天津大学

基于动态规划的 0/1 背包问题和最短路问题

王彦淞 许怿芃 曲晏林 吴炎 马凯

2020 年 12 月 23 日

姓名	学号	任务分工	成绩
许怿芃	3019244081	统筹规划，数据处理	
马凯	3019232189	编程实现	
曲晏林	3019244133	结果分析	
吴炎	3019244191	数据测试	
王彦淞	3019244191	实验报告撰写	

摘要

本实验针对 0/1 背包和最短路径的动态规划问题，利用动态规划的思想进行求解，对 Florida State University 的前三个数据集和数据量分别为 50、100、200 的数据集进行性能分析。实验结果表明动态规划问题不仅可以解决 0/1 问题，而且得到的是最优解。通过将递归求解过程进行无重复性优化，还可以把递归函数运行时间从不切实际的 $O(n^2)$ 降为切实可行的 $O\ cn$ 。动态规划算法求解时不仅递归易于实现，而且无重复性优化后复杂度急剧下降，利用迭代（复杂度和递归相同）还可以避免额外的附加的递归栈空间，由此可见优化良好的动态规划算法很适合求解 0/1 背包问题。代码只是无重复性优化之后的递归方法，并未实现迭代的方式，尽管迭代的方式因为避免了不必要的空间而更快，但主要目的是理解动态规划思想，暂不考虑同一思想中空间的影响。最短路径问题是图论研究中的一个经典算法问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。对于最短路问题，根据 Floyd 最短路径算法的三层循环，用堆来优化时间复杂度。

目录

1 实验目的

1.1 说明问题的背景及意义

1.1.1 0/1 背包问题

0/1 背包问题可以被应用到许多领域，在日常生活中也经常遇到类似的问题。比如每逢节假日各大商场都会推出各种促销活动，如购物满 500 元赠 200 元现金券, 并且现金券只能购物不找余额。这个购物问题就可以通过 0-1 背包问题解决, 假定要购买 10 件商品，每件商品有两项指标: 一个是重要度, 分为 4 等, 用 1 4 表示，数值越大表示该商品越重要; 另一个是 商品的价格。因此该购物问题就可以转换为如何选择这些商品使重要度价格的和最大，环超过 70 元的限制条件。0/1 背包问题是一个典型的 NP 难问题，可作为许多工业场合的应用模型，比如说资本预算、货物装载和存储分配问题等。对于该问题的求解，既有理论价值又有实际意义。

1.1.2 最短路径问题

最短路径问题广泛的应用于交通，网络寻优等领域。采用的算法也不仅仅是单运用于一般意义上的距离，也可以运用到其他的度量上。设计此算法是为了帮助我们在复杂的问题上，寻找到更为便捷，节约时间的结果。最短路径问题是图论研究中的一个经典算法问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。最短路径的现实意义是能广泛的应用于交通，城市出行的路径选择问题，通信网络的最可靠路，最大容量路问题等等。

1.2 课题的任务与目的

理解动态规划算法的原理和基本思想；通过本实验，加深对动态规划的理解，并且加深对 0/1 背包问题及 Floyd 最短路径问题的理解。利用动态规划策略实现 0/1 背包问题，并且比较动态规划算法对同一数据集的执行时间，将算法的时间复杂度降为 O（cn）。正如 1.1 所述，现实中有很多最短路径问题的例子。关于最短路径问题，目前人们采用最多的是 Dijkstra 算法，并在此基础上进行创新。Dijkstra 算法的思想是从 Vs 出发, 逐步地向外探寻最短路径。执行过程中, 与每 1 个点对应记录下 1 个数, 不断改变从 Vs 到该点的最短路径的上界最终求得最短路径。虽然这种算法被较广泛地应用, 但它只能解决单源点到达其余顶点的问题。我们试图从动态规划的角度和思路，根据 Floyd 算法，来实现任意两点间的最短路径问题，并且降低算法的时间复杂度。

1.3 解决方案及主要思路

1.3.1 0/1 背包问题

考虑到用动态规划解决此问题：
阶段：在前 N 件物品中，选取若干件物品放入背包中
状态：在前 N 件物品中，选取若干件物品放入所剩空间为 W 的背包中所能获得的最大价值。
决策：第 N 件物品放或者不放。
递归方程：

$$c[i,w]=\begin{cases}0,\ \text{if } i=0 \text{ or } w=0\\ c[i-1,w],\ \text{if } wi>w\\ \max(vi+c[i-1,w-w[i]],c[i-1,w],\ \text{if } i>0 \text{ and } w\geq w[i]\end{cases}$$

$c[i,w]$ 表示 其中将物品 i 装入背包中，记为“1”，并放入集合 S 中，将物品 i 不装入背包，记为“0”，最终可以形成子集 S，该子集可以最大化总价值。用堆来对算法进行优化，以降低算法的时间复杂度。
最短路径的 Floyd 算法，从图的带权邻接矩阵出发，其基本思想是从 vi 到 vj 的最短路径是以下各种可能路径中的长度最小者：若 $<vi,v1>,<v1,vj>$ 存在，则存在路径 $\{vi,v1,vj\}$(路径中所含顶点序号不大于 1)；
若 $\{vi,...,v2\}$ $\{v2,...,vj\}$ 存在，则存在一条路径 $\{vi,...,v2,...,vj\}$(路径中所含顶点不大于 2)。
以此类推，则 vi 到 vj 的最短路径是从上述这些路径中，路径长度的最小者。
现定义一个 n 阶方阵序列 $D^{(-1)},D^{(0)},D^{(0)},...,D^{(k)},...,D^{(n-1)}$, 其中：

$$D^{(-1)}[i][j]=G.arcs[i][j]$$
$$D^{(k)}[i][j]=MinD^{(k-1)}[i][j],D^{(k-1)}[i][k]+D^{(k-1)}[k][j],0\leq k\leq n-1$$

$D^{(1)}[i][j]$ 是从 v_i 到 v_j 的中间顶点的序号不大于 1 的最短路径长度；$D^{(k)}[i][j]$ 是从 v_i 到 v_j 的中间顶点的序号不大于 k 的最短路径长度；$D^{(k-1)}[i][j]$ 是从 v_i 到 v_j 的最短路径长度。这也是解决此算法的关键步骤

2 实验设计流程

2.1 流程图

针对 0/1 背包问题，采用如下流程图步骤；针对最短路问题，只设计前两步。

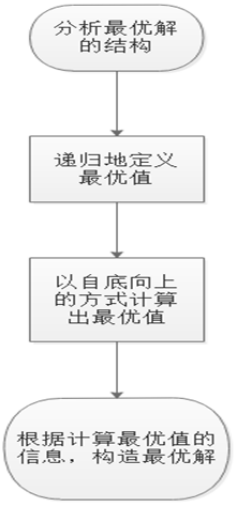

图 1: 流程图

2.2 0/1 背包问题的总体架构

2.2.1 分析最优解的结构

最优子结构即: 设 (x]. x2, ... xn) 是 0-1 背包问题的最优解，则 (x2. x3, ... xm) 是子问题 (从物品 2 至物品 n 中选取，装入背包) 的最优解。

2.2.2 递归定义最优值

在对该问题的解决中，有两种可能的情况，一种是背包容量不足以装入物品，背包不增加价值。另一种是背包容量可以装下物品，背包的价值增加了。对于第二种情况，可以再分为两种情况，第一种是把第 i 个物品放入背包中，背包的价值等于前 i-1 个物品装入容量 j-w[i] 中获得的价值加上第 i 个物品的价值；第二种是第 i 个物品的重量小于背包现在的容量，可以选择不装第 i 个物品，此时等于将前（i-1）个物品装入背包所获得的价值。我们的目标是使价值最大化，因此选这两种情况得到价值的最大值。通过对上述讨论，可以得到一个如下的递推公式：

$$c\left[i_{i,w}\right]=\begin{cases}0, \text{if } i=0 \text{ or } w=0\\ c[i-1,w], \text{ if } wi>w\\ \max(vi+c[i-1,w-w[i]], c[i-1,w], \text{ if } i>0 \text{ and } w\geq w[i]\end{cases}$$

将物品 i 装入背包中，记为“1”，并放入集合 S 中，将物品 i 不装入背包，记为“0”，最终可以形成子集 S，该子集可以最大化总价值。

2.2.3 计算最优值

最后，用一个二维数组 dp[i][j] 来记录迭代处理的结果，表示放入 0 i 这些物品，背包此时容量为 j。依次填入 dp[ ][ ] 得到迭代结果。

2.2.4 构造最优解

根据前面计算出的最优值 dp[][] 来构造最优解的 n 元 0-1 向量（x1，x2，…，xn）。代码中 if 条件语句的判断条件 dp[i][j] == dp[i - 1][j]) 表示当前物品 i 没有装入背包，否则 $j - w[i] >= 0 \&\& dp[i][j] == dp[i-1][j-w[i]] + v[i]$ 就表明当前物品 i 是装入背包的。

2.3 最短路的总体架构

2.3.1 分析最优解的结构

对于图 G<V,E> 的所有结点对最短路径的问题，一条最短路径的子路径都是最短路径。假设用邻接矩阵 W=w(i，j) 来表示输入带权图, 考虑从结点 i 到结点 j 的一条最短路径 p，如果 p 最多有 m(m 为有限值) 条边。若 i=j，则 p 的权值为 0 而且不包含其他边。若 i j，可以将 i 到 j 的路径转换为 i -> k、k->j。

2.3.2 递归定义最优值

根据最短路径问题思想和最优解的结构，可以得到最优解的递归公式：
$D[i][j] = MinD[i][j], D[i][k] + D[k][j], 0\ k\ n-1$
其中表示与边由连接的节点，如果 $i = j$，则 $D[i][j] = 0$。

2.4 0/1 背包问题的关键技术

关键技术 1：寻找最优解
约束关系：选择的所有物品总重量不超过背包的最大容纳量，就是
$W1X1 + W2X2 + \ldots + WnXn < capacity$ 递推关系式：
j<w(i) V(i,j)=V(i-1,j)
j>=w(i) V(i,j)=max {V(i-1,j)，V(i-1,j-w(i))+v(i)}
（1）包的剩余容量比物品小，则不选。那么选择的物品价值，重量不变。即 V(i,j)=V(i-1,j)
（2）有足够的容量装物品，但装了也不一定达到当前最优价值，所以在装与不装之间选择最优的一个，即 V(i,j)=max {V(i-1,j)，V(i-1,j-w(i))+v(i)}。其中 V(i-1,j) 表示不装，V(i-1,j-w(i))+v(i) 表示装了第 i 个商品，背包容量减少 w(i)，但价值增加了 v(i)；
关键技术 2：最优解回溯
目的：根据最优解回溯找出解的组成。

思想：V(i,j)=V(i-1,j) 时，说明没有选择第 i 个商品，则回到 V(i-1,j)；
V(i,j)=V(i-1,j-w(i))+v(i) 时，说明装了第 i 个商品，该商品是最优解组成的一部分，随后我们得回到装该商品之前，即回到 V(i-1,j-w(i))。一直进行到 i=0。

2.5 最短路的关键技术

关键技术 1：矩阵

将原本的点到点及其距离（加权），转化成一个初始矩阵。比如，初始数据：a b x，就说明了 a 到 b 有路径且权值为 x，则转化为矩阵中（a-1，b-1）的值为 x。

关键技术 2：Floyd 算法：

对矩阵的计算：十字交叉法

方法：两条线，从矩阵的左上角元素开始计算一直到右下角的元素。其核心语句如下：

```
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
           if(e[i][k]<inf && e[k][j]<inf&& e[i][j]>e[i][k]+e[k][j])
            e[i][j]=e[i][k]+e[k][j];
```

3 实验结果及复杂性分析

3.1 实验环境设置

x86-64 win10 编译器版本：g++ 8.1.0 codeblocks，vscode dev-C++

3.2 测试方法及性能评价指标

0/1 背包：由于数据量太小，而且计算时间的时候获取时间使用的函数是 now()，函数精度不够但是输出的精度足够，所以采用计算执行函数 1000000 次之后再人为除以次数输出时间的方式获取一次执行的平均时间，精度为小数点后 6 位。（使用语句 cout.setf(ios::fixed,ios::floatfield) 显示小数而不是科学计数法）

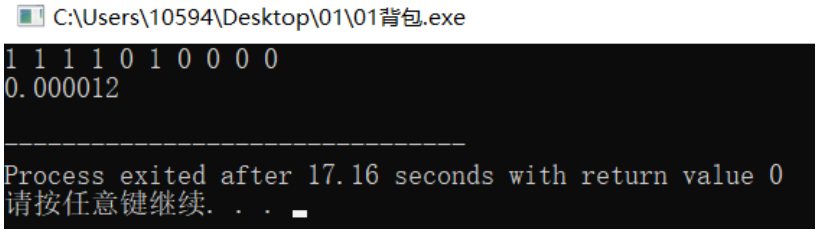

图 2: 数据集 1

图 3: 数据集 2

图 4: 数据集 3

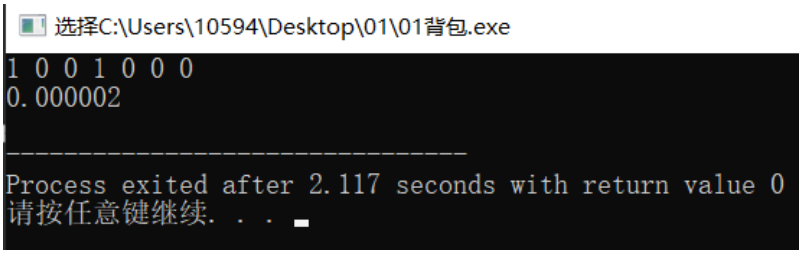

图 5: 数据集 4

图 6: 数据集 5

图 7: 数据集 6

3.3 实验结果分析

图 8: 最短路结果

3.4 实验结果分析

1. 0/1 背包问题：对所测试的数据集结果求平均值，结果如下：

数据集 / 测试次数	p_01	p_02	p_03	p_04	p_05	p_06
1	0.000012	0.000001	0.000005	0.000002	0.000004	0.000005
2	0.000012	0.000001	0.000005	0.000002	0.000004	0.000005
3	0.000007	0.000001	0.000005	0.000002	0.000004	0.000005
平均值	0.000010	0.000001	0.000005	0.000002	0.000004	0.000005

图 9: 01 背包问题结果

表中数据集数据的实例特征依次为 10，5，6，7，8，7。
能够看出数据集实例特征与测试结果的增长速度大致相同，所以使用 matlab 通过一次拟合得到图 3：

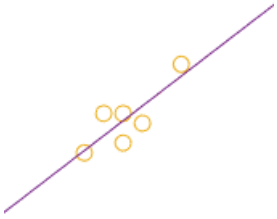

图 10: 01 背包拟合图像

由此可见，所测数据结果与直线十分接近，可以认为 0/1 背包问题的程序运行时间与实例特征成线性关系，即符合 t（n）= O（n），与理论分析的期望相符合。
2. 最短路问题：对所测试的数据集结果求平均值，结果如图 4：

数据集 / 测试次数	50	100	200	300	500
1	0.817	0.841	1.018	1.179	2.014
2	0.811	0.835	1.016	1.178	2.017
3	0.813	0.835	1.019	1.179	2.013
平均值	0.814	0.837	1.018	1.179	2.015

图 11: 最短路结果

可以看出随着数据集实例特征的增长，测试结果的增长速度越来越快，由于所测试数据相对较少，直观上仅能得到结果与数据集的实例特征之间并非一次线性相关，而是存在更高次幂的对应关系。

理论分析可知，程序运行的时间复杂度应该为 $O(n^3)$。使用 matlab 通过如下程序分别对二次和三次进行拟合，可以知道三次曲线的拟合更为准确。

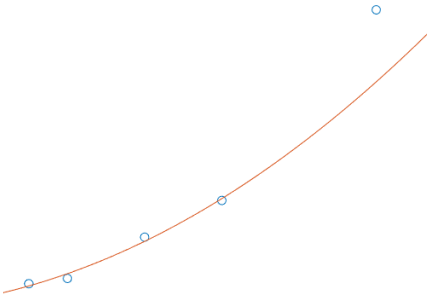

图 13: 二次曲线拟合

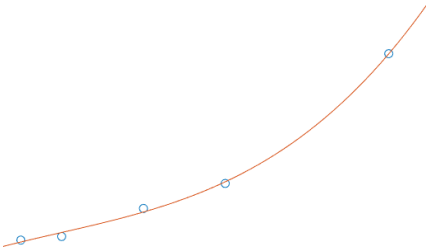

图 14: 三次曲线拟合

jie.m

```
x = [50 100 200 300 500]
y = [814 837 1018 1179 2015]
sum1 = 1000000;
c = 0;
for i = 2:3
    y2 = polyfit(x,y,i);
    Y = polyval(y2,x);
    if(sum((Y-y).^2) < sum1)
        c = i;
        sum1 = sum((Y-y).^2);
    end
end
c
```

命令行窗口

```
c =

     3
```

图 12: matlab 代码

从曲线图（图 13、图 14）也可更直观的得到结论：

4 代码实现

0/1 背包：

```
#include<iostream>
#include <bits/stdc++.h>
#include <windows.h>
#include <fstream>
#include <time.h>
#include <unistd.h>
#include <algorithm>
#include <chrono>
using namespace std;
using namespace chrono;
const int maxn = 1000;
const int times = 1000000;

int w[maxn];
int v[maxn];
const int maxv = 1000;
int dp[maxn][maxv] = { { 0 } };
int item[maxn];
int n, bagV;
void findMax() {            //动态规划
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= bagV; j++) {
      if (j < w[i])
        dp[i][j] = dp[i - 1][j];
      else
        dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
    }
  }
}

void findWhat(int i, int j) {        //最优解情况
  if (i >= 0) {
    if (dp[i][j] == dp[i - 1][j]) {
      item[i] = 0;
      findWhat(i - 1, j);
    }
    else if (j - w[i] >= 0 && dp[i][j] == dp[i - 1][j - w[i]] + v[i]) {
      item[i] = 1;
      findWhat(i - 1, j - w[i]);
    }
  }
}

void print() {
  for (int i = 1; i <= n; i++)       //最优解输出
```

```
46      cout << item[i] << ' ';
    cout << endl;
48  }

50  void totaltime() {
      for (int i = 0; i < times; i++)
52    {
        findMax();
54      findWhat(n, bagV);
      }
56  }

58  int main()
    {
60    w[0] = 0; v[0] = 0;
      freopen("p03_c.txt", "r", stdin);
62    cin >> n >> bagV;
      for (int i = 1; i <= n; i++) cin >> v[i];
64    for (int i = 1; i <= n; i++) cin >> w[i];
      fclose(stdin);
66
      findMax();
68    findWhat(n, bagV);
      print();
70    auto start = system_clock::now();
      totaltime();
72    auto end = system_clock::now();
      auto duration = duration_cast<microseconds>(end - start);
74    double timeCost1 = double(duration.count()) * microseconds::period::
          num / microseconds::period::den;
      cout.setf(ios::fixed, ios::floatfield);
76    cout << timeCost1 / times << endl;
      return 0;
78  }
```

最短路径问题：

```
    #include <cstdio>
2   #include <ctime>
    #include <iostream>
4   using namespace std;

6   const int MAP_SIZE = 1010;
    const double MAXN = 99999;
8
    double Graph[MAP_SIZE][MAP_SIZE];
10
```

```
    void init(int n) {
12      for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
14              Graph[i][j] = MAXN;
            }
16      for (int i = 0; i < n; i++) Graph[i][i] = 0;
    }
18
    void floyd(int n) {
20      int i, j, k;

22      for (j = 0; j < n; j++)
            for (i = 0; i < n; i++)
24              for (k = 0; k < n; k++) {
                    if (Graph[i][k] > Graph[i][j] + Graph[j][k])
26                      Graph[i][k] = Graph[i][j] + Graph[j][k];
                }
28  }

30  int main()
    {
32      freopen("200.txt", "r", stdin);
        freopen("out.txt", "w", stdout);
34
        int j, i; int ver = 200; double t;
36      init(ver);
        while (cin >> i >> j >> t) {
38          Graph[i][j] = t;
        }
40
        unsigned long long start = clock();
42      floyd(ver);
        unsigned long long end = clock();
44      for (int i = 0; i < ver; i++) {
            for (int j = 0; j < ver; j++) {
46              cout << Graph[i][j] << " ";
            }
48          printf("\n");
        }
50      fclose(stdin);
        fclose(stdout);
52      //printf("%lld\n", end - start);
        return 0;
54  }
```

5 结论与展望

本实验通过对 0/1 背包问题和最短路问题的实现及分析，验证了动态规划算法在实际中的应用。实验中，我们对所解决的问题的分析，并对算法进行实现，随后用代码随机生成了数据集，进行算法正确性的检测与性能分析。由于 0/1 背包数据量过小，为了检测算法运行时间，我们首先上网查询关于提高检测程序运行时间精度的函数，依旧无法测量，因此，我们选择采用统计程序多次运行时间来得到一次运行时间的办法，来提高测量的精度。对于最短路问题，我们利用代码随机生成数据量分别为 50、100、200 的数据集来进行时间分析，并且获得算法运行时间随数据范围的变化。与此同时，对于两个问题，我们都随机生成小数据进行算法正确性检测。美中不足的是，对于最短路问题，我们设计的算法对数据量要求较高（顶点个数必须小于 1000），并且对于两个问题都没有随机生成大数据，使用与暴力程序对拍的方法进行算法正确性验证，不够严谨。在日后对算法问题进行研究时，可以用对拍的方法严格地验证算法正确性。同时，希望在进一步的学习中能更深入理解动态规划算法在实际问题中的应用，解决由于栈空间的限制导致算法对顶点个数限制较大的问题。同时可以查阅更多资料，了解有关动态规划算法的优化，比如可以学习更加复杂的背包问题，如完全背包等，同时经过查阅资料，还可以将背包问题的空间复杂度降为 $O(n)$ 等。对于最短路问题，我们可以尝试进行优化，以降低算法的时间复杂度等。

6 参考文献

[1] 曹珊珊，动态规划在 0/1 背包问题中的应用与分析。
[2] 王茂波，Floyd 最短路径算法的动态优化。
[3] 严蔚敏, 吴伟民. 数据结构 [M]. 北京: 清华大学出版社, 1997:186-190.