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;
```

```
52
53      /*
54          MODELAGEM
55      */
56
57      // Variáveis de decisão
58      IloArray<IloNumVarArray> x(env);
59      for(int i = 0; i < vertexCount; i++){
60          x.add(IloNumVarArray(env));
61          for(int j = 0; j < vertexCount; j++){
62              x[i].add(IloIntVar(env, 0, A[i][j].u));
63              if(A[i][j].u != 0) numberVar++;
64          }
65      }
66
67      IloModel model(env);
68
69      // somatórios para restrições e função objetivo
70      IloExpr sum1(env);
71      IloExpr sum2(env);
72
73      // Função Objetivo:
74      sum1.clear();
75      for(int i = 0; i < vertexCount; i++){
76          for(int j = 0; j < vertexCount; j++){
77              // se a aresta existe
78              if(A[i][j].u != 0){
79                  sum1 += (A[i][j].cost * x[i][j]);
80              }
81          }
82      }
83      model.add(IloMinimize(env, sum1)); // Minimização
84
85      // Restrições:
86
87      // Restrição de oferta
88      for(int i = 0; i < offerCount; i++){
89          // somatorio de tudo que sai
90          sum1.clear();
91          for(int j = 0; j < vertexCount; j++){
92              // se a aresta existe
93              if(A[S[i].id][j].u != 0){
94                  sum1 += x[S[i].id][j];
95              }
96          }
97
98          // somatorio de tudo que entra
99          sum2.clear();
100         for(int k = 0; k < vertexCount; k++){
101             // se a aresta existe
102             if(A[k][S[i].id].u != 0){
103                 sum2 += x[k][S[i].id];
104             }
105         }
```

```
106
107         // tudo que sai - tudo que entra <= oferta
108         model.add(sum1 - sum2 <= S[i].offer);
109     }
110     numberRes++;
111
112     // Restrição da demanda
113     for(int i = 0; i < demandCount; i++){
114         // somatorio de tudo que sai
115         sum1.clear();
116         for(int j = 0; j < vertexCount; j++){
117             // se a aresta existe
118             if(A[D[i].id][j].u != 0){
119                 sum1 += x[D[i].id][j];
120             }
121         }
122
123         // somatorio de tudo que entra
124         sum2.clear();
125         for(int k = 0; k < vertexCount; k++){
126             // se a aresta existe
127             if(A[k][D[i].id].u != 0){
128                 sum2 += x[k][D[i].id];
129             }
130         }
131
132         // tudo que sai - tudo que entra <= -Demanda
133         model.add(sum1 - sum2 <= -D[i].demand);
134     }
135     numberRes++;
136
137
138     // Restrição da conservação de fluxo
139     for(int i = 0; i < transferCount; i++){
140         // somatorio de tudo que sai
141         sum1.clear();
142         for(int j = 0; j < vertexCount; j++){
143             // se a aresta existe
144             if(A[T[i].id][j].u != 0){
145                 sum1 += x[T[i].id][j];
146             }
147         }
148
149         // somatorio de tudo que entra
150         sum2.clear();
151         for(int k = 0; k < vertexCount; k++){
152             // se a aresta existe
153             if(A[k][T[i].id].u != 0){
154                 sum2 += x[k][T[i].id];
155             }
156         }
157
158         // tudo que sai - tudo que entra <= -Demanda
159         model.add(sum1 - sum2 == 0);
```

```
160     }
161     numberRes++;
162 
163     // restrições de capacidade
164     for(int i = 0; i < vertexCount; i++){
165         for(int j = 0; j < vertexCount; j++){
166             if(A[i][j].u != 0){
167                 model.add(x[i][j] <= A[i][j].u);
168             }
169         }
170     }
171     numberRes++;
172 
173     //------ EXECUCAO do MODELO ----------
174     time_t timer, timer2;
175     IloNum value, objValue;
176     double runTime;
177     string status;
178 
179     //Informacoes -------------------------------------------
180     printf("--------Informacoes da Execucao:----------\n\n");
181     printf("#Var: %d\n", numberVar);
182     printf("#Restricoes: %d\n", numberRes);
183     cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. *
    1024.) << " MB" << endl;
184 
185     IloCplex cplex(model);
186     cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.)
    << " MB" << endl;
187 
188     cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);
189 
190     time(&timer);
191     cplex.solve();//COMANDO DE EXECUCAO
192     time(&timer2);
193 
194     bool sol = true;
195     switch(cplex.getStatus()){
196         case IloAlgorithm::Optimal:
197             status = "Optimal";
198             break;
199         case IloAlgorithm::Feasible:
200             status = "Feasible";
201             break;
202         default:
203             status = "No Solution";
204             sol = false;
205     }
206 
207     cout << endl << endl;
208     cout << "Status da FO: " << status << endl;
209 
210     if(sol){
211         objValue = cplex.getObjValue();
```

```
212         runTime = difftime(timer2, timer);
213 
214         cout << "Variaveis de decisao: " << endl;
215 
216         for(int i = 0; i < vertexCount; i++){
217             for(int j = 0; j < vertexCount; j++){
218                 // se existe aresta
219                 if(A[i][j].u != 0){
220                     value = IloRound(cplex.getValue(x[i][j]));
221                     printf("x[%d][%d]: %.0lf\n", i, j, value);
222                 }
223             }
224         }
225 
226         cout << "Funcao Objetivo Valor = " << objValue << endl;
227         printf("..(%.6lf seconds).\n\n", runTime);
228 
229     }else{
230         printf("No Solution!\n");
231     }
232 
233     //Free Memory
234     cplex.end();
235     sum1.end();
236     sum2.end();
237 
238     cout << "Memory usage before end:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB"
    << endl;
239     env.end();
240 }
241 
242 int main(){
243     // Leitura dos dados
244     cin >> vertexCount >> edgeCount;
245     for (int i = 0; i < edgeCount; i++) {
246         int x, y, c, limL, limU;
247         cin >> x >> y >> c >> limL >> limU;
248         A[x][y] = {c, limL, limU};
249     }
250 
251     cin >> offerCount;
252     for (int i = 0; i < offerCount; i++)
253         cin >> S[i].id >> S[i].offer;
254 
255     cin >> demandCount;
256     for (int i = 0; i < demandCount; i++)
257         cin >> D[i].id >> D[i].demand;
258 
259     cin >> transferCount;
260     for (int i = 0; i < transferCount; i++)
261         cin >> T[i].id;
262 
263     // Impressão dos dados de entrada formatados
264     cout << "\nGrafo (arestas com custos e capacidades):\n";
```

```
265     for (int i = 0; i < vertexCount; i++) {
266         for (int j = 0; j < vertexCount; j++) {
267             if (A[i][j].u > 0) {
268                 cout << "Aresta " << i << " -> " << j << ": Custo = " << A[i][j].cost << ",
    Capacidade = [" << A[i][j].l << ", " << A[i][j].u << "]\n";
269             }
270         }
271     }
272 
273     cout << "\nNós de oferta:";
274     for (int i = 0; i < offerCount; i++)
275         cout << " (" << S[i].id << ", " << S[i].offer << ")";
276 
277     cout << "\nNós de demanda:";
278     for (int i = 0; i < demandCount; i++)
279         cout << " (" << D[i].id << ", " << D[i].demand << ")";
280 
281     cout << "\nNós de transferência:";
282     for (int i = 0; i < transferCount; i++)
283         cout << " " << T[i].id;
284     cout << "\n\n";
285 
286     cplex();
287     return 0;
288 
289     return 0;
290 }
291 
292 
293 
```

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
    */

```

```
52      // Variáveis de decisão
53      IloArray<IloNumVarArray> x(env);
54      for(int i = 0; i < vertexCount; i++){
55          x.add(IloNumVarArray(env));
56          for(int j = 0; j < vertexCount; j++){
57              if(A[i][j].cost == -1){
58                  x[i].add(IloIntVar(env, 0, 0));
59              }else{
60                  x[i].add(IloIntVar(env, 0, INT_MAX));
61                  numberVar++;
62              }
63          }
64      }
65
66      IloModel model(env);
67
68      // somatórios para restrições e função objetivo
69      IloExpr sum1(env);
70
71      // Função Objetivo:
72      sum1.clear();
73      for(int i = 0; i < vertexCount; i++){
74          for(int j = 0; j < vertexCount; j++){
75              // se a aresta existe
76              if(A[i][j].cost != -1){
77                  sum1 += (A[i][j].cost * x[i][j]);
78              }
79          }
80      }
81      model.add(IloMinimize(env, sum1)); // Minimização
82
83      // Restrições:
84
85      // Restrição de oferta
86      for(int i = 0; i < offerCount; i++){
87          // somatorio de tudo que sai
88          sum1.clear();
89          for(int j = 0; j < vertexCount; j++){
90              // se a aresta existe
91              if(A[S[i].id][j].cost != -1){
92                  sum1 += x[S[i].id][j];
93              }
94          }
95
96          // tudo que sai <= oferta
97          model.add(sum1 <= S[i].offer);
98      }
99      numberRes++;
100
101     // Restrição da demanda
102     for(int i = 0; i < demandCount; i++){
103         // somatorio de tudo que sai
104         sum1.clear();
105         for(int j = 0; j < vertexCount; j++){
```

```
106             // se a aresta existe
107             if(A[j][D[i].id].cost != -1){
108                 sum1 += x[j][D[i].id];
109             }
110         }
111 
112         // tudo que sai == Demanda
113         model.add(sum1 == D[i].demand);
114     }
115     numberRes++;
116 
117     //------ EXECUCAO do MODELO ----------
118     time_t timer, timer2;
119     IloNum value, objValue;
120     double runTime;
121     string status;
122 
123     //Informacoes ------------------------------------------
124     printf("--------Informacoes da Execucao:----------\n\n");
125     printf("#Var: %d\n", numberVar);
126     printf("#Restricoes: %d\n", numberRes);
127     cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. *
    1024.) << " MB" << endl;
128 
129     IloCplex cplex(model);
130     cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.)
    << " MB" << endl;
131 
132     cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);
133 
134     time(&timer);
135     cplex.solve();//COMANDO DE EXECUCAO
136     time(&timer2);
137 
138     bool sol = true;
139     switch(cplex.getStatus()){
140         case IloAlgorithm::Optimal:
141             status = "Optimal";
142             break;
143         case IloAlgorithm::Feasible:
144             status = "Feasible";
145             break;
146         default:
147             status = "No Solution";
148             sol = false;
149     }
150 
151     cout << endl << endl;
152     cout << "Status da FO: " << status << endl;
153 
154     if(sol){
155         objValue = cplex.getObjValue();
156         runTime = difftime(timer2, timer);
157 
```

```
158         cout << "Variaveis de decisao: " << endl;
159 
160         for(int i = 0; i < vertexCount; i++){
161             for(int j = 0; j < vertexCount; j++){
162                 // se existe aresta
163                 if(A[i][j].cost != -1){
164                     value = IloRound(cplex.getValue(x[i][j]));
165                     printf("x[%d][%d]: %.0lf\n", i, j, value);
166                 }
167             }
168         }
169 
170         cout << "Funcao Objetivo Valor = " << objValue << endl;
171         printf("..(%.6lf seconds).\n\n", runTime);
172 
173     }else{
174         printf("No Solution!\n");
175     }
176 
177     //Free Memory
178     cplex.end();
179     sum1.end();
180 
181     cout << "Memory usage before end:  " << env.getMemoryUsage() / (1024. * 1024.) << " MB"
    << endl;
182     env.end();
183 }
184 
185 int main(){
186     // Leitura dos dados
187     cin >> vertexCount >> edgeCount;
188     for (int i = 0; i < edgeCount; i++) {
189         int x, y, c;
190         cin >> x >> y >> c;
191         A[x][y].cost = c;
192     }
193 
194     cin >> offerCount;
195     for (int i = 0; i < offerCount; i++)
196         cin >> S[i].id >> S[i].offer;
197 
198     cin >> demandCount;
199     for (int i = 0; i < demandCount; i++)
200         cin >> D[i].id >> D[i].demand;
201 
202     // Impressão dos dados de entrada formatados
203     cout << "\nGrafo (arestas com custos e capacidades):\n";
204     for (int i = 0; i < vertexCount; i++) {
205         for (int j = 0; j < vertexCount; j++) {
206             if (A[i][j].cost != -1) {
207                 cout << "Aresta " << i << " -> " << j << ": Custo = " << A[i][j].cost <<
    "\n";
208             }
209         }
```

```
210     }
211
212     cout << "\nNós de oferta:";
213     for (int i = 0; i < offerCount; i++)
214         cout << " (" << S[i].id << ", " << S[i].offer << ")";
215
216     cout << "\nNós de demanda:";
217     for (int i = 0; i < demandCount; i++)
218         cout << " (" << D[i].id << ", " << D[i].demand << ")";
219     cout << "\n\n";
220
221     cplex();
222     return 0;
223
224     return 0;
225 }
226
227
228
```

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

```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";
```

```
52      cout << "#Variaveis: " << M << "\n";
53      cout << "#Restricoes: " << N << "\n";
54      cout << "Uso de memoria antes da solucao: " << env.getMemoryUsage() / (1024. * 1024.) <<
" MB\n";
55
56      cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);
57
58      time_t timer, timer2;
59      time(&timer);
60      cplex.solve();
61      time(&timer2);
62
63      bool sol = true;
64      string status;
65      switch (cplex.getStatus()) {
66          case IloAlgorithm::Optimal:
67              status = "Optimal";
68              break;
69          case IloAlgorithm::Feasible:
70              status = "Feasible";
71              break;
72          default:
73              status = "No Solution";
74              sol = false;
75      }
76
77      cout << "\nStatus da Funcao Objetivo: " << status << "\n";
78      if (sol) {
79          cout << "Custo minimo: " << cplex.getObjValue() << "\n";
80          cout << "Arestas no caminho minimo:\n";
81          for (int i = 0; i < M; i++) {
82              if (cplex.getValue(x[i]) > 0.5)
83                  cout << " " << arestas[i].origem << " -> " << arestas[i].destino << "
(Custo: " << arestas[i].custo << ")\n";
84          }
85          printf("Tempo de execucao: %.6lf segundos\n\n", difftime(timer2, timer));
86      } else {
87          cout << "Erro: O problema eh inviavel! Verifique se ha um caminho possivel entre
origem e destino.\n";
88      }
89
90      cout << "Uso de memoria apos solucao: " << env.getMemoryUsage() / (1024. * 1024.) << "
MB\n";
91
92      obj.end();
93      cplex.end();
94      model.end();
95      env.end();
96  }
97
98  int main() {
99      cin >> N >> M;
100     arestas.resize(M);
101     for (int i = 0; i < M; i++) {
```

```
102         cin >> arestas[i].origem >> arestas[i].destino >> arestas[i].custo;
103     }
104     cin >> origemFonte >> destinoDestino;
105 
106     cout << "Lista de Arestas:\n";
107     for (const auto& e : arestas) {
108         cout << " " << e.origem << " -> " << e.destino << " (Custo: " << e.custo << ")\n";
109     }
110     cout << "Origem: " << origemFonte << ", Destino: " << destinoDestino << "\n\n";
111 
112     cplex();
113     return 0;
114 }
115 
```

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);
```

```
106             }
107         }
108     }
109     numberRes++;
110 
111     //------ EXECUCAO do MODELO ----------
112     time_t timer, timer2;
113     IloNum value, objValue;
114     double runTime;
115     string status;
116 
117     //Informacoes ------------------------------------------
118     printf("--------Informacoes da Execucao:----------\n\n");
119     printf("#Var: %d\n", numberVar);
120     printf("#Restricoes: %d\n", numberRes);
121     cout << "Memory usage after variable creation:  " << env.getMemoryUsage() / (1024. *
    1024.) << " MB" << endl;
122 
123     IloCplex cplex(model);
124     cout << "Memory usage after cplex(Model):  " << env.getMemoryUsage() / (1024. * 1024.)
    << " MB" << endl;
125 
126     cplex.setParam(IloCplex::TiLim, CPLEX_TIME_LIM);
127 
128     time(&timer);
129     cplex.solve();//COMANDO DE EXECUCAO
130     time(&timer2);
131 
132     bool sol = true;
133     switch(cplex.getStatus()){
134         case IloAlgorithm::Optimal:
135             status = "Optimal";
136             break;
137         case IloAlgorithm::Feasible:
138             status = "Feasible";
139             break;
140         default:
141             status = "No Solution";
142             sol = false;
143     }
144 
145     cout << endl << endl;
146     cout << "Status da FO: " << status << endl;
147 
148     if(sol){
149         objValue = cplex.getObjValue();
150         runTime = difftime(timer2, timer);
151 
152         cout << "Variaveis de decisao: " << endl;
153 
154         for(int i = 0; i < vertexCount; i++){
155             for(int j = 0; j < vertexCount; j++){
156                 // se existe aresta
157                 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
```