《计算复杂性理论》
第6讲穷举法

山东师范大学信息科学与工程学院
段会川
2015年11月

目录

穷举法定义

- 在计算机科学中，穷举搜索(exhaustive search)或蛮力搜索(brute-force search), 也称为生成+测试法(generate and test), 是一种非常通用的问题求解方法，该方法由两部分组成，一是系统化地枚举问题各种可能的候选解，二是检查每一个解是否满足问题的求解要求。

维基百科

穷举法定义

- 进行蛮力搜索必须实现4个步骤，即首选(first), 再选(next), 验证(valid)和输出(output)，它们必须以问题的实例为输入参数，实现下面具体的功能:
  - 1. c=first (P): 产生问题P的第一个候选解.
  - 2. next (P, c): 从当前候选解c顺次产生下一个候选解.
  - 3. valid (P, c): 检查候选解c是否为问题P的解.
  - 4. output (P, c): 如果c为P的解则将其输出.
- 再选(next)步骤必须能判断是否还有下一个候选解，如果没有通常返回一个“空候选”(“null candidate“), 常以Λ表示。同样地，首选(first)步骤在实例P没有候选时也应该返回Λ.

维基百科

穷举法的通用算法

- 算法名称：通用穷举法(ExhaustiveSearch)
- 输入：问题实例P
- 输出：问题的解
- 1: c ← first(P)
- 2: while c ≠ Λ
- 3:   if valid(P,c) then
- 4:   output(P, c)
- 5:   c ← next(P)
- 6: end while

百元买百兔问题

- 兔翁一值钱五，兔母一值钱三，兔雏三值钱一。百钱买百兔，问兔翁、母、雏各几何？”
- 算法问题：n元买n兔问题
- 数学模型
  - $x+y+z=n$
  - $5x+3y+\frac{z}{3}=n$

百元买百兔问题

```
for x=1 to n
    for y=1 to n
        for z=1 to n
            if x+y+z=n then
                if z mod 3 = 0 then
                    if 5x+3y+z/3 = n then
                        print x,y,z
```


百元买百兔问题

```
for x=1 to n/5
    for y=1 to n/3
        z=n-x-y
        if z mod 3 = 0 then
            if 5x+3y+z/3 = n then
                print x,y,z
```


素数测试—试除法(trial division)

- 试除法是测试一个数N是否为素数的蛮力方法
  - 由于如果N有大于$\sqrt{N}$的因子p，则一定有一个小于$\sqrt{N}$因子q，因此只要用小于$\sqrt{N}$每个素数去试除N，如果找到一个数能够除尽N，则N就不是素数，如果所有的素数都除不尽N，则N必是素数
  - 上述方法未考虑计算$\sqrt{N}$的代价，但是算法设计过程中可以设法避免$\sqrt{N}$的计算

素数测试—朴素(na ïve)试除法伪代码

```
ret = true
for i=2 to sqrt(N)
    if N MOD i =0
        ret = false
        break
    end if
return ret
```


素数测试—朴素(na ïve)试除法伪代码

```
ret = true
i = 2
do
    if N MOD i =0
        ret = false
        break
    end if
    i = i+1
while i*i<=N
```


目录

0-1背包问题—形式化定义

- 给定n个重量为$w_1, w_2, \cdots, w_n$价值为$v_1, v_2, \cdots, v_n$的物品和容量为W的背包，其中$W < \sum_{i=1}^{n} w_i$且物品不可分割，问怎样装入物品可以获得最大的价值？
- 以$x_1, x_2, \cdots, x_n$表示物品的装入情况，其中$x_i \in \{0,1\}$，则0-1背包问题可以表达为如下所示的优化问题：

$$\max_{x_1, x_2, \cdots, x_n} V(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{n} x_i v_i,$$

$$s.t. \quad \sum_{i=1}^{n} x_i w_i \le W,$$

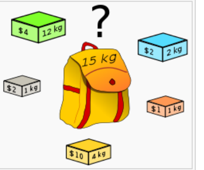
$$x_i \in \{0,1\}, i = 1, 2, \cdots, n.$$

0-1背包问题—示例

编号	1	2	3	4	5
重量	2	2	6	5	4
价值	6	3	5	4	6

背包容量：10　　王秋芬，P101

解：1,1,0,0,1
重量：2+2+4=8
价值：6+3+6=15



背包问题的一个例子：应该选择哪些盒子，才能使价格尽可能地大，而保持重量小于或等于15 kg？

i	1	2	3	4
w_i	5	4	6	3
v_i	10	40	30	50

$W = 10$

$X^* = (0, 1, 0, 1)$

$W^* = 4 + 3 = 7$

$V^* = 40 + 50 = 90$

http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf

0-1背包问题的状态空间树—子集树

子集树是使用回溯法解题时经常遇到的一种典型的解空间树。当所给的问题是从n个元素组成的集合S中找出满足某种性质的一个子集时，相应的解空间树称为子集树。此类问题解的形式为n元组$(x_1, x_2, \cdots, x_n)$，分量$x_i(i=1,2,\cdots,n)$表示第i个元素是否在要找的子集中。x_i的取值为0或1，$x_i=0$表示第i个元素不在要找的子集中；$x_i=1$表示第i个元素在要找的子集中。如图5-15所示是$n=3$时的子集树。

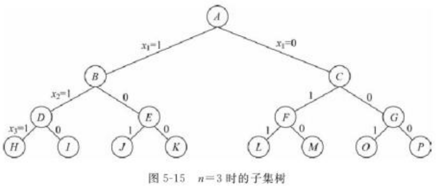


图 5-15　$n=3$时的子集树

P133

子集树—穷举搜索思路

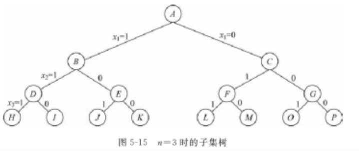


图 5-15　$n=3$时的子集树

- 使用深度优先搜索策略，用栈记录路径
  - 若当前为非叶节点
    - 将1压栈
    - 递归访问左子树
    - 将0压栈
    - 递归访问右子树
  - 若当前为叶节点
    - 打印路径

子集树穷举搜索伪代码

- 算法名称：　子集树的穷举搜索
- 输入：树的高度n
- 输出：从根到叶结点的所有路径

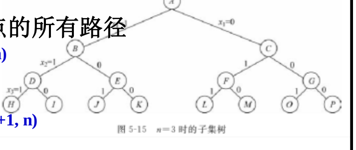


图 5-15　$n=3$时的子集树

```
1： ES-SubsetTree(s, c, n)
2： if c<=n then
3：     s.push(1)
4：     ES-SubsetTree(s, c+1, n)
5：     s.pop()
6：     s.push(0)
7：     ES-SubsetTree(s, c+1, n)
8：     s.pop()
9： else
10：    print(s)
```


子集树穷举搜索复杂度

- 算法名称：　子集树的穷举搜索
- 输入：树的高度n
- 输出：从根到叶结点的所有路径

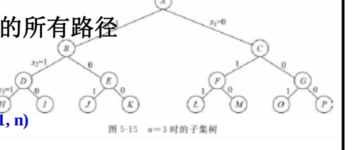


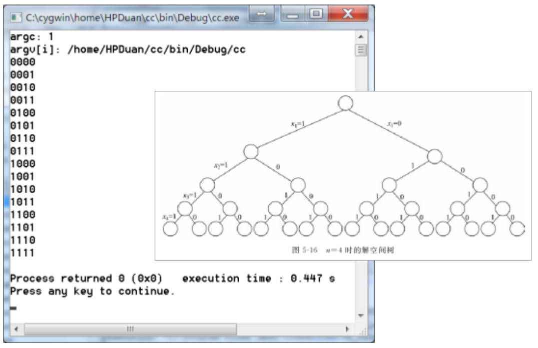
图 5-15　$n=3$时的子集树

```
1： ES-SubsetTree(s, c, n)
2： if c<=n then
3：     s.push(1)
4：     ES-SubsetTree(s, c+1, n)
5：     s.pop()
6：     s.push(0)
7：     ES-SubsetTree(s, c+1, n)
8：     s.pop()
9： else
10：    print(s)
```

子集树的穷举法—C++11实现

```
vector<int> s;
void ES-SubsetTree(int n) {
  if (n>0) {
    s.push_back(0);
    ES-SubsetTree (n-1);
    s.pop_back();
    s.push_back(1);
    ES-SubsetTree (n-1);
    KSv.pop_back(); }
  else {
    for (auto x:s)
      cout << x;
    cout << endl;
    return; }}
```


子集树穷举法—运行演示



0-1背包问题的穷举法—实现

```
vector<int> s;
void ES-SubsetTree (int n) {
  if (n>0) {
    s.push_back(0);
    ES-SubsetTree (n-1);
    s.pop_back();
    s.push_back(1);
    ES-SubsetTree (n-1);
    s.pop_back(); }
  else {
    for (auto x:s)
      cout << x;
    cout << endl;
    return; }}
```


目录

TSP问题—描述

（1）问题描述。

设有 n 个城市组成的交通图，一个售货员从住地城市出发，到其他城市各一次去推销货物，最后回到住地城市。假定任意两个城市 i,j 之间的距离 $d_{ij}(d_{ij}=d_{ji})$ 是已知的，问应该怎样选择一条最短的路线？

P152, 例5-10

（2）问题分析。

旅行商问题给定 n 个城市组成的无向带权图 $G=(V,E)$，顶点代表城市，权值代表城市之间的路径长度。要求找出以住地城市开始的一个排列，按照这个排列的顺序推销货物，所经路径长度是最短的。

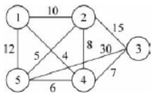
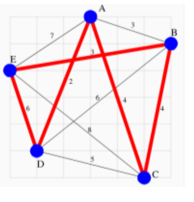


图 5-35 无向带权图

P153

TSP问题—示例



	A	B	C	D	E
A	0	3	4	2	7
B		0	4	6	3
C			0	5	8
D				0	6
E					0

http://people.sc.fsu.edu/~jburkardt/latex/genetic_2013_fsu/genetic_2013_fsu.html

TSP问题—示例

【例 5-1】 给定一个有向带权图 G=(V,E)，权非负，如图 5-1 所示。找出顶点 1→5 的最短路径及其长度。

问题分析：该问题所有可能的路径只有三条，分别是 1→2→4→5，1→3→4→5，1→4→5，采用穷举搜索的方法逐一检查这三条路径的长度，得出的最短路径为 1→2→4→5，其长度为 3。

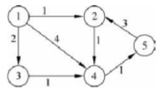
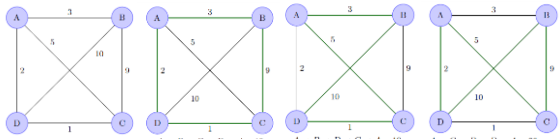


图 5-1 有向带权图



$A\to B\to C\to D\to A=15$ $A\to B\to D\to C\to A=19$ $A\to C\to B\to D\to A=26$

Integer linear programming formulation

For $i = 0, \ldots, n$, let u_i be an artificial variable, and finally take c_{ij} to be the distance from city i to city j. Then TSP can be written as the following integer linear programming problem:

$$\min \sum_{i=0}^{n} \sum_{j\neq i, j=0}^{n} c_{ij} x_{ij}$$

$$0 \le x_{ij} \le 1 \qquad i, j = 0, \cdots, n$$

$$u_i \in \mathbf{Z} \qquad i = 0, \cdots, n$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad j = 0, \cdots, n$$

$$\sum_{j=0, j\neq i}^{n} x_{ij} = 1 \qquad i = 0, \cdots, n$$

$$u_i - u_j + n x_{ij} \le n - 1 \qquad 1 \le i \neq j \le n$$

TSP问题的状态空间树—排列树

在排列树中，树的根结点表示初始状态（所有位置全部没有放置元素）；中间结点表示某种情况下的中间状态（中间结点之前的位置上已经确定了元素，中间结点之后的位置上还没有确定元素）；叶子结点表示结束状态（所有位置上的元素全部确定）；分支表示从一个状态过渡到另一个状态的行为（在特定位置上放置元素）；从根结点到叶子结点的路径表示一个可能的解（所有元素的一个排列）。排列树的深度等于问题的规模。

P146

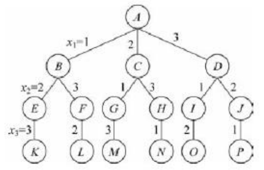


图 5-26 $n=3$ 的排列树

排列树—穷举搜索思路

- 使用深度优先搜索策略，用事先建立的线性表顺序存放城市号
  - 若当前为非叶节点，计序号为i
    - 遍历其下的所有子节点
    - 策略是让当前及后续的所有城市都有机会排在第i的位置上，通过交换实现
    - 递归访问各子树
  - 若当前为叶节点
    - 打印路径

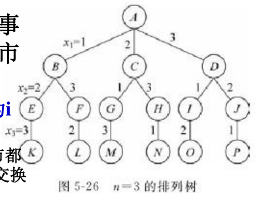


图 5-26 $n=3$ 的排列树

排列树穷举搜索伪代码

- 算法名称： 排列树的穷举搜索
- 输入：树的高度**n**，顺序的城市号列表**s**
- 输出：从根到叶结点的所有路径
  - **1：ES-PTree (s, c, n)**
  - **2：if c<=n then**
  - **3：　for i = c to n**
  - **4：　　swap(s[c], s[i])**
  - **5：　　ES-PTree(s, c+1, n)**
  - **6：　　swap(s[c], s[i])**
  - **7：else**
  - **8：　print(s)**

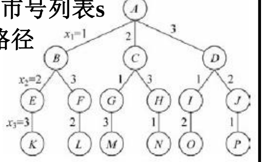


图 5-26 $n=3$ 的排列树

排列树穷举搜索复杂度

- 算法名称： 子集树的穷举搜索
- 输入：树的高度**n**
- 输出：从根到叶结点的所有路径
  - **1：ES-PTree (s, c, n)**
  - **2：if c<=n then**
  - **3：　for i = c to n**
  - **4：　　swap(s[c], s[i])**
  - **5：　　ES-PTree(s, c+1, n)**
  - **6：　　swap(s[c], s[i])**
  - **7：else**
  - **8：　print(s)**

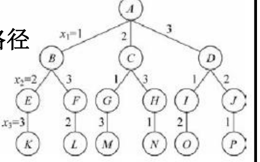


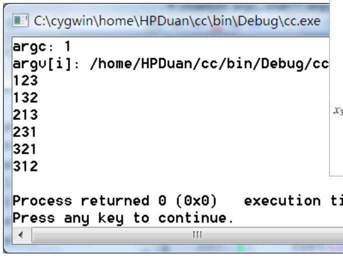
图 5-26 $n=3$ 的排列树

排列树穷举搜索—C++11实现

```
vector<int> s;
void ES-PTree(int i, int n) {
   if (i<n-1) {
      for (int j=i; j<n; ++j) {
      swap(s[i],s[j]);
      ES-PTree (i+1, n);
      swap(s[i],s[j]); }
   else {
      for (auto x:s)
         cout << x+1;
      cout << endl;
      return;
   }
}
```

```
void TSPt(int n)
{
   for (int i=0; i<n; ++i)
      s.push_back(i);
   ES-PTree (0,n);
}
```


排列树穷举搜索—示例



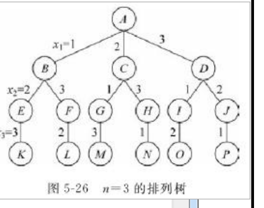


图 5-26 n=3 的排列树

TSP穷举搜索—C++11实现

```
vector<int> s;
void ES-PTree(int i, int n) {
   if (i<n-1) {
      for (int j=i; j<n; ++j) {
      swap(s[i],s[j]);
      ES-PTree (i+1, n);
      swap(s[i],s[j]); }
   else {
      for (auto x:s)
         cout << x+1;
      cout << endl;
      return;
   }
}
```

```
void TSPt(int n)
{
   for (int i=0; i<n; ++i)
      s.push_back(i);
   ES-PTree (0,n);
}
```


目录

- 穷举法的定义
  - 穷举法的通用算法
  - 百元买白兔问题的穷举法
  - 素数测试问题的穷举法
- 0-1背包问题及其穷举法
  - 0-1背包问题的定义
  - 子集树及其穷举法
  - 0-1背包问题及其穷举法
- TSP问题的穷举法
  - TSP问题的定义
  - 排列树及其穷举法
  - TSP问题及其穷举法

The End

Thanks！