**LAPORAN PRAKTIKUM**

**STRUKTUR DATA DAN ALGORITMA**

**MODUL 9**

**GRAPH DAN TREE**

Sebuah gambar berisi simbol, logo, Grafis, deasin

Deskripsi dibuat secara otomatis

**Disusun oleh**

**Faqih abdullah**

**2311102048**

**Dosen Pengampu**

**Wahyu Andi Saputra S,Pd., M.Eng.**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**2024**

**MODUL 9**

**GRAPH DAN TREE**

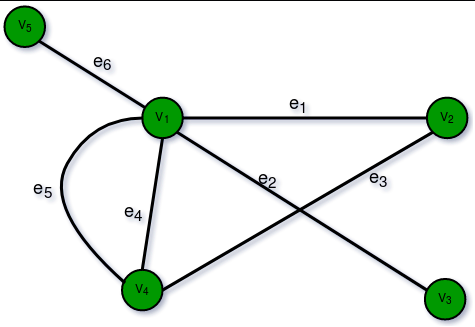
1. **DASAR TEORI**
2. **Graph**

**Pengertian**

Graf merupakan struktur data nonlinear yang terdiri dari simpul dan sisi. Simpul terkadang juga disebut sebagai simpul dan sisinya adalah garis atau busur yang menghubungkan dua simpul mana pun dalam grafik. Secara lebih formal, Graf terdiri dari himpunan simpul (V) dan himpunan sisi (E). Grafik tersebut dilambangkan dengan G(V,E).

Bayangkan sebuah permainan speak bola debagai sebuah jaringan koneksi, Dimana para pemain adalah simpulnya dan interaksi mereka dilapangan adalah ujung-ujungnya. Jaringan koneksi ini persis seperti yang diwakili oleh struktur data grafik, dan merupakan kunci untuk membuka wawasan tentang kinerja tim dan dinamika pemaindalam olahraga.

**Macam-macam Graph**

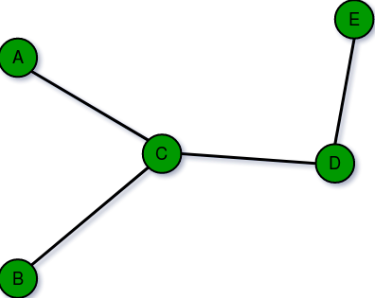
* Multi graf : Graf apapun yang memuat beberapa sisi pararel tetapi tidak memuat loop mandiri disebut multigraf. Misalnya peta jalan.
* Tepi Pararel : jika kedua simpul terhubung dengan lebih dari satu sisi, maka sisi-sisi tersebut disebut sisi pararel yang memiliki banyak rute tetapu satu tujuan
* Loop : Sisi suatu graf yang dimulai dari sebuah titik dan berakhir pada titik yang sama disebut loop atau self-loop.  
  

Gambar 1 : contoh gambar dari muti graf

* Grafik tak terbatas : suatu graf yang dikatakan tak terhingga jika graf tersebut mempunyai simpul yang tak terhingga dan juga jumlah sisi yang tak terhingga  
  Sebuah gambar berisi lingkaran, Warna-warni, garis, pola

  Deskripsi dibuat secara otomatis

Gambar 2: graf tak terbatas

* Grafik hingga : suatu graf dikatakan berhingga jika graf tersebut mempunyai jumlah simpul yang berhingga dan jumlah sisi yang berhingga. Graf berhingga adalah graf yang jumlah titik dan sisinya berhingga. Dengan kata lain, jumlah simpul dan jumlah sisi dalam suatu graf berhingga adalah terbatas dan dapat dihitung. Grafik terbatas sering digunakan untuk memodelkan situasi dunia nyata, Dimana terdapat sejumlah objek dan hubungan antar objek.  
  

Gambar 3 : graf hingga

* Grafik sepele : suatu graf dikatakan sepele jika suatu graf berhingga hanya mempunyai satu titik dan tidak mempunyai sisi. Graf sepele adalah graf yang hanya mempunyai satu titik dan tidak mempunyai sisi. Ia juga dikenal sebagai graf Tunggal atau graf simpul Tunggal. Graf sepele merupakan jenis graf yang paling sederhana dan sering digunakan sebagai titik awal untuk membuat graf yang lebih kompleks. Dalam teori graf, graf sepele dianggap sebagai kasus yang mengalami kemunduran dan biasanya tidak dipelajari secara mendetail

Sebuah gambar berisi Warna-warni, diagram, cuplikan layar

Deskripsi dibuat secara otomatis

Gambar 4 : graf sepele

* Grafik sederhana : Graf sederhana adalah graf yang tidak mempunyai lebih dari satu sisi diantara pasangan simpul. Rel kereta api sederhana yang menghubungkan berbagai kota merupakan contoh grafik sederhana.

Sebuah gambar berisi diagram, gambar, sketsa, lingkaran

Deskripsi dibuat secara otomatis

Gambar 5 : graf sederhana

* Graf nol : Graf bernode n dan berukuran nol adalah graf yang hanya terdapat simpul-simpul terisolasi yang terisolasi yang tidak memiliki sisi yang menghubungkan pasangan simpul mana pun. Graf nol adalah graf yang tidak memiliki sisi. Dengan kata lain, ini adlah graf yang hanya mempunyai simpul dan tidak ada hubungan di antara simpul-simpul tersebut. Graf nol juga dapat disebut graf tak bertepi, graf terisolasi, atau graf diskrit.

Sebuah gambar berisi diagram, deasin

Deskripsi dibuat secara otomatis

Gambar 6 : graf nol

**Representasi graf**

1. adjacency list

List array digunakan untuk menyimpan tepi antara dua simpul. Ukuran array sama dengan jumlah simpul (yaitu n). setiap indeks dalam array mewakili titik tertentu dalam graf. Entri pada indeks I dari array berisi daftar tertaut yang berisi simpul-simpul yang bertetangga dengan simpul i. anggaplah ada **n** sebagai **adjList[n].**

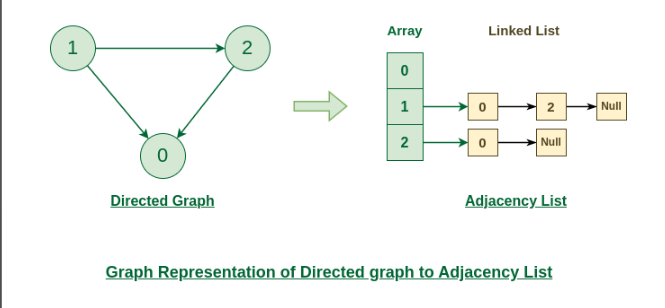
* **adjList[0]** akan memiliki semua node yang terhubung ke titik 0.
* **adjList[1]** akan memiliki semua node yang terhubung ke simpul 1 dan seterusnya.

Sebuah gambar berisi cuplikan layar, diagram, teks, garis

Deskripsi dibuat secara otomatis

**Representasi graf terarah ke adjacencyList**

graf berarah dibawah ini mempunyai 3 simpul. Jadi, array daftar akan dibuat dengan ukuran 3, Dimana setiap indeks mewakili simpul. Sekarang, simpul 0 tidak memiliki tetangga. Untuk simpul 1, ia memiliki dua tetangga yaitu, 0 dan 2. Jadi masukkan simpul 0 dan 2 pada indeks 1 dari array. Demikian pula untuk simpul 2, masukkan tetangganya ke dalam array daftar.



1. TREE

Dalam konsep tree di sebuah node ada namnaya Node Root. Node Root dalam sebuah tree yang dimaksud adalah suatu node yang memiliki hirarki tertinggi dan dapat juga memili node-node anak atau yang disebut shild node. Jadi semua node yang dapat ditelurusan dari node root tersebut. Node root dicirikan adalah suatu node khusus yang tercipta pertama kalinya. Sedangkan node-node lain dibawah node root saling terhubung satu sama lain disebut Subtree

Dalam konsep tree ini bisa dikorelasikan dalam contoh kehidupan dunia nyata seperti berikut :

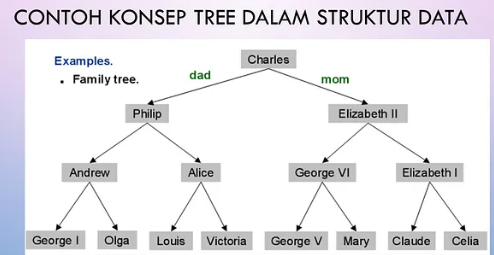
1. silsilah keluarga

2. parse tree (pada compiler)

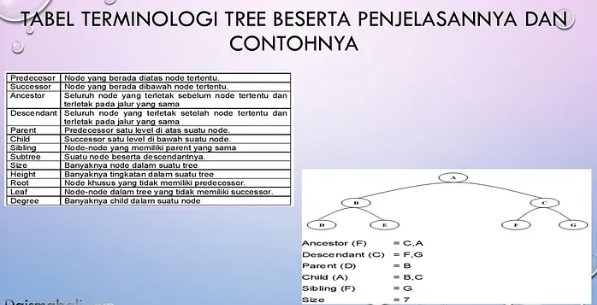
3. struktur file

4. pertandingan

Berikut salah satu contoh konsep Tree seperti silsilah keluarga dalam gambar berikut :



Konsep tree dalam struktur data memiliki istilah istilah terminologinya. Berikut istilah dan pengertian dari istilah terminology Tree dalam struktur data dalam gambar berikut :



Untuk operasi Tree dalam Struktur Data ada beberapa operasi yaitu :

1. Create () : fungsi untuk membentuk sebuah tree baru yang kosong biasanya berisi inisialisasi variable tree yang bernilai NULL seperti penulisan berikut :

Pohon = NULL;

2. Clear () : Fungsi untuk menghapus semua elemen tree dalam implementasinya cukup tambahkan syntax pohon = NULL; dalam fungsi clear ()

3. Empty () : Fungsi ini mengetahui apakah tree kosong atau tidak sebuah pemrograman. Untuk syntax Empty dalam pemrograman C++ dapat ditulis seperti berikut :

Int isEmpty(Tree \*pohon) {

If(pohon == NULL) return 1;

Else return 0;

}

4. Insert () : Fungsi yang menambah node ke dalam tree secara rekursif

5. traverse () : Fungsi ini merupakan operasi kunjungan (mengunjungi data) terhadap node-node dalam tree yang Dimana masing-masing node akan dikunjungi sekali. Adapun jenis traverse dalam konsep Tree sebagai berikut :

- Preorder

- InOrder

- PostOrder

1. **GUIDED**

program graph

Source Code

|  |
| --- |
| #include <iostream>  #include <iomanip>  using namespace std;  string simpul[7]={"Ciamis ","Bandung ","Bekasi ","Tasikmalaya ","Cianjur ","Purwokerto ","Yogyakarta "};  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},  {0,0,0,0,9,4,0},  {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;  } |

Screenshot Output

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Deskripsi dibuat secara otomatis

Deskripsi Program

Berikut deskripsi detail dari program tersebut

* Array ‘simpul’ berisi nama-nama kota yang akan menjadi simpul (node) dalam graf
* Matriks ‘busur’ berukuran 7x7 yang mempresentasikan bobot (weight) dari busur (edge) antara simpul-simpul dalam graf. Jika ‘busur’ [i][j]’ bernilai 0, artinya tidak ada bususr dari simpul ‘i’ ke simpul ‘j’
* Fungsi ‘tampilGraph” :
* fungsi ini bertugas untuk menampilkan graf berbobot dalam bentuk yang lebih mudah dibaca
* stiap simpul, fungsi ini mencetak nama simpul diikuti oleh daftar simpul tujuan dan bobot busur yang menghubungkannya.
* ‘setiosFlags(ios::left)’ dan ‘setw(15)’ digunakan untuk membuat tampilan nama simpul rata kiri dengan lebar tetap 15 karakter.
* Fungsi ‘main’
* Untuk memanggil fungsi ‘tampilGraph’

**GUIDED 2**

Program tree

Source code

|  |
| --- |
| #include <iostream>  #include <queue>  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() {  return root == NULL;  }  // Buat Node Baru  void buatNode(char data) {  if (isEmpty()) {  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()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  } else {  if (node->left != NULL) {  cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;  return NULL;  } else {  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 {  if (node->right != NULL) {  cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;  return NULL;  } else {  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;  }  }  }  // Ubah Data Tree  void update(char data, Pohon \*node) {  if (isEmpty()) {  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;  }  }  }  // Cari Data Tree  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 punya Child kanan)" << endl;  else  cout << "Child Kanan : " << node->right->data << endl;  }  }  }  // Penelurusan (Traversal)  // preOrder  void preOrder(Pohon \*node) {  if (!root) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  if (node != NULL) {  cout << " " << node->data << ", ";  preOrder(node->left);  preOrder(node->right);  }  }  }  // inOrder  void inOrder(Pohon \*node) {  if (!root) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  if (node != NULL) {  inOrder(node->left);  cout << " " << node->data << ", ";  inOrder(node->right);  }  }  }  // postOrder  void postOrder(Pohon \*node) {  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 terlebih dahulu!!" << endl;  } else {  deleteTree(root);  cout << "\nPohon berhasil dihapus." << endl;  }  }  // Cek Size Tree  int size(Pohon \*node) {  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) {  if (!root) {  cout << "\nBuat tree terlebih dahulu!" << endl;  return 0;  } else {  if (!node) {  return 0;  } else {  int heightKiri = height(node->left);  int heightKanan = height(node->right);  return max(heightKiri, heightKanan) + 1;  }  }  }  // Karakteristik Tree  void characteristic() {  cout << "\nSize Tree : " << size(root) << endl;  cout << "Height Tree : " << height(root) << endl;  cout << "Average Node of Tree : " << (float)size(root) / height(root) << endl;  }  // Display Child and Descendants  void displayChildAndDescendants(Pohon \*node) {  if (!node) {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  return;  }  cout << "\nData Node : " << node->data << endl;  if (node->left) {  cout << "Child Kiri : " << node->left->data << endl;  } else {  cout << "Child Kiri : (tidak punya Child kiri)" << endl;  }  if (node->right) {  cout << "Child Kanan : " << node->right->data << endl;  } else {  cout << "Child Kanan : (tidak punya Child kanan)" << endl;  }  cout << "Descendants : ";  queue<Pohon\*> q;  if (node->left) q.push(node->left);  if (node->right) q.push(node->right);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  cout << current->data << " ";  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  cout << endl;  }  // Menu  void menu() {  int pilihan;  char data;  Pohon \*node = nullptr;  do {  cout << "\nMenu:\n";  cout << "1. Buat Node\n";  cout << "2. Tambah Kiri\n";  cout << "3. Tambah Kanan\n";  cout << "4. Ubah Data\n";  cout << "5. Lihat Isi Data\n";  cout << "6. Cari Data\n";  cout << "7. PreOrder\n";  cout << "8. InOrder\n";  cout << "9. PostOrder\n";  cout << "10. Hapus SubTree\n";  cout << "11. Hapus Tree\n";  cout << "12. Karakteristik Tree\n";  cout << "13. Tampilkan Child dan Descendants\n";  cout << "0. Keluar\n";  cout << "Pilihan: ";  cin >> pilihan;  switch (pilihan) {  case 1:  cout << "Masukkan data node root: ";  cin >> data;  buatNode(data);  break;  case 2:  if (!isEmpty()) {  cout << "Masukkan data node: ";  cin >> data;  char parentData;  cout << "Masukkan data node parent: ";  cin >> parentData;  // Mencari node parent  Pohon\* parentNode = nullptr;  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == parentData) {  parentNode = current;  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  // Menambahkan node kiri jika ditemukan parent  if (parentNode) {  insertLeft(data, parentNode);  } else {  cout << "Node parent tidak ditemukan!" << endl;  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 3:  if (!isEmpty()) {  cout << "Masukkan data node: ";  cin >> data;  cout << "Masukkan data node parent: ";  cin >> node->data;  node = root;  if (node->data == root->data) {  insertRight(data, node);  }  else {  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == node->data) {  insertRight(data, current);  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 4:  if (!isEmpty()) {  cout << "Masukkan data baru: ";  cin >> data;  cout << "Masukkan data node yang ingin diubah: ";  cin >> node->data;  node = root;  if (node->data == root->data) {  update(data, node);  }  else {  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == node->data) {  update(data, current);  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 5:  if (!isEmpty()) {  cout << "Masukkan data node yang ingin dilihat: ";  cin >> node->data;  node = root;  if (node->data == root->data) {  retrieve(node);  }  else {  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == node->data) {  retrieve(current);  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 6:  if (!isEmpty()) {  cout << "Masukkan data node yang ingin dicari: ";  cin >> node->data;  node = root;  if (node->data == root->data) {  find(node);  }  else {  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == node->data) {  find(current);  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 7:  cout << "\nPreOrder :" << endl;  preOrder(root);  cout << "\n" << endl;  break;  case 8:  cout << "\nInOrder :" << endl;  inOrder(root);  cout << "\n" << endl;  break;  case 9:  cout << "\nPostOrder :" << endl;  postOrder(root);  cout << "\n" << endl;  break;  case 10:  if (!isEmpty()) {  cout << "Masukkan data node subtree yang ingin dihapus: ";  cin >> node->data;  node = root;  if (node->data == root->data) {  deleteSub(node);  }  else {  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == node->data) {  deleteSub(current);  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 11:  clear();  break;  case 12:  characteristic();  break;  case 13:  if (!isEmpty()) {  cout << "Masukkan data node yang ingin dilihat child dan descendants-nya: ";  cin >> data;  // Mencari node yang sesuai  Pohon\* targetNode = nullptr;  queue<Pohon\*> q;  q.push(root);  while (!q.empty()) {  Pohon\* current = q.front();  q.pop();  if (current->data == data) {  targetNode = current;  break;  }  if (current->left) q.push(current->left);  if (current->right) q.push(current->right);  }  // Jika node ditemukan, tampilkan child dan descendants-nya  if (targetNode) {  displayChildAndDescendants(targetNode);  } else {  cout << "Node tidak ditemukan!" << endl;  }  }  else {  cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree terlebih dahulu.\n";  }  break;  case 0:  cout << "Keluar dari program.\n";  break;  default:  cout << "Pilihan tidak valid! Silakan coba lagi.\n";  }  } while (pilihan != 0);  }  int main() {  menu();  return 0;  } |

Deskripsi Program

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Deskripsi dibuat secara otomatis

Deskripsi Program

* Struktur ‘pohon’ merepresentasikan sebuah node dalam pohon biner, yang memiliki data, pointer ke anak kiri, anak kanan, dan parent.
* Fungsi void init() : fungsi ini menginisialisasi root pohon sebagai ‘NULL’.
* Fungsi int isEmpty() : fungsi ini mengembalikan 1 jika pohon kosong, sebaliknya 0 jika pohon tidak kosong.
* Fungsi void buatNode(char data) : fungsi ini membuat node baru sebagai root jika pohon belum dibuat
* Pohon\* insertLeft(char data, pohon \*node) : untuk menambah anak kiri
* Pohon\* insertRight(char data, pohon \*node) : untuk menambah anak kanan
* Fungsi void update : ubah data
* Fungsi void retrieve(pohon \*node) : untuk menampilkan data node
* Fungsi void find(pohon \*node) : untuk mencari data node
* Fungsi void preorder(pohon \*node) : untuk traversal pohon
* Fungsi void inOrder(pohon \*node) : untuk inOrder traversal
* Fungsi void postOrder(pohon \*node) : untuk postOrder Traversal
* Fungsi void deleteSub(pohon \*node) : menghapus subtree
* Fungsi void clear() : menghapus tree
* Fungsi void height(pohon \*node) : untuk tinggi pohon
* Fungsi void characteristic() : karakteristik pohon

1. **UNGUIDED**

1. buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.  
output program

Sebuah gambar berisi teks, cuplikan layar, Font, deasin

Deskripsi dibuat secara otomatis

Source Code

|  |
| --- |
| #include <iostream>  #include <vector>  #include <unordered\_map>  #include <limits>  #include <string>  #include <iomanip>  using namespace std;  const int INF = numeric\_limits<int>::max();  struct Edge {  int dest;  int weight;  };  class Graph {  unordered\_map<string, int> cityIndex;  vector<string> cities;  vector<vector<int>> adjMatrix;  public:  void addCity(const string &name) {  if (cityIndex.find(name) == cityIndex.end()) {  cityIndex[name] = cities.size();  cities.push\_back(name);  for (auto &row : adjMatrix) {  row.push\_back(INF);  }  adjMatrix.push\_back(vector<int>(cities.size(), INF));  }  }  void addEdge(const string &from, const string &to, int weight) {  addCity(from);  addCity(to);  adjMatrix[cityIndex[from]][cityIndex[to]] = weight;  adjMatrix[cityIndex[to]][cityIndex[from]] = weight; // Assuming undirected graph  }  void printAdjMatrix() {  cout << setw(10) << " ";  for (const auto &city : cities) {  cout << setw(10) << city;  }  cout << endl;  for (size\_t i = 0; i < cities.size(); ++i) {  cout << setw(10) << cities[i];  for (size\_t j = 0; j < cities.size(); ++j) {  if (adjMatrix[i][j] == INF) {  cout << setw(10) << "INF";  } else {  cout << setw(10) << adjMatrix[i][j];  }  }  cout << endl;  }  }  };  int main() {  Graph graph;  int NIM\_2311102048;  cout << "Silakan masukan jumlah simpul: ";  cin >> NIM\_2311102048;  vector<string> cityNames(NIM\_2311102048);  for (int i = 0; i < NIM\_2311102048; ++i) {  cout << "Simpul " << i + 1 << ": ";  cin >> cityNames[i];  graph.addCity(cityNames[i]);  }  cout << "Silakan masukkan bobot antar simpul" << endl;  for (int i = 0; i < NIM\_2311102048; ++i) {  for (int j = 0; j < NIM\_2311102048; ++j) {  int weight;  if (i == j) {  weight = 0;  } else {  cout << cityNames[i] << "--> " << cityNames[j] << ": ";  cin >> weight;  }  graph.addEdge(cityNames[i], cityNames[j], weight);  }  }  cout << endl;  graph.printAdjMatrix();  return 0;  } |

Screenshot output

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Deskripsi dibuat secara otomatis

Deskripsi Program

*Program ini menerima input dari pengguna untuk membentuk sebuah graf, di mana simpul-simpulnya merepresentasikan kota-kota dan bobot pada sisi-sisinya mewakili jarak antara kota-kota tersebut. Program ini menampilkan matriks ketetanggaan (adjacency matrix) dari graf tersebut.*

* *Const int INF = numeric\_limits<int>::max(); : konstanta ‘INF’ menjadi representasi jarak tak terhingga*
* *Struct ’Edge’ : struktur ini mendefinisikan ‘Edge’ dengan tujuan (‘dest’) dan bobot (‘weight’). Dalam kode ini, struktur ‘Edge’ tidak digunakan, tetapi tetap didefinisikan.*
* *Class graph*

*Variable anggota*

* *cityIndex : peta yang memetakan nama kota ke indeks dalam vector ‘cities’.*
* *cities : vector yang menyimpan nama-nama kota*
* *adjMatrix : matriks ketetanggaan yang menyimpan bobot antar simpul*
* *method*
* *addCity : untuk menambahkan kota ke dalam graph*
* *addEdge : memberi bobot sisi diantara kota-kota yang di inputkan*
* *printAdjMatrix : untuk menampilkan adjacency matrix*
* *untuk menghitung kota yang di inputkan menggunakan nama variable NIM\_2311102048*

**unguide2**

modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinputkan!

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  // Deklarasi Pohon  struct Pohon {  char data;  Pohon \*left, \*right, \*parent;  };  Pohon \*root, \*baru;  // Inisialisasi  void init() {  root = NULL;  }  // Cek Node  int isEmpty() {  return root == NULL;  }  // Buat Node Baru  void buatNode(char data) {  if (isEmpty()) {  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()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  } else {  if (node->left != NULL) {  cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;  return NULL;  } else {  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 (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  return NULL;  } else {  if (node->right != NULL) {  cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;  return NULL;  } else {  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;  }  }  }  // Ubah Data Tree  void update(char data, Pohon \*node) {  if (isEmpty()) {  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 (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  if (!node) {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  } else {  cout << "\nData node : " << node->data << endl;  }  }  }  // Cari Data Tree  void find(Pohon \*node) {  if (isEmpty()) {  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 punya child kanan)" << endl;  } else {  cout << "Child Kanan : " << node->right->data << endl;  }  }  }  }  // Penelusuran (Traversal)  // preOrder  void preOrder(Pohon \*node = root) {  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  if (node != NULL) {  cout << " " << node->data << ", ";  preOrder(node->left);  preOrder(node->right);  }  }  }  // inOrder  void inOrder(Pohon \*node = root) {  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  if (node != NULL) {  inOrder(node->left);  cout << " " << node->data << ", ";  inOrder(node->right);  }  }  }  // postOrder  void postOrder(Pohon \*node = root) {  if (isEmpty()) {  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 (isEmpty()) {  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 (isEmpty()) {  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 (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!!" << endl;  } else {  deleteTree(root);  cout << "\nPohon berhasil dihapus." << endl;  }  }  // Cek Size Tree  int size(Pohon \*node = root) {  if (isEmpty()) {  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 (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  return 0;  } else {  if (!node) {  return 0;  } else {  int heightKiri = height(node->left);  int heightKanan = height(node->right);  return max(heightKiri, heightKanan) + 1;  }  }  }  // Karakteristik Tree  void characteristic() {  cout << "\nSize Tree : " << size() << endl;  cout << "Height Tree : " << height() << endl;  cout << "Average Node of Tree : " << size() / height() << endl;  }  // Tampilkan Child  void displayChild(Pohon \*node) {  if (!node) {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  } else {  if (node->left) {  cout << "Child Kiri : " << node->left->data << endl;  } else {  cout << "Child Kiri : (tidak punya child kiri)" << endl;  }  if (node->right) {  cout << "Child Kanan : " << node->right->data << endl;  } else {  cout << "Child Kanan : (tidak punya child kanan)" << endl;  }  }  }  // Tampilkan Descendants  void displayDescendants(Pohon \*node) {  if (!node) {  cout << "\nNode yang ditunjuk tidak ada!" << endl;  } else {  cout << "Descendants of " << node->data << ": ";  preOrder(node);  cout << endl;  }  }  // Menu  void menu() {  int choice;  char data;  Pohon \*node = NULL;  do {  cout << "\n=== Menu Binary Tree ===" << endl;  cout << "1. Buat Node Baru (Root)" << endl;  cout << "2. Tambah Kiri" << endl;  cout << "3. Tambah Kanan" << endl;  cout << "4. Ubah Data Node" << endl;  cout << "5. Lihat Data Node" << endl;  cout << "6. Cari Data Node" << endl;  cout << "7. PreOrder Traversal" << endl;  cout << "8. InOrder Traversal" << endl;  cout << "9. PostOrder Traversal" << endl;  cout << "10. Hapus SubTree" << endl;  cout << "11. Hapus Tree" << endl;  cout << "12. Karakteristik Tree" << endl;  cout << "13. Tampilkan Child Node" << endl;  cout << "14. Tampilkan Descendants" << endl;  cout << "0. Keluar" << endl;  cout << "Pilih menu: ";  cin >> choice;  switch (choice) {  case 1:  cout << "Masukkan data root: ";  cin >> data;  buatNode(data);  break;  case 2:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node baru: ";  cin >> data;  cout << "Masukkan data parent: ";  char parentData;  cin >> parentData;  node = root;  while (node && node->data != parentData) {  if (parentData < node->data) node = node->left;  else node = node->right;  }  insertLeft(data, node);  }  break;  case 3:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node baru: ";  cin >> data;  cout << "Masukkan data parent: ";  char parentData;  cin >> parentData;  node = root;  while (node && node->data != parentData) {  if (parentData < node->data) node = node->left;  else node = node->right;  }  insertRight(data, node);  }  break;  case 4:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin diubah: ";  char oldData;  cin >> oldData;  node = root;  while (node && node->data != oldData) {  if (oldData < node->data) node = node->left;  else node = node->right;  }  if (node) {  cout << "Masukkan data baru: ";  cin >> data;  update(data, node);  } else {  cout << "\nNode tidak ditemukan!" << endl;  }  }  break;  case 5:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin dilihat: ";  cin >> data;  node = root;  while (node && node->data != data) {  if (data < node->data) node = node->left;  else node = node->right;  }  retrieve(node);  }  break;  case 6:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin dicari: ";  cin >> data;  node = root;  while (node && node->data != data) {  if (data < node->data) node = node->left;  else node = node->right;  }  find(node);  }  break;  case 7:  cout << "\nPreOrder Traversal:" << endl;  preOrder(root);  cout << endl;  break;  case 8:  cout << "\nInOrder Traversal:" << endl;  inOrder(root);  cout << endl;  break;  case 9:  cout << "\nPostOrder Traversal:" << endl;  postOrder(root);  cout << endl;  break;  case 10:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin dihapus subtree-nya: ";  cin >> data;  node = root;  while (node && node->data != data) {  if (data < node->data) node = node->left;  else node = node->right;  }  if (node) {  deleteSub(node);  } else {  cout << "\nNode tidak ditemukan!" << endl;  }  }  break;  case 11:  clear();  break;  case 12:  characteristic();  break;  case 13:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin dilihat child-nya: ";  cin >> data;  node = root;  while (node && node->data != data) {  if (data < node->data) node = node->left;  else node = node->right;  }  displayChild(node);  }  break;  case 14:  if (isEmpty()) {  cout << "\nBuat tree terlebih dahulu!" << endl;  } else {  cout << "Masukkan data node yang ingin dilihat descendants-nya: ";  cin >> data;  node = root;  while (node && node->data != data) {  if (data < node->data) node = node->left;  else node = node->right;  }  displayDescendants(node);  }  break;  case 0:  cout << "Keluar..." << endl;  break;  default:  cout << "Pilihan tidak valid!" << endl;  break;  }  } while (choice != 0);  }  int main() {  init();  menu();  return 0;  } |

Screenchot Output

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Deskripsi dibuat secara otomatis

Deskripsi Program

*Program ini memungkinkan pengguna untuk membuat, mengedit, dan mengelola pohon biner, termasuk menambahkan node, menghapus node, mengubah data node, mencari data node, dan melakukan penelusuran (traversal) dalam pohon biner.*

**DAFTAR PUSTAKA**

(praktikum, 2024)

# References

praktikum, a. (2024). graph dan tree. In a. praktikum, *modul 9 graph dan tree* (pp. 2-5). purwokerto: institut teknologi telkom purwokerto.