << "Parent : " << node->parent->data << endl;  }  if (node->parent != NULL && node->parent->left != node && node->parent->right == node)  {  cout << "Sibling : " << node->parent->left->data << endl;  }  else if (node->parent != NULL && node->parent->right != node && node->parent->left == node)  {  cout << "Sibling : " << node->parent->right->data << endl;  }  else  {  cout << "Sibling : {tidak punya sibling}" << endl;  }  if (!node->left)  {  cout << "Child kiri : {tidak punya child kiri }" << endl;  }  else  {  cout << "Child kiri : " << node->left->data << endl;  }  if (!node->right)  {  cout << "Child kanan : {tidak pynya child kanan}" << endl;  }  else  {  cout << "Child kanan : " << node->right->data << endl;  }  }  }  }  // Penelusuran (Traversal)  // PreOrder  void preOrder(Pohon \*node = root)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  if (node != NULL)  {  cout << " " << node->data << ", ";  preOrder(node->left);  preOrder(node->right);  }  }  }  // in Order  void inOrder(Pohon \*node = root)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  if (node != NULL)  {  inOrder(node->left);  cout << " " << node->data << ", ";  inOrder(node->right);  }  }  }  // Post Order  void postOrder(Pohon \*node = root)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  if (node != NULL)  {  postOrder(node->left);  postOrder(node->right);  cout << " " << node->data << ", ";  }  }  }  // Hapus Node Tree  void deleteTree(Pohon \*node)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  if (node != NULL)  {  if (node != root)  {  node->parent->left = NULL;  node->parent->right = NULL;  }  deleteTree(node->left);  deleteTree(node->right);  if (node == root)  {  delete root;  root = NULL;  }  else  {  delete node;  }  }  }  }  // Hapus SubTree  void deleteSub(Pohon \*node)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu" << endl;  }  else  {  deleteTree(node->left);  deleteTree(node->right);  cout << "\nNode subtree" << node->data << " berhasil dihapus." << endl;  }  }  // Hapus Tree  void clear()  {  if (!root)  {  cout << "\nBuat tree terebih dahulu!!" << endl;  }  else  {  deleteTree(root);  cout << "\nPohon berhasil dihapus." << endl;  }  }  // Cek Size Tree  int size(Pohon \*node = root)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu!!" << endl;  return 0;  }  else  {  if (!node)  {  return 0;  }  else  {  return 1 + size(node->left) + size(node->right);  }  }  }  // Cek Height Level Tree  int height(Pohon \*node = root)  {  if (!root)  {  cout << "\nBuat tree terlebih dahulu!" << endl;  }  else  {  if (!node)  {  return 0;  }  else  {  int heightKiri = height(node->left);  int heightKanan = height(node->right);  if (heightKiri >= heightKanan)  {  return heightKiri + 1;  }  else  {  return heightKanan + 1;  }  }  }  }  // Karakteristik Tree  void characteristic()  {  cout << "\nSize Tree : " << size() << endl;  cout << "Height Tree : " << height() << endl;  cout << "Average Node of Tree : " << size() / height() << endl;  }  int main()  {  init();  int riyon\_221102241;  char data;  Pohon \*currentNode = NULL;  while (true)  {  cout << "\nMenu:" << endl;  cout << "1. Buat Node Baru" << endl;  cout << "2. Tambah Node Kiri" << endl;  cout << "3. Tambah Node Kanan" << endl;  cout << "4. Update Data Node" << endl;  cout << "5. Lihat Isi Data Node" << endl;  cout << "6. Cari Node" << endl;  cout << "7. PreOrder Traversal" << endl;  cout << "8. InOrder Traversal" << endl;  cout << "9. PostOrder Traversal" << endl;  cout << "10. Hapus SubTree" << endl;  cout << "11. Clear Tree" << endl;  cout << "12. Karakteristik Tree" << endl;  cout << "13. Exit" << endl;  cout << "Pilih menu: ";  cin >> riyon\_221102241;  switch (riyon\_221102241)  {  case 1:  if (isEmpty() == 1)  {  cout << "\nMasukkan data node baru: ";  cin >> data;  buatNode(data);  currentNode = root;  }  else  {  cout << "\nTree sudah dibuat!" << endl;  }  break;  case 2:  if (currentNode != NULL)  {  cout << "\nMasukkan data node kiri: ";  cin >> data;  currentNode = insertLeft(data, currentNode);  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 3:  if (currentNode != NULL)  {  cout << "\nMasukkan data node kanan: ";  cin >> data;  currentNode = insertRight(data, currentNode);  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 4:  if (currentNode != NULL)  {  cout << "\nMasukkan data node baru: ";  cin >> data;  update(data, currentNode);  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 5:  if (currentNode != NULL)  {  retrieve(currentNode);  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 6:  if (currentNode != NULL)  {  find(currentNode);  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 7:  preOrder();  break;  case 8:  inOrder();  break;  case 9:  postOrder();  break;  case 10:  if (currentNode != NULL)  {  deleteSub(currentNode);  currentNode = root;  }  else  {  cout << "\nPilih node terlebih dahulu!" << endl;  }  break;  case 11:  clear();  currentNode = NULL;  break;  case 12:  characteristic();  break;  case 13:  cout << "\nTerima kasih!" << endl;  exit(0);  default:  cout << "\nPilih menu yang valid!" << endl;  }  }  return 0;  } |

Output

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated with low confidence

A black background with white text

Description automatically generated with low confidence





Penjelasan

Program di atas adalah implementasi sederhana dari struktur data pohon biner menggunakan bahasa pemrograman C++. Pada program ini, Anda dapat membuat node baru, menambahkan node kiri atau kanan, mengubah data node, melihat isi data node, mencari node, melakukan penelusuran preOrder, inOrder, dan postOrder, menghapus subtree, menghapus seluruh pohon, dan melihat karakteristik pohon seperti ukuran (jumlah node), tinggi, dan rata-rata node per level.

**BAB IV**

**KESIMPULAN**

Pemahaman tentang tree dan graph di C++ sangat penting dalam pengembangan aplikasi yang melibatkan struktur data kompleks, seperti algoritma pencarian, optimasi jaringan, analisis data, dan banyak lagi. Dengan pemahaman yang baik tentang konsep ini, kita dapat membuat program yang efisien dan efektif dalam memanipulasi dan memproses data terkait struktur tree dan graph.