HW 5

肖桐 PB18000037

2020年11月17日

解 1.

```
S
    *MAKESET(int x)
  {
2
       pSet = (S *)malloc(sizeof(S)); //pSet 指向一个新的集合头节点
                                              //pNew 指向一个新的数据节点
       pNew = (Node *)malloc(sizeof(Node));
4
       pNew->Value = x;
5
       pNew->pNext = NULL;
6
       pNew->Set = pSet;
                           //指向头节点
       pSet->head = pNew;
       pSet->tail = pNew;
9
       pSet->Length = 1;
10
       return pSet;
11
  }
12
13
  S *FIND SET(Node *x)
14
  {
15
       return x->Set;
16
  }
17
18
  S *UNION(Node *p1, Node *p2)
19
   {
20
       S *pSet1 = FIND_SET(p1);
21
       S *pSet2 = FIND_SET(p2);
22
       if (pSet2->Length > pSet1->Length)
23
       {
24
           Node *z = pSet1->head;
25
           while (z != NULL)
26
           {
27
               z->Set = pSet2;
28
               z = z - pNext;
29
           }
30
           pSet1->tail->pNext = pSet2->head;
31
           pSet2->head = pSet1->head;
           pSet2->Length += pSet1->Length;
33
```

```
free(pSet1);
34
            return pSet2;
35
        }
36
        else
37
        {
38
            Node *z = pSet2->head;
39
            while (z != NULL)
            {
41
                 z->Set = pSet1;
42
                 z = z \rightarrow pNext;
43
             }
44
            pSet2->tail->pNext = pSet1->head;
45
            pSet1->head = pSet2->head;
46
            pSet1->Length += pSet2->Length;
47
            free(pSet2);
48
            return pSet1;
49
        }
   }
51
```

解 2. 设 Value[i,j] 为选择前 i 个物品中的其中几个加入背包中,且使得总重量不超过 j 所取得的最大价值. 初始化 Value[n+1,W+1]

设 n 件物品重量为 w_1, w_2, \ldots, w_n , 对应的价值分别为 v_1, v_2, \ldots, v_n .

则有最优子结构: $Value[i,j] = \max\{Value[i-1,j], Value[i-1,j-w_i] + v_i\}$.

理解为每次尝试将最新纳入考虑的第i件物品装入背包,若将第i件物品装入背包能够使得总价值增加的话,那就将第i件物品装入背包,否则保持原来的状态.

因此采取自底向上的方法可以写出伪代码:

```
for (int i = 0; i < n + 1; i++)</pre>
  {//初始化
       Value[i, 0] = 0;
3
4
  for (int j = 0; j < W + 1; j++)
  {//初始化
       Value[0, j] = 0;
  }
8
  for (int i = 1; i < n + 1; i++)
10
11
       for (int j = 1; j < W + 1; j++)
12
       {
13
           Value[i, j] = max{Value[i - 1, j], Value[i - 1, j - w_i] + v_i};
14
       }
15
16
```

因为内循环复杂度为 O(nW), 因此算法总复杂度为 O(nW).