第六次作业

一、算法分析题

6-1 0-1背包问题的栈式分支限界法

问题表达

对于给定的n个物体与一个背包，假设第i个物体的重量为w_i，价值为v_i，背包总容量为c。令$\vec{x}=(x_1,x_2,\ldots,x_n)$为其解集空间，$x_i\in\{0,1\}$代表是否装入第$i$个物体，则其解集空间为子集树。问题可表达为

$$\max\sum_{i=1}^{n}x_iv_i$$
$$s.t.\begin{cases}\sum_{i=1}^{n}x_iw_i\le c\\ x_i\in\{0,1\}\end{cases}$$

算法步骤

首先将根节点加入活结点列表并作为扩展结点，对于活结点列表，如果为空则输出最优解，否则判断左儿子节点是否为可行节点，可行则将其加入活结点列表。之后判断右儿子节点，由于右儿子节点肯定为可行节点，判断右儿子节点的最大可能价值是否大于最优价值，大于则加入活结点列表。之后从活结点列表中按照栈式后进先出的原则选取下一个节点。

伪代码

```
while (i != n+1) {  // 非叶结点
    // 检查当前扩展结点的左儿子结点
    Typew wt = cw + w[i];
    if (wt <= c) {
        // 左儿子结点为可行结点
        if (cp+p[i] > bestp)
            bestp = cp+p[i];
        AddLiveNode(up, cp+p[i], cw+w[i], true, i+1);
    }
    up = Bound(i+1);
    // 检查当前扩展结点的右儿子结点
    if (up >= bestp) // 右子树可能含最优解
        AddLiveNode(up, cp, cw, false, i+1);
    // 按照LIFO的原则选取下一个扩展结点
}
```

区别

栈式分支限界法与回溯法主要的区别在于对当前扩展结点的扩展方式不同，栈式分支界限法每个节点只扩展一次，并一次性产生所有子节点，二回溯法每个节点可多次扩展直到子节点扩展完。

二、算法设计题

6-1 最小长度电路板排列问题

问题表达

由于此问题为求所给定N块电路板的最佳排列，即解空间为排列树。令解向量为$\vec{x}=(x_1,x_2,\ldots,x_n)$，$x_i\in\{1,2,3,\ldots,n\}$，$\vec{x}$为$\{1,2,3,\ldots,n\}$的一个排列。对于一个特定排列$\vec{x}$及连接块$N_k$，设$f_k(\vec{x})=\max_{x_i,x_j\in N_k}\{|i-j|\}$为此时连接块的最大长度，则搜索目标为

$$\min_{\vec{x}}\{\max_{k}\{f_k(\vec{x})\}\}$$

算法步骤

电路板排列问题的解空间是一颗排列树。采用队列式分支限界法找出所给电路板的最小长度排列。算法采用队列，每一个的节点$node$包含域x，表示节点所相应的电路板排列；s表示该节点已确定的电路板排列$x[1:s]$；cd表示当前最大长度。

算法开始时，将排列树的根结点置为当前扩展结点。在do-while循环体内算法依次从队列中取出结点作为当前扩展结点。算法将当前扩展节点分两种情形处理：

1)首先考虑$s=n-1$的情形，当前扩展结点是排列树中的一个叶结点的父结点。x表示相应于该叶结点的电路板排列。计算出与x相应的最大长度并在必要时更新当前最优值和相应的当前最优解。

2)当$s<n-1$时，算法依次产生当前扩展结点的所有儿子结点。对于当前扩展结点的每一个儿子结点$node$，计算出其相应的最大长度$node.cd$。当$node.cd<bestd$时，将该儿子结点$node$插入到活结点队列中。

伪代码

```
int search(){
    queue<Node> q;
    Node enode;
    while(true){
        if(enode.dep == n-1){ //仅一个儿子结点，已经排完n-1个电路板，更新最优值
            if(enode.cd < bestd)
                bestd = enode.cd;
        }
        else{
            int cur = enode.dep + 1;
            for(i=enode.dep+1; i<=n; i++){  //产生当前扩展结点的所有儿子结点
                now.cd = now.len();
                if(now.cd < bestd){
                    q.push(now);
                }
            }
        }
        if(q.empty())
            break;
        else
            q.delete(enode);  //下一层扩展结点
    }
    return bestd;
}
```

6-2 最小权顶点覆盖问题

问题表达

对于一个无向图G，边集合E和节点集合V，令$x_i\in\{0,1\}$表示第i个节点是否选中，则其解空间为子集树，令集合U为选中节点组成的集合搜索目标为

$$\min \sum_{i=1}^{n} x_i w_i$$
$$s.t. \forall (u,v) \in E, \text{有} u \in U \ or \ v \in U$$

本问题可采用优先级队列，对于已经搜索到的节点，选取权值和最小的当作扩展结点。

可行性约束

选中的节点是否满足图G的一个顶点覆盖，若满足，更新最小值，并停止对该子树的搜索

限界约束

若已选节点的权重和下界大于当前最优值，则停止对该子树搜索。

边界条件

当搜索到叶子节点检查选中的节点是否满足图G的一个顶点覆盖，若满足，更新最小值，继续通过优先级队列搜索。

算法步骤

算法开始时，将子集树的根结点置为当前扩展结点。在do-while循环体内算法依次从优先队列中取出权值结点作为当前扩展结点。算法将当前扩展节点分三种情形处理，即可行性约束，限界约束与边界条件。

伪代码

```
int search()
{
    //建立初始空堆
    HeapNode E;
    int i=1,cn=0;
    while(true)
    {
        if(i>n)
        {
            if(cover(E))//边界条件
            {
                for(int j=1;j<=n;j++)
                    bestx[j]=E.x[j];
                bestn=cn;
                break;
            }
        }
        else
        {
            if(!cover(E))
                AddLiveNode(i,true);//加入结点标号为i 的结点到顶点覆盖集中，并把更新后的结点再插入堆中
            AddLiveNode(i,false); //不把结点标号为 i 的结点加入到顶点覆盖集中，并把更新后的结点插入堆中
        }
        if(H.IsEmpty())break;
        H.RemoveMin(E);  //优先级队列
        cn=E.cn;
        i=E.i+1;
    }
    reutrn bestn
}
```

三、算法实现题

6-3 最小重量机器设计问题

问题表达

设x_i表示第i个部件购得的供应商，则解空间为子集树，问题可表达为

$$\min \sum_{i=1}^{n} w[i][x_i]$$

$$s.t. \sum_{i=1}^{n} c[i][x_i] \leq d$$

算法

采用优先级队列，类似于单源最短路径，使用当前节点的重量作为优先级，重量小优先级高。由于w_{ij}不是负值，当前节点所对应的当前机器重量和是解空间中以该节点为根的子树的中所有节点所对应的重量和的下界。当不满足总价格不超过d的要求时，则剪枝。当搜索到深度n时，即搜索到了叶节点，不再进行扩展节点的操作，而是针对于叶节点所对应的最小值，反向求得该节点所对应的的路径。

代码

见附件，直接在附件文件夹运行$min_w.py$即可。