Aufgabe 5: Widerstand

Symbroson
Team-ID: 00165

24. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Um die Beste Schaltung ermitteln zu können muss man zwei Dinge implementieren: Zum einen muss man jede Mögliche Schaltung, also Kombinationen von Reihen- und Parallelschaltungen ausprobieren, und man muss die Widerstände selbst innerhalb der Schaltung tauschen, wobei das Tauschen zweier in Reihe oder Parallel geschalteter Widerstände vernachlässigbar ist. Schließlich muss von jeder Schaltung der Widerstandswert berechnet und die mit dem geringsten Unterschied zum gesuchten Widerstand in geeigneter Form ausgegeben werden.

2 Umsetzung

2.1 Eingabe

Die Eingabe des gesuchten Widerstandswertes erfolgt in der Konsole aus der Standardeingabe (stdin).

2.2 Permutation

Dann werden werden k Widerstände durch einen rekursiven Permutationsalgorithmus aus der gegebenen Liste ausgewählt. Dieser zählt jeweils mit, wie viele Elemente er bereits genommen hat, schreibt ab der Startposition jedes Element an die aktuellen Position in eine andere Liste und ruft sich selbst an der darauf folgenden Position als neuen Start auf. Die Schaltungen werden dann auf basis der generierten Widerstandsliste getestet.

Beim Testvorgang wurde auf eine dynamische Generierung der Schaltungen verzichtet und stattdessen jede mögliche Schaltung hardgecodet. Das hat unter Anderem die Folge, dass eine Schaltung eliminiert werden kann, weil sie einer anderen gleicht.

Die möglichen Permutationen der Widerstände innerhalb einer Schaltung sind, falls vorhanden, ebenfalls hardgecodet und als Array von Zeichenketten mit Indizes auf die Auswahlliste dargestellt. Dabei wurden all jene Permutationen Gestrichen, bei denen zwei Widerstände innerhalb einer Parallel- oder Reihenschaltung getauscht wurden, aber der Rest innerhalb der Schaltung einer anderen gleicht, da sie in ihrer Auswirkung in jedem Fall identisch wären.

2.3 Berechnung

Das Berechnen des Widerstandswertes einer Schaltung hängt von der Notation der Funktionsmakros ab. So ist zB. `ROW(R(0), PAR(R(1), R(2)))` eine Reihenschaltung mit dem Widerstand an erster Position und einer Parallelschaltung von den anderen zwei Widerständen an zweiter Position.

Die Formeln für Reihen- und Parallelschaltung sind jeweils $\sum R_i$ bzw $\frac{1}{\sum 1/R_i}$.

2.4 Ausgabe

Die Berechnung ist zugleich verbunden mit dem Schreiben der verwendeten Schaltungselemente in ein Array von Integern:

- `'R'`: Beginn einer Reihenschaltung
- `'P'`: Beginn einer Parallelschaltung
- `')'`: Ende einer Parallel/Reihenschaltung
- `0..k`: Position des Widerstandes in aktueller Auswahl
- `-1`: Ende der gesamten Schaltung

Somit ist eine effiziente Speicherung der Schaltung und später eine einfache Text Ausgabe möglich, bei der die Elemente jeweils ausformuliert geschrieben werden.

Die Elemente einer Reihen- (’Series’) oder Parallel- (’Parallel’) -schaltung werden jeweils in runden Klammern ’(...)’ mit einem Komma ’,’ getrennt notiert, und die Widerstandswerte mit einem ’Ω’ Zeichen gekennzeichnet.

3 Beispiele - widerstaende.txt

3.1 330 Ω

```
$ echo 330 | make run
resistor value (uint) in ohm: got 330 Ω
best: circuit 10 with 0.000000 Ω difference
330Ω -> 330.000000Ω
```

3.2 500 Ω

```
$ echo 500 | make run
resistor value (uint) in ohm: got 500 Ω
best: circuit 30 with 0.000000 Ω difference
Series(180Ω, 100Ω, 220Ω) -> 500.000000Ω
```

3.3 1037 Ω

```
$ echo 1037 | make run
resistor value (uint) in ohm: got 1037 Ω
best: circuit 47 with 0.005737 Ω difference
Parallel(Series(Parallel(1000Ω, 270Ω), 1200Ω), 3900Ω) -> 1036.994263Ω
```

3.4 2048 Ω

```
$ echo 330 | make run
resistor value (uint) in ohm: got 330 Ω
best: circuit 48 with 0.000000 Ω difference
Series(Parallel(Series(1000Ω, 4700Ω), 1800Ω), 680Ω) -> 2048.000000Ω
```

4 Quellcode

```
/* ... */

#define KMAX 4

vector<float> resistors; // Verfügbare Widerstände

cstr order,   // aktuelle Widerstandsreihenfolge
    best_o;   // beste Widerstandsreihenfolge

float best_r, // bester Widerstandswert
    best_d;   // bester Widerstandswertunterschied

uint len = 0,     // total r. count
    ck   = 1,     // current r. count
    cpos,         // current c. position
    srch_r,       // searched r. value
    perm[KMAX],   // current r. selection (k of n)
    best_p[KMAX]; // best r. permutation

char circuit[20], // aktuelle Schaltung
    best_c[20];   // beste Schaltung


/* ... */

// Speichert Eigenschaften der besten Schaltung
void saveBest(uint i, float d, float r) {
    best_o   = order;
    *circuit = i;
    best_d   = d;
    best_r   = r;
    memcpy(best_p, perm, sizeof(uint) * ck);
    memcpy(best_c, circuit, cpos + 1);
}

// Schreibt Schaltungselement in aktuelle Schaltung
inline void writeC(char c) {
    circuit[cpos++] = c;
}

// testet eine Widerstandspermutation
#define TEST(i, expr)                                                \
    order = "0123";                                                  \
    cpos  = 1;                                                       \
    if ((d = abs((r = (expr)) - srch_r)) < best_d || best_d == -1) { \
        writeC(-1);                                                  \
        saveBest(i, d, r);                                           \
    }

// testet mehrere gegebene Widerstandpermutationen
#define TEST2(i, expr, ...)                                              \
    for (cstr o: vector<cstr>(__VA_ARGS__)) {                            \
        order = o;                                                       \
        cpos  = 1;                                                       \
        if ((d = abs((r = (expr)) - srch_r)) < best_d || best_d == -1) { \
            writeC(-1);                                                  \
            saveBest(i, d, r);                                           \
        }                                                                \
    }

// gibt Widerstandswert abh. von Permutation und aktueller Auswahl zurück
inline float R(char i) {
    writeC('0' + i);
    return resistors[perm[order[i - 0] - '0']];
}
```

```
// berechnet und schreibt Reihenschaltung zweier Elemente
#define ROW(...) (writeC('R'), _ROW(__VA_ARGS__))
inline float _ROW(float a, float b, float c = 0, float d = 0) {
    writeC(')');
    return a + b + c + d;
}

// berechnet und schreibt Parallelschaltung zweier Elemente
#define PAR(...) (writeC('P'), _PAR(__VA_ARGS__))
inline float _PAR(float a, float b, float c = 0, float d = 0) {
    writeC(')');
    return 1 / (1 / a + 1 / b + (c ? 1 / c : 0) + (d ? 1 / d : 0));
}

// testet alle Schaltungen und Permutationen einer Widerstandsauswahl
void testPerm() {
    float d = -1, r = -1;

    if (!ck) { // kein Widerstand
        TEST(0, 0);
    } else if (ck == 1) { // 1 Widerstand
        TEST(10, R(0));
    } else if (ck == 2) { // 2 Widerstände
        TEST(20, ROW(R(0), R(1)));
        TEST(21, PAR(R(0), R(1)));
    } else if (ck == 3) { // 3 Widerstände
        TEST(30, ROW(R(0), R(1), R(2)));
        TEST(31, PAR(R(0), R(1), R(2)));

        TEST2(32, ROW(R(0), PAR(R(1), R(2))), {"012", "102", "201"});
        TEST2(33, PAR(R(0), ROW(R(1), R(2))), {"012", "102", "201"});

    } else if (ck == 4) { // 4 Widerstände

        TEST(40, ROW(R(0), R(1), R(2), R(3)));
        TEST(41, PAR(R(0), R(1), R(2), R(3)));

        TEST2(
            42, ROW(R(0), R(1), PAR(R(2), R(3))),
            {"0123", "0213", "0312", "1203", "1302", "2301"});
        // Entspricht oberer Schaltung
        // TEST2(
        //    42, PAR(R(0), R(1), ROW(R(2), R(3))),
        //    {"0123", "0213", "0312", "1203", "1302", "2301"});

        TEST2(
            43, PAR(R(0), ROW(R(1), R(2), R(3))),
            {"0123", "1023", "2013", "3012"});
        TEST2(
            44, ROW(R(0), PAR(R(1), R(2), R(3))),
            {"0123", "1023", "2013", "3012"});

        TEST2(
            45, PAR(ROW(R(0), R(1)), ROW(R(2), R(3))),
            {"0123", "0213", "0312", "1203", "1302", "2301"});
        TEST2(
            46, ROW(PAR(R(0), R(1)), PAR(R(2), R(3))),
            {"0123", "0213", "0312", "1203", "1302", "2301"});

        TEST2(
            47, PAR(R(0), ROW(R(1), PAR(R(2), R(3)))),
            {"0123", "0213", "0312", "1023", "1203", "1302", "2013", "2103",
             "2301", "3012", "3102", "3201"});
        TEST2(
            48, ROW(R(0), PAR(R(1), ROW(R(2), R(3)))),
            {"0123", "0213", "0312", "1023", "1203", "1302", "2013", "2103",
             "2301", "3012", "3102", "3201"});

    } else { // Fehler
        error("invalid resistor count: %i\n", ck);
        exit(1);
    }
}
```

```c
// Wählt alle Permutationen von k aus n Elementen aus
// d: aktuelle Rekursionstiefe, s: Startposition
void permut_mn(uint d, uint s) {
    if (d) {
        for (uint i = s; i < len; i++) {
            perm[d - 1] = i;
            permut_mn(d - 1, i + 1);
        }
    } else
        testPerm();
}


#undef R // Neudefinition auf beste Auswahl und Reihenfolge
#define R(i) resistors[best_p[best_o[i - '0'] - '0']]

int main(int argc, char* argv[]) {
    FILE* fp = NULL;
    int i;

    // Argumente und Eingabedatei einlesen
    /* ... */

    best_d = -1;

    // Gesuchten Widerstandswert einlesen
    printf("resistor value (uint) in ohm: ");
    do {
        if (fscanf(stdin, "%i", &srch_r) == 1) break;
        fprintf(stderr, "invalid!\n");
    } while (1);
    printf("got %i Ω\n", srch_r);

    // Schaltungen mit k aus n Wiederständen testen
    for (ck = 0; ck <= 4; ck++) permut_mn(ck, 0);

    // Ausgabe
    printf("best: circuit %i with %f Ω difference\n", *best_c, best_d);

    cpos = 1;

    if (!*best_c) printf("no resistor used");

    do {
        switch (best_c[cpos]) {
            case 'P': printf("Parallel("); break; // Parallel
            case 'R': printf("Series("); break;   // Reihe
            case ')': printf(")"); break;         // Elementende
            case -1: break;                       // Schaltungsende
            default:                              // Widerstand
                printf("%s%.lfΩ", cpos == 1 || strchr("PR",
                    best_c[cpos - 1]) ? "" : ", ", R(best_c[cpos]));
        }
    } while (best_c[cpos++] != -1 && cpos < 20);

    printf(" -> %fΩ\n", best_r);
    fclose(fp);
    return 0;
}
```