STUACM 第三次集训

2019/11/02

- **01**背包的优化
  - 再次观察01背包样例演示的过程，有没有发现前面一些状态对于后面求结果而言，都只是临时发挥作用而已，能不能从这里入手优化一下空间？

```
for(int i=1;i<=n;++i){
    for(int j=v;j>0;--j){
        if(j>=w[i])
            // 优化状态转移方程
            max_value[j]=max(max_value[j-w[i]]+v[i],max_value[j]);
    }
}
```

  - 重点理解第二层循环为什么j要求从v到0，而不能是从0到v？

    假设只有一个物品，重量**2**，价值**3**。背包容量为**7**。

    - 按优化后的代码模拟一次，第二层for循环j从**v**到**0**

```
for(int j=7;j>=0;--j){
    if(j>=w[1])    // 仅有一个物品，w[1]=2，v[1]=3
        max_value[j]=max(max_value[j-2]+3,max_value[j]);
}
```

j	max_value[j]
7	3
6	3
5	3
4	3
3	3
2	3
1	0
0	0

    - 我们探究一下，如果第二层for循环j从**0**到**v**，是什么意义？

```
for(int j=0;j<=7;++j){
    if(j>=w[1])    // 仅有一个物品，w[1]=2，v[1]=3
        max_value[j]=max(max_value[j-2]+3,max_value[j]);
}
```

j	max_Value[j]
0	0
1	0
2	3
3	3
4	6
5	6
6	9
7	9

效果是：不是一个物品，而是把它看作了一种物品，在背包容量允许范围内，可以无限次取！

- 为什么？

  因为优化为1维后，$max_Value[j]$的值又填回了原来的位置，j从小到大遍历时，小一点的值已经是考虑过物品i，然后把递推结果存回去了，到大一点的值的时候，又在小一点的值考虑过后的结果上有考虑了一次。而j从大到小遍历的时，可以保证，每一个值考虑物品i后，不会再被用于考虑第二遍。2维优化成1维时，需要在递推过程中注意时序问题。

- 01背包习题（开始上升为一般动态规划，先自行思考与“经典01背包”的相同及不同点，没有思路再参考题解或代码）

  音量调节

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mxn = 100, mxL = 2000;
bool dp[mxn][mxL] = {0};    // 与经典01背包不同的是，这里每个状态对应的不是最大价值，而是该状态可不可行
int cL[mxn];
int main(){
    int n, beginLevel, maxLevel;
    cin >> n >> beginLevel >> maxLevel;
    for(int i=1;i<=n;++i) cin >> cL[i];

    dp[0][beginLevel] = true;    // 初始化基本状态

    for(int i=1;i<=n;++i){
        for(int j=maxLevel;j>=0;--j){
            // 与经典01背包不同的是，这里不是选或不选，而是加或减
            if(j-cL[i]>=0){    // 通过上一状态加该变化转移而来
                dp[i][j] = dp[i][j] || dp[i-1][j-cL[i]];
            }
            if(j+cL[i]<=maxLevel){    // 通过上一状态减该变化转移而来
                dp[i][j] = dp[i][j] || dp[i-1][j+cL[i]];
            }
        }
    }

    int ansL = -1;
```

```
    // 找出最大的可行方案
    for(int j=maxLevel;j>=0;--j){
        if(dp[n][j]){
            ansL = j;
            break;
        }
    }
    cout << ansL << endl;
}
// 该题无法进行空间优化，为什么？
```

饭卡

Proud Merchants

其他以01背包为基础的变种题参考该博客

- 完全背包——动态规划中的一类题
  - 适用问题：有N种物品和一个容量为V的背包，每种物品都有无限件可用。第i种物品的费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。
  - 例题：Piggy-Bank
  - 核心代码：

```
for(int i=1;i<=n;++i){
    for(int j=0;j<=v;++j){
        if(j>=w[i])
            max_value[j]=max(max_value[j-w[i]]+v[i],max_value[j]);
    }
}
```

- 二分思想
  - 参考材料
  - 引子：
    - **猜数游戏**：两个人A，B做游戏，A从$[0,100)$之中选一个整数x，不告诉B，让B猜。若B猜y，**y>x**，则A反馈"比y小"；若B猜y，**y<x**，则A反馈"比y大"；若B猜y，**y=x**，则游戏结束。请问，**多次游戏**，用怎样的方法，可以使A**随机**选值，B猜中x所需次数的平均**值最小**？
    - 对于每局游戏，A随机选好一个x，B从0到99遍历一次的猜

```
// 伪代码
int x = ?;
for(int y=0;y<100;y++){
    if(check(y))    // 检测是否猜对。x==y -> 0; x>y -> 1; x<y -> -1
        break;
}
```

这样猜的平均次数是：$(1+100)*100/2/100=50.5$

    - 为什么效率这么低呢？
      这样猜的过程中，要么y=x游戏结束，要么y<x，A反馈"比y大"，然后y增大1，继续猜。每一轮B都只能排除一个值，一步一步的逼近答案，当然慢了。
    - 加入一个策略，不论x是什么，B第一次都先猜y=50

```
// 伪代码
int x = ?;
if(check(50)==-1){    // 检测是否猜对。x==y -> 0; x>y -> 1; x<y -> -1
  for(int y=51;y<100;y++){
      if(check(y))    // 检测是否猜对
          break;
  }
}else if(check(50)==0){

}else{
  for(int y=0;y<50;y++){
      if(check(y))    // 检测是否猜对
          break;
}
}
```

这样猜的平均次数是：
$(1/100)*1+(49/100)*(1/49)*(2+50)*49/2+(50/100)*(1/50)*(2+51)*50/2=26$

- 为什么只加入这样一个策略，平均次数就直接减为一半了？

  因为每次猜，要么直接猜对50，要么根据反馈，砍掉一半可能性，相当于下一次只用猜50或49个数中的一个，当然比一步一步逼近快啦！

- 由此得到启发

  每一次都猜所给范围中的中间值，那么，要么直接猜中，要么得到反馈后直接砍掉一半可能性。

- 有序序列中的二分查找
  - 给定序列，找序列中第一个大于等于x的值的下标

i	1	2	3	4	5	6	7	8
a[i]	2	4	5	7	12	13	15	17

  - 递归实现

    数据结构实验之查找四：二分查找

  - 非递归实现

```
int x;
int a[] = {0,2,4,5,7,12,13,15,17};
bool check(int i){
    return (a[i] >= x);
}
int binarySearch(int l, int r)    // 表示区间[l,r]
{
    while(l < r)
    {
        int mid = (l + r) / 2;
        if(check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

x=12， binarySearch(1,8)？

上面这个代码运行的时候有可能出问题吗？

x = 8，binarySearch(1,8)？

- 为什么会陷入死循环？

  因为 '/' 是整除，当$r = l + 1$时，$mid = (l + r)/2 = l$，又刚好条件使得$l = mid$，则会进入死循环。

- 推荐的写法：半开半闭

```
// 给定序列，找序列中第一个大于等于x的值的下标（求下界）
// x=8，binarySearch(0,8)
int x;
int a[] = {0,2,4,5,7,12,13,15,17};
bool check(int i){
    return (a[i] >= x);
}
int binarySearch(int l, int r)    // 表示区间(l,r]
{
    while(l + 1 < r)
    {
        int mid = (l + r) / 2;
        if(check(mid)) r = mid;
        else l = mid;
    }
    return r;
}
```

```
// 给定序列，找序列中尽量靠后的小于等于x的值的下标（求上界）
// x=8, binarySearch(1,9)
int x;
int a[] = {0,2,4,5,7,12,13,15,17};
bool check(int i){
    return (a[i] <= x);
}
int binarySearch(int l, int r)    // 表示区间[l,r)
{
    while(l + 1 < r)
    {
        int mid = (l + r) / 2;
        if(check(mid)) l = mid;
        else r = mid;
    }
    return l;
}
```

- 怎样的题可以采用二分来求解？
  - 思考：想找最小的i使b[i]为true，我可以从小到大遍历，直到b[i]==true结束。能否用二分的做法去找呢？

i	1	2	3	4	5	6	7	8
b[i]	false	false	true	true	false	false	true	false

  - 正在二分值x，x映射得到的结果为f(x)，f(x)呈现单调性。
- 二分步骤总结

        - 有一个暴力遍历（枚举），验证（check()）的算法
        - 判断是否可以用二分优化枚举过程
        - 若可以，改枚举过程为二分答案
    - 二分答案习题

      Aggressive cows

      Distributing Ballot Boxes

      Pie

      丢瓶盖

      书的复制