目录

[一.广义背包 2](#_Toc26370566)

[1.问题重述 2](#_Toc26370567)

[2.符号说明 3](#_Toc26370568)

[3.问题描述以及递推公式 3](#_Toc26370569)

[4.求出解向量 4](#_Toc26370570)

[5.优化 4](#_Toc26370571)

[6.证明最优子结构(反证法) 5](#_Toc26370572)

[7.优化后的算法以及时空复杂度 6](#_Toc26370573)

[8.测试用例 7](#_Toc26370574)

[二.TSP 8](#_Toc26370575)

[1.问题描述 8](#_Toc26370576)

[2.符号说明以及递推公式 8](#_Toc26370577)

[3.例子 8](#_Toc26370578)

[4.状态压缩 9](#_Toc26370579)

[5.算法实现以及算法时空复杂度 9](#_Toc26370580)

[6.数据测试 12](#_Toc26370581)

[三.附录 13](#_Toc26370582)

# 一.广义背包

### 1.问题重述

广义背包问题的描述如下：给定载重量为M的背包和n种物品，每种物品有一定的重量和价值，现在需要设计算法，在不超过背包载重量的前提下，巧妙选择物品，使得装入背包的物品的总价值最大化。规则是，每种物品均可装入背包多次或不装入（但不能仅装入物品的一部分）。

### 2.符号说明

|  |  |
| --- | --- |
|  | 第i件物品的重量(为正整数) |
|  | 第i件物品的价值(为自然数) |
|  | 解向量 |
|  | 第i件物品转入背包的数量 |
| M | 背包最大承载重量 |
| n | 物品种类数量 |
|  | 最大价值 |
|  | 背包现有价值 |
|  | 第i件物品转入背包的数量 |
| i | 选择的第i种物品 |
| j | 背包剩余承载量 |
| k | 指示变量. |

### 3.问题描述以及递推公式

#### 问题描述

#### 递推公式

①

### 4.求出解向量

通过例子说明:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 2 | 6 | 7 | 8 | 12 | 13 | 14 | 18 | 19 |

### 5.优化

②

与0-1背包的对比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0-1背包 | | | | 广义背包 | | | |
| 质量 |  |  | … |  |  |  | … |  |
| 价值 |  |  | … |  |  |  | … |  |
| 装入数 |  |  | … |  |  |  | … |  |

转换: 广义背包中第i件物品,选了次 🡪 0-1背包的种物品

即:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 广义背包 | | | | | | | | |
| 质量 |  |  | … |  |  |  | … |  | … |
| 价值 |  |  | … |  |  |  | … |  | … |
| 装入数 |  |  | … |  |  |  | … |  | … |

转换后的公式

|  |  |
| --- | --- |
|  | 背包现有价值 |
|  | 物品的种类 |
|  | 转换前的物品的种类 |
|  | 第i件物品的质量 |
|  | 第i件物品转入背包的数量 |
| i | 选择的第i种物品 |
| j | 背包剩余承载量 |
| k | 下标变量 |
|  | 最优解 |

递推公式:

### 6.证明最优子结构(反证法)

通过转化,则证明广义背包的最优子结构与0-1背包类似

证明:

假设是公式③所对应的最优解，

则是下列子问题的最优解。

反证法:

证明:

假设不是上述子问题的最优解, 而是上述子问题的最优解.

则

因此

故为该问题的最优解,与假设中为该问题的最优解相矛盾.

证毕.

### 7.优化后的算法以及时空复杂度

/\*

\*递推公式

\*时空复杂度

*M:背包最大承重量*

*n:物品种类个数*

|  |  |
| --- | --- |
| 时间复杂度 |  |
| 空间复杂度 |  |

\*输入

*m:物品质量数组*

*M:背包最大承重量*

*n:物品种类个数*

\*输出

最大价值与选择物品数量

\*/

function **GKP**(M, m, v) {

    var n = m.length

    var i, j;

var x = {} *//解向量*

    for (i = 0; i <= n; i++) {

        x[i] = 0

    }

    var path = new **Map**()

    var f = new **Array**(M + 1).**fill**(0)

    for (i = 1; i <= n; i++) {

        for (j = m[i - 1]; j <= M; j++) {

            var tmp = f[j - m[i - 1]] + v[i - 1]

            if (f[j] < tmp) {

                f[j] = tmp

                path[[i, j]] = 1//用hash表一次储存 (i,j)

            }

        }

    }

*//解路径*

    j = M

    i = n

    while (i > 0 && j > 0) {

        if (path[[i, j]] == 1) {

            x[i - 1]++

            j = j - m[i - 1]

        } else {

            i = i - 1

        }

    }

    return [f[M], x]

}

*//广义背包测试数据*

var M = 10

var m = [1, 6, 4, 3]

var v = [1, 3, 2, 6]

**console**.**log**(**GKP**(M, m, v))

M = 230

m = [20, 25, 40, 12, 31]

v = [1, 2, 3, 1, 5]

**console**.**log**(**GKP**(M, m, v))

### 8.测试用例

|  |  |  |
| --- | --- | --- |
| 组号 | 测试输入 | 算法输出 |
| 1 | var M = 10  var m = [1, 6, 4, 3]  var v = [1, 3, 2, 6] | 最少花费 =  19  解向量 = [1,0,0,3] |
| 2 | M = 230  m = [20, 25, 40, 12, 31]  v = [1, 2, 3, 1, 5] | 最少花费 =  36  解向量 = [0,0,0,1,7] |

# 二.TSP

### 1.问题描述

所谓TSP问题是指旅行商要去n个城市推销商品，其中每个城市到达且仅到达一次，并且要求所走的路程最短（该问题又称货郎担问题、邮递员问题、售货员问题等）。

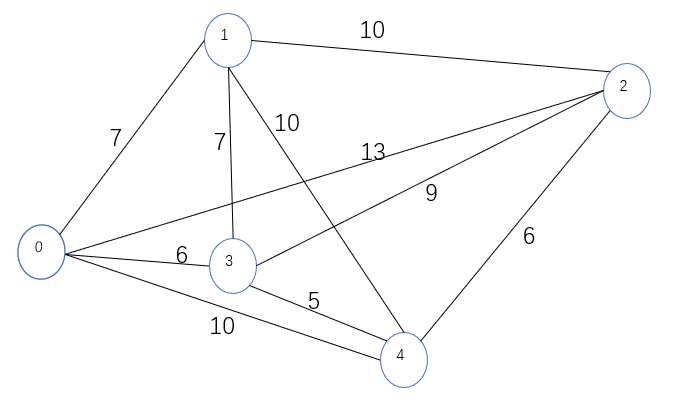
### 2.符号说明以及递推公式

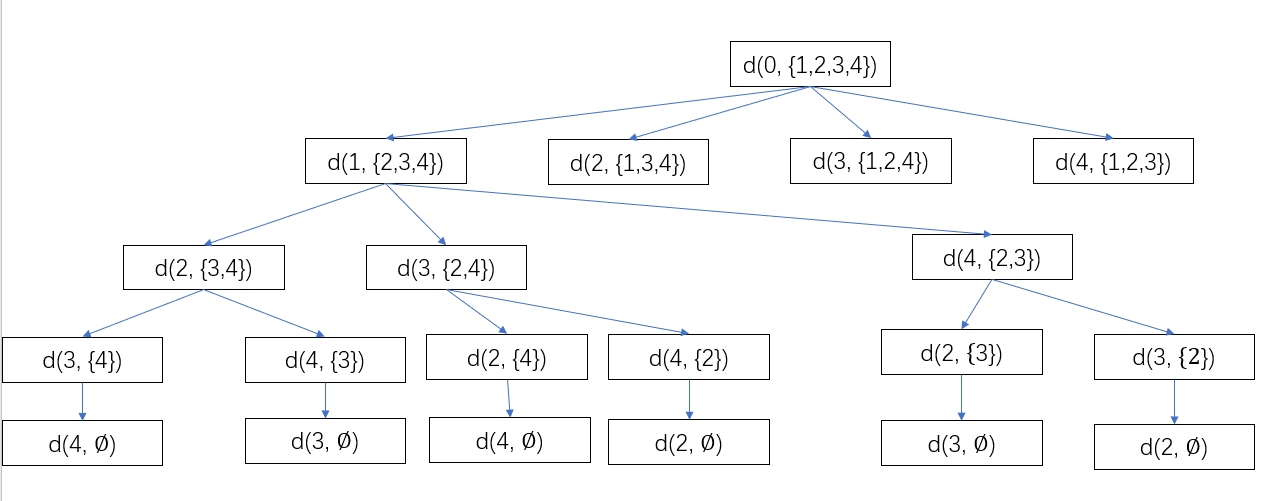
城市🡪点

|  |  |
| --- | --- |
| s | 起点 |
| V | 所有点的集合 |
|  | 从点i到点s包含所有点的集合 |
|  | 从点i到点j的花费 |
|  | 当前点i,经过所有点一次且仅一次到点s的最小花费 |

则的递推关系

### 3.例子





### 4.状态压缩

以集合S={1,2,3}为例

|  |  |
| --- | --- |
| 集合 | 状态压缩后二进制 |
|  | 000B |
| {1} | 001B |
| {2} | 010B |
| {3} | 100B |
| {1,2} | 011B |
| {1,3} | 101B |
| {2,3} | 110B |
| {1,2,3} | 111B |

### 5.算法实现以及算法时空复杂度

/\*

\*算法依据的递推公式

\*时空复杂度

\*n为点的个数

|  |  |
| --- | --- |
| 时间复杂度 |  |
| 空间复杂度 |  |

\*说明

\*只处理无向图

\**起点为0*

*\*输入*

*\*D 为二维数组(nxn矩阵)*

*\*输出*

*\* 最短路径,最短花费*

*\*代码实现*

\*JavaScript代码

\*/

**function** **TSP**(D) {

**const**INF=65535//*定义的最大值*

**var**n=D**.**length//*n的个数*

**var**i,j,k,min,tmp;

**var**b=1<<(n-1); //*点集状态总数*

**var**dp={}//*记录状态(状态压缩)*

/\*

状态压缩

将一个集合压缩为二进制位数

\*/

**var**bridge={}//*记录中间节点*

**for** (i = 0; i **<** n; i++) { //*初始化dp与bridge*

        dp[i] = {}

        bridge[i] = {}

**for** (j = 0; j **<** b; j++)

        {

            dp[i][j] = 0;

            bridge[i][j] = -1;

        }

    }

**for** (i = 0; i **<** n; i++) { //*初始化 dp的第0列*

        dp[i][0] = D[i][0]

    }

    //*初始化动态规划表结束*

    //*遍历dp*

**for** (i = 1; i **<** b - 1; i++) {

**for** (j = 1; j **<** n; j++) {

**if** ((1 << (j - 1) & i) **==** 0) {

                //*点j未访问*

*//*

                min = INF

**for** (k = 1; k **<** n; k++) { //*遍历点集*

**if** (1 << (k - 1) & i) {

                        //*点k在集合中*

                        //*松弛操作*

                        tmp = D[j][k] + dp[k][i - (1 << (k - 1))] //

**if** (tmp **<** min) {

                            min = tmp

                            dp[j][i] = min

                            bridge[j][i] = k

                        }

                    }

                }

            }

        }

    }

//*处理最后一列*

*//*

    min = INF

**for**(k=1;k**<**n;k++) {

        tmp = D[0][k] + dp[k][b-1-(1<<(k-1))]

**if**( tmp **<** min) {

            min = tmp

            dp[0][b-1] = min

            bridge[0][b-1] = k

        }

    }

**var**Vmax=dp[0][b-1]

    //构造路径

**var**path=[0]

**for**(i=b-1,j=0;i**>**0;) {

        j = bridge[j][i] //*下一个节点*

        i = i-( 1<<(j-1))

        path**.push**(j)

    }

    path**.push**(0)

**return** [path, Vmax]

}

//测试数据

**var**m=[

[0,7,6,10,13],

[7,0,7,10,10],

[6,7,0,5,9],

[10,10,5,0,6],

[13,10,9,6,0]

]

**var**res=**TSP**(m)

console**.**log(res)

### 6.数据测试

|  |  |  |
| --- | --- | --- |
| 组号 | 输入数据 | 算法输出 |
| 1 | var m = [      [0, 7, 6, 10, 13],      [7, 0, 7, 10, 10],      [6, 7, 0, 5, 9],      [10, 10, 5, 0, 6],      [13, 10, 9, 6, 0]  ] //输入的矩阵 |  |
| 2 | **var**n=22  **var**m=**new Array**(n)  **for** (**var**i=0; i **<** n; i++) {      m[i] = **new** **Array**(n)**.fill**(0)  }  **var**s1=**new Date**()**.getTime**()  res = **TSP**(m)  **var**s2=**new Date**()**.getTime**()  console**.**log("花费时间:",s2 - s1) |  |
| 3 | **var**n=23  **var**m=**new Array**(n)  **for** (**var**i=0; i **<** n; i++) {      m[i] = **new** **Array**(n)**.fill**(0)  } | FATAL ERROR:   JavaScript heap out of memory  详见附录  此时 |

# 三.附录

##### 1.参考资料

1.用动态规划方法求解广义背包问题 豆丁

<https://www.docin.com/p-1571618425.html>

2. 动态规划求解TSP(旅行商)问题

<https://blog.csdn.net/masibuaa/article/details/8236074>

##### 2.TSP运行错误信息

<--- Last few GCs --->

[9124:00000200ED644110] 6900 ms: Mark-sweep 1386.2 (1396.5) -> 1354.9 (1361.7) MB, 143.6 / 0.0 ms (+ 1.9 ms in 1 steps since start of marking, biggest step 1.9 ms, walltime since start of marking 151 ms) (average mu = 0.241, current mu = 0.142) alloc[9124:00000200ED644110] 7063 ms: Mark-sweep 1387.0 (1394.0) -> 1374.7 (1381.5) MB, 124.8 / 0.0

ms (+ 2.0 ms in 2 steps since start of marking, biggest step 2.0 ms, walltime since start of marking 146 ms) (average mu = 0.232, current mu = 0.223) alloc

<--- JS stacktrace --->

==== JS stack trace =========================================

0: ExitFrame [pc: 000001391CBDC5C1]

Security context: 0x01413de17991 <JSObject>

1: TSP [000003625F904939] [C:\Users\13298\Documents\Python Scripts\ds\_git\TSP\_DP.js:~2] [pc=000001391CC71FA5](this=0x01eb3b58d481 <JSGlobal Object>,D=0x03625f904979 <JSArray[23]>)

2: /\* anonymous \*/ [000003625F904A89] [C:\Users\13298\Documents\Python Scripts\ds\_git\TSP\_DP.js:87] [bytecode=000002FCD16D6DC1 offset=174](this=0x03625f904bb9 <Object map...

FATAL ERROR: Ineffective mark-compacts near heap limit Allocation failed - JavaScript heap out of memory

1: 00007FF7FA7EDD8A v8::internal::GCIdleTimeHandler::GCIdleTimeHandler+4506

2: 00007FF7FA7C8886 node::MakeCallback+4534

3: 00007FF7FA7C9200 node\_module\_register+2032

4: 00007FF7FAAE30DE v8::internal::FatalProcessOutOfMemory+846

5: 00007FF7FAAE300F v8::internal::FatalProcessOutOfMemory+639

6: 00007FF7FACC9804 v8::internal::Heap::MaxHeapGrowingFactor+9620

7: 00007FF7FACC07E6 v8::internal::ScavengeJob::operator=+24550

8: 00007FF7FACBEE3C v8::internal::ScavengeJob::operator=+17980

9: 00007FF7FACC7B87 v8::internal::Heap::MaxHeapGrowingFactor+2327

10: 00007FF7FACC7C06 v8::internal::Heap::MaxHeapGrowingFactor+2454

11: 00007FF7FADF1CD8 v8::internal::Factory::AllocateRawArray+56

12: 00007FF7FADF2652 v8::internal::Factory::NewFixedArrayWithFiller+66

13: 00007FF7FAE2748C v8::internal::Factory::NewCallHandlerInfo+157532

14: 00007FF7FAE27124 v8::internal::Factory::NewCallHandlerInfo+156660

15: 00007FF7FAE28275 v8::internal::Factory::NewCallHandlerInfo+161093

16: 00007FF7FAB649A8 v8::internal::SharedFunctionInfo::SetScript+23528

17: 00007FF7FAB47953 v8::internal::JSReceiver::class\_name+20595

18: 00007FF7FADE2F59 v8::internal::wasm::WasmCodeManager::LookupCode+15273

19: 00007FF7FADE6014 v8::internal::wasm::WasmCodeManager::LookupCode+27748

20: 000001391CBDC5C1

##### 3.运行环境

###### **nodejs版本**



###### 电脑环境



##### 4.结论

动态规划仅能解决复杂度低的的TSP问题. 对于更复杂的问题, 应选用其他算法(如蚁群算法)