《算法设计与分析》习题 9 作业

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/11/19　　得　分：

问题一：对于如果所示无向图，应用分支限定法求解从顶点 a 出发的 TSP 问题，请写出在解空间树上的搜索过程。

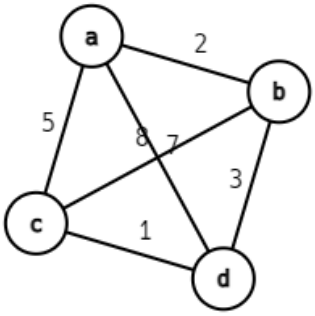

图 1 问题一配图

【过程】

①画出代价矩阵。

$$\begin{bmatrix} \infty & 2 & 5 & 7 \\ 2 & \infty & 8 & 3 \\ 5 & 8 & \infty & 1 \\ 7 & 3 & 1 & \infty \end{bmatrix}$$

图 2 无向图代价矩阵

②粗略估计上界（贪心法最优解$a->b->d->c->a=2+3+1+5=11$）和下界（分支限界法$(2+5+2+3+5+1+3+1)/2=11$），所以最终限定在[11,11]。

$$lb=(2\sum_{i=1}^{k-1}c[r_i][r_{i+1}]+\sum_{i=1,k}r_i \text{ 行不在路径上的最小元素}+\sum_{r_j\notin U}r_j \text{ 行最小的两个元素})/2$$

③利用限界函数计算解空间树上的搜索过程，每次选择一个节点后判断是够还在上下界中，在的话继续向下搜索，否则可抛弃该点作为剪枝优化。

【中间过程】

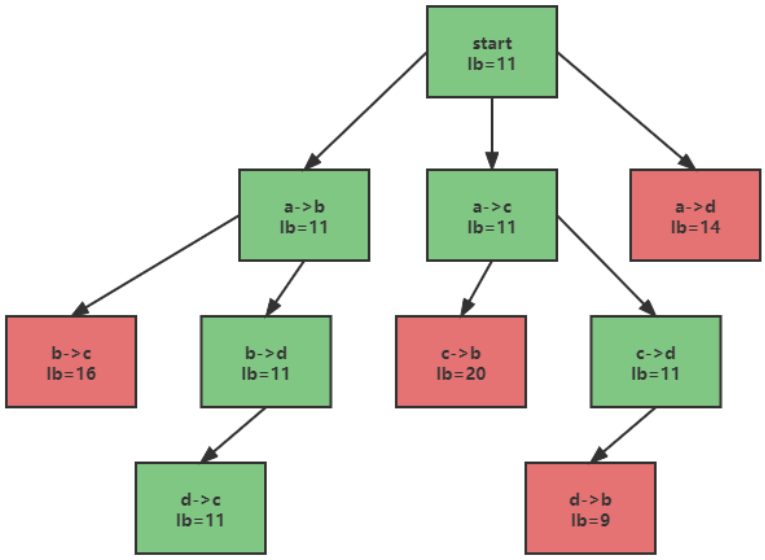

图 3 解空间树

【答案】

最终所求最优解为11，最优路径为$a-> b-> d-> c-> a$。

问题二：对图中所示任务分配问题，请写出在解空间树上的搜索过程。

$$C=\begin{bmatrix} & \text{任务 } 1 & \text{任务 } 2 & \text{任务 } 3 & \text{任务 } 4 \\ \text{人员 } a & 3 & 8 & 4 & 12 \\ \text{人员 } b & 9 & 12 & 13 & 5 \\ \text{人员 } c & 8 & 7 & 9 & 3 \\ \text{人员 } d & 12 & 7 & 6 & 8 \end{bmatrix}$$

图 4 问题二配图

【过程】

①粗略估计上界（贪心法最优解：任务 1->人员 a，任务 4->人员 b，任务 2->人员 c，任务 3->人员 d，总和为 21）和下界（不考虑实际情况，每个人都选最轻的任务：任务 1->人员 a，任务 4->人员 b，任务 4->人员 c，任务 3->人员 d，总和为 17），所以最终限定在$[17，21]$。

②一般情况下，假设当前已对人员$1-i$分配了任务，并且花费了成本v，则限界函数可以定义为：$lb=v+\sum_{k=i+1}^{n}$第k行的最小值，利用此限界函数计算解空间树上的搜索过程，每次选择一个节点后判断是够还在上下界中，在的话继续向下搜

索，否则可抛弃该点作为剪枝优化。

【中间过程】

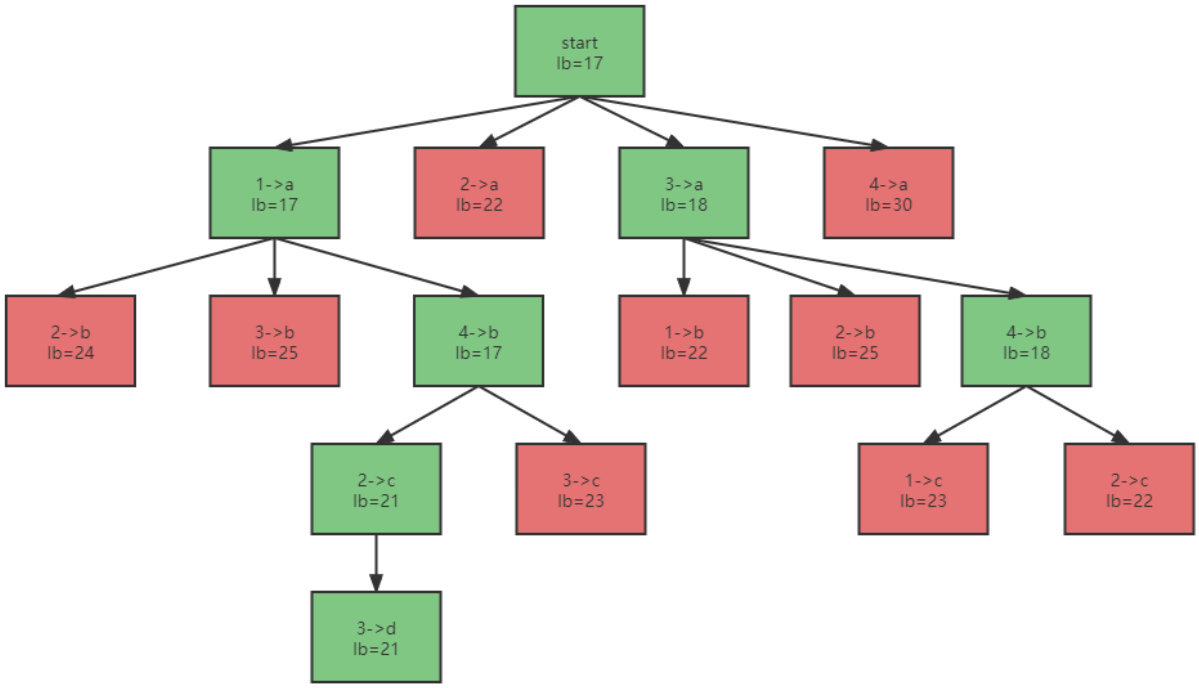

图 5 解空间树

【答案】

最终所求最优解为 21。任务分配方案如下：任务 1->人员 a，任务 4->人员 b，任务 2->人员 c，任务 3->人员 d。

问题三：0-1 背包分支限界代码设计。

【上下界确定】假设 n 中物品已按单位价值从大到小排序，一种最有效的下界就是贪心的根据价值重量比来选择物品，下界则考虑成每个物品可以选取多次，那么上界就是在背包容纳允许范围内全部装价值重量比最大的物品。

【中间过程】我们定义限界函数为：$ub = v + (W - w) * (v_{i+1}/w_{i+1})$。在背包处理每个物品的时候我们都考虑装或者不装两种情况，然后计算新状态的上界，如果符合初始估计的上下界那么我们就将新状态放入一个优先队列作为下次考虑的状态，否则说明可以进行剪枝优化。

【问题】给定n中物品和一个容量为W的背包，物品i的重量为w_i，其价值为v_{i}，对每种物品i只有两种选择：装入背包或不装入背包，如何选择装入背包的

物品，使得装入背包中物品的总价值最大？

【源码】

```
#include <bits/stdc++.h>
#pragma GCC optimize(2)
#pragma G++ optimize(2)
#define endl "\n"
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int maxn = 55;

int lb, ub;
pii a[maxn];

struct node {
    int w, v, bound, step;
    bool operator<(const node &A) const {
        return this->bound > A.bound;
    }
    node(int w, int v, int bound, int step) : w(w), v(v), bound(bound),
step(step) {}
};

void solve() {
    int n, W;
    cin >> n >> W;
    for (int i = 1; i <= n; ++i)
        cin >> a[i].first >> a[i].second;
    sort(a + 1, a + 1 + n, [=](pii A, pii B) {
        return A.second * 1.0 / A.first > B.second * 1.0 / B.first;
    });
    int tmp = W;
    for (int i = 1; i <= n; ++i)
        if (tmp >= a[i].first) tmp -= a[i].first, lb += a[i].second;
    tmp = W;
    ub = a[1].second * 1.0 / a[1].first * tmp;
    priority_queue<node> que;
    que.push(node(0, 0, 0, 0));
    int ans = 0;
    while (que.size()) {
        auto now = que.top();
        ans = max(ans, now.v);
        que.pop();
```

```
        auto nex = now;
        nex.step = now.step + 1;
        if (now.step == n + 1) continue;
        //先考虑不装进去
        auto tmp = nex;
        if (W - now.w >= a[tmp.step].first) {
            tmp.w = tmp.w + a[tmp.step].first;
            tmp.v = tmp.v + a[tmp.step].second;
            if (tmp.step != n)
                tmp.bound = tmp.v + (W - tmp.w) * a[tmp.step + 1].second
* 1.0 / a[tmp.step + 1].first;
            if (tmp.bound >= lb && tmp.bound <= ub) que.push(tmp);
        }
        tmp = nex;
        if (tmp.step != n)
            tmp.bound = tmp.v + (W - tmp.w) * a[tmp.step + 1].second *
1.0 / a[tmp.step + 1].first;
        if (tmp.bound >= lb && tmp.bound <= ub) que.push(tmp);
    }
    cout << ans << endl;
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    solve();
    return 0;
}
```