《算法设计与分析》习题 7 作业

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/11/4　　得　分：

问题一：用贪心法求解如下背包问题的最优解：有7个物品，重量分别为$(2,3,5,7,1,4,1)$，价值分别为$(10,5,15,7,6,18,3)$，背包容量$W=15$。写出求解过程。

【原理】

贪心法求解容易陷入求解局部最优而未能得到全局最优解的情况，因此设置合理的贪心策略更容易获得全局最优答案。这里我选择三个策略分别尝试进行求解，选取其中最优的答案：

- 选择价值最大的物品
- 选择重量最轻的物品
- 选择单位重量价值最大的物品

【调试】

为了更加清楚中间过程，设置三种策略的中间过程分别输出物品按照策略排好序后的物品序号、物品重量、物品价值以及是否选择将物品装入背包中。

选择价值最大物品策略

物品序号	物品重量	物品价值	是否选择
6	4	18	是
3	5	15	是
1	2	10	是
4	7	7	否
5	1	6	是
2	3	5	是
7	1	3	否

表 1 策略 1 中间过程

最终背包选择装入第 1、2、3、5、6 共 5 件物品，用掉 15 重量获得 54 价值。

选择重量最轻物品策略

物品序号	物品重量	物品价值	是否选择
5	1	6	是
7	1	3	是
1	2	10	是
2	3	5	是
6	4	18	是
3	5	15	否
4	7	7	否

表 2 策略 2 中间过程

最终背包选择装入第 1、2、5、6、7 共 5 件物品，用掉 11 重量获得 42 价值。

选择单位重量价值最大物品策略

物品序号	物品重量	物品价值	是否选择
5	1	6	是
1	2	10	是
6	4	18	是
7	1	3	是
3	5	15	是
2	3	5	否
4	7	7	否

表 3 策略 3 中间过程

最终背包选择装入第 1、3、5、6、7 共 5 件物品，用掉 13 重量获得 52 价值。

【源码】

```
#include <algorithm>
#include <iostream>
using namespace std;

struct thing {
    int weight, value, id;
} beg[10];

bool sortByValue(thing A, thing B) {    //物品价值大的优先
    return A.value > B.value;
```

```
}

bool sortByWeight(thing A, thing B) {   //物品重量轻的优先
    return A.weight < B.weight;
}

bool sortByRate(thing A, thing B) {     //物品价值重量比大的优先
    return A.value * 1.0 / A.weight > B.value * 1.0 / B.weight;
}

signed main() {
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    for (int i = 1; i <= 7; ++i)
        cin >> beg[i].weight;
    for (int i = 1; i <= 7; ++i)
        cin >> beg[i].value, beg[i].id = i;

    sort(beg + 1, beg + 8, sortByValue);
    int W = 15, ans = 0;
    cout << "id\t\tweight\tvalue\tchoose" << endl;
    for (int i = 1; i <= 7; ++i) {
        cout << beg[i].id << "\t\t" << beg[i].weight << "\t\t" <<
beg[i].value << "\t\t";
        if (beg[i].weight <= W)
            W -= beg[i].weight, ans += beg[i].value, cout << "Yes" <<
endl;
        else
            cout << "No" << endl;
    }
    cout << "Remain W: " << W << "\t\t" << "Earn V: " << ans << endl;

    sort(beg + 1, beg + 8, sortByWeight);
    W = 15, ans = 0;
    cout << "id\t\tweight\tvalue\tchoose" << endl;
    for (int i = 1; i <= 7; ++i) {
        cout << beg[i].id << "\t\t" << beg[i].weight << "\t\t" <<
beg[i].value << "\t\t";
        if (beg[i].weight <= W)
            W -= beg[i].weight, ans += beg[i].value, cout << "Yes" <<
endl;
        else
            cout << "No" << endl;
    }
```

```
    cout << "Remain W: " << W << "\t\t" << "Earn V: " << ans << endl;

    sort(beg + 1, beg + 8, sortByRate);
    W = 15, ans = 0;
    cout << "id\t\tweight\tvalue\tchoose" << endl;
    for (int i = 1; i <= 7; ++i) {
        cout << beg[i].id << "\t\t" << beg[i].weight << "\t\t" <<
beg[i].value << "\t\t";
        if (beg[i].weight <= W)
            W -= beg[i].weight, ans += beg[i].value, cout << "Yes" <<
endl;
        else
            cout << "No" << endl;
    }
    cout << "Remain W: " << W << "\t\t" << "Earn V: " << ans << endl;
    return 0;
}
```

【答案】综合上述三种策略，最终选择第一种策略，选择装入第 1、2、3、5、6 共 5 件物品，用掉 15 重量获得 54 价值。

问题二：写出7.2.1节最短链接策略求解TSP问题的算法对应的程序并上机实现。

【原理】

TSP问题的最短链接策略即每次选择边的时候不选择装入第 1、2、3、5、6 共 5 件物品，用掉 15 重量获得 54 价值。考虑全局的路径最短而仅仅着眼于当下局面，选择未走过的点距离当前位置最短的边作为最优解从而形成最终哈密顿回路。因此当从剩余边集E'中选择一条边（u,v）加入解集合S中，应该满足下述条件：

①边(u,v)是边集E'中代价最小的边；

②边(u,v)加入解集合S后，S中不产生回路；

③边(u,v)加入解集合S后，S中不产生分支。

【题目】

选择图7.2给出的无向图代价矩阵作为题目，上机计算过程设置记录中间过程并输出答案。

【源码】

```cpp
#include <iostream>
#include <vector>
using namespace std;
const int inf = 0x3f3f3f3f;
int G[7][7];
bool vis[7];

signed main() {
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    for (int i = 1; i <= 5; ++i)
        for (int j = 1; j <= 5; ++j)
            cin >> G[i][j];
    vis[1] = 1;  //起点从 1 出发
    int ans = 0, now = 1, node, minn;
    cout << 1 << "-";
    for (int i = 1; i < 5; ++i) {
        node = now, minn = inf;
        for (int j = 1; j <= 5; ++j)
            if (!vis[j] && G[now][j] < minn) minn = G[now][j], node = j;
        vis[node] = 1, now = node, ans += minn;
        cout << '(' << minn << ")->" << node << "-";
    }
    cout << '(' << G[now][1] << ")->" << 1 << endl;
    ans += G[node][1];
    cout << "TSP length: " << ans << endl;
    return 0;
}
/**
 * 1-(2)->4-(2)->3-(5)->5-(2)->2-(3)->1
 * TSP length: 14
 */
```

【答案】

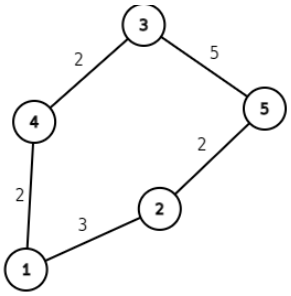

图 1 TSP 走法图解

最终计算得该贪心策略下 TSP 最优解为 14。

【复杂度分析】

n 个结点供进行 $n-1$ 次贪心，每次贪心遍历寻找未走过且距离最近的点，二重循环时间复杂度$O(n^2)$。

问题三：写出$Kruskal$算法对应的程序并上机实现。

【原理】

该算法又叫加边法，适用于稀疏图。采用的数据结构是并查集用于标记结点是否加入点集中，采用算法为贪心思想每次选择不在点集中且距离点集最近的点并放入点集中。

【证明】

若是选择的每一条边的权值是最小的，那么整个树的权值就是最小。符合贪心算法局部最优进而求出去全局最优的性质。

【操作】

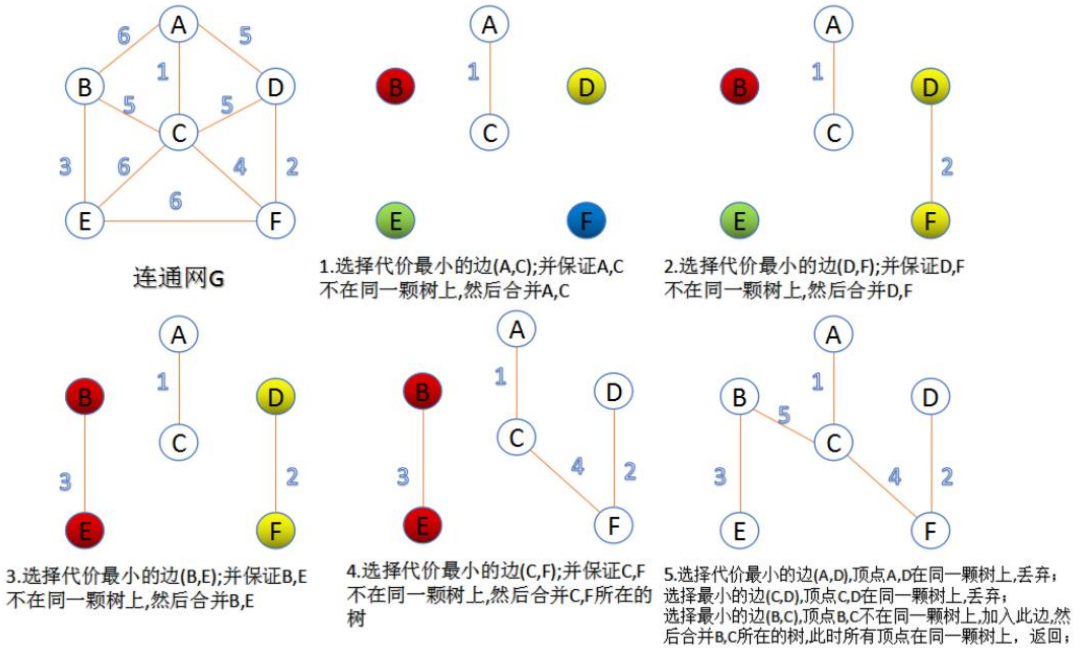

图 2 算法说明图

①把图中的所有边按代价从小到大排序（如此选择边的时候优先看权值小的）。

②把图中的 n 个顶点看成独立的 n 棵树组成的森林。

③按权值从小到大选择边，若是所选的边连接的两个顶点 u，v 属于两颗不同的树，则选择此边成为最小生成树的一条边，并将这两颗树合并作为一颗树。

④重复第三步操作直到所有顶点都在一颗树内或者有 $n-1$ 条边为止。

【源码】

以 HDU1233 为例，上机测试算法正确性，并输出中间过程。

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
int fa[105];  //父元素数组

struct node {
    int x;
    int y;
    int v;
} road[5005];

bool cmp(node x, node y) {  //比较声明，按升序排列
    return x.v < y.v;
}

void init() {
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}

int find(int x) {  //压缩路径的三目运算符
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

void join(int x, int y) {
    int fx = find(x), fy = find(y);
    if (fx == fy)
        return;
    else {
        fa[fy] = fx;
    }
}

signed main() {
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    ios::sync_with_stdio(false), cin.tie(0);
    while (cin >> n && n) {
```

```
        m = (n - 1) * n / 2;
        init();  //初始化
        for (int i = 1; i <= m; i++)
            cin >> road[i].x >> road[i].y >> road[i].v;
        sort(road + 1, road + 1 + m, cmp);  //对权值升序排序
        int sum = 0;
        for (int i = 1; i <= m; i++) {
            if (find(road[i].x) != find(road[i].y)) {  //找到没有通路的两点
                sum += road[i].v;                      //加和
                join(road[i].x, road[i].y);            //连接
                cout << "Step" << i << ": " << road[i].x << ' ' <<
road[i].y << ' ' << road[i].v << endl;
            }
        }
        cout << sum << endl;
    }
    return 0;
}
/*
 * Step1: 1 2 1
 * Step2: 1 3 2
 * 3
 * Step1: 1 2 1
 * Step2: 1 4 1
 * Step4: 2 3 3
 * 5
**/
```

【答案】

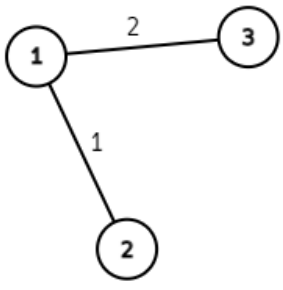

图 2 样例一图解

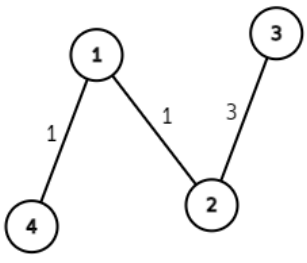

图 3 样例二图解

对于样例一求出来最小生成树的权值为 3，样例二求出来最小生成树的权值

为 5。

【复杂度分析】

初始化生成树的边集复杂度为$O(1)$，对每个结点的初始化为$O(V)$，由于集中于边操作所以复杂度为$O(nlogn)$。

问题四：$Dijkstra$是求解有向图最短路径的一个经典算法，也是应用贪心法的一个成功实例，请描述$Dijkstra$算法的贪心策略和具体算法。

【贪心策略】

不妨考虑以下场景，在途中所有的边上布满多米诺骨牌，相当于把骨牌看成图的边。一条边上的多米诺骨牌数量和边的权值成正比。规定所有骨牌倒下的速度都是一样的，如果在一个结点上推倒骨牌会导致这个结点上的所有骨牌都往后面倒下去。在起点s推导骨牌可以观察到从s开始它连接边上的骨牌都逐渐倒下，并到达所有能到达的结点。在某个结点t，可能先后从不同的线路倒骨牌过来；先倒过来的骨牌，其经过的路径肯定是从s到达t最短路径；后倒来的骨牌，对确定结点 t 的最短路径没有贡献不管它。

从整体看这就是一个从起点s扩散到整个图的过程，遵从的思想就是贪心，即“抄近路走，肯定能找到最短的路径”。

【具体步骤】

①在s所有直连的邻居中，最近的邻居u，骨牌肯定先到达。u是第一个确定最短路结点。从u直连到s的路径肯定是最短路径，因为如果u绕道别的结点到 s 肯定更远。

②然后把后面骨牌倒下的分成两个部分，一部分是从s继续倒下到s的其他直连邻居，另一部分是从u出发倒下到u的直连邻居。那么下一个到达的结点v必然是s或者u的一个直连邻居。v是第二个确定最短路径的结点。

③继续执行上述的步骤，在每次迭代的过程中都能确定一个结点的最短路径。

【源码】

```
const int maxn=1005;//顶点数
const int inf=0x3f3f3f3f;

int n;//记录顶点数
int dis[maxn];
```

```
int graph[maxn][maxn];
bool vis[maxn];

void init(){
        memset(vis,0,sizeof(vis));
        //graph 初始化，这是必须的
        //...其他的初始化
}

void dij(int start){//注意，start 不一定是哪一个点
        int x;//记录最近的未遍历点
        int minn;//临时保存当前的最短距离

    vis[start]=1; //标记起点
        //初始化 dis,里面存入 start 到各点的距离
        for(int i=1;i<=n;i++)
        dis[i]=graph[start][i];

    //一次性求出从距离近的点到远的点距离 start 的最短路径
        for(int i=1;i<n;i++){//i==n 的时候，到终点了就不需要再遍历了
                minn=inf;//初始化 minn 为最大，其实 minn 就是用来找最近的未遍历的点

        //遍历最小的 dis[i]
        for(int j=1;j<=n;j++)
                if(!vis[j]&&dis[j]<minn){
                        minn=dis[j];
                        x=j;//x 记录下来现在遍历了的点
                }
                vis[x]=1;//标记已经遍历过的点

        //j 就相当于 y,遍历与 x 相邻的 y 的点，更新 dis[j]的值
                for(int j=1;j<=n;j++){
                        if(!vis[j]&&dis[j]>dis[x]+graph[x][j])
                        dis[j]=dis[x]+graph[x][j];
                }
        }
}
```

【复杂度分析】

最朴素的 dijkstra 算法时间复杂度为标准的$O(n^2)$，其实在每次寻找最近点的步骤中可以对遍历过的点相邻的边使用堆优化，如此更具目的性的确定最近点，复杂度可以降到$O(ElogE)$