LintCode领扣题解 (/problem) / 旅行商问题 · Traveling Salesman Problem

旅行商问题 · Traveling Salesman Problem

中文

Undirected Graph (/problem/?tags=undirected-graph)

描述

给 `n` 个城市(从 `1` 到 `n`)，城市和无向道路 成本 之间的关系为3元组 `[A, B, C]` （在城市 `A` 和城市 `B` 之间有一条路，成本是 `C`）我们需要从1开始找到的旅行所有城市的付出最小的成本。

> 1. 一个城市只能通过一次。 2. 你可以假设你可以到达`所有`的城市。

在线评测地址: https://www.lintcode.com/problem/traveling-salesman-problem/ (https://www.lintcode.com/problem/traveling-salesman-problem/)

收起题目描述 ^

语言类型 ALL (21) java (10) python (8) cpp (3) 上传题解

九章用户JL5KVX

更新于 6/9/2020, 7:03:50 AM

DFS方法，每次枚举下一个没有去过的城市

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
// 九章算法求职训练营版本
public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */

    int n;
    int[][] g;
    boolean[] done;
    int res = Integer.MAX_VALUE;

    // dfs's parameters include level and current status
    // current city p
    // current cost c
    void dfs(int level, int p, int c) {
        if (c >= res) {
            // there's no reason to further recurse
            // because the cost can only become even bigger
            // so it cannot update res
            return;
        }

        if (level == n) {
            res = c;
            return;
        }

        int i;
        // next city i, from p
        // p-->i must have a road
        for (i = 0; i < n; ++i) {
            if (!done[i] && g[p][i] != Integer.MAX_VALUE) {
                done[i] = true;
                dfs(level + 1, i, c + g[p][i]);
                done[i] = false;
            }
        }
    }

    public int minCost(int nn, int[][] costs) {
        n = nn;
        int i, j, x, y;
        done = new boolean[n];
        for (i = 0; i < n; ++i) {
            done[i] = false;
        }

        g = new int[n][n];
        for (i = 0; i < n; ++i) {
            for (j = 0; j < n; ++j) {
                g[i][j] = Integer.MAX_VALUE; // no road between city i and j
            }
        }

        for (i = 0; i < costs.length; ++i) {
            x = costs[i][0] - 1;
            y = costs[i][1] - 1;
            g[x][y] = Math.min(g[x][y], costs[i][2]);
            g[y][x] = Math.min(g[y][x], costs[i][2]);
        }

        done[0] = true; // important
        dfs(1, 0, 0);
        return res;
    }
}
```

获赞 2   1 条评论

九章-加贺
更新于 11/9/2020, 7:36:18 PM

如果要保证正确性的最优做法。状态压缩动态规划。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ /
Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int minCost(int n, int[][] roads) {
        // Write your code here
        int inf = 1000000000;
        int[][] graph = constructGraph(roads, n);
        int stateSize = 1 << n;
        int[][] f = new int[stateSize][n + 1];
        for (int i = 0; i < stateSize; i++) {
            for (int j = 0; j < n + 1; j++) {
                f[i][j] = inf;
            }
        }
        f[1][1] = 0;

        for (int state = 0; state < stateSize; state++) {
            for (int i = 2; i < n + 1; i++) {
                if ((state & (1 << (i - 1))) == 0) {
                    continue;
                }
                int prevState = state ^ (1 << (i - 1));
                for (int j = 1; j < n + 1; j++) {
                    if ((prevState & (1 << (j - 1))) == 0) {
                        continue;
                    }
                    f[state][i] = Math.min(f[state][i], f[prevState][j] + graph[j][i]);
                }
            }
        }

        int minimalCost = inf;
        for (int i = 0; i < n + 1; i++) {
            minimalCost = Math.min(minimalCost, f[stateSize - 1][i]);
        }

        return minimalCost;
    }

    int[][] constructGraph(int[][] roads, int n) {
        int[][] graph = new int[n + 1][n + 1];
        int inf = 1000000000;
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < n + 1; j++) {
                graph[i][j] = inf;
            }
        }
        int roadsLength = roads.length;
        for (int i = 0; i < roadsLength; i++) {
            int a = roads[i][0], b = roads[i][1], c = roads[i][2];
            graph[a][b] = Math.min(graph[a][b], c);
            graph[b][a] = Math.min(graph[b][a], c);
        }

        return graph;
    }

}
```

获赞 1   添加评论

令狐冲
更新于 11/7/2020, 7:49:52 AM

如果要保证正确性的最优做法。状态压缩动态规划。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.construct_graph(roads, n)
        state_size = 1 << n
        f = [
            [float('inf')] * (n + 1)
            for _ in range(state_size)
        ]
        f[1][1] = 0
        for state in range(state_size):
            for i in range(2, n + 1):
                if state & (1 << (i - 1)) == 0:
                    continue
                prev_state = state ^ (1 << (i - 1))
                for j in range(1, n + 1):
                    if prev_state & (1 << (j - 1)) == 0:
                        continue
                    f[state][i] = min(f[state][i], f[prev_state][j] + graph[j][i])
        return min(f[state_size - 1])

    def construct_graph(self, roads, n):
        graph = {
            i: {j: float('inf') for j in range(1, n + 1)}
            for i in range(1, n + 1)
        }
        for a, b, c in roads:
            graph[a][b] = min(graph[a][b], c)
            graph[b][a] = min(graph[b][a], c)
        return graph
```

获赞 1   添加评论

令狐冲
更新于 10/31/2020, 3:11:10 AM

使用随机化算法，不保证正确性，但是可以处理很大的数据，得到近似答案。 调整策略是交换 i, j 两个点的位置，看看是否能得到更优解 测试中如果失败了可以多跑几次。

python   java

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
# increase RANDOM_TIMES or submit your code again
# if you got wrong answer.
RANDOM_TIMES = 1000

class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.construct_graph(roads, n)
        min_cost = float('inf')
        for _ in range(RANDOM_TIMES):
            path = self.get_random_path(n)
            cost = self.adjust_path(path, graph)
            min_cost = min(min_cost, cost)
        return min_cost

    def construct_graph(self, roads, n):
        graph = {
            i: {j: float('inf') for j in range(1, n + 1)}
            for i in range(1, n + 1)
        }
        for a, b, c in roads:
            graph[a][b] = min(graph[a][b], c)
            graph[b][a] = min(graph[b][a], c)
        return graph

    def get_random_path(self, n):
        import random

        path = [i for i in range(1, n + 1)]
        for i in range(2, n):
            j = random.randint(1, i)
            path[i], path[j] = path[j], path[i]
        return path

    def adjust_path(self, path, graph):
        n = len(graph)
        adjusted = True
        while adjusted:
            adjusted = False
            for i in range(1, n):
                for j in range(i + 1, n):
                    if self.can_swap(path, i, j, graph):
                        path[i], path[j] = path[j], path[i]
                        adjusted = True
        cost = 0
        for i in range(1, n):
            cost += graph[path[i - 1]][path[i]]
        return cost

    def can_swap(self, path, i, j, graph):
        before = self.adjcent_cost(path, i, path[i], graph)
        before += self.adjcent_cost(path, j, path[j], graph)
        after = self.adjcent_cost(path, i, path[j], graph)
        after += self.adjcent_cost(path, j, path[i], graph)
        return before > after

    def adjcent_cost(self, path, i, city, graph):
        cost = graph[path[i - 1]][city]
        if i + 1 < len(path):
            cost += graph[city][path[i + 1]]
        return cost
```

获赞 0 添加评论

九章-加贺
更新于 6/10/2020, 10:42:06 PM

使用了 pruning 的 DFS

```

/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Result {
    int minCost;
    public Result(){
        this.minCost = 1000000;
    }
}

public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int minCost(int n, int[][] roads) {
        int[][] graph = constructGraph(roads, n);
        Set<Integer> visited = new HashSet<Integer>();
        List<Integer> path = new ArrayList<Integer>();
        Result result = new Result();
        path.add(1);
        visited.add(1);

        dfs(1, n, path, visited, 0, graph, result);

        return result.minCost;
    }

    void dfs (int city,
              int n,
              List<Integer> path,
              Set<Integer> visited,
              int cost,
              int[][] graph,
              Result result) {

        if (visited.size() == n) {
            result.minCost = Math.min(result.minCost, cost);
            return ;
        }

        for(int i = 1; i < graph[city].length; i++) {
            if (visited.contains(i)) {
                continue;
            }
            if (hasBetterPath(graph, path, i)) {
                continue;
            }
            visited.add(i);
            path.add(i);
            dfs(i, n, path, visited, cost + graph[city][i], graph, result);
            visited.remove(i);
            path.remove(path.size() - 1);
        }
    }

    int[][] constructGraph(int[][] roads, int n) {
        int[][] graph = new int[n + 1][n + 1];
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < n + 1; j++) {
                graph[i][j] = 100000;
            }
        }
        int roadsLength = roads.length;
        for (int i = 0; i < roadsLength; i++) {
            int a = roads[i][0], b = roads[i][1], c = roads[i][2];
            graph[a][b] = Math.min(graph[a][b], c);
            graph[b][a] = Math.min(graph[b][a], c);
        }

        return graph;
    }

    boolean hasBetterPath(int[][] graph, List<Integer> path, int city) {
        int pathLength = path.size();
        for (int i = 1; i < pathLength; i++){
            int path_i_1 = path.get(i - 1);
            int path_i = path.get(i);
            int path_last = path.get(pathLength - 1);
            if (graph[path_i_1][path_i] + graph[path_last][city] >
                graph[path_i_1][path_last] + graph[path_i][city]) {
                return true;
            }
        }

        return false;
    }

}
```

获赞 0 添加评论

九章-加贺

更新于 6/10/2020, 4:12:17 AM

暴力 DFS 算法

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
```

```
class Result {
    int minCost;
    public Result(){
        this.minCost = 1000000;
    }
}

public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int minCost(int n, int[][] roads) {
        int[][] graph = constructGraph(roads, n);
        Set<Integer> visited = new HashSet<Integer>();
        Result result = new Result();
        visited.add(1);

        dfs(1, n, visited, 0, graph, result);

        return result.minCost;
    }

    void dfs (int city,
              int n,
              Set<Integer> visited,
              int cost,
              int[][] graph,
              Result result) {

        if (visited.size() == n) {
            result.minCost = Math.min(result.minCost, cost);
            return ;
        }

        for(int i = 1; i < graph[city].length; i++) {
            if (visited.contains(i)) {
                continue;
            }
            visited.add(i);
            dfs(i, n, visited, cost + graph[city][i], graph, result);
            visited.remove(i);
        }
    }

    int[][] constructGraph(int[][] roads, int n) {
        int[][] graph = new int[n + 1][n + 1];
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < n + 1; j++) {
                graph[i][j] = 100000;
            }
        }
        int roadsLength = roads.length;
        for (int i = 0; i < roadsLength; i++) {
            int a = roads[i][0], b = roads[i][1], c = roads[i][2];
            graph[a][b] = Math.min(graph[a][b], c);
            graph[b][a] = Math.min(graph[b][a], c);
        }

        return graph;
    }

}
```

获赞 0   添加评论

九章算法助教团队
更新于 6/9/2020, 7:04:12 AM

状压dp $dp[i][j]$表示到第i个点，当前通过的点的状态为j的最短路。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class node{
        int w, x, y;
        node(int w, int x, int y){
                this.w = w;
                this.x = x;
                this.y = y;
        }
}
public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
        int[][] dp = new int[12][4096];
        int[][] mp = new int[12][12];
        Comparator<node> nodecmp = new Comparator<node>() {
        @Override
        public int compare(node o1, node o2) {
            return o1.w - o2.w;
        }
    };
        Queue<node> q = new PriorityQueue<>(nodecmp);
    public int minCost(int n, int[][] roads) {
        // Write your code here
        for(int i = 0; i < roads.length; ++i){
            int x = roads[i][0] - 1, y = roads[i][1] - 1;
                if(mp[x][y] == 0)mp[x][y] = mp[y][x] = roads[i][2];
                else mp[x][y] = mp[y][x] = Math.min(mp[x][y], roads[i][2]);
        }
        for(int i = 0; i < n; ++i) {
                for(int j = 0; j < (1 << n); ++j) {
                        dp[i][j] = 100000000;
                }
        }
        q.add(new node(0,0,1));
        dp[0][1] = 0;
        while(!q.isEmpty()) {
                node tmp = q.poll();
                int w = tmp.w, x = tmp.x, y = tmp.y;
                if(w > dp[x][y])continue;
                for(int i = 0; i < n; ++i)if(mp[x][i] != 0 && (y & (1 << i)) == 0) {
                        int ny = (y | (1 << i));
                        if(dp[i][ny] > dp[x][y] + mp[x][i]) {
                                dp[i][ny] = dp[x][y] + mp[x][i];
                                q.add(new node(dp[i][ny], i, ny));
                        }
                }
        }
        int min = 100000000;
        for(int i = 0; i < n; ++i) {
                min = Math.min(min, dp[i][(1 << n) - 1]);
        }
        return min;
    }
}
```

获赞 0 添加评论

九章算法助教团队
更新于 6/9/2020, 7:04:12 AM

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution {
public:
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    int minCost(int n, vector<vector<int>> &roads) {
        int dp[12][4096];
        int mapp[12][12];
        memset(mapp,0x3f,sizeof(mapp));
        memset(dp,0x3f,sizeof(dp));
        for (auto & i: roads){
            int x = i[0] - 1, y = i[1] - 1, c = i[2];
            mapp[x][y] = mapp[y][x] = min(mapp[x][y], c);
        }
        pair<int,pair<int,int>> tmp;
        tmp.first = 0; tmp.second.first = 0; tmp.second.second = 1;
        priority_queue<pair<int,pair<int,int>>> eq;
        eq.push(tmp);
        dp[0][1] = 0;
        while (!eq.empty()){
            pair<int,pair<int,int>> now = eq.top();
            eq.pop();
                int w = now.first, x = now.second.first, y = now.second.second;
                if(w > dp[x][y])continue;
                for(int i = 0; i < n; ++i){
                    if((y & (1 << i)) == 0){
                                int ny = (y | (1 << i));
                                if(dp[i][ny] > dp[x][y] + mapp[x][i]) {
                                        dp[i][ny] = dp[x][y] + mapp[x][i];
                                        tmp.first = dp[i][ny]; tmp.second.first = i; tmp.second.second = ny;
                                        eq.push(tmp);
                            }
                    }
                }
        }
        int minn = dp[0][0];
        for(int i = 0; i < n; i++) {
                minn = min(minn, dp[i][(1 << n) - 1]);
        }
        return minn;
    }
};
```

获赞 0 添加评论

令狐冲
更新于 6/9/2020, 7:04:00 AM

暴力 DFS 算法

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Result:
    def __init__(self):
        self.min_cost = float('inf')

class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.construct_graph(roads, n)
        result = Result()
        self.dfs(1, n, set([1]), 0, graph, result)
        return result.min_cost

    def dfs(self, city, n, visited, cost, graph, result):
        if len(visited) == n:
            result.min_cost = min(result.min_cost, cost)
            return

        for next_city in graph[city]:
            if next_city in visited:
                continue
            visited.add(next_city)
            self.dfs(next_city, n, visited, cost + graph[city][next_city], graph, result)
            visited.remove(next_city)

    def construct_graph(self, roads, n):
        graph = {i: {} for i in range(1, n + 1)}
        for a, b, c in roads:
            if b not in graph[a]:
                graph[a][b] = c
            else:
                graph[a][b] = min(graph[a][b], c)
            if a not in graph[b]:
                graph[b][a] = c
            else:
                graph[b][a] = min(graph[b][a], c)
        return graph
```

获赞 0　　添加评论

令狐冲
更新于 6/9/2020, 7:04:00 AM

使用了 pruning 的 DFS

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Result:
    def __init__(self):
        self.min_cost = float('inf')

class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.construct_graph(roads, n)
        result = Result()
        self.dfs(1, n, [1], set([1]), 0, graph, result)
        return result.min_cost

    def dfs(self, city, n, path, visited, cost, graph, result):
        if len(visited) == n:
            result.min_cost = min(result.min_cost, cost)
            return

        for next_city in graph[city]:
            if next_city in visited:
                continue
            if self.has_better_path(graph, path, next_city):
                continue
            visited.add(next_city)
            path.append(next_city)
            self.dfs(
                next_city,
                n,
                path,
                visited,
                cost + graph[city][next_city],
                graph,
                result,
            )
            path.pop()
            visited.remove(next_city)

    def construct_graph(self, roads, n):
        graph = {
            i: {j: float('inf') for j in range(1, n + 1)}
            for i in range(1, n + 1)
        }
        for a, b, c in roads:
            graph[a][b] = min(graph[a][b], c)
            graph[b][a] = min(graph[b][a], c)
        return graph

    def has_better_path(self, graph, path, city):
        for i in range(1, len(path)):
            if graph[path[i - 1]][path[i]] + graph[path[-1]][city] >\
                    graph[path[i - 1]][path[-1]] + graph[path[i]][city]:
                return True
        return False
```

获赞 0 添加评论

令狐冲
更新于 6/9/2020, 7:04:00 AM

另外一种随机化。调整策略是反转(reverse)中间的 i~j 这一段看看是否可以得到更优解。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
# increase RANDOM_TIMES or submit your code again
# if you got wrong answer.
RANDOM_TIMES = 1000

class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.construct_graph(roads, n)
        min_cost = float('inf')
        for _ in range(RANDOM_TIMES):
            path = self.get_random_path(n)
            cost = self.adjust_path(path, graph)
            min_cost = min(min_cost, cost)
        return min_cost

    def construct_graph(self, roads, n):
        graph = {
            i: {j: float('inf') for j in range(1, n + 1)}
            for i in range(1, n + 1)
        }
        for a, b, c in roads:
            graph[a][b] = min(graph[a][b], c)
            graph[b][a] = min(graph[b][a], c)
        return graph

    def get_random_path(self, n):
        import random

        path = [i for i in range(1, n + 1)]
        for i in range(2, n):
            j = random.randint(1, i)
            path[i], path[j] = path[j], path[i]
        return path

    def adjust_path(self, path, graph):
        n = len(graph)
        adjusted = True
        while adjusted:
            adjusted = False
            for i in range(1, n):
                for j in range(i + 1, n):
                    if self.can_reverse(path, i, j, graph):
                        self.reverse(path, i, j)
                        adjusted = True
        cost = 0
        for i in range(1, n):
            cost += graph[path[i - 1]][path[i]]
        return cost

    def can_reverse(self, path, i, j, graph):
        before = graph[path[i - 1]][path[i]]
        if j + 1 < len(path):
            before += graph[path[j]][path[j + 1]]
        after = graph[path[i - 1]][path[j]]
        if j + 1 < len(path):
            after += graph[path[i]][path[j + 1]]
        return before > after

    def reverse(self, path, i, j):
        while i < j:
            path[i], path[j] = path[j], path[i]
            i += 1
            j -= 1
```

获赞 0 添加评论

九章-加贺

更新于 6/9/2020, 7:04:00 AM

使用随机化算法，不保证正确性，但是可以处理很大的数据，得到近似答案。 调整策略是交换 i, j 两个点的位置，看看是否能得到更优解 测试中如果失败了可以多跑几次。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
/**
 * increase RANDOM_TIMES or submit your code again
 * if you got wrong answer.
 */

public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int RANDOM_TIMES = 1000;
    public int inf = 1000000;
    public Random rand =new Random(10000);
    public int minCost(int n, int[][] roads) {
        int[][] graph = constructGraph(roads, n);
        int minimalCost = inf;
        for (int i = 0; i < RANDOM_TIMES; i++) {
            int[] path = getRandomPath(n);
            int cost = adjustPath(path, graph);
            minimalCost = Math.min(minimalCost, cost);
        }

        return minimalCost;
    }

    int[][] constructGraph(int[][] roads, int n) {
        int[][] graph = new int[n + 1][n + 1];
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < n + 1; j++) {
                graph[i][j] = inf;
            }
        }
        int roadsLength = roads.length;
        for (int i = 0; i < roadsLength; i++) {
            int a = roads[i][0], b = roads[i][1], c = roads[i][2];
            graph[a][b] = Math.min(graph[a][b], c);
            graph[b][a] = Math.min(graph[b][a], c);
        }

        return graph;
    }

    int[] getRandomPath(int n) {
        int[] path = new int[n];
        for (int i = 0; i < n; i++) {
            path[i] = i + 1;
        }
        for (int i = 2; i < n; i++) {
            int j = rand.nextInt(10000) % i + 1;
            int tmp = path[i];
            path[i] = path[j];
            path[j] = tmp;
        }

        return path;
    }

    int adjustPath(int[] path, int[][] graph) {
        int n = graph.length - 1;
        boolean adjusted = true;
        while (adjusted) {
            adjusted = false;
            for (int i = 1; i < n; i++) {
                for (int j = i + 1; j < n; j++) {
                    if (canSwap(path, i, j, graph)) {
                        int tmp = path[i];
                        path[i] = path[j];
                        path[j] = tmp;
                        adjusted = true;
                    }
                }
            }
        }
        int cost = 0;
        for (int i = 1; i < n; i++) {
            cost += graph[path[i - 1]][path[i]];
        }

        return cost;
    }

    boolean canSwap(int[] path, int i, int j, int[][] graph) {
        int before = adjcentCost(path, i, path[i], graph);
        before += adjcentCost(path, j, path[j], graph);
        int after = adjcentCost(path, i, path[j], graph);
        after += adjcentCost(path, j, path[i], graph);

        return before > after;
    }

    int adjcentCost(int[] path, int i, int city, int[][] graph) {
        int cost = graph[path[i - 1]][city];
        if (i + 1 < path.length) {
            cost += graph[city][path[i + 1]];
        }

        return cost;
    }

}
```

获赞 0 添加评论

九章-加贺
更新于 6/9/2020, 7:04:00 AM

另外一种随机化。调整策略是反转(reverse)中间的 i~j 这一段看看是否可以得到更优解

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
/**
 * increase RANDOM_TIMES or submit your code again
 * if you got wrong answer.
 */

public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int RANDOM_TIMES = 2000;
    public int inf = 1000000;
    public Random rand =new Random(10000);
    public int minCost(int n, int[][] roads) {
        int[][] graph = constructGraph(roads, n);
        int minimalCost = inf;
        for (int i = 0; i < RANDOM_TIMES; i++) {
            int[] path = getRandomPath(n);
            int cost = adjustPath(path, graph);
            minimalCost = Math.min(minimalCost, cost);
        }

        return minimalCost;
    }

    int[][] constructGraph(int[][] roads, int n) {
        int[][] graph = new int[n + 1][n + 1];
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < n + 1; j++) {
                graph[i][j] = inf;
            }
        }
        int roadsLength = roads.length;
        for (int i = 0; i < roadsLength; i++) {
            int a = roads[i][0], b = roads[i][1], c = roads[i][2];
            graph[a][b] = Math.min(graph[a][b], c);
            graph[b][a] = Math.min(graph[b][a], c);
        }

        return graph;
    }

    int[] getRandomPath(int n) {
        int[] path = new int[n];
        for (int i = 0; i < n; i++) {
            path[i] = i + 1;
        }
        for (int i = 2; i < n; i++) {
            int j = rand.nextInt(10000) % i + 1;
            int tmp = path[i];
            path[i] = path[j];
            path[j] = tmp;
        }

        return path;
    }

    int adjustPath(int[] path, int[][] graph) {
        int n = graph.length - 1;
        boolean adjusted = true;
        while (adjusted) {
            adjusted = false;
            for (int i = 1; i < n; i++) {
                for (int j = i + 1; j < n; j++) {
                    if (canReverse(path, i, j, graph)) {
                        reverse(path, i, j);
                        adjusted = true;
                    }

                }
            }
        }
        int cost = 0;
        for (int i = 1; i < n; i++) {
            cost += graph[path[i - 1]][path[i]];
        }

        return cost;
    }

    boolean canReverse(int[] path, int i, int j, int[][] graph) {
        int before = graph[path[i - 1]][path[i]];
        if (j + 1 < path.length) {
            before += graph[path[j]][path[j + 1]];
        }
        int after = graph[path[i - 1]][path[j]];
        if (j + 1 < path.length) {
            after += graph[path[i]][path[j + 1]];
        }

        return before > after;
    }

    void reverse(int[] path, int i, int j) {
        while (i < j) {
            int tmp = path[i];
            path[i] = path[j];
            path[j] = tmp;
            i++;
            j--;
        }
    }

}
```

获赞 0   添加评论

辛同学
更新于 12/23/2020, 4:25:11 PM

这道题有一个很脑残的地方， 就是会出现重复的边， 权值还不一样。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int minCost(int n, int[][] roads) {
        // Write your code here
        int[][] cost = new int[n + 1][n + 1];
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                cost[i][j] = Integer.MAX_VALUE;
            }
        }

        for (int[] road: roads) {
            int u = road[0], v = road[1], p = road[2];
            cost[v][u] = Math.min(cost[v][u], p);
            cost[u][v] = Math.min(cost[u][v], p);
        }


        boolean[] visited = new boolean[n + 1];
        visited[1] = true;
        int[] res = {Integer.MAX_VALUE};
        findMinCostPath(1, n - 1, 0, visited, cost, res);
        return res[0];
    }

    private void findMinCostPath(int cur, int k, int cost, boolean[] visited, int[][] price, int[] res) {
        if (cost >= res[0]) {
            return;
        }
        if (k == 0) {
            res[0] = cost;
            return;
        }

        for (int city = 2; city < visited.length; ++city) {
            if (!visited[city] && price[cur][city] != Integer.MAX_VALUE) {
                visited[city] = true;
                findMinCostPath(city, k - 1, cost + price[cur][city], visited, price, res);
                visited[city] = false;
            }
        }
    }
}
```

👍 获赞 1 💬 3 条评论

L同学

更新于 12/6/2020, 9:25:18 PM

BFS 用 priority queue。每次都poll最小cost的然后往下查找，当找到某个点，所有城市都走遍了，就是答案。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ /
Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
public class Solution {

    class Pair implements Comparable<Pair> {
        int cur, remaining, cost;
        boolean[] visited;

        Pair(int cur, int remaining, int cost, boolean[] visited) {
            this.cur = cur;
            this.remaining = remaining;
            this.cost = cost;
            this.visited = Arrays.copyOf(visited, visited.length);
            this.visited[cur] = true;
        }

        public int compareTo(Pair other) {
            return this.cost - other.cost;
        }
    }

    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    public int minCost(int n, int[][] roads) {
        // Write your code here
        if (roads == null || roads.length == 0 || n <= 1) return 0;

        Map<Integer, List<int[]>> srcToDesMap = getSrcToDesMap(roads);

        Queue<Pair> pq = new PriorityQueue<>();
        pq.offer(new Pair(1, n - 1, 0, new boolean[n+1]));

        while (!pq.isEmpty()) {
            Pair curPos = pq.poll();
            if (curPos.remaining == 0) return curPos.cost;

            for (int[] next : srcToDesMap.get(curPos.cur)) {
                if (curPos.visited[next[0]]) continue;

                pq.offer(new Pair(next[0], curPos.remaining - 1, curPos.cost + next[1], curPos.visited));
            }
        }
        return 0;
    }

    private Map<Integer, List<int[]>> getSrcToDesMap(int[][] roads) {
        Map<Integer, List<int[]>> result = new HashMap<>();
        for (int[] road : roads) {
            int c1 = road[0], c2 = road[1], cost = road[2];
            result.putIfAbsent(c1, new ArrayList<>());
            result.get(c1).add(new int[]{c2, cost});

            result.putIfAbsent(c2, new ArrayList<>());
            result.get(c2).add(new int[]{c1, cost});
        }

        return result;
    }
}
```

获赞 1   添加评论

奶威是大腿

更新于 10/23/2020, 3:39:38 PM

DFS+剪枝。建图的时候如果不存在的边被设为INT_MAX，在判断是否剪枝的时候要注意溢出（因为涉及到加法）

```

/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ /
Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution {
public:
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    int minCost(int n, vector<vector<int>> &roads) {
        // Write your code here
        if (n <= 1 || roads.empty()) return 0;
        unordered_map<int, unordered_map<int, int>> graph;
        buildGraph(n, graph, roads);
        int shortest = INT_MAX;
        set<int> visited = {1};
        vector<int> path = {1};
        findShortest(n, graph, shortest, visited, path, 0);
        return shortest;
    }

    void buildGraph(int n, unordered_map<int, unordered_map<int, int>> &graph, vector<vector<int>> &roads) {
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j++) {
                graph[i][j] = 9999;
                graph[j][i] = 9999;
            }
        }
        for (vector<int> &tuple : roads) {
            graph[tuple[0]][tuple[1]] = min(graph[tuple[0]][tuple[1]], tuple[2]);
            graph[tuple[1]][tuple[0]] = min(graph[tuple[1]][tuple[0]], tuple[2]);
        }
    }

    void findShortest(int n, unordered_map<int, unordered_map<int, int>> &graph, int &shortest, set<int> &visited, vector<i
nt> &path, int cost) {
        if (path.size() == n) {
            shortest = min(shortest, cost);
            return;
        }
        for (auto &next : graph[path.back()]) {
            if (visited.find(next.first) != visited.end()) continue;
            if (hasBetter(path, next.first, graph)) continue;
            cost += next.second;
            visited.insert(next.first);
            path.push_back(next.first);
            findShortest(n, graph, shortest, visited, path, cost);
            path.pop_back();
            visited.erase(next.first);
            cost -= next.second;
        }
    }

    bool hasBetter(vector<int> &path, int next, unordered_map<int, unordered_map<int, int>> &graph) {
        for (int i = 1; i < path.size() - 1; i++) {
            if (graph[path[i-1]][path[path.size()-1]] + graph[path[i]][next]
            <= graph[path[i-1]][path[i]] + graph[path[path.size()-1]][next])
                return true;
        }
        return false;
    }
};
```

获赞 1 添加评论

我要AC

更新于 8/1/2020, 12:28:02 AM

这题是个马夹题，其实就是 1到n个数字，然后求n-1的permutation。

题目有一个大坑，也可以说是题目交代的不清。

- a -> b 和 b -> a 的距离可以是不一样的，需要求两个之间的 min。

复杂度

- time: `O((n-1)!)`
- space: `O(n)`

优化

- 如果当前cost已经超过计算得到的最小cost了的话，以后cost只会更大，及时剪纸。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution:
    """
 @param n: an integer,denote the number of cities
 @param roads: a list of three-tuples,denote the road between cities
 @return: return the minimum cost to travel all cities
 """

    def minCost(self, n, roads):
        # Write your code here
        """
        1->    2->3
        1->    3->2

        cities: 1...n
        permutation: (n-1)!

        time: O((n-1)!)
        space: O(n)
        """

        def dfs(n, visited, prev_city, cost, weights):
            # print((visited, prev_city, cost))
            if len(visited) == n:
                # print((visited, cost))
                self.min_cost = min(self.min_cost, cost)
                return

            if cost > self.min_cost:
                return

            for i in range(2, n + 1):
                if not i in visited:
                    visited.add(i)

                    # weights is a dict records cost from city1 to city2
                    # here cost + the cost from prev_city to i city.
                    min_dist_between = min(
                        weights.get((i, prev_city), float('inf')),
                        weights.get((prev_city, i), float('inf')))
                    dfs(n, visited, i, cost + min_dist_between, weights)

                    visited.remove(i)

        self.min_cost = float('inf')
        weights = {(cf, ct): cost for cf, ct, cost in roads}
        # print(weights)
        dfs(n, set([1]), 1, 0, weights)
        return self.min_cost
```

获赞 0 添加评论

九章用户B4UZID
更新于 6/9/2020, 7:04:07 AM

dynamic programming 经典的DP解法套用到本题上 注意一些奇怪的test case

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution {
public:
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */
    int minCost(int n, vector<vector<int>> &roads) {
        if (n < 2) {
            return 0;
        }
        vector<vector<int>> path = findDis(roads);
        vector<vector<int>> dp(1 << n, vector<int>(n, INT_MAX));

        dp[1][0] = 0;
        for (int i = 1; i < (1 << n); ++i) {
            for (int j = 0; j < n; ++j) {
                if (dp[i][j] == INT_MAX) {
                    continue;
                }
                for (int k = 0; k < n; ++k) {
                    if ((i & (1 << k)) == 0 && path[j][k] != INT_MAX) {
                        dp[i | (1 << k)][k] = min(dp[i | (1 << k)][k], dp[i][j] + path[j][k]);
                     }
                }
            }
        }

        int ans = INT_MAX;
        for (int i = 0; i < n; ++i) {
            ans = min(ans, dp[(1 << n) - 1][i]);
        }
        return ans;
    }
private:
    vector<vector<int>> findDis(vector<vector<int>>& roads) {
        int n = roads.size();
        vector<vector<int>> ans(n, vector<int>(n, INT_MAX));
        for (auto& item : roads) {
            int x = item[0] - 1;
            int y = item[1] - 1;
            int cost = item[2];
            ans[x][y] = min(ans[x][y], cost);
            ans[y][x] = min(ans[x][y], cost);
        }
        return ans;
    }
};
```

获赞 0 添加评论

九章用户CD9WI2

更新于 6/9/2020, 7:04:03 AM

1. 将 ((1,2,1),(2,1,3)...) tuple list 转变成2D数组。 2D数组下标为出发A和到达B城市数 - 1，数组存储内容是A到B或B到A 之间最小cost

2. 运用permutation九章算法班模板。从第二个开始dfs枚举每一个城市并累计计算花销。当所有城市遍历完为一组，更新minCost。
3. 其中加入一个pruning即如果recursion中途累计cost已经大于minCost，退出recursion其答案一定不会影响最终答案。

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
public class Solution {
    /**
     * @param n: an integer,denote the number of cities
     * @param roads: a list of three-tuples,denote the road between cities
     * @return: return the minimum cost to travel all cities
     */


    int minCost = Integer.MAX_VALUE;

    public int minCost(int n, int[][] roads) {

        int[][] cost = new int[n][n];

        for(int i = 0; i < n; i++) {

            for(int j = 0; j < n; j++) {
                cost[i][j] = Integer.MAX_VALUE;
            }
        }

        for(int i = 0; i < roads.length; i++) {

            int cityA = roads[i][0];
            int cityB = roads[i][1];

            cost[cityA - 1][cityB - 1] = Math.min(cost[cityA - 1][cityB - 1], roads[i][2]);
            cost[cityB - 1][cityA - 1] =  Math.min(cost[cityB - 1][cityA - 1], roads[i][2]);
        }

        boolean[] visited = new boolean[n];
        List<Integer> subset = new ArrayList<>();
        subset.add(1);
        visited[0] = true;

        dfs(n, subset, visited, cost, 0);

        return minCost;

    }

    void dfs(int n, List<Integer> subset, boolean[] visited, int[][] cost, int costSum) {

        if(subset.size() == n) {
            minCost = Math.min(minCost, costSum);
            return;
        }

        if(costSum > minCost) {
            return;
        }

        for(int i = 1; i < n + 1; i++) {

            int start = subset.get(subset.size() - 1);
            if(visited[i - 1] || cost[start - 1][i - 1] == Integer.MAX_VALUE) {
                continue;
            }

            subset.add(i);
            visited[i - 1] = true;
            dfs(n, subset, visited, cost, costSum + cost[start - 1][i - 1]);
            subset.remove(subset.size() - 1);
            visited[i - 1] = false;
        }
    }



}
```

获赞 0 添加评论

Tianshu
更新于 6/9/2020, 7:04:03 AM

来个python的，这个会TLE，因为没有剪枝。

递归的定义： 从当前节点出发，走完所有城市的最小值. 我们用一个visited来记录已经走过的城市。 当然，这个visited的长度，也可以告诉我们已经走了多少个城市。

递归的拆解： 从当前节点的每个孩子出发，走完所有城市的最小值 然后对于每个孩子，加上从当前城市到每个孩子的距离。 然后打擂台， 得到从当前节点出发走遍所有城市最小值

(/) 课程 (/course/) 旗舰课 (/premium-course/) 1对1私教 (/1on1/) 免费课 (/seminar/) 下载APP 邀请有礼 题解 成功案例 消息 更多... 任务 我的课程 (/accounts/profile/) (/accounts/

递归的出口： 显然，如果已经走完所有的城市，那么返回 0

坑点： 两个城市之间的直接道路也可以有多条，挑最短的那条。

数据结构： 建图，用了 dict of dict。 dictu () = w 表示城市u到v的距离为w

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def minCost(self, n, roads):
        graph = self.buildGraph(n, roads)

        return self.dfs(graph, 1, set([1]))


    def dfs(self, graph, start, visited):

        if len(visited) == len(graph):
            return 0

        minC = float('Inf')
        for next_city in graph[start]:
            if next_city in visited:
                continue

            visited.add(next_city)
            next_cost = self.dfs(graph, next_city, visited)
            minC = min(minC, graph[start][next_city] + next_cost)
            visited.remove(next_city)

        return minC



    def buildGraph(self, n, roads):
        graph = { i : {} for i in range(1, n + 1)}

        for u, v, w in roads:
            if v not in graph[u]:
                graph[u][v] = w
            else:
                graph[u][v] = min(graph[u][v], w)

            if u not in graph[v]:
                graph[v][u] = w
            else:
                graph[v][u] = min(graph[v][u], w)

        return graph
```

获赞 0 添加评论

Tianshu

更新于 6/9/2020, 7:04:03 AM

这个是剪枝版本。 用一个全局变量来看已经走了路径的最小值

递归的定义： 从当前节点出发，走完所有城市， 更新全局变量的最小值 我们用一个visited来记录已经走过的城市。 当然，这个visited的长度，也可以告诉我们已经走了多少个城市。 如果已经走完，更新全局变量的最小值。 这里，我们用一个变量来记录从1出发走到当前节点的距离 如果这个距离已经超过了self.result就不需要再走了。

递归的拆解： 从当前节点的每个孩子出发，走完所有城市的最小值 然后对于每个孩子，加上从当前城市到每个孩子的距离。 然后打擂台， 得到从当前节点出发走遍所有城市最小值

递归的出口： 显然，如果已经走完所有的城市，那么更新self.result, 返回

坑点： 两个城市之间的直接道路也可以有多条，挑最短的那条。

数据结构： 建图，用了 dict of dict。 dictu () = w 表示城市u到v的距离为w

```
/**
* 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
* - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
* - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
* - Design类课程包括：系统设计 System Design，面向对象设计 OOD
* - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
* - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
*/
class Solution:
    """
    @param n: an integer,denote the number of cities
    @param roads: a list of three-tuples,denote the road between cities
    @return: return the minimum cost to travel all cities
    """
    def __init__(self):
        self.result = float('Inf')

    def minCost(self, n, roads):
        graph = self.buildGraph(n, roads)

        self.dfs(graph, 1, 0, set([1]))

        return self.result


    def dfs(self, graph, start, cmc, visited):

        if len(visited) == len(graph):
            self.result = min(self.result, cmc)
            return

        for next_city in graph[start]:
            if next_city in visited:
                continue

            if cmc > self.result:
                continue

            cur_cost = graph[start][next_city]
            if cur_cost > self.result:
                continue

            visited.add(next_city)
            self.dfs(graph, next_city, cmc + cur_cost, visited)
            visited.remove(next_city)


    def buildGraph(self, n, roads):
        graph = { i : {} for i in range(1, n + 1)}

        for u, v, w in roads:
            if v not in graph[u]:
                graph[u][v] = w
            else:
                graph[u][v] = min(graph[u][v], w)

            if u not in graph[v]:
                graph[v][u] = w
            else:
                graph[v][u] = min(graph[v][u], w)

        return graph
```

获赞 0   添加评论