**UJIAN PRAKTIKUM PERANCANGAN ANALISIS ALGORITMA**

Nama : Tohur Ma’ali Aushof

NPM : 51422590

Kelas : 3IA05

Tanggal : 23 Januari 2025

Ketua Asisten :

**TATA CARA PENGERJAAN**

1. Kerjakan ujian dengan jujur dengan kemampuan masing – masing.
2. **Dilarang** **mencontek**, **bekerja sama, berdiskusi**, dan **menggunakan AI** untuk mengerjakan soal ujian.
3. Sebelum mengerjakan ujian, silahkan untuk melakukan absensi terlebih dahulu dengan penanggungjawab praktikum.
4. Sebelum mengerjakan esai ujian, silahkan mengerjakan soal pre-test terlebih dahulu.
5. Bacalah soal dengan teliti dan cermat. Diharapkan untuk melampirkan jawaban berupa screenshot **codingan** dan **output.**
6. Wajib untuk mengisi soal bagian **soal wajib** yang berisi 5 soal dengan bobot total 50 points.
7. Untuk soal pilihan, dibebaskan untuk memilih soal dengan kemampuan masing – masing peserta. Namun bobot total keseluruhan dari soal esai ini adalah 100 points. Sehingga diharapkan untuk praktikan dapat memenuhi point tertinggi sebisa mungkin.
8. Manfaatkan waktu dengan sebaik mungkin. Jangan terlalu terpaku dengan soal yang terbilang sulit.
9. Diperbolehkan untuk melaporkan soal yang kurang jelas kepada asisten atau penanggungjawab yang bertugas.

**Apabila terdapat praktikan yang melanggar tata peraturan diatas ini, nilai ujian akan bernilai 0 dan tidak mendapat nilai ujian.**

**Goodluck.**

**SOAL UJIAN**

**SOAL WAJIB (50 Point)**

1. Jelaskan apa itu algoritma greedy? **(10 Point)**

Algoritma greedy adalah metode pemrograman yang menyelesaikan masalah dengan cara memilih solusi terbaik yang tampak pada setiap langkahnya tanpa mempertimbangkan keputusan di masa mendatang.

1. Sebutkan keuntungan dan kekurangan dari algoritma DNC **(5 Point)**

**Keuntungan :**

* **Penyederhanaan Masalah:**  
  Algoritma DNC membagi masalah besar menjadi submasalah yang lebih kecil dan lebih sederhana untuk dipecahkan, sehingga lebih mudah untuk dianalisis dan diselesaikan.
* **Penerapan Paralelisme:**  
  Submasalah yang dihasilkan sering kali dapat diselesaikan secara paralel, sehingga meningkatkan efisiensi ketika diterapkan pada sistem dengan kemampuan komputasi paralel.
* **Reusabilitas Solusi Submasalah:**  
  Jika ada submasalah yang sama muncul kembali, solusinya dapat digunakan kembali, seperti yang diterapkan dalam algoritma **Dynamic Programming** yang merupakan turunan dari DNC.
* **Cocok untuk Struktur Rekursif:**  
  Banyak masalah yang memiliki sifat rekursif, seperti pengurutan atau pencarian, dapat diselesaikan secara alami dengan pendekatan DNC.
* **Efisiensi pada Masalah Tertentu:**  
  Algoritma DNC sering kali memberikan waktu komputasi yang optimal untuk masalah besar, seperti **merge sort**, **quick sort**, atau **binary search**, dibandingkan dengan metode iteratif murni.

**Kekurangan Algoritma Divide and Conquer (DNC):**

* **Overhead Rekursi:**Pendekatan rekursif dalam DNC dapat menyebabkan overhead yang signifikan, seperti penggunaan memori untuk tumpukan pemanggilan rekursi dan pengelolaan parameter.
* **Kelebihan Pemrosesan untuk Submasalah:**  
  Jika masalah tidak terstruktur dengan baik, membagi masalah menjadi terlalu banyak submasalah dapat meningkatkan waktu komputasi secara keseluruhan.
* **Tidak Efisien untuk Masalah Kecil:**Algoritma DNC mungkin menjadi tidak efisien jika digunakan pada masalah dengan ukuran kecil, karena overhead pembagian masalah lebih besar daripada keuntungan yang diperoleh.
* **Kebutuhan Ruang Tambahan:**  
  Beberapa algoritma DNC, seperti merge sort, membutuhkan ruang tambahan yang signifikan untuk menampung data sementara selama proses penggabungan.
* **Kesulitan Implementasi:**  
  Untuk beberapa masalah, merancang solusi dengan algoritma DNC bisa lebih kompleks dibandingkan pendekatan langsung, terutama jika pembagian masalah dan penggabungan solusinya tidak intuitif.

1. Sebutkan kelebihan dan kekurangan DFS dan BFS **(30 point)**

**DFS**

Kelebihan:

1. Efisiensi Memori:  
   DFS hanya menyimpan simpul pada jalur yang sedang dieksplorasi dalam stack, sehingga lebih hemat memori dibandingkan BFS.
2. Cepat Menemukan Jalur Dalam:  
   DFS sangat cocok untuk masalah yang membutuhkan eksplorasi mendalam, seperti pencarian solusi hingga kedalaman tertentu.
3. Penerapan Mudah dengan Rekursi:  
   DFS sering kali lebih sederhana untuk diimplementasikan menggunakan teknik rekursi.
4. Cocok untuk Masalah yang Memerlukan Semua Jalur:  
   DFS digunakan untuk pencarian jalur atau solusi kombinatorial, seperti masalah pencarian Hamiltonian atau solusi teka-teki.
5. Efisien pada Graf Jarang:  
   DFS lebih efektif untuk graf yang jarang karena tidak perlu menjelajahi semua simpul di setiap level.

Kekurangan:

1. Tidak Optimal:  
   DFS tidak selalu menemukan solusi optimal, terutama jika simpul yang dicari terletak di dekat simpul awal tetapi pada cabang yang berbeda.
2. Berisiko Terjebak pada Siklus:  
   Tanpa penanganan eksplisit untuk siklus (misalnya, dengan daftar simpul yang telah dikunjungi), DFS bisa memasuki loop tak terbatas pada graf berarah atau tidak berarah.
3. Sensitif terhadap Kedalaman:  
   DFS dapat gagal atau menghasilkan error (stack overflow) jika kedalamannya terlalu besar.
4. Tidak Menjamin Solusi Terbaik:  
   DFS hanya memastikan pencarian selesai jika semua jalur dieksplorasi, tetapi solusi optimal tidak dijamin.

**BFS**

Kelebihan:

1. Menemukan Solusi Optimal (Shortest Path):  
   BFS memastikan solusi optimal pada graf tak berbobot karena eksplorasi dilakukan berdasarkan level atau kedalaman simpul.
2. Tidak Terjebak pada Siklus:  
   Dengan penggunaan daftar simpul yang telah dikunjungi, BFS tidak akan terjebak dalam loop.
3. Sesuai untuk Masalah dengan Solusi Dekat:  
   BFS lebih efisien dalam menemukan simpul yang dekat dengan simpul awal.
4. Cocok untuk Graf dengan Banyak Cabang:  
   BFS mengeksplorasi semua simpul pada satu level sebelum melangkah ke level berikutnya, sehingga ideal untuk masalah dengan cabang yang luas.
5. Memastikan Semua Solusi Terlihat:  
   BFS menjamin eksplorasi semua simpul, sehingga cocok untuk masalah seperti enumerasi simpul.

Kekurangan:

1. Penggunaan Memori yang Besar:  
   BFS membutuhkan banyak ruang untuk menyimpan semua simpul pada level saat ini dan berikutnya, terutama pada graf yang lebar.
2. Waktu Eksekusi Lambat pada Jalur Dalam:  
   BFS kurang efisien untuk masalah yang membutuhkan eksplorasi jalur yang dalam karena menjelajahi semua level sebelum mencapai kedalaman tertentu.
3. Tidak Efisien pada Graf Besar:  
   Pada graf yang sangat besar, BFS dapat menjadi sangat lambat karena harus menjelajahi semua simpul di setiap level sebelum bergerak lebih dalam.
4. Kompleksitas Implementasi Lebih Tinggi:  
   Implementasi BFS lebih kompleks dibandingkan DFS karena memerlukan antrian eksplisit untuk menyimpan simpul yang akan dieksplorasi.
5. Tidak Cocok untuk Masalah yang Memerlukan Kedalaman Maksimal:  
   BFS tidak efisien jika solusi berada jauh di dalam graf karena perlu menjelajahi semua level sebelumnya terlebih dahulu.
6. Berikan saran dan kritik kepada asisten & penanggungjawab yang bertugas! **(5 Point)**

**SOAL PILIHAN**

1. Lanjutkan kode dibawah sehingga bisa menghasilkan output yang diinginkan **(20 Point)**

**Kode:**

// import library stdio.h

#include <stdio.h>

// define size itu bernilai 99

#define size 99

// buat prototype untuk function sort

void sort(int[], int);

// function main()

int main() {

  // deklarasi variable

  int jenis\_koin[size], i, uang, banyak\_jenis\_koin, hasil[size], jumlah;

  // beri nilai jumlah sama dengan 0

  jumlah =  0;

  // meminta banyak jenis koin dari user

  printf("\nBanyak Jenis Koin : ");

  scanf("%d", &banyak\_jenis\_koin);

  // meminta jenis koin apa saja dari user

  printf("\nMasukan Jenis Koin : \n");

  for (i = 0; i < banyak\_jenis\_koin; i++) {

    scanf("%d", &jenis\_koin[i]);

  }

  // melakukan sort terhadap jenis koin, dan jumlah nya

  sort(jenis\_koin, banyak\_jenis\_koin);

  // print koin yang tersedia dari besar hingga terkecil

  printf("\nKoin yang tersedia : \n");

  for (i = 0; i < banyak\_jenis\_koin; i++) {

    printf("%d\n", jenis\_koin[i]);

  }

  // meminta nilai yang ingin dipecah dari user

  printf("\nMasukan Nilai yang dipecah : ");

  scanf("%d", &uang);

  printf("\n");

  // for loop untuk memecah uang yang diminta user

  for (i = 0; i < banyak\_jenis\_koin; i++) {

    // memberi nilai kepada variable hasil indeks ke-i dengan nilai bulat dari pembagian uang dan jenis koin indeks ke-i

    hasil[i] = uang / jenis\_koin[i];

    // beri nilai uang dari modulo uang dengan jenis\_koin indeks ke-i

    uang = uang % jenis\_koin[i];

    // tambahkan ke jumlah total koin

    jumlah += hasil[i];

  }

  // for loop pecahan uang yang sudah dihitung

  for (i = 0; i < banyak\_jenis\_koin; i++) {

    printf("Keping %d-an sebanyak : %d\n", jenis\_koin[i], hasil[i]);

  }

  // tampilkan sisa uang dan jumlahnya

  printf("\nSisanya adalah %d\n", uang);

  printf("Jumlah koin minimum adalah %d\n", jumlah);

  return 0;

}

// fungsi untuk mengurutkan array secara descending

void sort(int array[], int siz) {

  // deklarasi variable

  int i, hold, j;

  // loop pertama untuk loop semua jenis koin

  for (i = 0; i < siz - 1; i++) {

    // loop kedua untuk membandingkan array indeks ke-j dengan array indeks ke-j+1

    for (j = 0; j < siz - i - 1; j++) {

      // perbandingan jika array j lebih kecil dari array j + 1

      if (array[j] < array[j + 1]) {

        // menukarkan value dari array j+1 ke array j

        hold = array[j];

        array[j] = array[j + 1];

        array[j + 1] = hold;

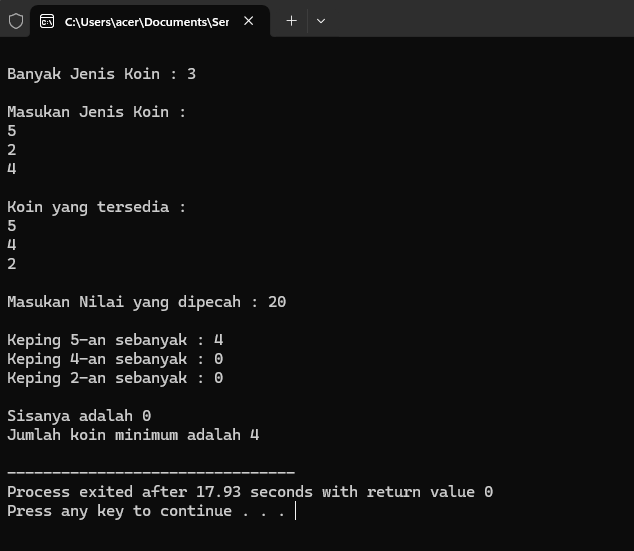
      }

    }

  }

}

**Output :**



1. Buatlah program agar menampilkan seperti output dibawah ini **(30 point)**

#include <stdio.h>

int main() {

    // Deklarasi variabel

    int n, i, angka[100], max, min;

    printf("Maximum Dan Minimum\n\n");

    // Meminta jumlah angka dari pengguna

    printf("Masukkan Banyak Angka: ");

    scanf("%d", &n);

    // Validasi jumlah angka harus lebih dari 0

    if (n <= 0) {

        printf("Jumlah angka harus lebih dari 0.\n");

        return 1;

    }

    printf("\nMasukkan Angka-angkanya:\n");

    // Memasukkan angka pertama dan menginisialisasi nilai max dan min

    scanf("%d", &angka[0]);

    max = angka[0];

    min = angka[0];

    // Memasukkan angka lainnya dan mencari max serta min

    for (i = 1; i < n; i++) {

        scanf("%d", &angka[i]);

        if (angka[i] > max) {

            max = angka[i];

        }

        if (angka[i] < min) {

            min = angka[i];

        }

    }

    // Menampilkan angka-angka yang dimasukkan

    printf("\nAngka-angkanya adalah: ");

    for (i = 0; i < n; i++) {

        printf("%d ", angka[i]);

    }

    printf("\n\n");

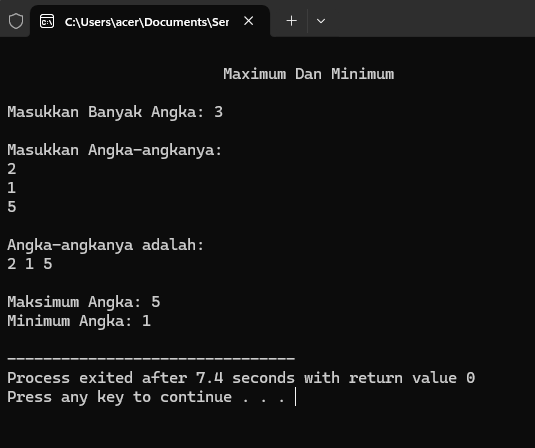
    // Menampilkan hasil max dan min

    printf("Maksimum Angka: %d\n", max);

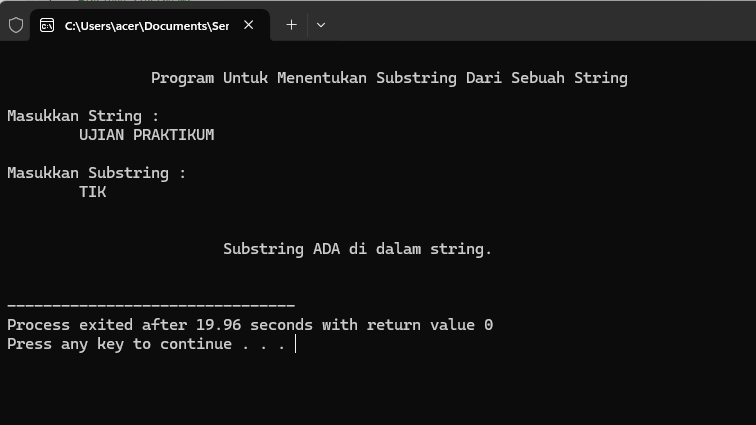
    printf("Minimum Angka: %d\n", min);

    return 0;

}



1. Buatlah program agar menampilkan seperti output dibawah ini **(30 point)**



#include <stdio.h>

#include <string.h>

int main() {

    char str[100], substr[50];

    printf("Program Untuk Menentukan Substring Dari Sebuah String\n\n");

    // Memasukkan string utama

    printf("Masukkan String : ");

    fgets(str, sizeof(str), stdin);

    str[strcspn(str, "\n")] = '\0'; // Menghapus newline di akhir input

    // Memasukkan substring

    printf("\nMasukkan Substring : ");

    fgets(substr, sizeof(substr), stdin);

    substr[strcspn(substr, "\n")] = '\0'; // Menghapus newline di akhir input

    // Mencari substring dalam string utama

    if (strstr(str, substr) != NULL) {

        printf("\n\tSubstring ADA di dalam string.\n");

    } else {

        printf("\n\tSubstring TIDAK ADA di dalam string.\n");

    }

    return 0;

}

1. Lengkapi kode dibawah ini sehingga menghasilkan output yang diinginkan **(20 point)**

**Kode:**

#include <iostream>

#include <queue>

#include <stack>

#include <vector>

using namespace std;

// Kelas untuk merepresentasikan graf

class Graf {

private:

    int vertices;  // Jumlah simpul (vertices) dalam graf

    vector<vector<int> > adjacencyList;  // List adjacency untuk menyimpan sisi-sisi graf

public:

    // Konstruktor untuk membuat graf dengan jumlah simpul v

    Graf(int v) : vertices(v), adjacencyList(v + 1) {}

    // Fungsi untuk menambah sisi (edge) antara dua simpul v dan w

    void tambahSisi(int v, int w) {

        adjacencyList[v].push\_back(w);  // Menambahkan sisi v -> w

        adjacencyList[w].push\_back(v);  // Menambahkan sisi w -> v untuk graf tak berarah

    }

    // Fungsi untuk melakukan traversal BFS (Breadth-First Search)

    void BFS(int mulai) {

        // Membuat vektor untuk melacak simpul yang sudah dikunjungi

        vector<bool> dikunjungi(vertices + 1, false);

        queue<int> q;  // Antrian untuk BFS

        q.push(mulai);  // Menambahkan simpul awal ke dalam antrian

        dikunjungi[mulai] = true;  // Menandai simpul awal sebagai sudah dikunjungi

        // Selama antrian tidak kosong

        while (!q.empty()) {

            int saatIni = q.front();  // Mengambil simpul yang ada di depan antrian

            cout << saatIni << " ";    // Mencetak simpul saat ini

            q.pop();  // Mengeluarkan simpul dari antrian

            // Menelusuri tetangga-tetangga dari simpul saat ini

            for (vector<int>::iterator it = adjacencyList[saatIni].begin(); it != adjacencyList[saatIni].end(); ++it) {

                int tetangga = \*it;  // Mendapatkan tetangga dari simpul saat ini

                if (!dikunjungi[tetangga]) {  // Jika tetangga belum dikunjungi

                    q.push(tetangga);  // Menambahkan tetangga ke dalam antrian

                    dikunjungi[tetangga] = true;  // Menandai tetangga sebagai sudah dikunjungi

                }

            }

        }

    }

    // Fungsi untuk melakukan traversal DFS (Depth-First Search)

    void DFS(int mulai) {

        // Membuat vektor untuk melacak simpul yang sudah dikunjungi

        vector<bool> dikunjungi(vertices + 1, false);

        stack<int> s;  // Tumpukan untuk DFS

        s.push(mulai);  // Menambahkan simpul awal ke dalam tumpukan

        // Selama tumpukan tidak kosong

        while (!s.empty()) {

            int saatIni = s.top();  // Mendapatkan simpul yang ada di atas tumpukan

            s.pop();  // Mengeluarkan simpul dari tumpukan

            if (!dikunjungi[saatIni]) {  // Jika simpul belum dikunjungi

                cout << saatIni << " ";  // Mencetak simpul saat ini

                dikunjungi[saatIni] = true;  // Menandai simpul saat ini sebagai sudah dikunjungi

            }

            // Menelusuri tetangga-tetangga dari simpul saat ini (dengan urutan terbalik)

            for (vector<int>::reverse\_iterator it = adjacencyList[saatIni].rbegin(); it != adjacencyList[saatIni].rend(); ++it) {

                int tetangga = \*it;  // Mendapatkan tetangga dari simpul saat ini

                if (!dikunjungi[tetangga]) {  // Jika tetangga belum dikunjungi

                    s.push(tetangga);  // Menambahkan tetangga ke dalam tumpukan

                }

            }

        }

    }

};

int main() {

    int vertices, edges;

    // Meminta input jumlah simpul dan sisi dari pengguna

    cout << "Masukkan jumlah simpul: ";

    cin >> vertices;

    Graf graf(vertices);  // Membuat objek graf dengan jumlah simpul yang dimasukkan

    cout << "Masukkan jumlah sisi: ";

    cin >> edges;

    cout << "Masukkan sisi-sisi (format: sumber tujuan):" << endl;

    // Meminta input sisi-sisi dan menambahkannya ke dalam graf

    for (int i = 0; i < edges; ++i) {

        int sumber, tujuan;

        cin >> sumber >> tujuan;

        graf.tambahSisi(sumber, tujuan);  // Menambahkan sisi ke dalam graf

    }

    int pilihan;

    // Menampilkan menu untuk memilih antara BFS, DFS, atau keluar

    do {

        cout << "\nMenu:\n1. Breadth-First Search (BFS)\n2. Depth-First Search (DFS)\n0. Keluar\nMasukkan pilihan: ";

        cin >> pilihan;

        switch (pilihan) {

            case 1:  // Pilihan untuk menjalankan BFS

                cout << "Masukkan simpul awal untuk BFS: ";

                int bfsMulai;

                cin >> bfsMulai;

                cout << "Hasil traversal BFS: ";

                graf.BFS(bfsMulai);  // Menjalankan BFS dari simpul yang dipilih

                cout << endl;

                break;

            case 2:  // Pilihan untuk menjalankan DFS

                cout << "Masukkan simpul awal untuk DFS: ";

                int dfsMulai;

                cin >> dfsMulai;

                cout << "Hasil traversal DFS: ";

                graf.DFS(dfsMulai);  // Menjalankan DFS dari simpul yang dipilih

                cout << endl;

                break;

            case 0:  // Pilihan untuk keluar dari program

                cout << "Keluar dari program. Selamat tinggal!\n";

                break;

            default:  // Jika input pilihan tidak valid

                cout << "Pilihan tidak valid. Harap masukkan opsi yang benar.\n";

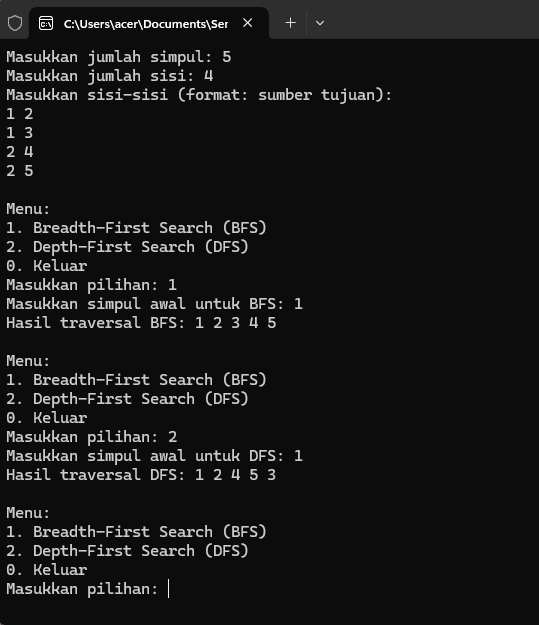
        }

    } while (pilihan != 0);  // Mengulang hingga pengguna memilih untuk keluar

    return 0;  // Mengembalikan nilai 0 untuk menandakan program selesai

}

**Output:**

****