旅行商问题实验报告

一、问题分析

- 处理对象：将每个城市看作一个点，它们之间的道路看作做图的一条边，则处理对象可看作一幅简单无向图，以邻接矩阵表示
- 实现功能：需计算求出从 0 号顶点（城市）经过所有顶点后回到 0 号顶点的最小成本和最短路径。
- 结果显示：输出两行。一行为最小成本，一行为该成本的最短路径（如果有多条，则优先输出序号小的点），如果不存在最优方案，输出-1 。
- 输入样例及求解过程：

4 6

0 1 30

0 2 6

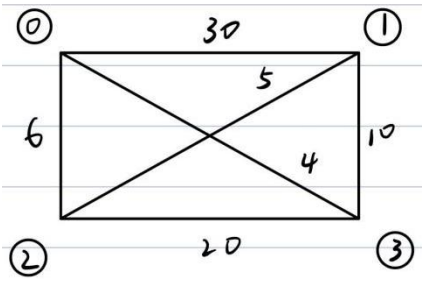
0 3 4

1 2 5

1 3 10

2 3 20

构成如下图：



从顶点 0 出发，则按 0->2->1->3->0 路径可得到最小成本为 6 + 5 + 10 + 4 = 25

输出：

25

0 2 1 3 0

二、数据结构和算法设计

1.抽象数据类型设计：

即抽象类 Graph，其中封装了无向图结构的多种函数接口，如无向图的构造、析构、初始化、获取顶点/边数、获取某个顶点在邻接矩阵中的第一个相邻顶点、 获取某一顶点的访问情况以及获取某一条边权重等对图进行基本操作的函数，具体设计如下:

```
// Graph abstract class. This ADT assumes that the number
// of vertices is fixed when the graph is created.
class Graph {
private:
    void operator = (const Graph&) {}   // Protect assignment
    Graph (const Graph&) {}     // Protect copy constructor

public:
    Graph () {}            // Default constructor
    virtual ~Graph () {}   // Base destructor

    // Initialize a graph of n vertices
    virtual void Init (int n) = 0;

    // Return: the number of vertices and edges
    virtual int n() = 0;
    virtual int e() = 0;

    // Return v's first neighbor
    virtual int first (int v) = 0;

    // Return v's next neighbor
    virtual int next (int v, int w) = 0;

    // Set the weight for an edge
    // i, j: The vertices
    // wgt: Edge weight
    virtual void setEdge (int v1, int v2, int wght) = 0;
```

```
    // Delete an edge
    // i, j: The vertices
    virtual void delEdge(int v1,int v2) = 0;

    // Determine if an edge is in the graph
    // i, j: The vertices
    // Return: true if edge i, j has non-zero weight
    virtual bool isEdge (int i, int j) = 0;

    // Return an edge's weight
    // i, j: The vertices
    // Return: The weight of edge i,j, or zero
    virtual int weight (int v1, int v2) =0;

    // Get and Set the mark value for a vertex
    // v: The vertex
    // val: The value to set
    virtual int getMark (int v) =0;
    virtual void setMark (int v, int val) =0;
};
```

2.物理数据类型设计：

用邻接矩阵表示无向图，Graphm 类公有继承 Graph 具体实现 Graph 中接口内部的函数，实现实例化，且设置私有成员数据 numVertex(顶点数)，numEdge(边数)，二维数组 matrix(表示邻接矩阵)，一维数组 mark(记录每个顶点访问情况)，数据类型均为 int 型。

3.算法思想设计：

1. 定义全局二维 vector 容器 halmiton_cycle 储存图中哈密尔顿回路，以及 vector 容器 path 储存当前判断路径，储存变量类型为 int，根据输入的数据，构造无向图 map，调用设计好的 Hamilton 函数，得到图中全部哈密尔顿回路。

2. Hamilton 函数采用回溯法的算法思想：通过回溯剪枝得到解空间树，约束条件为两点之间存在边且将要经过的点没被访问，若不满足则此路径不是解，回溯到上一节点，满足则继

续前进判断是否满足约束条件，当 step==n 时，判断能不能回到 0 顶点，若能则是一条哈密尔顿回路，将每次得到哈密尔顿回路存入 hamilton_path。

3. 遍历得到的所有哈密尔顿回路，找到成本最低路径，输出该最低成本及对应路径，如果不存在哈密尔顿回路，则输出-1 。

4. 关键功能和算法步骤：

1. 回溯法寻找哈密尔顿回路：

①将 0 号顶点作为搜索树的根结点，将其加入 path，并标记为已访问，记录已走过顶点数 step，采用深度优先遍历的方法，访问 0 的下一个邻接顶点 w（按邻接矩阵的顺序访问，即可得到优先输出序号小的路径），满足约束条件则，将其作为下一个扩展节点，当前位置 site = w，step+1 继续下一层判断，否则将对应顶点所在子树剪枝，回溯到根结点访问另一邻接顶点。

②继续访问当前位置 site 的邻接顶点 w，若满足约束条件 getMark(w) == 1，则将 w 加入 path，标记为已访问，更新 site = w， step+1，不满足约束条件则剪枝回溯，回到活节点深度优先搜索另一颗子树。

③当递归到最后一层，即 step == n 时，判断最后一个顶点能否回到顶点 0，即

```
weight(path[step - 1], 0) == 1
```

如果满足，则是一条哈密尔顿回路，将这条 path 存入 hamilton_path 中。再按深度优先搜索递归遍历解空间，找出所有哈密尔顿回路。

2. 判断是否存在哈密尔顿回路：

获取 hamilton_path 的容量大小，为 0 则不存在回路，输出-1，存在则遍历比较所有哈密尔顿回路的成本,得到最低成本的路径,存入 best_path 中,输出最低成本 Min 及 best_path。

三、算法性能分析

1. 对于完全图，回溯算法需要枚举所有可能路径，而完全图的可能路径为 n！，因此时间复杂度为 O（n！），对于大规模的问题，回溯法计算时间较长，但有点儿在于容易理解、实现，另外可以采用高效的剪枝函数优化算法。

2. 算法用到二维向量 hamilton_path 储存哈密尔顿回路，若有 c 条哈密尔顿回路，则空间复杂度为 O（cn）。

3. 本题测试用例的规模较小，使用回溯算法比较容易实现，本次实验我只采用了约束函数进行回溯，可以使用限界函数对搜索树剪枝优化算法。