算法第六次作业

李雨轩

计算机2205

2204112913

修改解旅行售货员问题

1. 题目描述

试修改解旅行售货员问题的分支限界法，使得 $s = n - 2$ 的节点不插入优先队列，而是将当前最优排列存储于bestp中。经这样修改后，算法在下一个扩展结点满足条件Lcost≥bestc时结束。

2. 算法设计

1. **初始化**：
   - 创建一个优先队列 Q 来存储活动节点，按 $lcost$ 的值进行排序。
   - 定义一个全局变量 bestc，初始化为无穷大，用来存储当前发现的最低成本。
   - 定义一个数组 bestp 用来存储当前最优路径。
2. **初始节点设置**：
   - 创建一个初始节点，包括起始城市、当前成本 $cc = 0$、已访问城市列表等。
   - 计算初始节点的 $lcost$ 并将其放入优先队列。
3. **节点扩展**：
   - 当优先队列不为空时，从中取出 $lcost$ 最小的节点。
   - 如果当前节点的 $s = n - 2$，则直接计算通过包括最后一个城市和返回起始城市的完整路径成本：
     - 如果这个成本小于 bestc，更新 bestc 和 bestp。
     - 继续从优先队列取节点进行处理。
   - 否则，对每个未访问的城市生成新的节点：
     - 计算新节点的 cc 和 $lcost$。
     - 如果 $lcost < bestc$，则将新节点加入优先队列。

- 检查取出的节点的 $lcost$ 是否大于等于 bestc，如果是，则算法终止。

4. **结束**：
   - 当优先队列为空或找到了满足条件的解，算法结束。
   - 输出最优路径 bestp 和成本 bestc。

3. 算法说明

修改了算法以避免将 $s = n - 2$ 的节点插入优先队列，并改为更新已知最佳路径（$bestp$）和成本（$bestc$）：

1. **在 $s = n - 2$ 时直接完成**：当节点达到 $s = n - 2$ 时，意味着只剩下一个城市未完成路径。通过添加唯一未访问的城市，然后返回起始城市，直接计算完整路径的成本。如果这个计算的路径成本小于 $bestc$，则更新 $bestc$ 和 $bestp$。
2. **提前终止**：如果在任何时候从优先队列中提取的节点的 $lcost$ 大于或等于 $bestc$，则终止过程。这使用最佳已知解决方案作为界限，以消除不可能比已经找到的解更好的路径。
3. **优先队列效率**：通过不将 $s = n - 2$ 的节点添加到优先队列中，可以减少队列的大小和管理它的开销。这导致计算量减少，可能更快地收敛到最优解。

4. 源代码

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <algorithm>

using namespace std;

struct Node
{
  int level;
  vector<int> path;
  int reducedCost;
  int cost;
};

struct CompareNode
{
  bool operator()(Node const &n1, Node const &n2)
  {
    return n1.cost > n2.cost;
  }
};

int firstMin(vector<vector<int>> &cost, int i)
{
  int min = INT_MAX;
  for (int k = 0; k < cost.size(); k++)
```

```
    if (cost[i][k] < min && i != k)
      min = cost[i][k];
  return min;
}

void solveTSP(vector<vector<int>> &cost)
{
  priority_queue<Node, vector<Node>, CompareNode> pq;
  vector<int> bestp;
  int bestc = INT_MAX;

  vector<int> initialPath;
  initialPath.push_back(0);

  Node root = {0, initialPath, 0, 0};
  root.cost = firstMin(cost, 0);

  pq.push(root);

  while (!pq.empty())
  {
    Node min = pq.top();
    pq.pop();

    int last = min.path[min.path.size() - 1];

    if (min.level == cost.size() - 2)
    {
      for (int i = 0; i < cost.size(); i++)
      {
        if (find(min.path.begin(), min.path.end(), i) == min.path.end())
        {
          int currentCost = min.reducedCost + cost[last][i] + cost[i][0];
          if (currentCost < bestc)
          {
            bestc = currentCost;
            bestp = min.path;
            bestp.push_back(i);
            bestp.push_back(0);
          }
          break;
        }
      }
    }
    else
    {

      for (int i = 0; i < cost.size(); i++)
      {
        if (find(min.path.begin(), min.path.end(), i) == min.path.end())
        {
          Node child = {min.level + 1, vector<int>(min.path), min.reducedCost + cost[last]
[i], 0};
          child.path.push_back(i);
          child.cost = child.reducedCost + firstMin(cost, i);
          if (child.cost < bestc)
          {
```

```
            pq.push(child);
          }
        }
      }
    }
  }

  cout << "Best Cost: " << bestc << endl;
  cout << "Best Path: ";
  for (int i : bestp)
  {
    cout << i << " ";
  }
  cout << endl;
}

int main()
{
  vector<vector<int>> cost = {
      {INT_MAX, 30, 6, 4},
      {30, INT_MAX, 5, 10},
      {6, 5, INT_MAX, 20},
      {4, 10, 20, INT_MAX}};

  solveTSP(cost);
  return 0;
}
```