PFCM\pfcm.cpp

```cpp
/*---------------- File: pfcm.cpp  ----------------------+
| Problema da de Fluxo de Custo Mínimo (PFCM)          |
|                                                      |
| Adaptado por Guilherme Francis e Lucas Rocha         |
+------------------------------------------------+ */

#include <bits/stdc++.h>
#include <ilcplex/ilocplex.h>

using namespace std;
ILOSTLBEGIN; //MACRO - "using namespace" for ILOCPEX

#define CPLEX_TIME_LIM 3600 //3600 segundos
#define MAX_INT 100
// DADOS PARA O PROBLE PFCM

typedef struct edges
{
    int cost, l, u;
} Edges;

typedef struct nodeOffer{
    int id, offer;
} NodeOffer;

typedef struct nodeDemand{
    int id, demand;
} NodeDemand;

typedef struct nodeTransfer{
    int id;
} NodeTransfer;

int vertexCount;
int edgeCount;
int offerCount;
int demandCount;
int transferCount;

vector<vector<Edges>> A(MAX_INT, vector<Edges>(MAX_INT));
vector<NodeOffer> S(MAX_INT);
vector<NodeDemand> D(MAX_INT);
vector<NodeTransfer> T(MAX_INT);

void cplex(){
    // CPLEX
    IloEnv env; // Define o ambiente do CPLEX

    // Variaveis
    int numberVar = 0;
    int numberRes = 0;

    /*
        MODELAGEM
    */

    // Variáveis de decisão
    IloArray<IloNumVarArray> x(env);
    for(int i = 0; i < vertexCount; i++){
        x.add(IloNumVarArray(env));
        for(int j = 0; j < vertexCount; j++){
            x[i].add(IloIntVar(env, 0, A[i][j].u));
            if(A[i][j].u != 0) numberVar++;
        }
    }

    IloModel model(env);

    // somatórios para restrições e função objetivo
    IloExpr sum1(env);
    IloExpr sum2(env);

    // Função Objetivo:
    sum1.clear();
    for(int i = 0; i < vertexCount; i++){
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[i][j].u != 0){
                sum1 += (A[i][j].cost * x[i][j]);
            }
        }
    }
    model.add(IloMinimize(env, sum1)); // Minimização

    // Restrições:

    // Restrição de oferta
    for(int i = 0; i < offerCount; i++){
        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[S[i].id][j].u != 0){
                sum1 += x[S[i].id][j];
            }
        }

        // somatorio de tudo que entra
        sum2.clear();
        for(int k = 0; k < vertexCount; k++){
            // se a aresta existe
            if(A[k][S[i].id].u != 0){
                sum2 += x[k][S[i].id];
            }
        }

        // tudo que sai - tudo que entra <= oferta
        model.add(sum1 - sum2 <= S[i].offer);
    }
    numberRes++;

    // Restrição da demanda
    for(int i = 0; i < demandCount; i++){
        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[D[i].id][j].u != 0){
                sum1 += x[D[i].id][j];
            }
        }

        // somatorio de tudo que entra
        sum2.clear();
        for(int k = 0; k < vertexCount; k++){
            // se a aresta existe
            if(A[k][D[i].id].u != 0){
                sum2 += x[k][D[i].id];
            }
        }

        // tudo que sai - tudo que entra <= -Demanda
        model.add(sum1 - sum2 <= -D[i].demand);
    }
    numberRes++;


    // Restrição da conservação de fluxo
    for(int i = 0; i < transferCount; i++){
        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[T[i].id][j].u != 0){
                sum1 += x[T[i].id][j];
            }
        }

        // somatorio de tudo que entra
        sum2.clear();
        for(int k = 0; k < vertexCount; k++){
            // se a aresta existe
            if(A[k][T[i].id].u != 0){
                sum2 += x[k][T[i].id];
            }
        }

        // tudo que sai - tudo que entra <= -Demanda
        model.add(sum1 - sum2 == 0);
    }
    numberRes++;

    // restrições de capacidade
    for(int i = 0; i < vertexCount; i++){
        for(int j = 0; j < vertexCount; j++){
            if(A[i][j].u != 0){
                model.add(x[i][j] <= A[i][j].u);
            }
        }
    }
    numberRes++;

    //------ EXECUCAO do MODELO ----------
    time_t timer, timer2;
    IloNum value, objValue;
    double runTime;
    string status;

    //Informacoes -------------------------------------------
    printf("--------Informacoes da Execucao:----------\n\n");
    printf("#Var: %d\n", numberVar);
    printf("#Restricoes: %d\n", numberRes);
    cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    IloCplex cplex(model);
    cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);

    time(&timer);
    cplex.solve();//COMANDO DE EXECUCAO
    time(&timer2);

    bool sol = true;
    switch(cplex.getStatus()){
        case IloAlgorithm::Optimal:
            status = "Optimal";
            break;
        case IloAlgorithm::Feasible:
            status = "Feasible";
            break;
        default:
            status = "No Solution";
            sol = false;
    }

    cout << endl << endl;
    cout << "Status da FO: " << status << endl;

    if(sol){
        objValue = cplex.getObjValue();
        runTime = difftime(timer2, timer);

        cout << "Variaveis de decisao: " << endl;

        for(int i = 0; i < vertexCount; i++){
            for(int j = 0; j < vertexCount; j++){
                // se existe aresta
                if(A[i][j].u != 0){
                    value = IloRound(cplex.getValue(x[i][j]));
                    printf("x[%d][%d]: %.0lf\n", i, j, value);
                }
            }
        }

        cout << "Funcao Objetivo Valor = " << objValue << endl;
        printf("..(%.6lf seconds).\n\n", runTime);

    }else{
        printf("No Solution!\n");
    }

    //Free Memory
    cplex.end();
    sum1.end();
    sum2.end();

    cout << "Memory usage before end:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;
    env.end();
}

int main(){
    // Leitura dos dados
    cin >> vertexCount >> edgeCount;
    for (int i = 0; i < edgeCount; i++) {
        int x, y, c, limL, limU;
        cin >> x >> y >> c >> limL >> limU;
        A[x][y] = {c, limL, limU};
    }

    cin >> offerCount;
    for (int i = 0; i < offerCount; i++)
        cin >> S[i].id >> S[i].offer;

    cin >> demandCount;
    for (int i = 0; i < demandCount; i++)
        cin >> D[i].id >> D[i].demand;

    cin >> transferCount;
    for (int i = 0; i < transferCount; i++)
        cin >> T[i].id;

    // Impressão dos dados de entrada formatados
    cout << "\nGrafo (arestas com custos e capacidades):\n";
    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < vertexCount; j++) {
            if (A[i][j].u > 0) {
                cout << "Aresta " << i << " -> " << j << ": Custo = " << A[i][j].cost << ", Capacidade = [" << A[i][j].l << ", " << A[i][j].u << "]\n";
            }
        }
    }

    cout << "\nNós de oferta:";
    for (int i = 0; i < offerCount; i++)
        cout << " (" << S[i].id << ", " << S[i].offer << ")";

    cout << "\nNós de demanda:";
    for (int i = 0; i < demandCount; i++)
        cout << " (" << D[i].id << ", " << D[i].demand << ")";

    cout << "\nNós de transferência:";
    for (int i = 0; i < transferCount; i++)
        cout << " " << T[i].id;
    cout << "\n\n";

    cplex();
    return 0;

    return 0;
}


```

PT\pt.cpp

```cpp
/*---------------- File: pt.cpp  ----------------------+
| Problema do Transporte (PT)                           |
|                                                       |
| Adaptado por Guilherme Francis e Lucas Rocha          |
+----------------------------------------------------+ */

#include <bits/stdc++.h>
#include <ilcplex/ilocplex.h>

using namespace std;
ILOSTLBEGIN; //MACRO - "using namespace" for ILOCPEX

#define CPLEX_TIME_LIM 3600 //3600 segundos
#define MAX_INT 100
// DADOS PARA O PROBLE PFCM

typedef struct edges
{
    int cost = -1;
} Edges;

typedef struct nodeOffer{
    int id, offer;
} NodeOffer;

typedef struct nodeDemand{
    int id, demand;
} NodeDemand;


int vertexCount;
int edgeCount;
int offerCount;
int demandCount;

vector<vector<Edges>> A(MAX_INT, vector<Edges>(MAX_INT));
vector<NodeOffer> S(MAX_INT);
vector<NodeDemand> D(MAX_INT);

void cplex(){
    // CPLEX
    IloEnv env; // Define o ambiente do CPLEX

    // Variaveis
    int numberVar = 0;
    int numberRes = 0;

    /*
        MODELAGEM
    */

    // Variáveis de decisão
    IloArray<IloNumVarArray> x(env);
    for(int i = 0; i < vertexCount; i++){
        x.add(IloNumVarArray(env));
        for(int j = 0; j < vertexCount; j++){
            if(A[i][j].cost == -1){
                x[i].add(IloIntVar(env, 0, 0));
            }else{
                x[i].add(IloIntVar(env, 0, INT_MAX));
                numberVar++;
            }
        }
    }

    IloModel model(env);

    // somatórios para restrições e função objetivo
    IloExpr sum1(env);

    // Função Objetivo:
    sum1.clear();
    for(int i = 0; i < vertexCount; i++){
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[i][j].cost != -1){
                sum1 += (A[i][j].cost * x[i][j]);
            }
        }
    }
    model.add(IloMinimize(env, sum1)); // Minimização

    // Restrições:

    // Restrição de oferta
    for(int i = 0; i < offerCount; i++){
        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[S[i].id][j].cost != -1){
                sum1 += x[S[i].id][j];
            }
        }

        // tudo que sai <= oferta
        model.add(sum1 <= S[i].offer);
    }
    numberRes++;

    // Restrição da demanda
    for(int i = 0; i < demandCount; i++){
        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
```

```cpp
            // se a aresta existe
            if(A[j][D[i].id].cost != -1){
                sum1 += x[j][D[i].id];
            }
        }

        // tudo que sai == Demanda
        model.add(sum1 == D[i].demand);
    }
    numberRes++;

    //------ EXECUCAO do MODELO ----------
    time_t timer, timer2;
    IloNum value, objValue;
    double runTime;
    string status;

    //Informacoes ---------------------------------------------
    printf("--------Informacoes da Execucao:----------\n\n");
    printf("#Var: %d\n", numberVar);
    printf("#Restricoes: %d\n", numberRes);
    cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    IloCplex cplex(model);
    cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);

    time(&timer);
    cplex.solve();//COMANDO DE EXECUCAO
    time(&timer2);

    bool sol = true;
    switch(cplex.getStatus()){
        case IloAlgorithm::Optimal:
            status = "Optimal";
            break;
        case IloAlgorithm::Feasible:
            status = "Feasible";
            break;
        default:
            status = "No Solution";
            sol = false;
    }

    cout << endl << endl;
    cout << "Status da FO: " << status << endl;

    if(sol){
        objValue = cplex.getObjValue();
        runTime = difftime(timer2, timer);

        cout << "Variaveis de decisao: " << endl;

        for(int i = 0; i < vertexCount; i++){
            for(int j = 0; j < vertexCount; j++){
                // se existe aresta
                if(A[i][j].cost != -1){
                    value = IloRound(cplex.getValue(x[i][j]));
                    printf("x[%d][%d]: %.0lf\n", i, j, value);
                }
            }
        }

        cout << "Funcao Objetivo Valor = " << objValue << endl;
        printf("..(%.6lf seconds).\n\n", runTime);

    }else{
        printf("No Solution!\n");
    }

    //Free Memory
    cplex.end();
    sum1.end();

    cout << "Memory usage before end:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;
    env.end();
}

int main(){
    // Leitura dos dados
    cin >> vertexCount >> edgeCount;
    for (int i = 0; i < edgeCount; i++) {
        int x, y, c;
        cin >> x >> y >> c;
        A[x][y].cost = c;
    }

    cin >> offerCount;
    for (int i = 0; i < offerCount; i++)
        cin >> S[i].id >> S[i].offer;

    cin >> demandCount;
    for (int i = 0; i < demandCount; i++)
        cin >> D[i].id >> D[i].demand;

    // Impressão dos dados de entrada formatados
    cout << "\nGrafo (arestas com custos e capacidades):\n";
    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < vertexCount; j++) {
            if (A[i][j].cost != -1) {
                cout << "Aresta " << i << " -> " << j << ": Custo = " << A[i][j].cost << "\n";
            }
        }
    }

    cout << "\nNós de oferta:";
    for (int i = 0; i < offerCount; i++)
        cout << " (" << S[i].id << ", " << S[i].offer << ")";

    cout << "\nNós de demanda:";
    for (int i = 0; i < demandCount; i++)
        cout << " (" << D[i].id << ", " << D[i].demand << ")";
    cout << "\n\n";

    cplex();
    return 0;

    return 0;
}
```

PD\pd.cpp

```cpp
/*---------------- File: pd.cpp  -----------------------+
| Problema da Designação (PD)                           |
|                                                       |
| Adaptado por Guilherme Francis e Lucas Rocha          |
+-----------------------------------------------------+ */

#include <bits/stdc++.h>
#include <ilcplex/ilocplex.h>

using namespace std;
ILOSTLBEGIN

#define CPLEX_TIME_LIM 3600

int N;
vector<vector<int>> custo;

void cplex() {
    IloEnv env;
    IloModel model(env);
    IloCplex cplex(model);

    IloArray<IloNumVarArray> x(env, N);
    for (int i = 0; i < N; i++) {
        x[i] = IloNumVarArray(env, N, 0, 1, ILOBOOL);
    }

    // Função Objetivo: Minimizar o custo total
    IloExpr obj(env);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            obj += custo[i][j] * x[i][j];
        }
    }
    model.add(IloMinimize(env, obj));

    // Restrições
    // Cada agente deve executar exatamente uma tarefa
    for (int i = 0; i < N; i++) {
        IloExpr sum(env);
        for (int j = 0; j < N; j++) {
            sum += x[i][j];
        }
        model.add(sum == 1);
        sum.end();
    }

    // Cada tarefa deve ser atribuída a exatamente um agente
    for (int j = 0; j < N; j++) {
        IloExpr sum(env);
        for (int i = 0; i < N; i++) {
            sum += x[i][j];
        }
        model.add(sum == 1);
        sum.end();
    }

    // Informações antes da execução
    cout << "-------- Informações da Execução ----------\n";
    cout << "#Variáveis: " << (N * N) << "\n";
    cout << "#Restrições: " << (2 * N) << "\n";
    cout << "Uso de memória antes da solução: " << env.getMemoryUsage() / (1024. * 1024.) << " MB\n";

    cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);

    time_t timer, timer2;
    time(&timer);
    cplex.solve();
    time(&timer2);

    bool sol = true;
    string status;
    switch (cplex.getStatus()) {
        case IloAlgorithm::Optimal:
            status = "Optimal";
            break;
        case IloAlgorithm::Feasible:
            status = "Feasible";
            break;
        default:
            status = "No Solution";
            sol = false;
    }

    cout << "\nStatus da Função Objetivo: " << status << "\n";
    if (sol) {
        cout << "Custo mínimo: " << cplex.getObjValue() << "\n";
        cout << "Atribuições encontradas:\n";
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (cplex.getValue(x[i][j]) > 0.5) {
                    cout << "Agente " << i << " -> Tarefa " << j << " (Custo: " << custo[i][j] << ")\n";
                }
            }
        }
        printf("Tempo de execução: %.6lf segundos\n\n", difftime(timer2, timer));
    } else {
        cout << "Nenhuma solução ótima encontrada.\n";
    }

    cout << "Uso de memória após solução: " << env.getMemoryUsage() / (1024. * 1024.) << " MB\n";

    obj.end();
    cplex.end();
    model.end();
    env.end();
}

int main() {
    cin >> N;
    custo.resize(N, vector<int>(N));

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cin >> custo[i][j];
        }
    }

    cout << "Matriz de Custos Lida:\n";
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << custo[i][j] << " ";
        }
        cout << endl;
    }

    cplex();
    return 0;
}

```

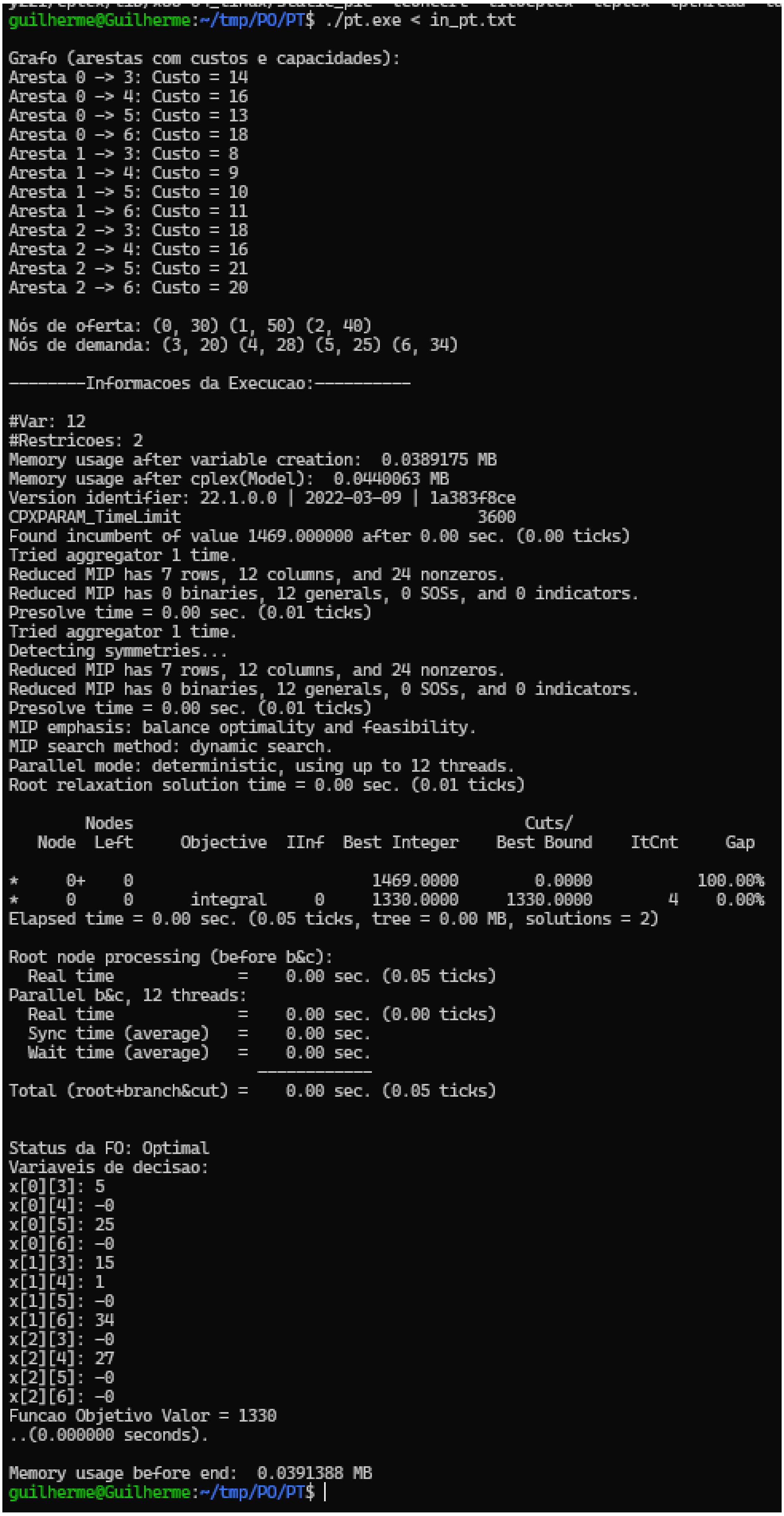
PCM\pcm.cpp

```
/*---------------- File: pcm.cpp  ---------------------+
| Problema do Caminho Minimo (PCM)                     |
|                                                      |
| Adaptado por Guilherme Francis e Lucas Rocha         |
+-------------------------------------------------------+ */

#include <bits/stdc++.h>
#include <ilcplex/ilocplex.h>

using namespace std;
ILOSTLBEGIN

#define CPLEX_TIME_LIM 3600

struct Edge {
    int origem, destino, custo;
};

int N, M;
vector<Edge> arestas;
int origemFonte, destinoDestino;

void cplex() {
    IloEnv env;
    IloModel model(env);
    IloCplex cplex(model);

    IloNumVarArray x(env, M, 0, 1, ILOBOOL);

    IloExpr obj(env);
    for (int i = 0; i < M; i++) {
        obj += arestas[i].custo * x[i];
    }
    model.add(IloMinimize(env, obj));

    for (int v = 0; v < N; v++) {
        IloExpr fluxoEntrada(env);
        IloExpr fluxoSaida(env);
        for (int i = 0; i < M; i++) {
            if (arestas[i].origem == v) fluxoSaida += x[i];
            if (arestas[i].destino == v) fluxoEntrada += x[i];
        }
        if (v == origemFonte)
            model.add(fluxoSaida - fluxoEntrada >= 1);
        else if (v == destinoDestino)
            model.add(fluxoEntrada - fluxoSaida >= 1);
        else
            model.add(fluxoEntrada - fluxoSaida == 0);
    }

    cout << "-------- Informacoes da Execucao ----------\n";
```

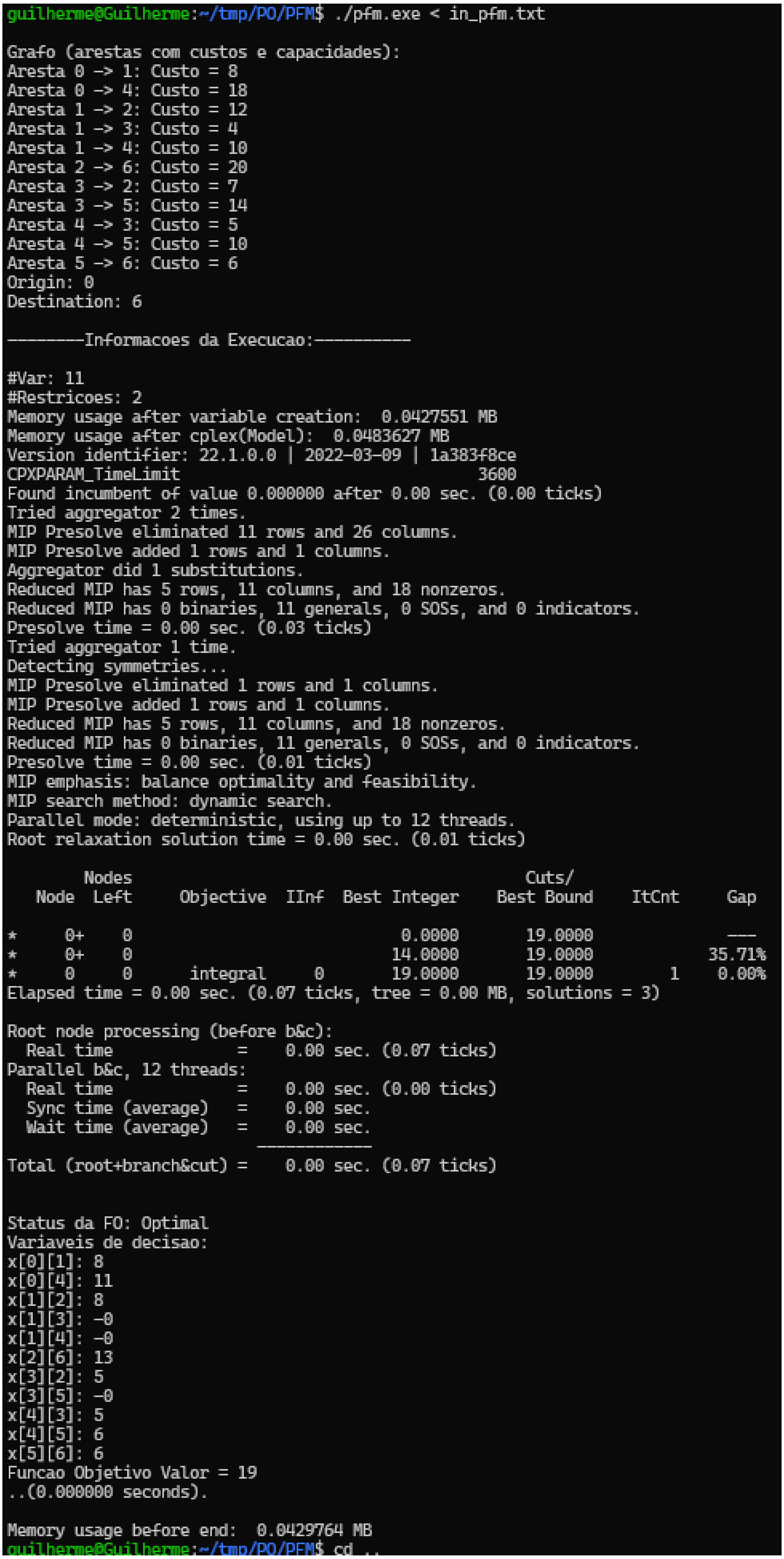
```cpp
    cout << "#Variaveis: " << M << "\n";
    cout << "#Restricoes: " << N << "\n";
    cout << "Uso de memoria antes da solucao: " << env.getMemoryUsage() / (1024. * 1024.) << " MB\n";

    cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);

    time_t timer, timer2;
    time(&timer);
    cplex.solve();
    time(&timer2);

    bool sol = true;
    string status;
    switch (cplex.getStatus()) {
        case IloAlgorithm::Optimal:
            status = "Optimal";
            break;
        case IloAlgorithm::Feasible:
            status = "Feasible";
            break;
        default:
            status = "No Solution";
            sol = false;
    }

    cout << "\nStatus da Funcao Objetivo: " << status << "\n";
    if (sol) {
        cout << "Custo minimo: " << cplex.getObjValue() << "\n";
        cout << "Arestas no caminho minimo:\n";
        for (int i = 0; i < M; i++) {
            if (cplex.getValue(x[i]) > 0.5)
                cout << "  " << arestas[i].origem << " -> " << arestas[i].destino << " (Custo: " << arestas[i].custo << ")\n";
        }
        printf("Tempo de execucao: %.6lf segundos\n\n", difftime(timer2, timer));
    } else {
        cout << "Erro: O problema eh inviavel! Verifique se ha um caminho possivel entre origem e destino.\n";
    }

    cout << "Uso de memoria apos solucao: " << env.getMemoryUsage() / (1024. * 1024.) << " MB\n";

    obj.end();
    cplex.end();
    model.end();
    env.end();
}

int main() {
    cin >> N >> M;
    arestas.resize(M);
    for (int i = 0; i < M; i++) {
        cin >> arestas[i].origem >> arestas[i].destino >> arestas[i].custo;
    }
    cin >> origemFonte >> destinoDestino;

    cout << "Lista de Arestas:\n";
    for (const auto& e : arestas) {
        cout << " " << e.origem << " -> " << e.destino << " (Custo: " << e.custo << ")\n";
    }
    cout << "Origem: " << origemFonte << ", Destino: " << destinoDestino << "\n\n";

    cplex();
    return 0;
}

```

PFM\pfm.cpp

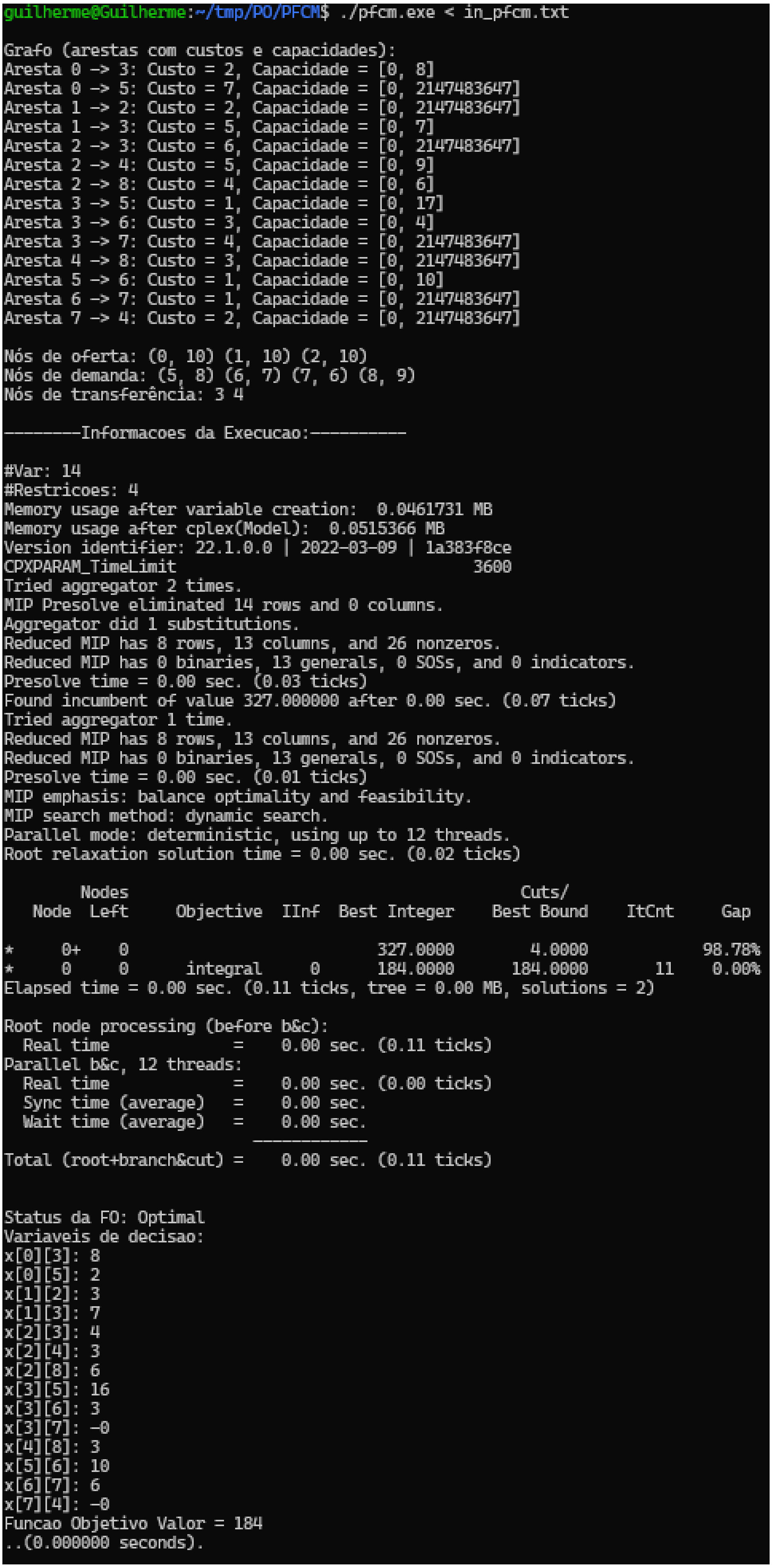
```cpp
/*---------------- File: pfm.cpp  ----------------------+
| Problema de Fluxo Máximo (PFM)                       |
|                                                      |
| Adaptado por Guilherme Francis e Lucas Rocha         |
+----------------------------------------------------+ */


#include <bits/stdc++.h>
#include <ilcplex/ilocplex.h>

using namespace std;
ILOSTLBEGIN; //MACRO - "using namespace" for ILOCPEX

#define CPLEX_TIME_LIM 3600 //3600 segundos
#define MAX_INT 100
// DADOS PARA O PROBLE PFCM

typedef struct edges
{
    int maximum_capacity = -1;
} Edges;

int vertexCount;
int edgeCount;
int origin;
int destination;

vector<vector<Edges>> A(MAX_INT, vector<Edges>(MAX_INT));

void cplex(){
    // CPLEX
    IloEnv env; // Define o ambiente do CPLEX

    // Variaveis
    int numberVar = 0;
    int numberRes = 0;

    /*
        MODELAGEM
    */

    // Variáveis de decisão
    IloArray<IloNumVarArray> x(env);
    for(int i = 0; i < vertexCount; i++){
        x.add(IloNumVarArray(env));
        for(int j = 0; j < vertexCount; j++){
            if(A[i][j].maximum_capacity != -1){
                x[i].add(IloIntVar(env, 0, A[i][j].maximum_capacity));
                numberVar++;
            }else{
                x[i].add(IloIntVar(env, 0, 0));
            }
        }
    }

    IloModel model(env);

    // somatórios para restrições e função objetivo
    IloExpr sum1(env);
    IloExpr sum2(env);

    // Função Objetivo:
    sum1.clear();
    for(int i = 0; i < vertexCount; i++){
        // se a aresta existe
        if(A[origin][i].maximum_capacity != -1){
            sum1 += x[origin][i];
        }
    }
    model.add(IloMaximize(env, sum1)); // Maximização

    // Restrições:

    // Restrição da conservação de fluxo
    for(int i = 0; i < vertexCount; i++){
        if(i == origin || i == destination) continue;

        // somatorio de tudo que sai
        sum1.clear();
        for(int j = 0; j < vertexCount; j++){
            // se a aresta existe
            if(A[i][j].maximum_capacity != -1){
                sum1 += x[i][j];
            }
        }

        // somatorio de tudo que entra
        sum2.clear();
        for(int k = 0; k < vertexCount; k++){
            // se a aresta existe
            if(A[k][i].maximum_capacity != 0){
                sum2 += x[k][i];
            }
        }

        // tudo que sai  == tudo que entra
        model.add(sum1 == sum2);
    }
    numberRes++;

    // restrições de capacidade
    for(int i = 0; i < vertexCount; i++){
        for(int j = 0; j < vertexCount; j++){
            if(A[i][j].maximum_capacity != -1){
                model.add(x[i][j] <= A[i][j].maximum_capacity);
            }
        }
    }
    numberRes++;

    //------ EXECUCAO do MODELO ----------
    time_t timer, timer2;
    IloNum value, objValue;
    double runTime;
    string status;

    //Informacoes ---------------------------------------------
    printf("--------Informacoes da Execucao:----------\n\n");
    printf("#Var: %d\n", numberVar);
    printf("#Restricoes: %d\n", numberRes);
    cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    IloCplex cplex(model);
    cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;

    cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);

    time(&timer);
    cplex.solve();//COMANDO DE EXECUCAO
    time(&timer2);

    bool sol = true;
    switch(cplex.getStatus()){
        case IloAlgorithm::Optimal:
            status = "Optimal";
            break;
        case IloAlgorithm::Feasible:
            status = "Feasible";
            break;
        default:
            status = "No Solution";
            sol = false;
    }

    cout << endl << endl;
    cout << "Status da FO: " << status << endl;

    if(sol){
        objValue = cplex.getObjValue();
        runTime = difftime(timer2, timer);

        cout << "Variaveis de decisao: " << endl;

        for(int i = 0; i < vertexCount; i++){
            for(int j = 0; j < vertexCount; j++){
                // se existe aresta
                if(A[i][j].maximum_capacity != -1){
                    value = IloRound(cplex.getValue(x[i][j]));
                    printf("x[%d][%d]: %.0lf\n", i, j, value);
                }
            }
        }

        cout << "Funcao Objetivo Valor = " << objValue << endl;
        printf("..(%.6lf seconds).\n\n", runTime);

    }else{
        printf("No Solution!\n");
    }

    //Free Memory
    cplex.end();
    sum1.end();
    sum2.end();

    cout << "Memory usage before end:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB" << endl;
    env.end();
}

int main(){
    // Leitura dos dados
    cin >> vertexCount >> edgeCount;
    for (int i = 0; i < edgeCount; i++) {
        int x, y, c, limL, limU;
        cin >> x >> y >> c;
        A[x][y].maximum_capacity = c;
    }

    cin >> origin >> destination;

    // Impressão dos dados de entrada formatados
    cout << "\nGrafo (arestas com custos e capacidades):\n";
    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < vertexCount; j++) {
            if (A[i][j].maximum_capacity != -1) {
                cout << "Aresta " << i << " -> " << j << ": Custo = " << A[i][j].maximum_capacity << "\n";
            }
        }
    }

    cout << "Origin: " << origin << endl;
    cout << "Destination: " << destination << endl << endl;

    cplex();
    return 0;

    return 0;
}
```



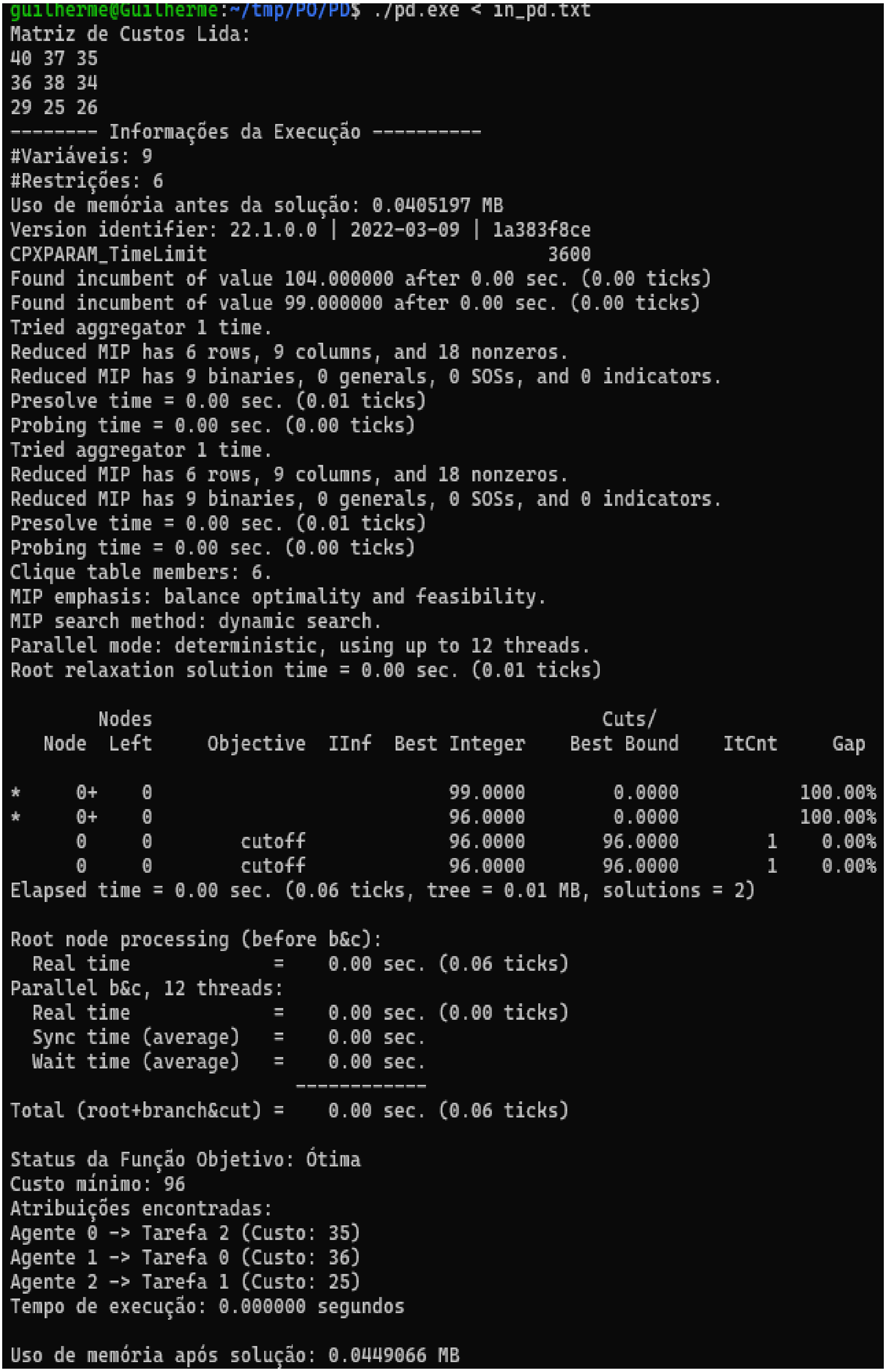
```
guilherme@Guilherme:~/tmp/PO/PT$ ./pt.exe < in_pt.txt

Grafo (arestas com custos e capacidades):
Aresta 0 -> 3: Custo = 14
Aresta 0 -> 4: Custo = 16
Aresta 0 -> 5: Custo = 13
Aresta 0 -> 6: Custo = 18
Aresta 1 -> 3: Custo = 8
Aresta 1 -> 4: Custo = 9
Aresta 1 -> 5: Custo = 10
Aresta 1 -> 6: Custo = 11
Aresta 2 -> 3: Custo = 18
Aresta 2 -> 4: Custo = 16
Aresta 2 -> 5: Custo = 21
Aresta 2 -> 6: Custo = 20

Nós de oferta: (0, 30) (1, 50) (2, 40)
Nós de demanda: (3, 20) (4, 28) (5, 25) (6, 34)

--------Informacoes da Execucao:----------

#Var: 12
#Restricoes: 2
Memory usage after variable creation:  0.0389175 MB
Memory usage after cplex(Model):  0.0440063 MB
Version identifier: 22.1.0.0 | 2022-03-09 | 1a383f8ce
CPXPARAM_TimeLimit                               3600
Found incumbent of value 1469.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 7 rows, 12 columns, and 24 nonzeros.
Reduced MIP has 0 binaries, 12 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 7 rows, 12 columns, and 24 nonzeros.
Reduced MIP has 0 binaries, 12 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.01 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         1469.0000        0.0000           100.00%
*     0     0      integral     0     1330.0000     1330.0000        4    0.00%
Elapsed time = 0.00 sec. (0.05 ticks, tree = 0.00 MB, solutions = 2)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.05 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.05 ticks)


Status da FO: Optimal
Variaveis de decisao:
x[0][3]: 5
x[0][4]: -0
x[0][5]: 25
x[0][6]: -0
x[1][3]: 15
x[1][4]: 1
x[1][5]: -0
x[1][6]: 34
x[2][3]: -0
x[2][4]: 27
x[2][5]: -0
x[2][6]: -0
Funcao Objetivo Valor = 1330
..(0.000000 seconds).

Memory usage before end:  0.0391388 MB
guilherme@Guilherme:~/tmp/PO/PT$ |
```



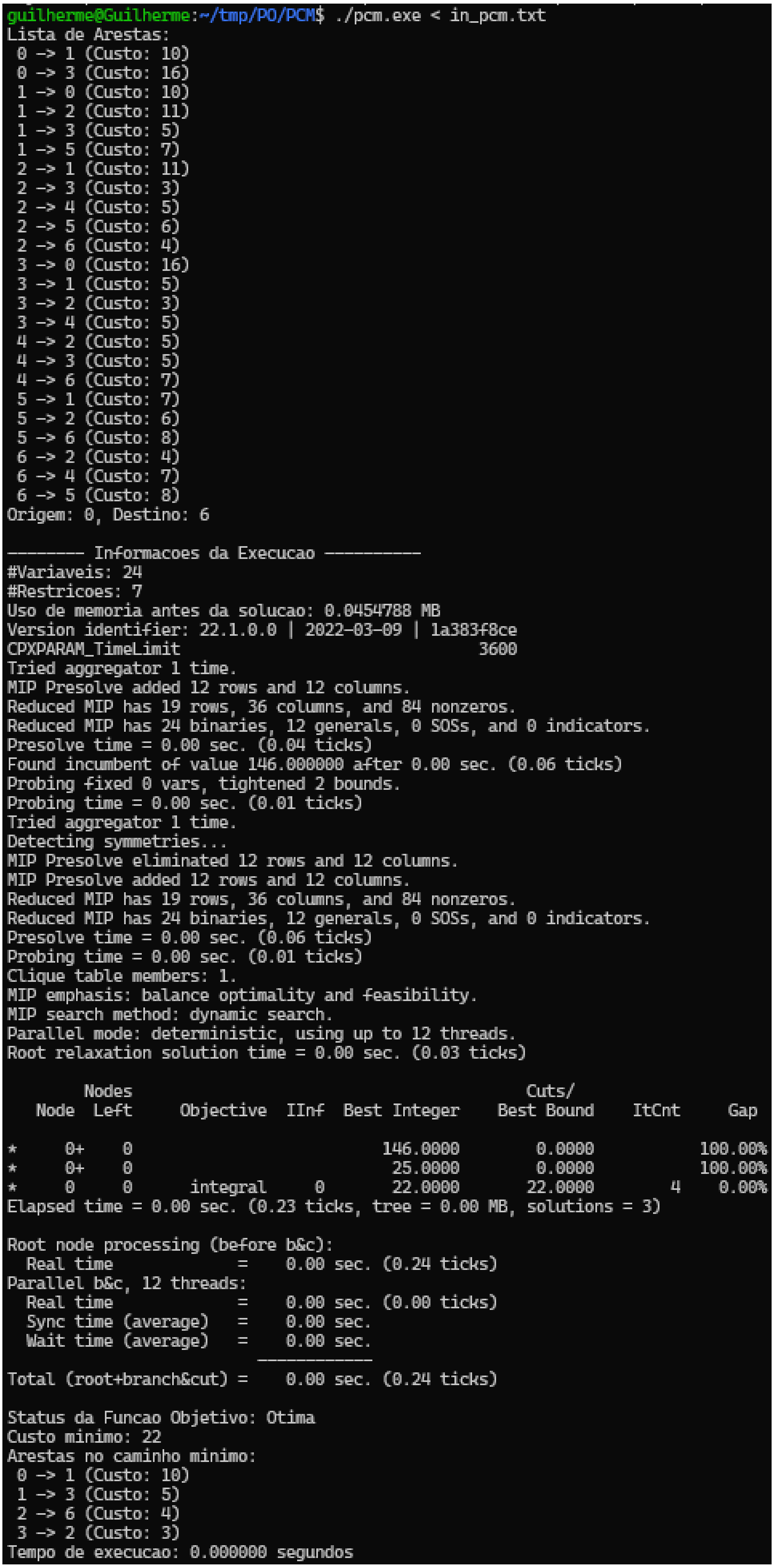
```
guilherme@Guilherme:~/tmp/PO/PFM$ ./pfm.exe < in_pfm.txt

Grafo (arestas com custos e capacidades):
Aresta 0 -> 1: Custo = 8
Aresta 0 -> 4: Custo = 18
Aresta 1 -> 2: Custo = 12
Aresta 1 -> 3: Custo = 4
Aresta 1 -> 4: Custo = 10
Aresta 2 -> 6: Custo = 20
Aresta 3 -> 2: Custo = 7
Aresta 3 -> 5: Custo = 14
Aresta 4 -> 3: Custo = 5
Aresta 4 -> 5: Custo = 10
Aresta 5 -> 6: Custo = 6
Origin: 0
Destination: 6

--------Informacoes da Execucao:----------

#Var: 11
#Restricoes: 2
Memory usage after variable creation:  0.0427551 MB
Memory usage after cplex(Model):  0.0483627 MB
Version identifier: 22.1.0.0 | 2022-03-09 | 1a383f8ce
CPXPARAM_TimeLimit                               3600
Found incumbent of value 0.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 2 times.
MIP Presolve eliminated 11 rows and 26 columns.
MIP Presolve added 1 rows and 1 columns.
Aggregator did 1 substitutions.
Reduced MIP has 5 rows, 11 columns, and 18 nonzeros.
Reduced MIP has 0 binaries, 11 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.03 ticks)
Tried aggregator 1 time.
Detecting symmetries...
MIP Presolve eliminated 1 rows and 1 columns.
MIP Presolve added 1 rows and 1 columns.
Reduced MIP has 5 rows, 11 columns, and 18 nonzeros.
Reduced MIP has 0 binaries, 11 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.01 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                            0.0000       19.0000           ---
*     0+    0                           14.0000       19.0000            35.71%
*     0     0      integral     0       19.0000       19.0000        1    0.00%
Elapsed time = 0.00 sec. (0.07 ticks, tree = 0.00 MB, solutions = 3)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.07 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.07 ticks)


Status da FO: Optimal
Variaveis de decisao:
x[0][1]: 8
x[0][4]: 11
x[1][2]: 8
x[1][3]: -0
x[1][4]: -0
x[2][6]: 13
x[3][2]: 5
x[3][5]: -0
x[4][3]: 5
x[4][5]: 6
x[5][6]: 6
Funcao Objetivo Valor = 19
..(0.000000 seconds).

Memory usage before end:  0.0429764 MB
guilherme@Guilherme:~/tmp/PO/PFM$ cd ..
```



```
guilherme@Guilherme:~/tmp/PO/PFCM$ ./pfcm.exe < in_pfcm.txt

Grafo (arestas com custos e capacidades):
Aresta 0 -> 3: Custo = 2, Capacidade = [0, 8]
Aresta 0 -> 5: Custo = 7, Capacidade = [0, 2147483647]
Aresta 1 -> 2: Custo = 2, Capacidade = [0, 2147483647]
Aresta 1 -> 3: Custo = 5, Capacidade = [0, 7]
Aresta 2 -> 3: Custo = 6, Capacidade = [0, 2147483647]
Aresta 2 -> 4: Custo = 5, Capacidade = [0, 9]
Aresta 2 -> 8: Custo = 4, Capacidade = [0, 6]
Aresta 3 -> 5: Custo = 1, Capacidade = [0, 17]
Aresta 3 -> 6: Custo = 3, Capacidade = [0, 4]
Aresta 3 -> 7: Custo = 4, Capacidade = [0, 2147483647]
Aresta 4 -> 8: Custo = 3, Capacidade = [0, 2147483647]
Aresta 5 -> 6: Custo = 1, Capacidade = [0, 10]
Aresta 6 -> 7: Custo = 1, Capacidade = [0, 2147483647]
Aresta 7 -> 4: Custo = 2, Capacidade = [0, 2147483647]

Nós de oferta: (0, 10) (1, 10) (2, 10)
Nós de demanda: (5, 8) (6, 7) (7, 6) (8, 9)
Nós de transferência: 3 4

--------Informacoes da Execucao:----------

#Var: 14
#Restricoes: 4
Memory usage after variable creation:  0.0461731 MB
Memory usage after cplex(Model):  0.0515366 MB
Version identifier: 22.1.0.0 | 2022-03-09 | 1a383f8ce
CPXPARAM_TimeLimit                               3600
Tried aggregator 2 times.
MIP Presolve eliminated 14 rows and 0 columns.
Aggregator did 1 substitutions.
Reduced MIP has 8 rows, 13 columns, and 26 nonzeros.
Reduced MIP has 0 binaries, 13 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.03 ticks)
Found incumbent of value 327.000000 after 0.00 sec. (0.07 ticks)
Tried aggregator 1 time.
Reduced MIP has 8 rows, 13 columns, and 26 nonzeros.
Reduced MIP has 0 binaries, 13 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.02 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                           327.0000        4.0000            98.78%
*     0     0      integral     0      184.0000      184.0000       11    0.00%
Elapsed time = 0.00 sec. (0.11 ticks, tree = 0.00 MB, solutions = 2)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.11 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.11 ticks)


Status da FO: Optimal
Variaveis de decisao:
x[0][3]: 8
x[0][5]: 2
x[1][2]: 3
x[1][3]: 7
x[2][3]: 4
x[2][4]: 3
x[2][8]: 6
x[3][5]: 16
x[3][6]: 3
x[3][7]: -0
x[4][8]: 3
x[5][6]: 10
x[6][7]: 6
x[7][4]: -0
Funcao Objetivo Valor = 184
..(0.000000 seconds).
```



```
guilherme@Guilherme:~/tmp/PO/PD$ ./pd.exe < in_pd.txt
Matriz de Custos Lida:
40 37 35
36 38 34
29 25 26
-------- Informações da Execução ----------
#Variáveis: 9
#Restrições: 6
Uso de memória antes da solução: 0.0405197 MB
Version identifier: 22.1.0.0 | 2022-03-09 | 1a383f8ce
CPXPARAM_TimeLimit                                     3600
Found incumbent of value 104.000000 after 0.00 sec. (0.00 ticks)
Found incumbent of value 99.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 6 rows, 9 columns, and 18 nonzeros.
Reduced MIP has 9 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 6 rows, 9 columns, and 18 nonzeros.
Reduced MIP has 9 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Clique table members: 6.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.01 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         99.0000        0.0000           100.00%
*     0+    0                         96.0000        0.0000           100.00%
      0     0        cutoff           96.0000       96.0000        1    0.00%
      0     0        cutoff           96.0000       96.0000        1    0.00%
Elapsed time = 0.00 sec. (0.06 ticks, tree = 0.01 MB, solutions = 2)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.06 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.06 ticks)

Status da Função Objetivo: Ótima
Custo mínimo: 96
Atribuições encontradas:
Agente 0 -> Tarefa 2 (Custo: 35)
Agente 1 -> Tarefa 0 (Custo: 36)
Agente 2 -> Tarefa 1 (Custo: 25)
Tempo de execução: 0.000000 segundos

Uso de memória após solução: 0.0449066 MB
```



```
guilherme@Guilherme:~/tmp/PO/PCM$ ./pcm.exe < in_pcm.txt
Lista de Arestas:
 0 -> 1 (Custo: 10)
 0 -> 3 (Custo: 16)
 1 -> 0 (Custo: 10)
 1 -> 2 (Custo: 11)
 1 -> 3 (Custo: 5)
 1 -> 5 (Custo: 7)
 2 -> 1 (Custo: 11)
 2 -> 3 (Custo: 3)
 2 -> 4 (Custo: 5)
 2 -> 5 (Custo: 6)
 2 -> 6 (Custo: 4)
 3 -> 0 (Custo: 16)
 3 -> 1 (Custo: 5)
 3 -> 2 (Custo: 3)
 3 -> 4 (Custo: 5)
 4 -> 2 (Custo: 5)
 4 -> 3 (Custo: 5)
 4 -> 6 (Custo: 7)
 5 -> 1 (Custo: 7)
 5 -> 2 (Custo: 6)
 5 -> 6 (Custo: 8)
 6 -> 2 (Custo: 4)
 6 -> 4 (Custo: 7)
 6 -> 5 (Custo: 8)
Origem: 0, Destino: 6

--------- Informacoes da Execucao ----------
#Variaveis: 24
#Restricoes: 7
Uso de memoria antes da solucao: 0.0454788 MB
Version identifier: 22.1.0.0 | 2022-03-09 | 1a383f8ce
CPXPARAM_TimeLimit                               3600
Tried aggregator 1 time.
MIP Presolve added 12 rows and 12 columns.
Reduced MIP has 19 rows, 36 columns, and 84 nonzeros.
Reduced MIP has 24 binaries, 12 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.04 ticks)
Found incumbent of value 146.000000 after 0.00 sec. (0.06 ticks)
Probing fixed 0 vars, tightened 2 bounds.
Probing time = 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Detecting symmetries...
MIP Presolve eliminated 12 rows and 12 columns.
MIP Presolve added 12 rows and 12 columns.
Reduced MIP has 19 rows, 36 columns, and 84 nonzeros.
Reduced MIP has 24 binaries, 12 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.06 ticks)
Probing time = 0.00 sec. (0.01 ticks)
Clique table members: 1.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.03 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         146.0000        0.0000           100.00%
*     0+    0                          25.0000        0.0000           100.00%
*     0     0      integral     0       22.0000       22.0000        4    0.00%
Elapsed time = 0.00 sec. (0.23 ticks, tree = 0.00 MB, solutions = 3)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.24 ticks)
Parallel b&c, 12 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.24 ticks)

Status da Funcao Objetivo: Otima
Custo minimo: 22
Arestas no caminho minimo:
 0 -> 1 (Custo: 10)
 1 -> 3 (Custo: 5)
 2 -> 6 (Custo: 4)
 3 -> 2 (Custo: 3)
Tempo de execucao: 0.000000 segundos
```