# LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

**MODUL 9**



# Dosen : Wahyu Andi Saputra, S.Pd., M.Eng.

**Disusun oleh:**

# ABDA FIRAS RAHMAN

# 2311102049

# IF-11-B

**PROGRAM STUDI S1 INFORMATIKA FAKULTAS INFORMATIKA**

# INSTITUT TEKNOLOGI TELKOM PURWOKERTO 2024

**BAB I**

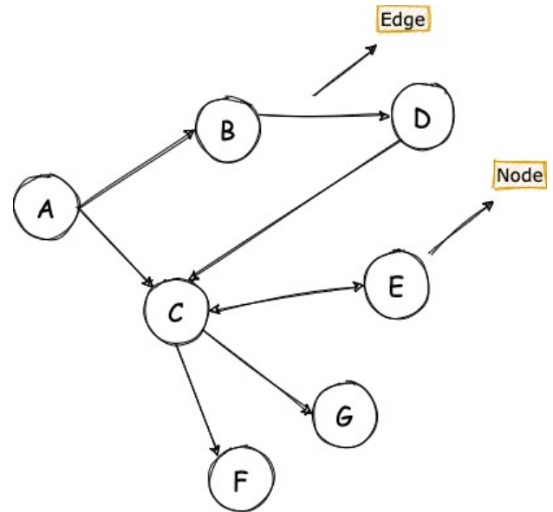
**DASAR TEORI**

GRAF DAN TREE

1. **Pengertian**

* 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 sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

G = (V, E)

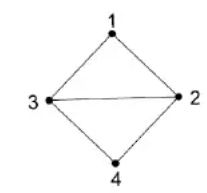


**Jenis jenis graf**

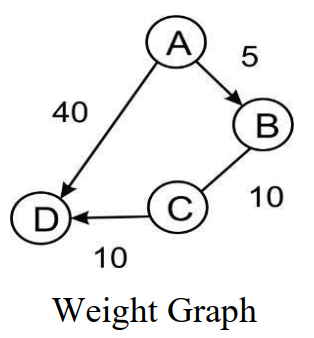
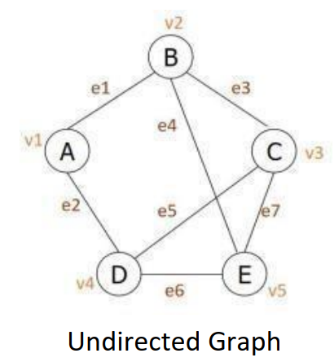
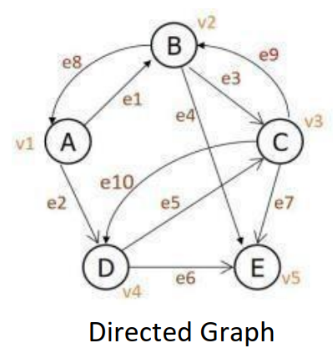
1. Graf Sederhana (Simple Graph)

Graf yang tidak mengandung gelang maupun sisi ganda dinamakangraf sederhana. Pada graf sederhana, sisi adalah pasangan tak terurut (unordered pairs).

Jadi, menuliskan sisi (u,v) sama saja dengan (v,u).kita dapat juga mendefinisikan graf sederhana G=(V,E) terdiri darihimpunan tidak kosong simpul-simpul dan E adalah himpunanpasangan tak terurut yang berbeda yang disebut sisi.

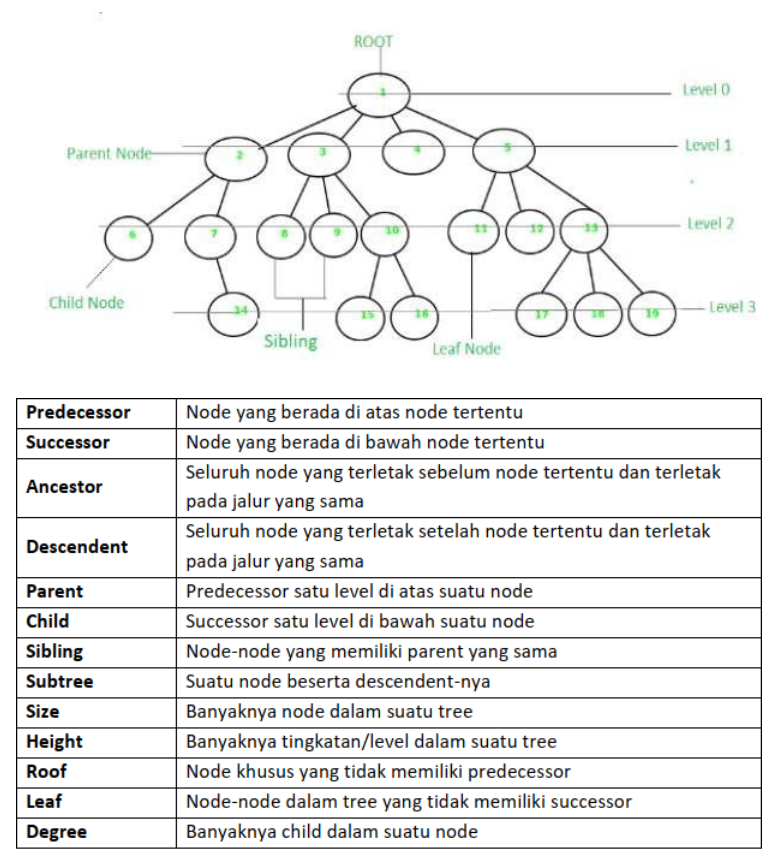


1. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
2. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
3. Weight Graph : Graph yang mempunyai nilai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



TREE ATAU POHON

Dalam ilmu komputer, pohon/tree 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, dimana 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 hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



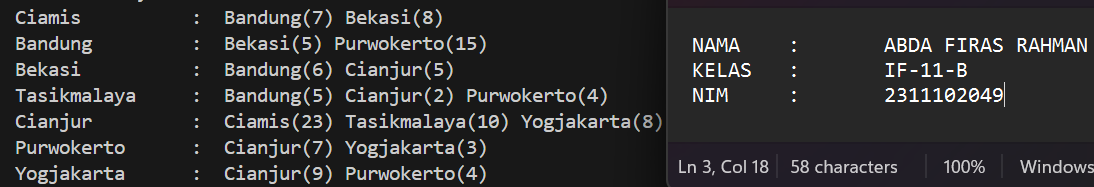
Operasi pada Tree

1. Create: digunakan untuk membentuk binary tree baru yang masih kosong.
2. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
3. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
4. Insert: digunakan untuk memasukkan sebuah node kedalam tree.
5. Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
6. f. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
7. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
8. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
9. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
10. Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

**GUIDED (1)**

|  |
| --- |
| #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**

****

**DESKRIPSI PROGRAM**

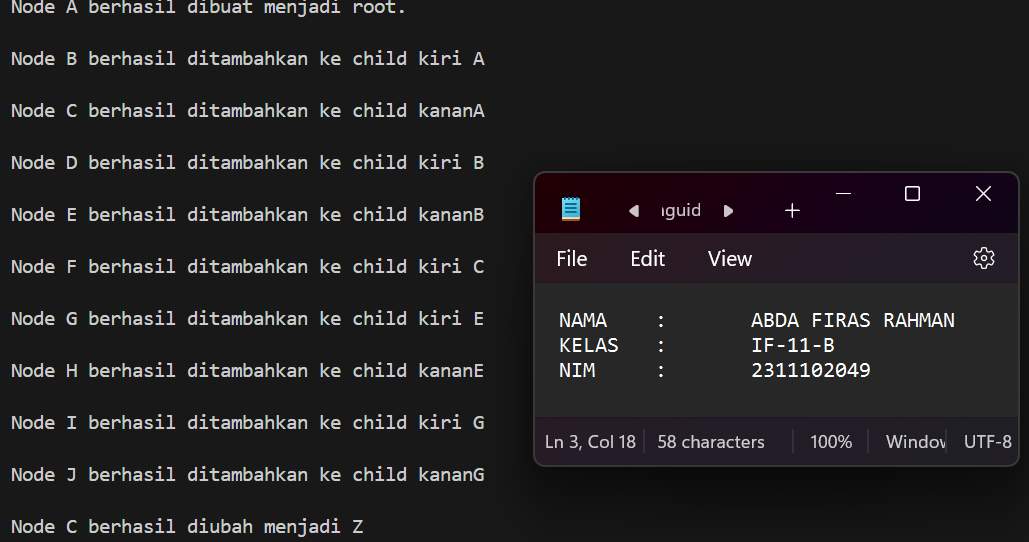
Program ini membahas tentang representasi graf berbobot yang menghubungkan beberapa kota di Jawa Barat dan sekitarnya. Graf tersebut diwakili oleh dua array, yaitu array satu dimensi "simpul" yang menyimpan nama-nama kota dan array dua dimensi "busur" yang menyimpan jarak antara setiap pasangan kota. Array "simpul" berisi tujuh elemen string, masing-masing merepresentasikan kota Ciamis, Bandung, Bekasi, Tasikmalaya, Cianjur, Purwokerto, dan Yogyakarta. Sementara array "busur" adalah array dua dimensi 7x7 yang menyimpan jarak dalam satuan tertentu antara setiap pasangan kota. Nilai 0 dalam array "busur" menunjukkan tidak ada hubungan langsung antara dua kota.

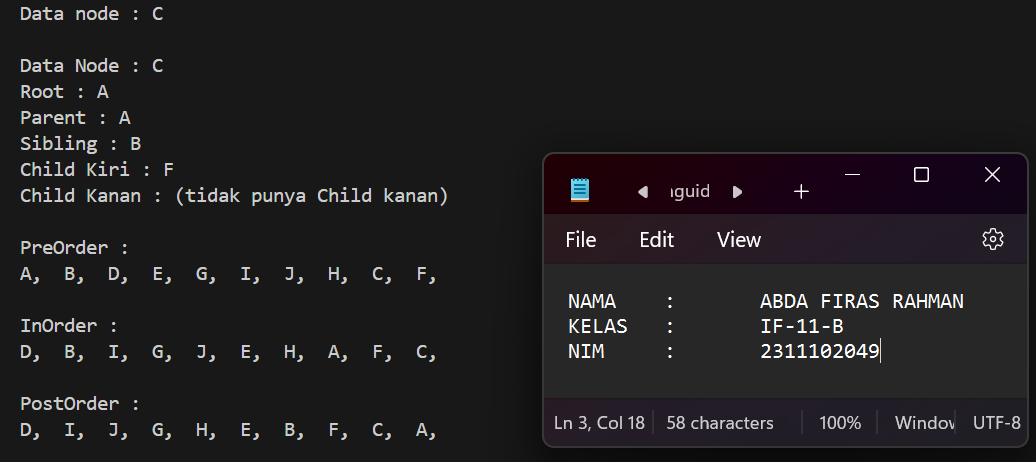
Program memiliki fungsi bernama "tampilGraph()" yang digunakan untuk menampilkan representasi graf secara visual di layar. Fungsi ini mencetak setiap baris yang merepresentasikan satu kota dan mencetak kota-kota terhubung beserta jaraknya dalam tanda kurung. Pada akhirnya, program memanggil fungsi "tampilGraph()" di dalam fungsi "main()" untuk menampilkan graf tersebut kepada pengguna.

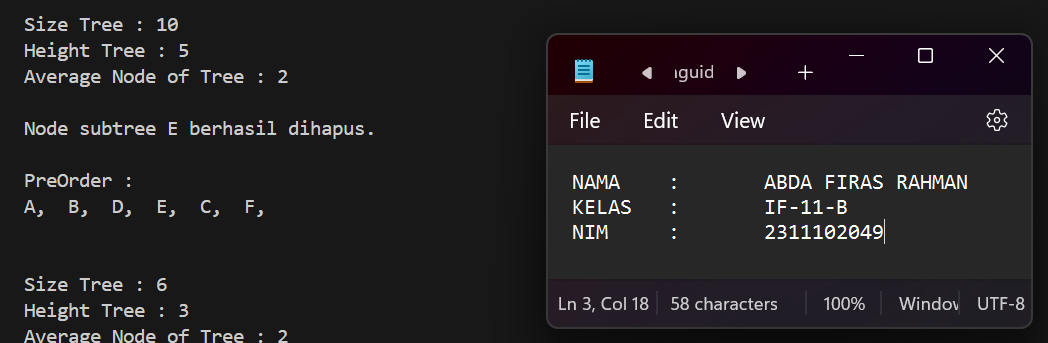
**GUIDED (2)**

|  |
| --- |
| #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; // true      else          return 0; // false  }  // 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 << "\n Node " << data << " berhasil dibuat menjadi root."               << endl;      }      else      {          cout << "\n Pohon sudah dibuat" << endl;      }  }  // Tambah Kiri  Pohon \*insertLeft(char data, Pohon \*node)  {      if (isEmpty() == 1)      {          cout << "\n Buat tree terlebih dahulu!" << endl;          return NULL;      }      else      {          // cek apakah child kiri ada atau tidak          if (node->left != NULL)          {              // kalau ada              cout << "\n Node " << node->data << " sudah ada child kiri!"                   << endl;              return NULL;          }          else          {              // kalau tidak ada              baru = new Pohon();              baru->data = data;              baru->left = NULL;              baru->right = NULL;              baru->parent = node;              node->left = baru;              cout << "\n Node " << data << " berhasil ditambahkan ke child kiri "                   << baru->parent->data << endl;              return baru;          }      }  }  // Tambah Kanan  Pohon \*insertRight(char data, Pohon \*node)  {      if (root == NULL)      {          cout << "\n Buat tree terlebih dahulu!" << endl;          return NULL;      }      else      {          // cek apakah child kanan ada atau tidak          if (node->right != NULL)          {              // kalau ada              cout << "\n Node " << node->data << " sudah ada child kanan!"                   << endl;              return NULL;          }          else          {              // kalau tidak ada              baru = new Pohon();              baru->data = data;              baru->left = NULL;              baru->right = NULL;              baru->parent = node;              node->right = baru;              cout << "\n Node " << data << " berhasil ditambahkan ke child kanan" << baru->parent->data << endl;                  return baru;          }      }  }  // Ubah Data Tree  void update(char data, Pohon \*node)  {      if (isEmpty() == 1)      {          cout << "\n Buat tree terlebih dahulu!" << endl;      }      else      {          if (!node)              cout << "\n Node 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;          }      }  }  // Cari Data Tree  void find(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;              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 punya Child kanan)" << endl;              else                  cout << " Child Kanan : " << node->right->data << endl;          }      }  }  // Penelurusan (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);          }      }  }  // inOrder  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);          }      }  }  // postOrder  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 terlebih 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;          return 0;      }      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);      update('Z', nodeC);      update('C', nodeC);      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**

****

****

****

**DESKRIPSI PROGRAM**

Program ini merupakan implementasi konsep Binary Tree dalam bahasa pemrograman C++. Binary Tree adalah struktur data tree di mana setiap node memiliki maksimum dua anak, yaitu anak kiri (left child) dan anak kanan (right child). Program ini menyediakan berbagai fungsi untuk membuat, memanipulasi, dan melakukan operasi pada sebuah binary tree.

Program dimulai dengan menyertakan library iostream yang digunakan untuk input/output stream. Kemudian, program mendefinisikan sebuah struct Pohon yang merepresentasikan sebuah node dalam binary tree. Struct Pohon memiliki tiga atribut, yaitu data (yang menyimpan nilai node), left (pointer ke anak kiri), right (pointer ke anak kanan), dan parent (pointer ke parent node).

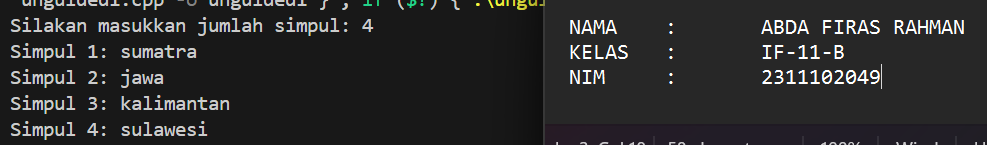
Program menyediakan fungsi-fungsi utama seperti init() untuk menginisialisasi binary tree, isEmpty() untuk memeriksa apakah binary tree kosong, buatNode() untuk membuat node baru sebagai root, insertLeft() dan insertRight() untuk menambahkan anak kiri dan kanan pada suatu node, update() untuk mengubah nilai data pada suatu node, retrieve() dan find() untuk melihat dan mencari data pada suatu node, preOrder(), inOrder(), dan postOrder() untuk melakukan traversal pada binary tree, deleteTree() untuk menghapus seluruh node pada binary tree, deleteSub() untuk menghapus subtree dari suatu node, clear() untuk menghapus seluruh isi binary tree, size() untuk menghitung jumlah node pada binary tree, dan height() untuk menghitung tinggi (level) dari binary tree.

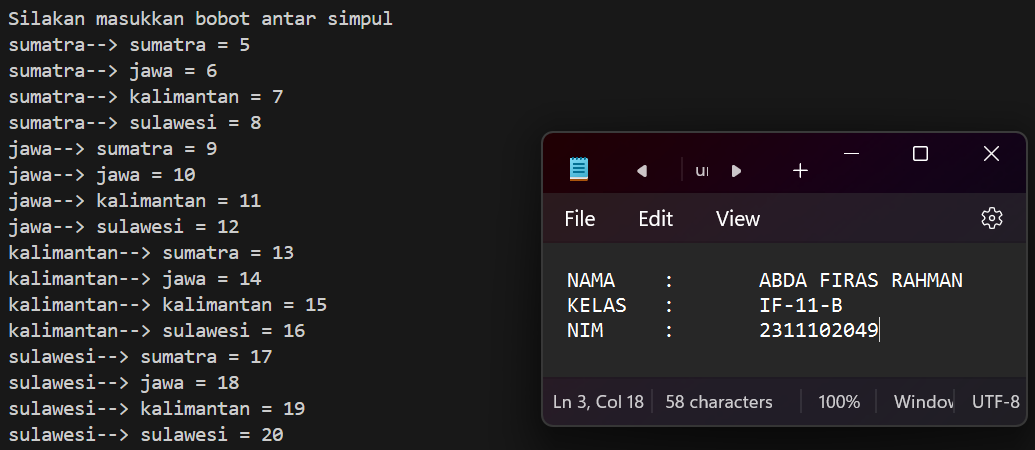
Di dalam fungsi main(), program membuat beberapa node secara manual dan melakukan operasi-operasi seperti update data, retrieve data, cari data, traversal, menghitung ukuran dan tinggi binary tree, serta menghapus subtree.

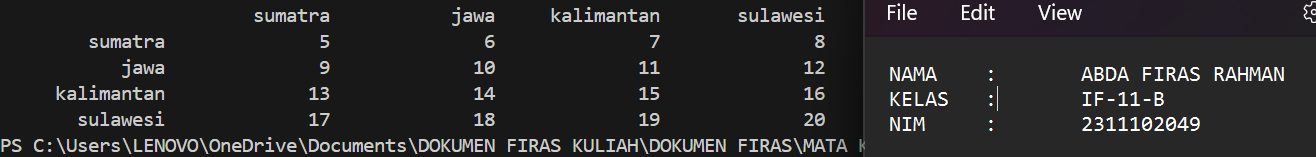
**UNGUIDED (1)**

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <string>  using namespace std;  void AbdaFirasR\_2311102049() {      int jumlahSimpul;      // Meminta pengguna memasukkan jumlah simpul      cout << "Silakan masukkan jumlah simpul: ";      cin >> jumlahSimpul;      string \*simpul = new string[jumlahSimpul];      int \*\*bobot = new int\*[jumlahSimpul];      for (int i = 0; i < jumlahSimpul; ++i) {          bobot[i] = new int[jumlahSimpul];      }      // Meminta pengguna memasukkan nama-nama simpul      for (int i = 0; i < jumlahSimpul; i++) {          cout << "Simpul " << i + 1 << ": ";          cin >> simpul[i];      }      // Meminta pengguna memasukkan bobot antar simpul      cout << "\nSilakan masukkan bobot antar simpul" << endl;      for (int i = 0; i < jumlahSimpul; i++) {          for (int j = 0; j < jumlahSimpul; j++) {              cout << simpul[i] << "--> " << simpul[j] << " = ";              cin >> bobot[i][j];          }      }      // Menampilkan hasil input pengguna      cout << "\n";      cout << setw(15) << " ";      for (int i = 0; i < jumlahSimpul; i++) {          cout << setw(15) << simpul[i];      }      cout << "\n";      for (int i = 0; i < jumlahSimpul; i++) {          cout << setw(15) << simpul[i];          for (int j = 0; j < jumlahSimpul; j++) {              cout << setw(15) << bobot[i][j];          }          cout << endl;      }      // Menghapus memori yang dialokasikan secara dinamis      delete[] simpul;      for (int i = 0; i < jumlahSimpul; ++i) {          delete[] bobot[i];      }      delete[] bobot;  }  int main() {      AbdaFirasR\_2311102049();      return 0;  } |

**OUTPUT**

****

****

****

**DESRIPSI PROGRAM**

Program C++ ini merupakan sebuah program yang memungkinkan pengguna untuk membuat representasi graf berbobot. Program ini meminta pengguna untuk memasukkan jumlah simpul (node) dalam graf, kemudian meminta pengguna untuk memasukkan nama-nama simpul tersebut. Setelah itu, program akan meminta pengguna untuk memasukkan bobot (nilai) yang menghubungkan setiap pasangan simpul.

Program ini menggunakan array dinamis untuk menyimpan data simpul dan bobot. Array dinamis dipilih untuk memungkinkan pengguna memasukkan jumlah simpul yang bervariasi sesuai kebutuhan.

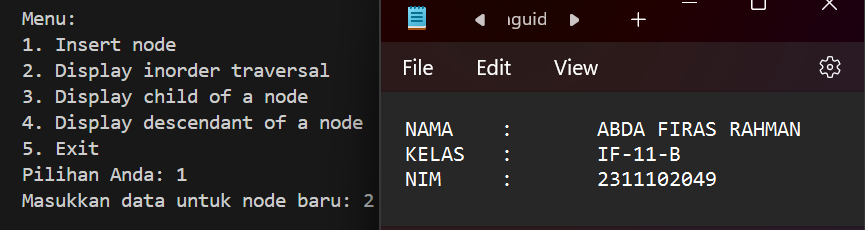
Setelah semua data dimasukkan, program akan menampilkan tabel yang merepresentasikan graf berbobot tersebut. Tabel ini berisi nama-nama simpul di baris pertama dan kolom pertama, sedangkan nilai bobot antar simpul diwakili oleh nilai-nilai di dalam sel-sel tabel.

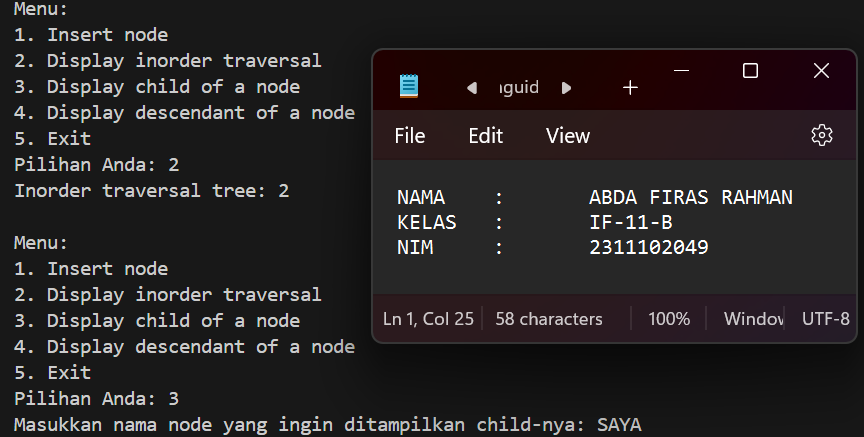
Setelah menampilkan tabel, program akan menghapus memori yang telah dialokasikan secara dinamis untuk array simpul dan bobot. Hal ini dilakukan untuk mencegah kebocoran memori (memory leak).

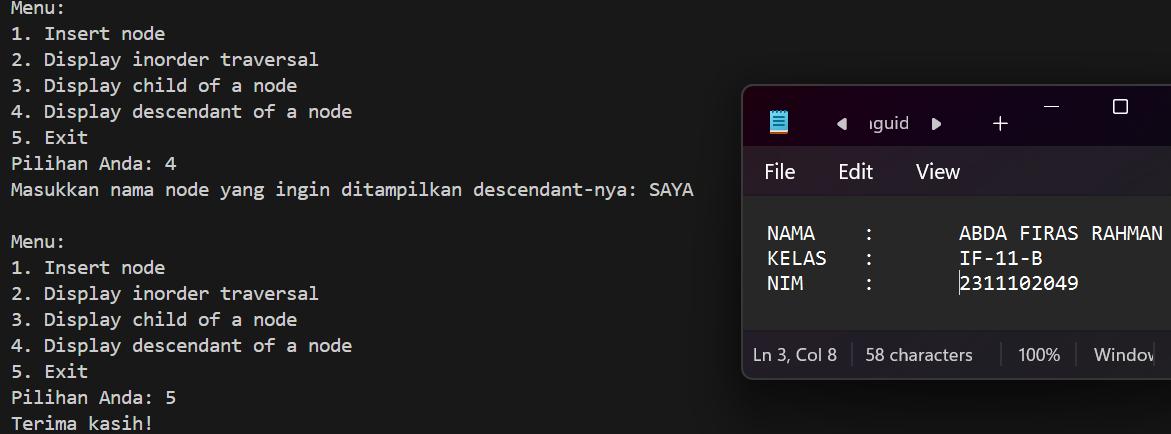
**UNGUIDED 2**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  // Node tree  struct Node {      string data;      Node\* left;      Node\* right;  };  // Fungsi untuk membuat node baru  Node\* createNode(string data) {      Node\* newNode = new Node();      newNode->data = data;      newNode->left = NULL;      newNode->right = NULL;      return newNode;  }  // Fungsi untuk menambahkan node ke tree  Node\* insertNode(Node\* root, string data) {      if (root == NULL) {          root = createNode(data);      } else if (data <= root->data) {          root->left = insertNode(root->left, data);      } else {          root->right = insertNode(root->right, data);      }      return root;  }  // Fungsi untuk menampilkan inorder traversal tree  void inorderTraversal(Node\* root) {      if (root == NULL) return;      inorderTraversal(root->left);      cout << root->data << " ";      inorderTraversal(root->right);  }  // Fungsi untuk menampilkan child dari suatu node  void displayChild(Node\* root, string parent) {      if (root == NULL) return;      if (root->data == parent) {          if (root->left != NULL)              cout << "Child kiri dari " << parent << ": " << root->left->data << endl;          if (root->right != NULL)              cout << "Child kanan dari " << parent << ": " << root->right->data << endl;          return;      }      displayChild(root->left, parent);      displayChild(root->right, parent);  }  // Fungsi untuk menampilkan descendant dari suatu node  void displayDescendant(Node\* root, string parent) {      if (root == NULL) return;      if (root->data == parent) {          cout << "Descendant dari " << parent << ": ";          inorderTraversal(root->left);          inorderTraversal(root->right);          cout << endl;          return;      }      displayDescendant(root->left, parent);      displayDescendant(root->right, parent);  }  // Fungsi utama sesuai NIM  void piras\_049() {      Node\* root = NULL;      int choice;      string data, parent;      do {          cout << "\nMenu:\n";          cout << "1. Insert node\n";          cout << "2. Display inorder traversal\n";          cout << "3. Display child of a node\n";          cout << "4. Display descendant of a node\n";          cout << "5. Exit\n";          cout << "Pilihan Anda: ";          cin >> choice;          switch (choice) {              case 1:                  cout << "Masukkan data untuk node baru: ";                  cin >> data;                  root = insertNode(root, data);                  break;              case 2:                  cout << "Inorder traversal tree: ";                  inorderTraversal(root);                  cout << endl;                  break;              case 3:                  cout << "Masukkan nama node yang ingin ditampilkan child-nya: ";                  cin >> parent;                  displayChild(root, parent);                  break;              case 4:                  cout << "Masukkan nama node yang ingin ditampilkan descendant-nya: ";                  cin >> parent;                  displayDescendant(root, parent);                  break;              case 5:                  cout << "Terima kasih!\n";                  break;              default:                  cout << "Pilihan tidak valid!\n";          }      } while (choice != 5);  }  int main() {      piras\_049();      return 0;  } |

**OUTPUT**

****

****

****

**DESKRIPSI PROGRAM**

Program C++ ini merupakan implementasi struktur data Binary Search Tree (BST). BST adalah jenis struktur data tree di mana setiap node memiliki maksimal dua anak, yaitu anak kiri (left child) dan anak kanan (right child). Dalam BST, semua node di subtree kiri memiliki nilai yang lebih kecil dari node induk (parent), sedangkan semua node di subtree kanan memiliki nilai yang lebih besar dari node induk.

Program dimulai dengan mendefinisikan struct Node yang merepresentasikan setiap node dalam BST. Struct Node memiliki tiga atribut, yaitu data (untuk menyimpan nilai node), left (pointer ke anak kiri), dan right (pointer ke anak kanan). Kemudian, program menyediakan fungsi-fungsi utama seperti createNode untuk membuat node baru, insertNode untuk memasukkan node baru ke dalam BST sesuai dengan aturan BST, inorderTraversal untuk melakukan inorder traversal pada BST dan mencetak data setiap node, displayChild untuk mencari node dengan data tertentu dan mencetak anak-anaknya, serta displayDescendant untuk mencari node dengan data tertentu dan mencetak descendant-nya.

Program utama dijalankan melalui fungsi piras\_049(), di mana pengguna dapat memilih opsi dari menu yang disediakan. Opsi-opsi tersebut meliputi memasukkan node baru ke dalam BST, menampilkan inorder traversal dari BST, menampilkan anak-anak dari suatu node dengan data tertentu, menampilkan descendant dari suatu node dengan data tertentu, dan keluar dari program. Dengan demikian, program ini dapat digunakan sebagai dasar untuk mempelajari konsep BST dan implementasinya dalam C++, serta dapat dikembangkan lebih lanjut dengan menambahkan fungsi-fungsi lain terkait dengan operasi-operasi pada BST.

DAFTAR PUSTAKA

1. Data Structure : Mengenal Graph & Tree

<https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>

1. Jenis-jenis graf:

<https://id.scribd.com/doc/266040286/Jenis-Jenis-Graf>