Report 3

Team information (B23-ISE-02):

- Team member 1 (leader): Kirill Efimovich (k.efimovich@innopolis.university), grade 5/5
- Team member 2: Arsen Galiev (a.galiev@innopolis.university), grade: 5/5
- Team member 3: Asqar Arslanov (a.arslanov@innopolis.university), grade: 5/5
- Team member 4: Ilya-Linh Nguen (i.nguen@innopolis.university), grade: 5/5

Link to the product:

- https://github.com/quintet-sdr/optimization-pt3

Programming language:

- C++ (CMake)
- To launch the code:
  - for Unix-like: `$ ./run.sh`
  - for Windows: `> .\run.cmd`

Transportation problem:

- North-West Corner Method
- Vogel’s Approximation Method
- Russell’s Approximation Method

Tests:

You may find and edit the tests inside the `tests/` directory.

(1) **Input:**

- Name: Test-1.
- Supply: $\begin{bmatrix}251 & 300 & 400\end{bmatrix}$.
- Costs: $\begin{bmatrix}11 & 13 & 17 & 14\\ 16 & 18 & 14 & 10\\ 21 & 24 & 13 & 10\end{bmatrix}$.
- Demand: $\begin{bmatrix}200 & 225 & 275 & 250\end{bmatrix}$.

Output:

- Answer: The problem is not balanced.

(2) **Input:**

- Name: Test-2.
- Supply: $\begin{bmatrix}160 & 140 & 170\end{bmatrix}$.
- Costs: $\begin{bmatrix}7 & 8 & 1 & 2\\ 4 & 5 & 9 & 8\\ 9 & 2 & 5 & 6\end{bmatrix}$.
- Demand: $\begin{bmatrix}120 & 50 & 190 & 110\end{bmatrix}$.

Output:

- For the North-West Corner Method, the answer $x^0 = 3{,}320$.
- For Vogel's Approximation Method, the answer $x^0 = 1{,}330$.
- For Russell's Approximation Method, the answer $x^0 = 1{,}530$.

(3) **Input:**

- Name: Test-3.
- Supply: $\begin{bmatrix}251 & 300 & 400\end{bmatrix}$.
- Costs: $\begin{bmatrix}11 & 13 & 17 & 14\\ 16 & 18 & 14 & 10\\ 21 & 24 & 13 & 10\end{bmatrix}$.
- Demand: $\begin{bmatrix}200 & 225 & 275 & 250\end{bmatrix}$.

Output:

- For the North-West Corner Method, the answer $x^0 = 1{,}015$.
- For Vogel's Approximation Method, the answer $x^0 = 779$.
- For Russell's Approximation Method, the answer $x^0 = 807$.

Code:

src/main.cpp

```
#include "headers/matrix.hpp"
#include "headers/north_west.hpp"
#include "headers/vogel.hpp"
#include "headers/russel.hpp"
#include "headers/parser.hpp"

using namespace std;

int main() {
    JsonParser parser;
    vector<string> json_files = {
        "tests/Test-1.json",
        "tests/Test-2.json",
        "tests/Test-3.json"
    };
     for (const string& file : json_files) {
        cout << endl << string(65, '*') << endl;
        cout << "Processing: " << file << endl;
        string json_data = parser.read_json_file(file);
        vector<int> supply = parser.parse_supply(json_data);
        vector<int> demand = parser.parse_demand(json_data);
        vector<vector<int>> costs = parser.parse_costs(json_data);
        TransportMatrix tm(supply, costs, demand);
        cout << tm;
        if (!tm.check_the_balance(tm)) {
            continue;
        }
        TransportMatrix vogel_matrix = tm;
        TransportMatrix russel_matrix = tm;
        TransportMatrix north_west_matrix = tm;
        north_west(north_west_matrix);
        vogel(vogel_matrix);
        russel(russel_matrix);
    }
    return 0;
}
```

src/headers/matrix.cpp

```
#include <utility>
#include <iomanip>
#include "matrix.hpp"

using namespace std;

TransportMatrix::TransportMatrix() : supply(0), demand(0), costs(0) {};

TransportMatrix::TransportMatrix(const vector<int>& S, vector<vector<int>> C, const vector<int>& D) : supply(S

bool TransportMatrix::check_the_balance(const TransportMatrix& tm) {
  int sum_S = 0, sum_D = 0;
        for (int i : tm.supply) {
            sum_S += i;
        }
        for (int i : tm.demand) {
            sum_D += i;
        }
        if (sum_S != sum_D) {
            cout << "The problem is not balanced" << endl;
            return false;
        }
        return true;
}

ostream& operator<< (ostream& out, TransportMatrix& tm) {
    out << left << setw(15) << "SOURCE";

    out << left << setw(6 * tm.costs[0].size()) << "COST";

    out << left << setw(8) << "SUPPLY" << endl;

    out << string(tm.costs[0].size() * 6 + 28, '-') << endl;

    int source_i = 1;
    for (int i = 0; i < tm.costs.size(); i++) {
      out << left << setw(10) << "A" + to_string(source_i++);
      for (int j : tm.costs[i]) {
        out << right << setw(6) << j;
      }
      out << right << setw(6) << tm.supply[i] << endl;
    }

    out << string(tm.costs[0].size() * 6 + 28, '-') << endl;

    out << left << setw(10) << "DEMAND";
    for (int i : tm.demand) {
      out << right << setw(6) << i;
    }
    out << endl;
//    out << left << setw(8) << endl;

    return out;
}
```

src/headers/north_west.cpp

```
#include "north_west.hpp"
#include <algorithm>

using namespace std;

void north_west(TransportMatrix& tm) {
  cout << endl << string(tm.costs[0].size() * 6 + 40, '-') << endl;
  cout << "The North-West corner method" << endl;
  vector<pair<int, int>> peaked, answer;
  for (int i = 0; i < tm.costs.size(); i++) {
    for (int j = 0; j < tm.costs[0].size(); j++) {
      if (find(peaked.begin(), peaked.end(), make_pair(i, j)) == peaked.end()) {
        int cell = tm.costs[i][j];
        int supply = tm.supply[i], demand = tm.demand[j];

        int minimum = min(supply, demand);
        tm.demand[j] -= minimum;
        tm.supply[i] -= minimum;
        answer.emplace_back(cell, minimum);
        if (minimum == supply) {
          for (int k = j; k < tm.costs[i].size(); k++) {
            peaked.emplace_back(i, k);
          }
        } else {
          for (int k = i; k < tm.costs.size(); k++) {
            peaked.emplace_back(k, j);
          }
        }
      }
    }
  }
  int x0 = 0;
  for (int i = 0; i < answer.size(); i++) {
    x0 += answer[i].first * answer[i].second;
  }
  cout << "Answer: x0 = " << x0 << endl;
  cout << string(tm.costs[0].size() * 6 + 40, '-') << endl;
}
```

src/headers/russel.cpp

```
#include "russel.hpp"

using namespace std;

struct pair_hash {
  template<class T1, class T2>
  size_t operator()(const pair<T1, T2> &p) const {
    auto hash1 = hash<T1>{}(p.first);
    auto hash2 = hash<T2>{}(p.second);
    return hash1 ^ (hash2 << 1);
  }
};

pair<pair<int, int>, int> findMaxAbsValue(const vector<vector<pair<pair<int, int>, int>>> &matrix) {
  pair<pair<int, int>, int> maxElement = {{0, 0}, 0};
  int maxAbsValue = 0;

  for (const auto &row : matrix) {
    for (const auto &element : row) {
      int absValue = abs(element.second);
      if (absValue > maxAbsValue) {
        maxAbsValue = absValue;
        maxElement = element;
      }
    }
  }
  return maxElement;
}

void russel(TransportMatrix &tm) {
  cout << endl << string(tm.costs[0].size() * 6 + 40, '-') << endl;
  cout << "The Russel's Approximation method" << endl;
  unordered_set<pair<int, int>, pair_hash> peaked;
  vector<pair<int, int>> answer;
  int r = tm.costs.size(), c = tm.costs[0].size();
  while (peaked.size() != tm.costs.size() * tm.costs[0].size()) {
    vector<int> colMax, rowMax;
    for (int i = 0; i < tm.costs[0].size(); i++) {
      int maximum = numeric_limits<int>::min();
      for (int j = 0; j < tm.costs.size(); j++) {
        if (maximum < tm.costs[j][i] && peaked.find({j, i}) == peaked.end()) {
          maximum = tm.costs[j][i];
        }
      }
      if (maximum != numeric_limits<int>::min()) {
        colMax.push_back(maximum);
      }
    }

    for (int i = 0; i < tm.costs.size(); i++) {
      int maximum = numeric_limits<int>::min();
      for (int j = 0; j < tm.costs[i].size(); j++) {
        if (tm.costs[i][j] > maximum && peaked.find({i, j}) == peaked.end()) {
          maximum = tm.costs[i][j];
        }
      }
      if (maximum != numeric_limits<int>::min()) {
        rowMax.push_back(maximum);
      }
    }
```

```
    vector<vector<pair<pair<int, int>, int>>> temporary(r, vector<pair<pair<int, int>, int>>(c));
    int o = 0;
    for (int i = 0; i < tm.costs.size(); i++) {
      int l = 0;
      bool flag = false;
      for (int j = 0; j < tm.costs[i].size(); j++) {
        if (peaked.find({i, j}) == peaked.end()) {
          temporary[o][l].first = make_pair(i, j);
          temporary[o][l].second = tm.costs[i][j] - colMax[l] - rowMax[o];
          l++;
          flag = true;
        }
      }
      if (flag) {
        o++;
      }
    }

    pair<pair<int, int>, int> minVal = findMaxAbsValue(temporary);
    int i = minVal.first.first, j = minVal.first.second;
    int cell = tm.costs[i][j];
    int supply = tm.supply[i], demand = tm.demand[j];
    int minimum = min(supply, demand);
    tm.demand[j] -= minimum;
    tm.supply[i] -= minimum;
    answer.emplace_back(cell, minimum);
    if (minimum == supply) {
      for (int k = 0; k < tm.costs[i].size(); k++) {
        peaked.emplace(i, k);
      }
      r--;
    } else {
      for (int k = 0; k < tm.costs.size(); k++) {
        peaked.emplace(k, j);
      }
      c--;
    }
  }

  int x0 = 0;
  for (int i = 0; i < answer.size(); i++) {
    x0 += answer[i].first * answer[i].second;
  }
  cout << "Answer: x0 = " << x0 << endl;
  cout << string(tm.costs[0].size() * 6 + 40, '-') << endl;
}
```

src/headers/vogel.cpp

```
#include "vogel.hpp"
#include <algorithm>

using namespace std;

pair<vector<int>, vector<int>> find_row_col_diff(TransportMatrix& tm) {
    vector<int> row_diff;
    vector<int> col_diff;
    int rows = tm.costs.size();
    int cols = tm.costs[0].size();
    for (int i = 0; i < rows; i++) {
        vector<int> arr = tm.costs[i];
        sort(arr.begin(), arr.end());
        row_diff.push_back(arr[1] - arr[0]);
    }
    for (int i = 0; i < cols; i++) {
        vector<int> temp;
        for (int j = 0; j < rows; j++) {
            temp.push_back(tm.costs[j][i]);
        }
        sort(temp.begin(), temp.end());
        col_diff.push_back(temp[1] - temp[0]);
    }

    return make_pair(row_diff, col_diff);
}

int find_min_cost_in_row(TransportMatrix& tm, int row) {
    return *min_element(tm.costs[row].begin(), tm.costs[row].end());
}

int find_min_cost_in_column(TransportMatrix& tm, int col) {
    int min_cost = 1000;
    for (int j = 0; j < tm.costs.size(); j++) {
        min_cost = min(min_cost, tm.costs[j][col]);
    }
    return min_cost;
}

void update_supply_demand(TransportMatrix& tm, int supply_index, int demand_index, int amount, int cost, int&
    solution += amount * cost;
    tm.supply[supply_index] -= amount;
    tm.demand[demand_index] -= amount;

    if (tm.demand[demand_index] == 0) {
        for (int r = 0; r < tm.costs.size(); r++) {
            tm.costs[r][demand_index] = 1000;
        }
    } else {
        fill(tm.costs[supply_index].begin(), tm.costs[supply_index].end(), 1000);
    }
}

void vogel(TransportMatrix& tm) {
    cout << endl << string(tm.costs[0].size() * 6 + 40, '-') << endl;
    cout << "The Vogel Approximation method:" << endl;

    int solution = 0;
    while (*max_element(tm.supply.begin(), tm.supply.end()) != 0 || *max_element(tm.demand.begin(), tm.demand.
        pair<vector<int>, vector<int>> diffs = find_row_col_diff(tm);
```

```
        int max_in_rows = *max_element(diffs.first.begin(), diffs.first.end());
        int max_in_columns = *max_element(diffs.second.begin(), diffs.second.end());

        if (max_in_rows >= max_in_columns) {
            for (int i = 0; i < diffs.first.size(); i++) {
                if (diffs.first[i] == max_in_rows) {
                    int min_cost = find_min_cost_in_row(tm, i);
                    for (int j = 0; j < tm.costs[i].size(); j++) {
                        if (tm.costs[i][j] == min_cost) {
                            int amount = min(tm.supply[i], tm.demand[j]);
                            update_supply_demand(tm, i, j, amount, min_cost, solution);
                            break;
                        }
                    }
                    break;
                }
            }
        } else {
            for (int i = 0; i < diffs.second.size(); i++) {
                if (diffs.second[i] == max_in_columns) {
                    int min_cost = find_min_cost_in_column(tm, i);
                    for (int c = 0; c < tm.costs.size(); c++) {
                        if (tm.costs[c][i] == min_cost) {
                            int amount = min(tm.supply[c], tm.demand[i]);
                            update_supply_demand(tm, c, i, amount, min_cost, solution);
                            break;
                        }
                    }
                    break;
                }
            }
        }
    }
    cout << "Answer: x0 = " << solution << endl;
    cout << string(tm.costs[0].size() * 6 + 40, '-') << endl;
}
```