算法基础 HW5

PB18111697 王章瀚

2020 年 11 月 12 日

1

使用链表表示和加权合并启发式策略，写出Make-Set, Find-Set和Union操作的伪代码

详细说明已经注释清楚, 就不再赘述了.

Algorithm 1 链表表示和加权合并启发式策略的不相交集——Make-Set

```
1: function Make-Set(int key)
2:     node = new Node
3:     disjset = new DisjSet
4:     node.key = key
5:     node.set = list                          ▷ 该结点的所属集合指向该链表
6:     disjset.head = node
7:     disjset.tail = node                      ▷ 链表头尾指向该结点
8:     disjset.size = 1
9:     return disjset
```

Algorithm 2 链表表示和加权合并启发式策略的不相交集——Find-Set

```
1: function Find-Set(x)
2:     return x.set                             ▷ 直接返回其对应集合
```

Algorithm 3 链表表示和加权合并启发式策略的不相交集——Union

```
1: function Union(disjset1, disjset2)
2:     if disjset1.size ¡ disjset2.size then
3:         Swap(disjset1, disjset2)             ▷ 启发式: 保证disjset1大小比较小
4:     for node in disjset2 do                  ▷ 将disjset2的结点的集合改为disjset1
5:         node.set = disjset1
6:     disjset1.tail.next = disjset2.head
7:     disjset1.tail = disjset2.tail            ▷ 将disjset2的结点加入disjset1
8:     disjset2.head = zdisjset2.tail = NULL    ▷ 清空disjset2
9:     disjset1.size = disjset1.size + disjset2.size   ▷ 更新disjset1大小
10:    disjset2.size = 0                        ▷ 更新disjset2大小
```

2

设定动态规划算法求解0-1背包问题，要求运行时间为O(nW),n为商品数量，W是小偷能放进背包的最大商品总重量

Algorithm 4 0-1背包问题

Require: weight: 重量数组, value: 价值数组, n: 数目, W: 背包最大重

```
1: function PACKAGE01(weight, value, n, W)
2:     r = new array[W]                          ▷ r[i]表示大小为r[i]的包最大价值
3:     for i = 1 → W do                          ▷ 放第0个物品
4:         r[i] = weight[0] ≤ i ? value[0] : 0   ▷ 当放得下物品i的时候才放
5:     for i = 1 → n-1 do                        ▷ 考虑1-n的每个物品是否放
6:         for j = W → weight[i] do              ▷ 考虑所有能放下物品i的包大小
7:             r[j] = max(r[j], r[j-weight[i]]+value[i])   ▷ 当取了总价值能变大才取
       return r[W]                               ▷ 返回最大总价值
```