BAB 5 IMPLEMENTASI SISTEM

5.1 Implementasi Kode Program

5.1.1 Inisialisasi Populasi Awal

Proses inisialisasi populasi awal digunakan untuk merepresentasikan kromosom sesuai dengan jumlah populasi (*popsize*), dan nilai permutasi yang digunakan untuk mengisi nilai pada masing-masing gen. Panjang kromosom dalam sehari yaitu 18 yang merupakan representasi dari 6 sumber bahan makanan meliputi karbohidrat, protein hewani, protein nabati, lemak, sayuran dan buah-buahan. Masing-masing merepresentasikan sebanyak 3 kandungan bahan makanan sesuai dengan jumlah waktu yaitu pagi, siang, dan malam. Kode 5.1 merupakan implementasi program untuk inisialisasi populasi awal.

```
1   public int[][][] Inisialisasi() {
2        for (int h = 0; h < hari; h++) {
3          for (int i = 0; i < popsize; i++) {
4            for (int j = 0; j < panjangKromosom; j++) {
5              populasi[h][i][j] =
6              ThreadLocalRandom.current().nextInt(1, 65);
7              pop_gab[h][i][j] = populasi[h][i][j];
8             }
9           }
10       }
11       return populasi;
12  }
```

Kode Program 5.1 Inisialisasi Populasi Awal

Penjelasan untuk Kode Program 5.1 sebagai berikut:

1. Baris 1 digunakan untuk membuat method inisialisasi yang memiliki panjang *array* 3 dimensi dengan tipe data *integer*.
2. Baris 2-4, perulangan untuk mengisi *array populasi* sebanyak jumlah hari, *popsize,* dan *panjangKromosom.*
3. Baris 5-6, digunakan untuk *random* nilai indeks gen dengan batas bawah yaitu 1 dan batas atas 65 dan disimpan pada variabel *populasi.*
4. Baris 7, digunakan untuk menyimpan populasi awal (populasi) kedalam *array pop_gab.*
5. Baris 11, digunakan untuk mengembalikan nilai *array populasi.*

5.1.2 *C*rossover

*C*rossover yang digunakan yaitu *extended intermediete crossover.* Langkah pertama yaitu dengan menggalikan nilai *cr* dengan jumlah *popsize,* untuk menentukan jumlah *child (offspring)* yang dihasilkan dari proses *crossover.* Kode Program 5.2 merupakan implementasi proses *crossover.*

```
1   public int[][][] crossover(int[][][] populasi) {
2     Random Rand = new Random();
3     double[][][]hasilTemp = new double[hari]
4     [bykchildCros + 1][panjangKromosom];
5     int[][][] hasil_crossover = new int[hari]
6     [bykchildCros][panjangKromosom];
7       for (int h = 0; h < hari; h++) {
8         for (int i = 0, childKe = 0; i <
9         bykPerulanganCros; i++, childKe += 2) {
10        int indexp1 = 0, indexp2 = 0;
11
12          indexp1 = Rand.nextInt((popsize - 1) - 0 + 1);
13          do {
14          indexp2 = Rand.nextInt((popsize - 1) - 0 + 1);
15          } while (indexp1 == indexp2);
16          (indexp1 + 1) + ",P2 = p" + (indexp2 + 1));
17
18            nilai_a1 = new double[panjangKromosom];
19            for (int j = 0; j < nilai_a1.length; j++) {
20              nilai_a1[j] = ThreadLocalRandom.current()
21              .nextDouble(-0.25, 1.25);
22            }
23            nilai_a2 = new double[panjangKromosom];
24              for (int j = 0; j < nilai_a2.length; j++){
25                nilai_a2[j]= ThreadLocalRandom.current()
26                .nextDouble(-0.25, 1.25);
27              }
28      for(int j=0; j < panjangKromosom;j++) {
29        hasilTemp[h][childKe][j] = populasi[h][indexp1]
30         [j]+ (nilai_a1[j] * (populasi[h][indexp2][j] -
31         populasi[h][indexp1][j]));
32        hasilTemp[h][childKe + 1][j] = populasi[h]
33         [indexp2][j] +(nilai_a2[j]*populasi[h][indexp1]
34         [j]-populasi[h][indexp2][j]));
35      }
36
37     for (int k = 0, a = popsize; k < bykchildCros; k++,
38  a++) {
39      for (int j = 0; j < panjangKromosom; j++) {
40        hasil_crossover[h][k][j] = (int)Math.round
41        (hasilTemp[h][k][j]);
42        if (hasil_crossover[h][k][j] < 1) {
43            hasil_crossover[h][k][j] = 1;
44        }
45        if (j == 0 || j == 6 || j == 12) { //29; KH
46           if (hasil_crossover[h][k][j] > 29) {
47            hasil_crossover[h][k][j] =
48            hasil_crossover[h][k][j] - 29;          }
49        } else if (j == 1 || j == 7 || j == 13) {//37;PH
50           if (hasil_crossover[h][k][j] > 37) {
51            hasil_crossover[h][k][j] =
52            hasil_crossover[h][k][j] - 37;          }
53        } else if (j == 2 || j == 8 || j == 14) {//13;PN
```

```
54          if (hasil_crossover[h][k][j] > 12) {
55             hasil_crossover[h][k][j] =
56             hasil_crossover[h][k][j] - 13;          }
57       } else if (j == 3 || j == 9 || j == 15) {
58          if (hasil_crossover[h][k][j] > 8) {//8;Lmak
59              hasil_crossover[h][k][j] =
60              hasil_crossover[h][k][j] - 8;
61          }
62       } else if (j == 4 || j == 10 || j == 16) {
63          if (hasil_crossover[h][k][j] > 37) {//37;buah
64             hasil_crossover[h][k][j] =
65             hasil_crossover[h][k][j] - 37;          }
66       } else if (j == 5 || j == 11 || j == 17) {
67          if (hasil_crossover[h][k][j] > 45) {//32;Syur
68             hasil_crossover[h][k][j] =
69             hasil_crossover[h][k][j] - 45;          }
70       }
71       pop_gab[h][a][j]=(int)hasil_crossover[h][k][j];
72     }
73    }
74    return hasil_crossover;
75 }
```

Kode Program 5.2 Implementasi *Crossover*

Penjelasan untuk Kode Program 5.2 sebagai berikut:

1. Baris 1 digunakan untuk memanggil variabel *populasi.*

2. Baris 3-4, digunakan untuk inisialisasi *hasiltemp array* 3 dimensi yang berisi hari, byk*childCr*oss dan *panjangKromosom.*

3. Baris 5-6, digunakan untuk inisialisasi *hasil_crossover array* 3 dimensi yang berisi hari, byk*childCr*oss dan *panjangKromosom.*

4. Baris 7, digunakan untuk perulangan untuk jumlah hari.

5. Baris 8-9, digunakan untuk melakukan perulangan dalam proses *cr*ossover.

6. Baris 10, inisialisasi *indexP1* dan *indexP2.*

7. Baris 12-16, digunakan untuk pemilihan parent secara *random parent* pada *index1* dan *index2*.

8. Baris 18, digunakan untuk inisialisasi *array nilai_a1* berisi *panjangKromosom.*

9. Baris 19, perulangan yang digunakan untuk mengisi nilai *nilai_a1* sesuai *panjangKromosom.*

10. Baris 20-22, digunakan untuk *random nilai_a1* dengan batas bawah -0,25 dan batas atas 1,25.

11. Baris 23, digunakan untuk inisialisasi *array nilai_a2* berisi *panjangKromosom.*

12. Baris 24, perulangan yang digunakan untuk mengisi nilai *nilai_a2* sesuai *panjangKromosom.*

13. Baris 25-27, digunakan untuk *random* nilai_a2 dengan batas bawah -0,25 dan batas atas 1,25.
14. Baris 28, perulangan yang digunakan untuk menghitung *child* pada masing-masing kromosom (*panjangKromosom*).
15. Baris 29-31, digunakan untuk menghitung nilai *child1* sesuai Persamaan yang sudah ditentukan kemudian menyimpan hasil perhitungan *cr*ossover *extended intermediat*e pada *array hasilTemp index1.*
16. Baris 32-35, digunakan untuk menghitung nilai *child*2 sesuai Persamaan yang sudah ditentukan kemudian menyimpan hasil perhitungan crossover extended intermediate pada *array hasilTemp index2.*
17. Baris 37-39, perulangan yang digunakan untuk normalisasi hasil nilai *cr*ossover dari bilangan decimal ke bilangan bulat dan nilai *cr*ossover yang melebihi batasan indeks bahan makanan.
18. Baris 40-41, digunakan membulatkan ke bilangan terdekat dan mengubah *hasiltemp* menjadi bilangan *integer*, kemudian disimpan dalam *array hasil_crossover.*
19. Baris 42-44, digunakan untuk kondisi jika *hasilcrossover* memiliki nilai kurang dari 1 maka nilai *hasil_crossover* akan diubah menjadi nilai 1.
20. Baris 45, digunakan untuk mengisi pada *array* ke 0, 6, dan 12.
21. Baris 46-48, digunakan untuk kondisi jika *hasil_crossover* melebihi nilai 29 maka nilai *hasil_crossover* dikurangi dengan batas atas jumlah bahan makanan yaitu 29.
22. Baris 49, digunakan untuk mengisi pada *array* ke 1, 7, dan 13.
23. Baris 50-52, digunakan untuk kondisi jika hasil_*cr*ossover melebihi nilai 37 maka nilai hasil_*cr*ossover dikurangi dengan batas atas jumlah bahan makanan yaitu 37.
24. Baris 53, digunakan untuk mengisi pada *array* ke 2, 8, dan 14.
25. Baris 54-56, digunakan untuk kondisi jika hasil_*cr*ossover melebihi nilai 13 maka nilai hasil_*cr*ossover dikurangi dengan batas atas jumlah bahan makanan yaitu 13.
26. Baris 57, digunakan untuk mengisi pada *array* ke 3,9, dan 15.
27. Baris 58-60, digunakan untuk kondisi jika hasil_*cr*ossover melebihi nilai 8 maka nilai hasil_*cr*ossover dikurangi dengan batas atas jumlah bahan makanan yaitu 8.
28. Baris 61, digunakan untuk mengisi pada *array* ke 4, 10, dan 16.
29. Baris 62-64, digunakan untuk kondisi jika hasil_*cr*ossover melebihi nilai 37 maka nilai hasil_*cr*ossover dikurangi dengan batas atas jumlah bahan makanan yaitu 37.

30. Baris 65, digunakan untuk mengisi pada *array* ke 5, 11, dan 17.
31. Baris 66-69, digunakan untuk kondisi jika hasil_*cr*ossover melebihi nilai 45 maka nilai hasil_*cr*ossover dikurangi dengan batas atas jumlah bahan makanan yaitu 45.
32. Baris 70, digunakan untuk menyimpan *hasil_crossover* yang sudah dinormalisasi ke *array pop_gab.*
33. Baris 73, digunakan untuk mengembalikan nilai *hasil_crossover*.

5.1.3 Mutasi

Mutasi yang digunakan yaitu yaitu *reciprocal exchange mutation.* Langkah pertama yaitu menghitung nilai *offspring* dengan cara menggalikan nilai *mr* dengan jumlah *popsize.* Selanjutnya menentukan 1 *parent* secara acak, kemudian menentukan nilai *xp1* dan *xp2* untuk ditukar. Kode 5.3 merupakan implementasi proses mutasi.

```
1   public int[][][] mutasi(int[][][] populasi) {
2      int[][][] hasil_mutasi = new int[hari]
3      [byk child Mutasi][panjangKromosom];
4      int[][][] child = new int[hari]
5      [byk child Mutasi][panjangKromosom];
6        for (int k = 0, a = popsize + byk childCros;
7        k < byk child Mutasi; k++, a++) {
8         int parent = (int) (Math.random() * popsize);
9         for (int h = 0; h < hari; h++) {
10          for (int j = 0; j < panjangKromosom; j++) {
11           hasil_mutasi[h][k][j] = populasi[h]
12           [parent][j];
13          }
14          int xp1 = (int) (Math.random() *
15          panjangKromosom);
16          int xp2 = (int) (Math.random() *
17          panjangKromosom);
18          int temp1 = populasi[h][parent][xp1];
19          hasil_mutasi[h][k][xp1] = hasil_mutasi[h]
20          [k][xp2];
21          hasil_mutasi[h][k][xp2] = temp1;
22        for (int n = 0; n < panjangKromosom; n++) {
23          pop_gab[h][a][n] = hasil_mutasi[h][k][n];
24        }
25      }
26      return hasil_mutasi;
27  }
```

Kode Program 5.3 Implementasi Mutasi

Penjelasan untuk Kode Program 5.3 sebagai berikut:

1. Baris 1, digunakan untuk memanggil nilai *array* populasi.
2. Baris 2-3, digunakan untuk inisaialisasi hasil_mutasi yang berisi *array* 3 dimensi yaitu hari, byk*child*Mutasi dan panjangKromosom.

3. Baris 4-5, inisialisasi *array child* 3 dimensi yang berisi hari, bnyk*child*mutasi dan *panjangKromosom.*
4. Baris 6-8, perulangan yang digunakan untuk pemilihan parent secara acak dengan cara *random* sebanyak jumlah *popsize.*
5. Baris 9-13, perulangan yang digunakan untuk menyimpan hasil parent terpilih yang disimpan dalam *array* hasil_mutasi.
6. Baris 14-15, pemilihan titik *exchange* secara acak pada *xp1* dengan cara *random* dikalikan p*anjangKromosom* yang disimpan dalam variabel *xp1*.
7. Baris 16-17, pemilihan titik *exchange* secara acak pada *xp2* dengan cara *random* dikalikan *panjangKromosom* yang disimpan dalam variabel *xp2*.
8. Baris 18, digunakan untuk menyimpan populasi yang berisi *xp1* ke dalam variabel *temp1*.
9. Baris 19, digunakan untuk menukar *hasil_mutasi* yang berisi *xp1* dengan *hasil_mutasi* yang berisi xp2.
10. Baris 20, digunakan untuk menyimpan variabel *temp1* pada *array hasil_mutasi* yang berisi *xp2*.
11. Baris 22-24, digunakan untuk menyimpan *hasil_mutasi* yang sudah ditukar ke dalam *array* pop_gab.
12. Baris 26, digunakan untuk mengembalikan nilai *hasil_mutasi*.

5.1.4 Penalti Gizi

Penalti gizi digunakan untuk mengetahui nilai selisih dari kebutuhan gizi balita dengan kandungan gizi perhari. Hasil dari perhitungan penalti digunakan untuk menghitung nilai *fitness*. Kode Program 5.4 merupakan implementasi proses penalti gizi.

```
1  Public double[] penaltiGizi(double kebutuhan, double[]
2  Rata2) {
3    double penalti[] = new double[pop_gab[0].length];
4    for (int i = 0; i < Rata2.length; i++) {
5       penalti[i] = (kebutuhan - Rata2[i]);
6    }
7    return penalti;
8  }
```

Kode Program 5.4 Implementasi Penalti

Penjelasan untuk Kode Program 5.4 sebagai berikut:
1. Baris 1-2, digunakan untuk memanggil nilai kebutuhan dan rata2.
2. Baris 3, digunakan untuk inisialisasi *array* penalti yang berisi *pop_gab.length.*
3. Baris 4, perulangan untuk mengisi nilai penalti sebanyak rata-rata.
4. Baris 5 , perhitungan nilai penalti pada setiap individu dengan cara mengurangi kebutuhan gizi balita dengan rata2 kandungan zat gizi yang direprensentasikan rata2.
5. Baris 8, mengembalikan nilai penalti.

5.1.5 *Fitness*

Parameter untuk menghitung *fitness* yaitu totalpenalti, total harga dan variasi. Kode Program 5.5 merupakan implementasi proses perhitungan nilai *fitness*.

```
1  public double[] fitness(double totalPenalti[], double[]
2  TtlHarga, int[] count) {
3    double[] fitness = new double[pop_gab[0].length];
4    for (int j = 0; j < pop_gab[0].length; j++) {
5       fitness[j] = (1000000 / TtlHarga[j])
6       + (count[j] * 0.5) + (10000 / totalPenalti[j]));
7    }
8    return fitness;
9  }
```

Kode Program 5.5 Implementasi *Fitness*

Penjelasan untuk Kode Program 5.5 sebagai berikut:

1. Baris 1-2, digunakan untuk memanggil parameter *fitness* yaitu total penalti, total harga dan variasi.
2. Baris 3, digunakan untuk inisaialisasi nilai *array fitness* yang berisi *pop_gab* .
3. Baris 4, perulangan untuk mengisi nilai *fitness* sebanyak *pop-gab*.
4. Baris 5-6, digunakan untuk menghitung nilai *fitness* pada masing-masing individu.
5. Baris 8, digunakan untuk mengembalikan nilai *fitness*.

5.1.6 Seleksi

Proses seleksi digunakan untuk mengambil nilai *fitness* terbesar yang lolos ke generasi berikutnya. Seleksi yang digunakan yaitu *elitism selection.* Kode Program 5.6 merupakan kode proses seleksi.

```
1  public int[][][] seleksi(double[] fitness, int
2  hasilkonversi[][][]) throws SQLException,
3  ClassNotFoundException {
4        for (int i = 0; i < fitness.length; i++) {
5         for (int j = 1; j < fitness.length; j++) {
6           if (fitness[j - 1] < fitness[j]) {
7              double temp = fitness[j];
8              fitness[j] = fitness[j - 1];
9              fitness[j - 1] = temp;
10              for (int k = 0; k < hari; k++) {
11                for (int l = 0; l < panjangKromosom;
12                l++)
13               {
14             int temp2 = hasilkonversi[k][j - 1][l];
15             hasilkonversi[k][j - 1][l] =
16             hasilkonversi[k][j][l];
17             hasilkonversi[k][j][l] = temp2;
18     for (int i = 0; i < fitness.length; i++) {
19       fitnessSorted[i] = fitness[i];
20     }
```

```
21          for (int i = 0; i < popsize; i++) {
22            for (int h = 0; h < hari; h++) {
23              for (int j = 0; j < panjangKromosom; j++) {
24                populasi[h][i][j]=hasilkonversi[h][i][j];
25               }
26             }
27          }
28          return hasilkonversi;
29  }
```

Kode Program 5.6 Proses Seleksi

Penjelasan untuk Kode Program 5.6 sebagai berikut:

1. Baris 1-3, digunakan untuk memanggil parameter seleksi yaitu *fitness* dan *hasilkonversi*.
2. Baris 4-5, perulangan untuk mengisi nilai *fitness* yang diurutkan.
3. Baris 6-9, digunakan untuk kondisi jika nilai *fitness* [j – 1] kurang dari nilai *fitness*[j], maka nilai *fitness* akan diurutkan dari yang terbesar sampai yang terkecil.
4. Baris 10-13, digunakan untuk perulangan menampilkan susunan kromosom dari masing-masing nilai *fitness* yang sudah diurutkan.
5. Baris 14, digunakan untuk menyimpan *hasilkonversi* pada variabel *temp2*.
6. Baris 15-16, digunakan untuk menyimpan *array hasil_konversi[k][j][l]* pada *array hasil_konversi[k][j][l].*
7. Baris 17, digunakan untuk menyimpan *temp2* pada *array hasil_konversi[k][j][l].*
8. Baris 18-19, digunakan untuk menyimpan hasil nilai *fitness* pada method *fitnessSorted.*
9. Baris 21-23, perulangan yang digunakan untuk mengambil nilai *fitness* sebanyak jumlah *popsize.*
10. Baris 24, digunakan untuk menyimpan susunan kromosom *hasil_konversi* pada *array* populasi.
11. Baris 28, digunakan untuk mengembalikan nilai *hasilkonversi*.

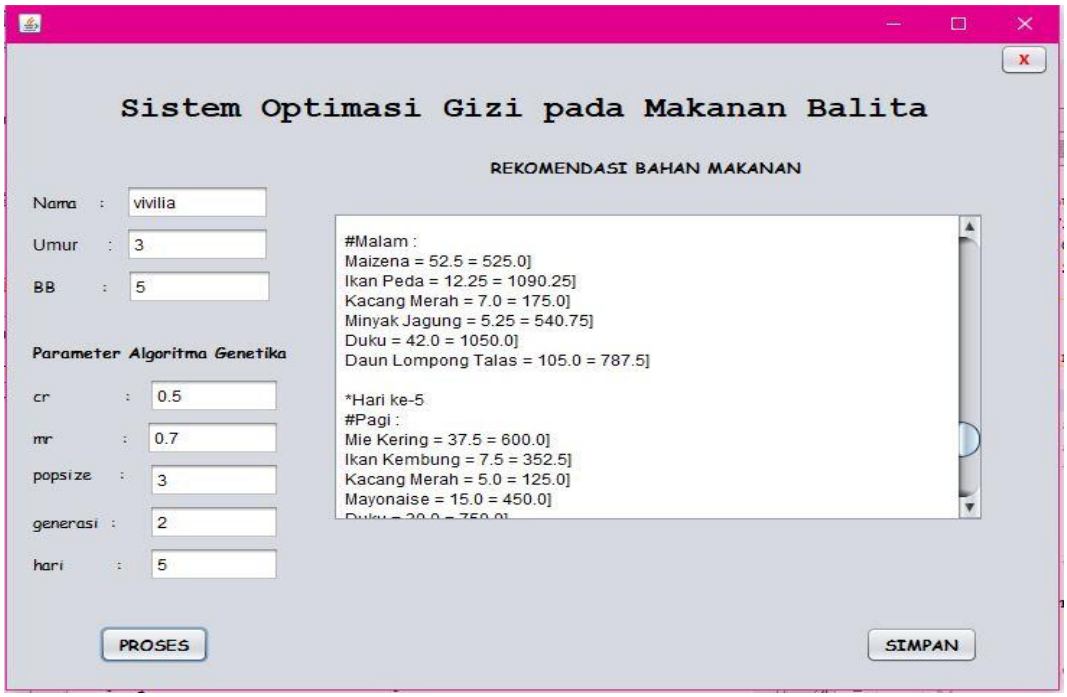
5.2 Implementasi Antarmuka Sistem

Antarmuka sistem merupakan tampilan sistem yang digunakan untuk interaksi antara pengguna dengan sistem. Gambar 5.1 merupakan tampilan awal sistem. Terdapat 2 form yang digunakan untuk memasukkan nilai data balita dan parameter algoritme genetika. Pengguna dapat memasukkan nilai data balita sesuai dengan data yang ada, dengan batasan nilai umur 1 sampai 5. Selanjutnya untuk parameter genetika juga dapat dimasukan sesuai dengan kebutuhan. Setelah semua data dimasukan, maka pengguna harus memlilih fitur proses untuk meproses data yang tela dimasukkan oleh pengguna. Hasil dari proses tersebut akan ditampilkan disamping form. Gambar 5.1 merupakan implementasi tampilan sistem optimasi gizi pada bahan makanan balita menggunakan algoritme genetika.

Gambar 5.1 Tampilan Awal Sistem

5.2.1 Halaman Hasil Rekomendasi Sistem

Halaman hasil rekomendasi sistem menampilkan rekomendasi bahan makanan perhari yang terdiri 3 waktu yaitu pagi, siang dan malam disertai harga masing-masing bahan makanan. Hasil rekomendasi bahan makanan dapat disimpan dalam bentuk file txt, ketika pengguna menekan tombol simpan. Gambar 5.2 merupakan hasil rekomendasi sistem.



Gambar 5.2 Hasil Rekomendasi Sistem