```c
/* Ciclo Hamiltoniano e Caixeiro Viajante */

#include <stdio.h>
#include <stdlib.h>

#define false 0
#define true 1

#define INVALIDO -1

typedef int bool;

typedef struct {
  int** dists; /* matriz de distancia entre cidades */
  int cidades; /* numero de cidades */
} MAPA;

bool* cidadesVisitadas;
int* caminhoAtual;
int* melhorCaminho;
int custoAtual;
int melhorCusto;

/* Cria um mapa com n cidades */
MAPA novoMapa(int n) {
  int i,j;
  MAPA m;
  m.cidades = n;
  m.dists = (int**)malloc(sizeof(int*)*n);
  for (i=0;i<n;i++){
    m.dists[i] = (int*)malloc(sizeof(int)*n);
    for (j=0;j<n;j++) m.dists[i][j] = INVALIDO;
  }
  return m;
}

/* Exibe um ciclo formado por n cidades */
void exibirCiclo(int* caminho, int n){
  int i;
  printf("Ciclo [%i]:", n);
  for (i=0;i<n;i++) printf(" %i",caminho[i]);
  printf("\n");

}
```

```
/* Funcao recursiva para resolver o problema do Ciclo
Hamiltoniano */
bool cicloHamiltonianoAux(MAPA m, int numVisitadas){
  int i, atual;
  if (numVisitadas == m.cidades){
    if (m.dists[caminhoAtual[m.cidades-1]][caminhoAtual[0]] !=
INVALIDO)
      return true;
    else return false;
  }
  atual = caminhoAtual[numVisitadas-1];
  for (i=0;i<m.cidades;i++){
    if (!cidadesVisitadas[i] && m.dists[atual][i] != INVALIDO){
      cidadesVisitadas[i] = true;
      caminhoAtual[numVisitadas] = i;
      if (cicloHamiltonianoAux(m, numVisitadas+1)) return true;
      caminhoAtual[numVisitadas] = -1;
      cidadesVisitadas[i] = false;
    }
  }
  return false;
}

/* Funcao inicial para a resolucao do problema do Ciclo
Hamiltoniano */
bool cicloHamiltoniano(MAPA m){
  int i;
  if (m.cidades<1) return false;
  cidadesVisitadas = (bool*)malloc(sizeof(bool)*m.cidades);
  caminhoAtual = (int*)malloc(sizeof(int)*m.cidades);
  for(i=0;i<m.cidades;i++) cidadesVisitadas[i] = false;
  cidadesVisitadas[0] = 1;
  caminhoAtual[0] = 0;
  if (cicloHamiltonianoAux(m,1)){
    printf("Ciclo Hamiltoniano encontrado!\n");
    exibirCiclo(caminhoAtual, m.cidades);
  }
  free(cidadesVisitadas);
  free(caminhoAtual);
}
```

```
/* Funcao recursiva para a resolucao do problema do Caixeiro
Viajante */
void caixeiroViajanteAux(MAPA m, int numVisitadas){
  int i, atual, custo;
  if (custoAtual>=melhorCusto) return;
  if (numVisitadas == m.cidades){
    custo = m.dists[caminhoAtual[m.cidades-1]][caminhoAtual[0]];
    if (custo != INVALIDO){
      if (custoAtual+custo < melhorCusto){
        melhorCusto = custoAtual+custo;
        for(i=0;i<m.cidades;i++) melhorCaminho[i] =
caminhoAtual[i];
      }
    }
    return;
  }
  atual = caminhoAtual[numVisitadas-1];
  for (i=0;i<m.cidades;i++){
    if (!cidadesVisitadas[i] && m.dists[atual][i] != INVALIDO){
      cidadesVisitadas[i] = true;
      custoAtual += m.dists[atual][i];
      caminhoAtual[numVisitadas] = i;
      caixeiroViajanteAux(m, numVisitadas+1);
      caminhoAtual[numVisitadas] = -1;
      custoAtual -= m.dists[atual][i];
      cidadesVisitadas[i] = false;
    }
  }
}
```

```
/* Funcao inicial para a resolucao do problema do Caixeiro
Viajante */
bool caixeiroViajante(MAPA m){
  int i;
  if (m.cidades<1) return false;
  cidadesVisitadas = (bool*)malloc(sizeof(bool)*m.cidades);
  caminhoAtual = (int*)malloc(sizeof(int)*m.cidades);
  melhorCaminho = (int*)malloc(sizeof(int)*m.cidades);
  for(i=0;i<m.cidades;i++) cidadesVisitadas[i] = false;
  cidadesVisitadas[0] = true;
  caminhoAtual[0] = 0;
  custoAtual = 0;
  melhorCusto = 999999999;
  caixeiroViajanteAux(m,1);
  if (melhorCusto>0){
    printf("Custo do Problema do Caixeiro Viajante: %i\
n",melhorCusto);
    exibirCiclo(melhorCaminho, m.cidades);
  }
  free(cidadesVisitadas);
  free(caminhoAtual);
  free(melhorCaminho);
}
```

```c
int main(){
  MAPA mapa1 = novoMapa(5);
  /* Sao Paulo, Guarulhos, Jundiai, Sorocaba e Campinas */
  mapa1.dists[0][1] = 43;
  mapa1.dists[0][2] = 56;
  mapa1.dists[0][3] = 102;
  mapa1.dists[0][4] = INVALIDO;
  mapa1.dists[1][0] = 43;
  mapa1.dists[2][0] = 56;
  mapa1.dists[3][0] = 102;
  mapa1.dists[4][0] = INVALIDO;

  mapa1.dists[1][2] = 130;
  mapa1.dists[1][3] = INVALIDO;
  mapa1.dists[1][4] = INVALIDO;
  mapa1.dists[2][1] = 112;
  mapa1.dists[3][1] = INVALIDO;
  mapa1.dists[4][1] = INVALIDO;

  mapa1.dists[2][3] = 93;
  mapa1.dists[2][4] = 38;
  mapa1.dists[3][2] = 93;
  mapa1.dists[4][2] = 38;

  mapa1.dists[3][4] = 89;
  mapa1.dists[4][3] = 89;

  printf("Verificando a existencia de um Ciclo Hamiltoniano\n");
  cicloHamiltoniano(mapa1);

  printf("\nProcurando pelo ciclo de menor custo.\n");
  caixeiroViajante(mapa1);

  return 0;
}


/* SAIDA

Verificando a existencia de um Ciclo Hamiltoniano
Ciclo Hamiltoniano encontrado!
Ciclo [5]: 0 1 2 4 3

Procurando pelo ciclo de menor custo.
Custo do Problema do Caixeiro Viajante: 384
Ciclo [5]: 0 3 4 2 1

*/
```