Министерство образования Республики Беларусь

Учреждение Образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных средств

Лабораторная работа № 7

«ИЗУЧЕНИЕ АЛГОРИТМОВ РАЗМЕЩЕНИЯ ЭЛЕМЕНТОВ»

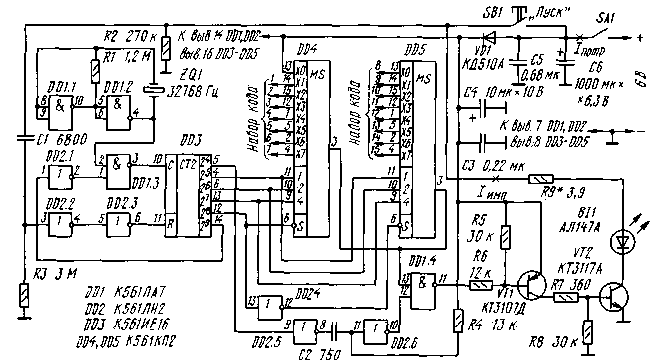
|  |  |
| --- | --- |
| Выполнили:  ст. гр. 850702  Маковский Р. А.  Турко В. Д. | Проверил:  Станкевич А. В. |
|  |  |

Минск 2020

Цель работы:

Изучить алгоритмы размещения конструктивных элементов на печатной плате.

Исходные данные(Вариант 3):



final Matrix POSITIONS = Matrix([[5, 6, 7], [2, 3, 4], [null, 1, null]]);

const Line A = Line('a', 2);

const Line B = Line('b', 2);

const Line C = Line('c', 2);

const Line D = Line('d', 3);

const Line E = Line('e', 2);

const Line F = Line('f', 3);

const Line G = Line('g', 2);

final List<Element> ELEMENTS = [

  Element('1', [A, B]),

  Element('2', [A, C]),

  Element('3', [B, C, D]),

  Element('4', [F, G]),

  Element('5', [D, F]),

  Element('6', [E, F, G]),

  Element('7', [D, E])

];

Ход работы:

1. **Алгоритм случайного поиска.**

void randomMethod(List<Element> elements, int numberOfAttempts) {

  List<Attempt> attemptsResults = [];

  for (int k = 0; k < numberOfAttempts; k++) {

    attemptsResults.add(\_random(elements));

  }

  print('Количество попыток: ${attemptsResults.length}');

  final bestAttemp = attemptsResults.indexOf(attemptsResults.min());

  print('Лучшая попытка №$bestAttemp');

  print('Суммарная взвешенная длина соединений: ${attemptsResults[bestAttemp].resultSum}');

  for (int i = 0; i < elements.length; i++) {

    print('${elements[i].name} -> ${attemptsResults[bestAttemp].positions[i]}');

  }

  final worst = attemptsResults.indexOf(attemptsResults.max());

  print('Худшая попытка №$worst');

  print('Суммарная взвешенная длина соединений: ${attemptsResults[worst].resultSum}');

  for (int i = 0; i < elements.length; i++) {

    print('${elements[i].name} -> ${attemptsResults[worst].positions[i]}');

  }

}

Attempt \_random(List<Element> elements) {

  double sum = 0;

  List<int> positions = [2, 3, 4, 5, 6, 7];

  positions.shuffle();

  positions.insert(0, 1);

  for (int i = 0; i < elements.length; i++) {

    for (int j = 0; j < elements.length; j++) {

      final coordOfCurrent = POSITIONS.indexOf(positions[i]);

      final coordOfConnecting = POSITIONS.indexOf(positions[j]);

      sum += elements[i].connectionWeightTo(elements[j]) \* coordOfCurrent.relativeLengthTo(coordOfConnecting);

    }

  }

  return Attempt(positions, sum);

}

Количество попыток: 100

Лучшая попытка №1

Суммарная взвешенная длина соединений: 1233.3333333333333

1 -> 1

2 -> 2

3 -> 3

4 -> 5

5 -> 6

6 -> 7

7 -> 4

Худшая попытка №74

Суммарная взвешенная длина соединений: 1966.6666666666665

1 -> 1

2 -> 7

3 -> 2

4 -> 5

5 -> 3

6 -> 4

7 -> 6

1. **Алгоритм последовательного поиска.**

void sequential(List<Element> elements) {

  print('Составим матрицу относительных расстояний');

  Matrix \_positionsMatrix = \_stateMatrix();

  \_outputState(\_positionsMatrix);

  List<int> \_minRow = \_findMinRow(\_positionsMatrix);

  List<int> \_sortedPositions = \_sortRow(\_minRow);

  print('\nВидно, что наименьшая сумма относительных расстояний достигается при использовании вектора очередности позициий $\_sortedPositions\n');

  print('Составим матрицу взвешенной связанности');

  Matrix \_linksWeight = \_lengthsMatrix();

  \_outputLengths(\_linksWeight);

  List<Element> \_sortedWeights = \_sortWeightRows(\_linksWeight);

  final \_fixed = \_fixPositions(\_sortedWeights, \_sortedPositions);

  \_sortedWeights = \_fixed.keys.first;

  \_sortedPositions = \_fixed[\_sortedWeights];

  Attempt res = result(\_sortedWeights, \_sortedPositions);

  print('Суммарная взвешенная длина соединений: ${res.resultSum}');

  for (int i = 0; i < \_sortedWeights.length; i++) {

    print('${\_sortedWeights[i].name} -> ${res.positions[i]}');

  }

}

Matrix \_stateMatrix() {

  Matrix matrix;

  POSITIONS.forEach((item) {

    if (item != null) {

      List<int> lengths = [];

      POSITIONS.forEach((comparing) {

        if (comparing != null) {

          final coordOfCurrent = POSITIONS.indexOf(item);

          final coordOfConnecting = POSITIONS.indexOf(comparing);

          final result = coordOfCurrent.relativeLengthTo(coordOfConnecting);

          lengths.add(result);

        }

      });

      if (matrix == null) {

        matrix = Matrix([lengths]);

      } else {

        matrix.addRow(lengths);

      }

    }

  });

  return matrix;

}

void \_outputState(Matrix matrix) {

  String header = 'P | ';

  String divider = '--|-';

  List<int> headers = [];

  POSITIONS.forEach((item) {

    if (item != null) {

      headers.add(item);

      header += '$item ';

      divider += '--';

    }

  });

  header += '| Sum';

  divider += '|----';

  print(header);

  print(divider);

  int i = 0;

  int j = 0;

  for (j = 0; j < matrix.n; j++) {

    String line = '${headers[j]} | ';

    int resultLength = 0;

    for (i = 0; i < matrix.m; i++) {

      if (matrix[i][j] != null) {

        final result = matrix[i][j];

        resultLength += result;

        line += '$result ';

      }

    }

    line += '| $resultLength';

    print('$line');

  }

}

List<int> \_findMinRow(Matrix matrix) {

  List<int> sum;

  int i = 0;

  for (i = 0; i < matrix.m; i++) {

    final rowSum = matrix[i].sum;

    if (sum == null) {

      sum = matrix[i];

    } else if (sum.sum > rowSum) {

      sum = matrix[i];

    }

  }

  return sum;

}

List<int> \_sortRow(List<int> positions) {

  List<int> positionsHeaders = [];

  POSITIONS.forEach((item) {

    if (item != null) {

      positionsHeaders.add(item);

    }

  });

  for (int j = 0; j < positions.length; j++) {

    for (int i = positions.length - 1; i > 0; i--) {

      final cur = positions[i];

      final next = positions[i - 1];

      if(cur < next) {

        positions.swap(i - 1, i);

        positionsHeaders.swap(i - 1, i);

      } else if (cur == next) {

        if (positionsHeaders[i] < positionsHeaders[i - 1]) {

          positions.swap(i - 1, i);

          positionsHeaders.swap(i - 1, i);

        }

      }

    }

  }

  return positionsHeaders;

}

Matrix \_lengthsMatrix() {

  Matrix matrix;

  ELEMENTS.forEach((item) {

    if (item != null) {

      List<double> lengths = [];

      ELEMENTS.forEach((comparing) {

        if (comparing != null) {

          if (item == comparing) {

            lengths.add(0);

          } else {

            final result = item.connectionWeightTo(comparing);

            lengths.add(result);

          }

        }

      });

      if (matrix == null) {

        matrix = Matrix([lengths]);

      } else {

        matrix.addRow(lengths);

      }

    }

  });

  return matrix;

}

void \_outputLengths(Matrix matrix) {

  String header = 'L | ';

  String divider = '--|-';

  List<int> headers = [];

  for (int i = 0; i < ELEMENTS.length; i++) {

    headers.add(i);

    header += '${ELEMENTS[i].name} ';

    divider += '--';

  }

  header += '| Sum';

  divider += '|----';

  print(header);

  print(divider);

  int i = 0;

  int j = 0;

  for (j = 0; j < matrix.n; j++) {

    String line = '${ELEMENTS[j].name} | ';

    double resultLength = 0;

    for (i = 0; i < matrix.m; i++) {

      if (matrix[i][j] != null) {

        final result = matrix[i][j];

        resultLength += result;

        line += '${result.toStringAsFixed(1)} ';

      }

    }

    line += '| ${resultLength.toStringAsFixed(1)}';

    print('$line');

  }

}

List<Element> \_sortWeightRows(Matrix weights) {

  List<Element> weightsHeaders = [];

  ELEMENTS.forEach((item) {

    weightsHeaders.add(item);

  });

  for (int j = 0; j < weights.m; j++) {

    for (int i = 0; i < weights.m - 1; i++) {

      final cur = weights[i].sum;

      final next = weights[i + 1].sum;

      if(cur > next) {

        weights.swap(i, i + 1);

        weightsHeaders.swap(i, i + 1);

      }

    }

  }

  return weightsHeaders;

}

Map<List<Element>, List<int>> \_fixPositions(List<Element> elements, List<int> positions) {

  final itemIndex = elements.indexOf(ELEMENTS[0]);

  final posIndex = positions.indexOf(1);

  print('\nВектор очередности позиций');

  print(positions);

  print('Вектор очередности размещения');

  elements.show();

  if (itemIndex != posIndex) {

    print('В условии задано, что соединитель (мы выбрали элемент с индексом 1 в качестве соединителя) должен быть на 1 позиции.\nДля этого скорректируем вектор очередности размещения элементов так, чтобы выполнялось условие.');

    elements.swap(0, 1);

  }

  print('Вектор очередности позиций');

  print(positions);

  print('Вектор очередности размещения');

  elements.show();

  return {elements: positions};

}

Attempt result(List<Element> elements, List<int> positions) {

  double sum = 0;

  for (int i = 0; i < elements.length; i++) {

    for (int j = 0; j < elements.length; j++) {

      final coordOfCurrent = POSITIONS.indexOf(positions[i]);

      final coordOfConnecting = POSITIONS.indexOf(positions[j]);

      sum += elements[i].connectionWeightTo(elements[j]) \* coordOfCurrent.relativeLengthTo(coordOfConnecting);

    }

  }

  return Attempt(positions, sum);

}

Составим матрицу относительных расстояний

P | 5 6 7 2 3 4 1 | Sum

--|---------------|----

5 | 0 1 2 1 2 3 3 | 12

6 | 1 0 1 2 1 2 2 | 9

7 | 2 1 0 3 2 1 3 | 12

2 | 1 2 3 0 1 2 2 | 11

3 | 2 1 2 1 0 1 1 | 8

4 | 3 2 1 2 1 0 2 | 11

1 | 3 2 3 2 1 2 0 | 13

Видно, что наименьшая сумма относительных расстояний достигается при использовании вектора очередности позициий [3, 1, 2, 4, 6, 5, 7]

Составим матрицу взвешенной связанности

L | 1 2 3 4 5 6 7 | Sum

--|------------------------------------|----

1 | 0.0 50.0 50.0 0.0 0.0 0.0 0.0 | 100.0

2 | 50.0 0.0 50.0 0.0 0.0 0.0 0.0 | 100.0

3 | 50.0 50.0 0.0 0.0 33.3 0.0 33.3 | 166.7

4 | 0.0 0.0 0.0 0.0 33.3 83.3 0.0 | 116.7

5 | 0.0 0.0 33.3 33.3 0.0 33.3 33.3 | 133.3

6 | 0.0 0.0 0.0 83.3 33.3 0.0 50.0 | 166.7

7 | 0.0 0.0 33.3 0.0 33.3 50.0 0.0 | 116.7

Вектор очередности позиций

[3, 1, 2, 4, 6, 5, 7]

Вектор очередности размещения

[1, 2, 7, 4, 3, 5, 6]

В условии задано, что соединитель (мы выбрали элемент с индексом 1 в качестве соединителя) должен быть на 1 позиции.

Для этого скорректируем вектор очередности размещения элементов так, чтобы выполнялось условие.

Вектор очередности позиций

[3, 1, 2, 4, 6, 5, 7]

Вектор очередности размещения

[2, 1, 7, 4, 3, 5, 6]

Суммарная взвешенная длина соединений: 1466.6666666666667

2 -> 3

1 -> 1

7 -> 2

4 -> 4

3 -> 6

5 -> 5

6 -> 7

Вывод:

В лабораторной работе мы познакомились задачами размещения.