Nama : Muhammad Riza Ballafi

NIM : 101210061

Mata Kuliah / SKS : Algoritma dan Kompleksitas / 2

Prodi / Kelas : Teknologi Informasi / TF21C

Dosen Pengampu : Nur Hadian, M.Kom.

Petunjuk Pengerjaan :

1. Isikan terlebih dahulu identitas Anda di lembar jawab yang disediakan!
2. Bacalah Doa sebelum mengerjakan!
3. Periksa Kembali jawaban sebelum dikumpulkan!

Soal

1. Case Study: Pemilihan Kursi Konser

   Anda adalah seorang event organizer yang sedang mengatur konser musik dengan kursi yang diberi nomor. Setiap kursi memiliki rating yang menunjukkan seberapa baik pandangan penonton dari kursi tersebut. Anda ingin memilih subset kursi yang akan dijual dengan total rating maksimum.

   • Tentukan langkah-langkah algoritma Greedy untuk menyelesaikan masalah ini.
   • Implementasikan algoritma Greedy tersebut dalam bahasa pemrograman pilihan Anda.

2. Case Study: Penyelesaian Maze

   Anda ingin mengembangkan sebuah program untuk menyelesaikan maze. Anda diberikan sebuah grid yang mewakili maze dengan rintangan dan jalan bebas. Anda harus menemukan jalur dari titik awal ke titik akhir dalam maze tersebut dengan menggunakan algoritma backtracking.

   • Tentukan langkah-langkah algoritma Backtracking untuk menyelesaikan masalah ini.
   • Implementasikan algoritma Backtracking tersebut dalam bahasa pemrograman pilihan Anda.

3. Case Study: Pencarian Maksimum dalam Array

   Anda memiliki sebuah array bilangan bulat dan ingin mencari nilai maksimum di dalam array tersebut menggunakan algoritma Divide & Conquer.

   • Tentukan langkah-langkah algoritma Divide & Conquer untuk menyelesaikan masalah ini.
   • Implementasikan algoritma Divide & Conquer tersebut dalam bahasa pemrograman pilihan Anda.

SELAMAT MENGERJAKAN SEMOGA SUKSES

Jawaban!

1.A. langkah-langkah algoritma Greedy untuk menyelesaikan masalah ini ialah:

1. Inisialisasi : Tentukan jumlah kursi yang teersedia dan urutkan berdasarkan kriteria awal yang dianggap penting untuk pandangan penonton, seperti jarak dari pangggung.

2. Pilih kursi teratas : Pilih kursi dengan peringkat tertinggi atau yang dianggap memiliki pandangan terbaik berdasarkankriteria yang telah ditentukan.

3. Evaluasi ketersediaan : Periksa apakah kursi yang dipilih masih tersedia atau sudah ditempati. Jika kursi tersebut masih tersedia, lanjutkan ke langkah berikutnya,jika tidak, lanjutkan ke langkah 5.

4. Simpan kursi yang dipilih : Tandai kursi tersebut sebagai kursi yang telah dipilih atau tambahkan ke daftar kursi yang dipilih.

5. Pilih alternatif: jika kursi yang dipilih tidak tersedia, pilih kursi alternatif dengan peringkat tertinggi berikutnya dari daftar kursi yang tersedia. Ulangi langkah 3 dan 4 untuk kursi alternatif yang dipilih.

6. Evaluasi solusi : setelah semua kursi telah dievaluasi,evaluasi solusi yang dihasilakan. Anda dapat menggunakan metrik khusus untuk mengukur seberapa baik pandangan penonton, Misalnya dengan menghitung total skor kursi yang dipilih berdasarkan kriteria yang ditetapkan.

7. Output hasil : Setelah selesai, Keluarkan hasil berupa daftar kursi yang dipilih dan informasi yang terkait, seperti urutan kursi dan skor pandangan penonton yang diperoleh.

B. Implementasikan algoritma Greedy tersebut dalam bahasa pemrograman Java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

class Seat {
    private int id;
    private int distance;
    private boolean available;

public Seat(int id, int distance, boolean available) {
        this.id = id;
        this.distance = distance;
        this.available = available;
            }

    public int getId() {
            return id;
```

```
        }

        public int getDistance() {
                return distance;
        }

        public boolean isAvailable() {
                return available;
        }

        public void setAvailable(boolean available) {
                this.available = available;
        }
}

        public class GreedySeatSelection {
        public static List<Seat> greedySeatSelection(List<Seat> seats, int numDesiredSeats,
                        String criteria) {
                        List<Seat> selectedSeats = new ArrayList<>();

        // Urutkan kursi berdasarkan kriteria
                        seats.sort(Comparator.comparingInt(seat -> {
                        if (criteria.equals("distance")) {
                        return seat.getDistance();
                        } else {
                // Tambahan untuk kriteria lainnya (jika ada)
                return 0;
            }
        }));

        for (Seat seat : seats) {
            if (seat.isAvailable()) {
                selectedSeats.add(seat);
                seat.setAvailable(false); // tandai kursi sebagai dipilih
            }

            if (selectedSeats.size() == numDesiredSeats) {
                break;
            }
        }

        return selectedSeats;
    }

    public static void main(String[] args) {
```

```
        // Data kursi (contoh)
        List<Seat> seats = new ArrayList<>();
        seats.add(new Seat(1, 10, true));
        seats.add(new Seat(2, 5, true));
        seats.add(new Seat(3, 15, true));
        seats.add(new Seat(4, 8, true));
        seats.add(new Seat(5, 12, true));

        int numDesiredSeats = 3; // Jumlah kursi yang diinginkan
        String criteria = "distance"; // Kriteria untuk menentukan pandangan yang baik (misalnya, jarak dari panggung)

        // Panggil fungsi greedySeatSelection
        List<Seat> selectedSeats = greedySeatSelection(seats, numDesiredSeats, criteria);

        // Output hasil
        System.out.println("Kursi yang dipilih:");
        for (Seat seat : selectedSeats) {
            System.out.println("Kursi ID: " + seat.getId() + ", Jarak: " + seat.getDistance());
        }
    }
}
```

2. A. Tentukan langkah-langkah algoritma Backtracking untuk menyelesaikan masalah

Penyelesain Maze :

1. Tentukkan titik awal dan titik akhir dalam maze.
2. Inisialisasi suatu jalur kosong sebagai jalur saat ini dan tandai titik awal sebagai kunjungan pertama dalam jalur saat ini.
3. Mulai proses backtraking:

a. Periksa apakah jalur saat ini mencapai titik akhir dalam maze. Jila iya, maka jalur saat ini adalah solusi yang valid. proses backtracking berakhir.

b. Jika jalur saat ini belum mencapai titik akhir, coba semua kemungkinan langkah berikutnya dari titik saat ini

c. Untuk setiap langkah mungkin :

- periksa apakah langkah tersebut valid
- jika langkah tersebut valid, tandai langkah tersebut sebagai kunjungan dalam jalur saat ini.
- Lanjutkan proses backtracking dengan memanggil diri sendiri secara rekrusif untuk langkah berikutnya.
- jika panggilan rekrusif menghasilkan solusi kembaliksan solusi tersebut.
- jjika panggilan rekrusif tidak menghasilkan solusi, batalkan langkah tersebut dan tandai langkah tersebut sebagai belum dikunjungi dalam jalur ini.

4. Jika proses backtracking telah selesai, kembalikan solusi yang ditemukan (jika ada) atau tampilkan pesan bahwa tidak ada solusi yang ditemukan.

B. Implementasikan algoritma Backtracking tersebut dalam bahasa pemrograman java :

```
public class MazeSolver {
    private static final int[][] DIRECTIONS = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // Up, Down, Left, Right
    private int[][] maze;
    private int numRows;
    private int numCols;

    public MazeSolver(int[][] maze) {
        this.maze = maze;
        this.numRows = maze.length;
        this.numCols = maze[0].length;
    }

    public boolean solveMaze() {
        int[][] solution = new int[numRows][numCols];
        if (solve(0, 0, solution)) {
            printSolution(solution);
            return true;
        } else {
            System.out.println("No solution found.");
            return false;
        }
    }

    private boolean solve(int row, int col, int[][] solution) {
        if (row == numRows - 1 && col == numCols - 1) {
            solution[row][col] = 1; // Mark the final position
            return true;
        }

        if (isSafe(row, col)) {
            solution[row][col] = 1; // Mark current position as part of the solution

            for (int[] direction : DIRECTIONS) {
                int nextRow = row + direction[0];
                int nextCol = col + direction[1];

                if (isValidMove(nextRow, nextCol)) {
                    if (solve(nextRow, nextCol, solution)) {
                        return true;
                    }
                }
```

```
            }

            solution[row][col] = 0; // Backtrack if no valid move is found
        }

        return false;
    }

    private boolean isSafe(int row, int col) {
        return row >= 0 && row < numRows && col >= 0 && col < numCols &&
maze[row][col] == 1;
    }

    private boolean isValidMove(int row, int col) {
        return isSafe(row, col) && maze[row][col] != 0;
    }

    private void printSolution(int[][] solution) {
        System.out.println("Solution:");
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                System.out.print(solution[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] maze = {
                {1, 0, 0, 0},
                {1, 1, 0, 1},
                {0, 1, 0, 0},
                {1, 1, 1, 1}
        };

        MazeSolver solver = new MazeSolver(maze);
        solver.solveMaze();
    }
}
```

3. A. Tentukan langkah-langkah algoritma Divide & Conquer untuk menyelesaikan masalah Pencarian Maksimum dalam Array.

Berikut adalah langkah-langkah algoritma Divide & Conquer untuk menyelesaikan masalah Pencarian Maksimum dalam Array:

1. Tentukan array bilangan bulat yang ingin dicari nilai maksimumnya.

2. Tentukan batasan masalah awal. Dalam kasus ini, batasan awal adalah seluruh array.

3. Tentukan kondisi dasar (base case) untuk membatasi rekursi. Dalam kasus ini, jika array hanya memiliki satu elemen, kembalikan elemen tersebut sebagai nilai maksimum.

4. Bagi array menjadi dua bagian yang lebih kecil. Misalnya, bagi array menjadi dua bagian yang sama besar atau hampir sama besar.

5. Panggil rekursif fungsi Pencarian Maksimum untuk setiap bagian array yang lebih kecil. Kembalikan nilai maksimum dari dua nilai yang dikembalikan oleh pemanggilan rekursif tersebut.

6. Kembalikan nilai maksimum yang didapatkan.

B. Implementasikan algoritma Divide & Conquer tersebut dalam bahasa pemrograman pilihan anda.

```
public class MaxValueFinder {
    public static int findMax(int[] array) {
        return findMaxHelper(array, 0, array.length - 1);
    }

    private static int findMaxHelper(int[] array, int start, int end) {
        if (start == end) {
            return array[start];
        }

        int mid = (start + end) / 2;

        int leftMax = findMaxHelper(array, start, mid);
        int rightMax = findMaxHelper(array, mid + 1, end);

        return Math.max(leftMax, rightMax);
    }

    public static void main(String[] args) {
        int[] array = {2, 7, 1, 8, 3};
        int max = findMax(array);
        System.out.println("Maximum value: " + max);
    }
}
```