### 0-1背包问题

1. 实验要求
   1. 0-1背包问题：物品或者被装入背包，或者不被装入背包，设表示物品装入背包的情况，则当时，表示物品没有被装入背包，时，表示物品被装入背包。
   2. 编写满足下面要求的0-1背包算法

根据问题的要求，有如下约束条件和目标函数：

目标函数：

约束条件：

寻找一个满足约束条件，并使目标函数式达到最大的解向量：

* 证明该问题满足最优子结构
* 给出递推式子
* 基于动态规划实现算法
* 分析算法的复杂度
  1. 实现对上述0-1背包问题的改进
     1. 上述算法要求所给物品的重量必须是整数，而实际处理问题时无法避免物品的重量是小数的情况，试编写一个能够处理重量为小数的情况。
     2. 当背包容量很大时，算法需要计算的时间很大，该算法的时间复杂度在时为，在算法中，注意到是阶梯状单调不减函数。请试图改进该算法，提高算法复杂度。

1. 实验环境

VS Code 2019,C++

1. 实验内容
   1. 满足最优子结构的证明

设是所给0-1背包问题的一个最优解，则是下面一个子问题的最优解：

目标函数：

约束条件：

如若不然，设是上述子问题的一个最优解，则：



因此，



这说明是所给0-1背包问题比更优的最优解，导致矛盾。

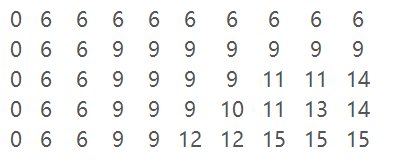
* 1. 递推式

令表示在前个物品中能够装入容量为的背包中的物品的最大值，则可得到如下递推式：



* 1. 动态规划算法
     1. 核心代码：

1. **for**(**int** i=1;i<=MAX\_NUM;i++){
2. **for**(**int** j=1;j<=maxCapacity;j++){
3. **if**(weight[i-1]>j)
4. dp[i][j]=dp[i-1][j];
5. **else**{
6. dp[i][j]=max(dp[i-1][j],
7. dp[i-1][j-weight[i-1]]+value[i-1]);
8. }
9. }
10. }
    * 1. 结果展示：
         1. 初始化：
11. **int** maxCapacity=10;
12. **int** weight[MAX\_NUM]={2,2,6,5,4};
13. **int** value[MAX\_NUM]={6,3,5,4,6};
    * + 1. 输出：



* 1. 算法复杂度的分析

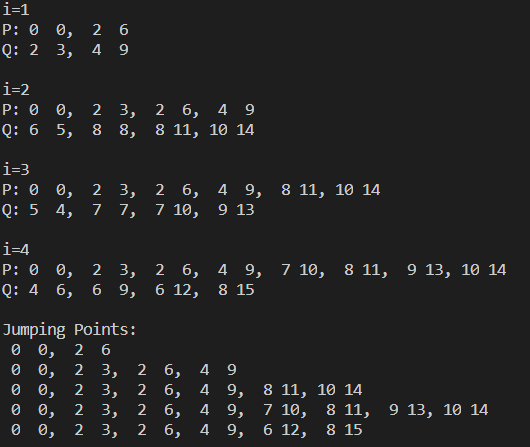
因为算法有两重循环，所以需要空间复杂度与时间复杂度均为

* 1. 大容量0-1背包问题
     1. 跳跃点定义
* 对于的递推公式容易证明，在一般情况下，对于每个确定的，函数是关于变量的**阶梯状单调不减函数**
* **跳跃点**是指在函数中函数值发生跃迁的点，可以用一个实数序偶描述，跳跃点是阶梯函数的描述特征，函数可由其全部跳跃点唯一确定
* 对于每个确定的，用一个表存储的全部跳跃点
  + 1. 跳跃点的递归求解：
       1. 定义跳跃点表**，**存储第个物品的跳跃点集合
       2. 函数的递归定义：

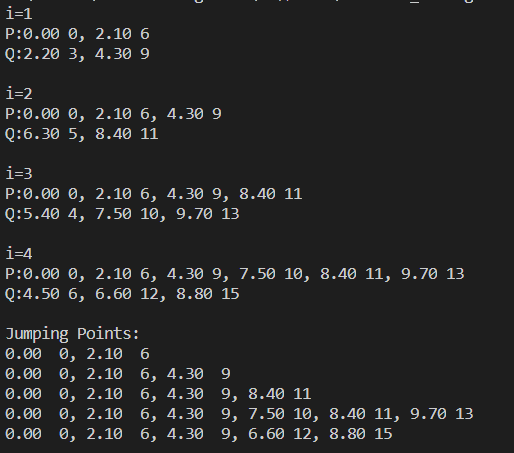


* + - 1. 跳跃点集：由函数决定，对应于不装第个物品的所有装包情形
      2. 跳跃点集：由函数决定，对应于装上第个物品的所有情形
      3. 计算：在跳跃点集与中除去这样的：且
    1. 核心代码：

1. jumpingPoint v[MAX\_NUM];
2. v[0].push\_back(make\_pair(0,0));
3. v[0].push\_back(make\_pair(weight[0],value[0]));
4. **for**(**int** i=1;i<MAX\_NUM;i++){
5. jumpingPoint p=v[i-1];
6. jumpingPoint q;
7. **int** len1=p.size();
8. **for**(**int** j=0;j<len1;j++)
9. **if**(weight[i]+p[j].first<=maxCapacity)
10. q.push\_back(make\_pair(weight[i]+p[j].first,value[i]+p[j].second));
12. **int** j=1;
13. **int** k=0;
14. **int** len2=q.size();
15. v[i].push\_back(make\_pair(0,0));
16. **while**(j!=len1&&k!=len2){
17. **if**(p[j].first<q[k].first){
18. **if**(p[j].second>v[i][v[i].size()-1].second)
19. v[i].push\_back(p[j]);
20. j++;
21. }
22. **else**{
23. **if**(q[k].second>v[i][v[i].size()-1].second)
24. v[i].push\_back(q[k]);
25. k++;
26. }
27. }
28. **if**(j==len1&&k!=len2){
29. **while**(k<len2){
30. **if**(q[k].second>v[i][v[i].size()-1].second)
31. v[i].push\_back(q[k]);
32. k++;
33. }
34. }
    * 1. 结果展示：
         1. 初始化：
    1. **int** maxCapacity=10;
    2. **int** weight[MAX\_NUM]={2,2,6,5,4};
    3. **int** value[MAX\_NUM]={6,3,5,4,6};
       * 1. 输出：



* 1. 小数情况的0-1背包问题
     1. 同样使用跳跃点方法，只是将数据类型改为double
     2. 结果展示：
        1. 初始化：
     3. **int** maxCapacity=10;
     4. **int** weight[MAX\_NUM]={2,2,6,5,4};
     5. **int** value[MAX\_NUM]={6,3,5,4,6};
        1. 输出：



1. 实验总结

动态规划用于解决多阶段决策问题，其特点有：

* 重复子问题
* 最优子结构
* 无后效性

动态规划的要点在于状态转移函数，可以从最小的子问题处思考。

1. 附录。

0-1bag.cpp

1. #include<stdio.h>
2. #define MAX\_NUM 5
3. #define max(a,b) ( ((a)>(b)) ? (a):(b) )
5. **int** maxCapacity;
7. **void** init(){
8. maxCapacity=10;
9. }
10. **void** solution(){
11. init();
12. **int** weight[MAX\_NUM]={2,2,6,5,4};
13. **int** value[MAX\_NUM]={6,3,5,4,6};
14. **int** dp[MAX\_NUM+1][maxCapacity+1];
15. **for**(**int** i=0;i<MAX\_NUM;i++)
16. **for**(**int** j=0;j<=maxCapacity;j++)
17. dp[i][j]=0;
18. **for**(**int** i=1;i<=MAX\_NUM;i++){
19. **for**(**int** j=1;j<=maxCapacity;j++){
20. **if**(weight[i-1]>j)
21. dp[i][j]=dp[i-1][j];
22. **else**{
23. dp[i][j]=max(dp[i-1][j],dp[i-1][j-weight[i-1]]+value[i-1]);
24. }
25. }
26. }
27. **for**(**int** i=1;i<=MAX\_NUM;i++){
28. **for**(**int** j=1;j<=maxCapacity;j++){
29. printf("%2d ",dp[i][j]);
30. }
31. printf("\n");
32. }
33. }
34. **int** main(){
35. solution();
36. }

Improved-0-1bag.cpp

1. #include<stdio.h>
2. #include<vector>
3. #define MAX\_NUM 5
4. #define max(a,b) ( ((a)>(b)) ? (a):(b) )
5. **using** **namespace** std;
7. **typedef** vector<pair<**int**,**int**> > jumpingPoint;
9. **int** maxCapacity;
11. **void** init(){
12. maxCapacity=10;
13. }
15. **void** solution(){
16. init();
17. **int** weight[MAX\_NUM]={2,2,6,5,4};
18. **int** value[MAX\_NUM]={6,3,5,4,6};
19. jumpingPoint v[MAX\_NUM];
20. v[0].push\_back(make\_pair(0,0));
21. v[0].push\_back(make\_pair(weight[0],value[0]));
22. **for**(**int** i=1;i<MAX\_NUM;i++){
24. jumpingPoint p=v[i-1];
25. jumpingPoint q;
26. **int** len1=p.size();
27. **for**(**int** j=0;j<len1;j++)
28. **if**(weight[i]+p[j].first<=maxCapacity)
29. q.push\_back(make\_pair(weight[i]+p[j].first,value[i]+p[j].second));
31. **int** j=1;
32. **int** k=0;
33. v[i].push\_back(make\_pair(0,0));
34. printf("i=%d\n",i);
35. printf("P:");
36. **for**(**int** a=0;a<len1;a++)
37. **if**(a!=len1-1)
38. printf("%2d %2d, ",p[a].first,p[a].second);
39. **else**
40. printf("%2d %2d\n",p[a].first,p[a].second);
41. printf("Q:");
42. **int** len2=q.size();
43. **for**(**int** a=0;a<len2;a++)
44. **if**(a!=len2-1)
45. printf("%2d %2d, ",q[a].first,q[a].second);
46. **else**
47. printf("%2d %2d\n",q[a].first,q[a].second);
48. **while**(j!=len1&&k!=len2){
49. **if**(p[j].first<q[k].first){
50. **if**(p[j].second>v[i][v[i].size()-1].second)
51. v[i].push\_back(p[j]);
52. j++;
53. }
54. **else**{
55. **if**(q[k].second>v[i][v[i].size()-1].second)
56. v[i].push\_back(q[k]);
57. k++;
58. }
59. }
60. **if**(j==len1&&k!=len2){
61. **while**(k<len2){
62. **if**(q[k].second>v[i][v[i].size()-1].second)
63. v[i].push\_back(q[k]);
64. k++;
65. }
66. }
67. **else** **if**(j!=len1&&k==len2){
68. **while**(j<len1){
69. **if**(p[j].second>v[i][v[i].size()-1].second)
70. v[i].push\_back(p[j]);
71. j++;
72. }
73. }
74. printf("\n");
75. }
76. printf("Jumping Points:\n");
77. **for**(**int** i=0;i<MAX\_NUM;i++){
78. **int** len=v[i].size();
79. **for**(**int** j=0;j<len;j++)
80. **if**(j!=len-1)
81. printf("%2d %2d, ",v[i][j].first,v[i][j].second);
82. **else**
83. printf("%2d %2d\n",v[i][j].first,v[i][j].second);
84. }
85. }
87. **int** main(){
88. solution();
89. }

Demical-0-1bag.cpp

1. #include<stdio.h>
2. #include<vector>
3. #define MAX\_NUM 5
4. #define max(a,b) ( ((a)>(b)) ? (a):(b) )
5. **using** **namespace** std;
7. **typedef** vector<pair<**double**,**int**> > jumpingPoint;
9. **int** maxCapacity;
11. **void** init(){
12. maxCapacity=10;
13. }
15. **void** solution(){
16. init();
17. **double** weight[MAX\_NUM]={2.1,2.2,6.3,5.4,4.5};
18. **int** value[MAX\_NUM]={6,3,5,4,6};
19. jumpingPoint v[MAX\_NUM];
20. v[0].push\_back(make\_pair(0.0,0));
21. v[0].push\_back(make\_pair(weight[0],value[0]));
22. **for**(**int** i=1;i<MAX\_NUM;i++){
24. jumpingPoint p=v[i-1];
25. jumpingPoint q;
26. **int** len1=p.size();
27. **for**(**int** j=0;j<len1;j++)
28. **if**(weight[i]+p[j].first<=maxCapacity)
29. q.push\_back(make\_pair(weight[i]+p[j].first,value[i]+p[j].second));
31. **int** j=1;
32. **int** k=0;
33. **int** len2=q.size();
34. v[i].push\_back(make\_pair(0.0,0));
35. printf("i=%d\n",i);
36. printf("P:");
37. **for**(**int** a=0;a<len1;a++)
38. **if**(a!=len1-1)
39. printf("%.2f %d, ",p[a].first,p[a].second);
40. **else**
41. printf("%.2f %d\n",p[a].first,p[a].second);
42. printf("Q:");
43. **for**(**int** a=0;a<len2;a++)
44. **if**(a!=len2-1)
45. printf("%.2f %d, ",q[a].first,q[a].second);
46. **else**
47. printf("%.2f %d\n",q[a].first,q[a].second);
48. **while**(j!=len1&&k!=len2){
49. **if**(p[j].first<q[k].first){
50. **if**(p[j].second>v[i][v[i].size()-1].second)
51. v[i].push\_back(p[j]);
52. j++;
53. }
54. **else**{
55. **if**(q[k].second>v[i][v[i].size()-1].second)
56. v[i].push\_back(q[k]);
57. k++;
58. }
59. }
60. **if**(j==len1&&k!=len2){
61. **while**(k<len2){
62. **if**(q[k].second>v[i][v[i].size()-1].second)
63. v[i].push\_back(q[k]);
64. k++;
65. }
66. }
67. **else** **if**(j!=len1&&k==len2){
68. **while**(j<len1){
69. **if**(p[j].second>v[i][v[i].size()-1].second)
70. v[i].push\_back(p[j]);
71. j++;
72. }
73. }
74. printf("\n");
75. }
76. printf("Jumping Points:\n");
77. **for**(**int** i=0;i<MAX\_NUM;i++){
78. **int** len=v[i].size();
79. **for**(**int** j=0;j<len;j++)
80. **if**(j!=len-1)
81. printf("%.2f %2d, ",v[i][j].first,v[i][j].second);
82. **else**
83. printf("%.2f %2d\n",v[i][j].first,v[i][j].second);
84. }
85. }
87. **int** main(){
88. solution();
89. }