ナップザック問題
枝切付数え上げ法のプログラムと解説

SHIMURA Masato
http://japla.sakura.ne.jp

2014 年 12 月 4 日

目次

2014/9 の JAPLA のワークショップで山本洋一氏から材料取りをテーマとする数え上げ問題が出された。指数よりも急激に膨張する迷路のような組み合わせのスクリプトを作成している折に、ナップザック問題を参照した。ナップザック問題は歴史ある計算科学問題であり、その構造が材料取りと似ているので、数え上げ法のスクリプトを作成してみた。

1　ナップザック問題

例題：高級縫いぐるみ工場に押し入って、人気の熊さん人形を物色している。

- 熊さん人形は 4 種類あって次の表のとおり。人形は沢山ある。

SS	$2kg$	16 万円
S	$3kg$	19 万円
M	$4kg$	23 万円
L	$5kg$	28 万円

- 逃走用のナップザックは古く、7 ☒より多くの荷物を入れると底が抜けてしまう。
- 転売価格が最大になるにはどのような熊さん人形を持って逃げればよいか

出題　久保　p124

1.1　材料取りとナップザック問題

材料取り	ナップザック
指定の長さの部材を指定個数母材から切り出す。母材は何本使ってもよい	指定重量の範囲で商品を詰め込む。商品の個数には制限はない

1.2　ナップザック問題とアルゴリズム

入力　次のように縦型とする

```
   DATA
2 16
3 19
4 23
5 28
```

ソート　スクリプトでは入力に昇順ソート　(/ :~) をかけて、出力もソート済

みである。結果を読むときに、入力もソートしておくと便利である。

```
sort=: /:~
```

sort DATA

最大個数　n = $\frac{\text{制限重量}}{\text{最軽ピースの重量}}$ の切り捨て (floor <.) 値

組み合わせの方法　数学の順列や組み合わせは個数を求める。ここでは内訳が必要。

$$ {}_nH_r = \frac{(n+r-1)!}{(n-1)!r!} = {}_{n+r-1}C_r $$

4 人の子供 (n) に 5 個のみかん (r) を重複を許して分ける方法

$$ {}_4H_5 = \frac{8!}{5!3!} = {}_8C_5 = 56 $$

組み合わせの内訳を求めるのに J のイディオム `tap=:i.@! A. i.` があり、順列 ${}_nP_n$ とした全個数を打ち出す。ここでは重複が入り、重複具合が分からないので、愚直に全組み合わせを打ち出す

アルゴリズム　ナップザック問題のアルゴリズム

ここでは単純な力技法を用い次のアルゴリズムによる。

1. **n** を元に全組み合わせを作成する
2. 途中に重量チェックを組み入れる (枝切、分岐限定プロセス）
3. 価格による最適評価は重量制限下で可能な全組み合わせ求めたうえで一度に一覧を打ち出す
4. 整数でなくともよい

1.3　例題の実行

0 は何も取らない。2 3 4 5kg を指標 1 オリジンとして順に取った

```
   7 napsack DATA
```

指標の組み合わせ重量とその価格を付加

組み合わせの指標

```
   7 napsack0 DATA
0 0 0
1 0 0
1 1 0
1 1 1
2 0 0
2 1 0
2 1 1
2 2 0
3 0 0
3 1 0
3 2 0
4 0 0
4 1 0
```

```
0 0 0 0  0
1 0 0 2 16
2 0 0 3 19
3 0 0 4 23
4 0 0 5 28
1 1 0 4 32
2 1 0 5 35
2 2 0 6 38
3 1 0 6 39
3 2 0 7 42
4 1 0 7 44
1 1 1 6 48
2 1 1 7 51
index kg/price
```

最適化の順にソートしている

久保は最大値 51 を求めるのにツリー構造や Bellman の動的計画法を用いて、価格まで同時に探索して最大価格を求めている。マシンパワーを生かした力技法の方がブラックボックスがないので明快である

2　経過と説明

2.1　組み合わせを求める

```
napsack0=: 4 : 0
NB. Usage: 7 napsack0 DATA
limit=. x
' weight price'=. { |: /:~  y   NB. upsort
maxnum=. {. <. limit % weight        NB.floor
```

```
ans=:  (maxnum # 0) ,   >({@> >:i.maxnum) # (L:0) 1
for_ctr. i. <: # y do.
   ind0=. }. i. maxnum
   left=. ({@> ind0) # (L:0) 2+ ctr
   index=. maxnum - L:0 {@> ind0
   right=. index {."1 (L:0) ans
   left=.>L:1((# ans) # L:1 { L:0 left)        NB. fine
   tmp0=:(,./,./>,.left ,. L:0 right), maxnum # 2+ctr
  ans=.  ~. (limit;weight) check_weight ans, tmp0
        NB. nub

end.
)
```

- 軽い順にソートし、weight price に代入

  ```
  ' weight price'=. { :/: y — NB. upsort
  ```

- 最大個数 (n) を求める

  ```
  maxnum=. {. <. limit % weight  NB.floor
  ```

- n と 1 番目の指標の組み合わせ。1 まで処理完了

  ```
  ans=:  (maxnum # 0) ,   >({@> >:i.maxnum) # (L:0) 1

     ans
  0 0 0
  1 0 0
  1 1 0
  1 1 1
  ```

- 2 番目の組み合わせ

  ```
    ({@> }. i.3) # (L:0) 2
  +-+---+
  |2|2 2|
  +-+---+
  ```

これを ans の行数分コピーして、右に足りない列分の ans を取り出し拡張する

```
   tmp0
2 0 0
2 1 0
2 1 1
2 1 1
------
2 2 0
2 2 1
2 2 1
2 2 1
-------
2 2 2
```

最後に 2 2 2 2 を加える

- ここで次項のリミットチェックを行う。数値はインデックス
- 枝切である

```
   7 napsack0 DATA
0 0 0
1 0 0
1 1 0
1 1 1
2 0 0
2 1 0
2 1 1
2 2 0
```

- 3 項目.. と繰り返す
- L:1　は 2 重ボックスの中での演算

2.2 重量チェック

```
check_weight=: 4 : 0
'limit weight'=. x
 ind=.limit>: +/"1 y { 0, weight
 ind# y
)
```

- 0,weight で 0 番目の 0 を入れる
- 重量内かオーバー化の１０の指標を作成し、指標に従って 1 の部分をコピーして取り出す

2.3 最適化

ブラックボックスで最適解を求めるより、一覧表の方がよくわかる。

```
napsack=: 4 : 0
NB. Usage: 7 napsack DATA
limit=. x
' weight price'=.{. { |: /:~  y   NB. upsort
ind=. limit napsack0 y
ind,.  |: > +/"1 L:0 ind{ L:0 {|: 0,y
)
```

- 取り出した指標ごとの価格を計算し、一覧表を作る。

```
   7 napsack DATA
```

指標の組み合わせ重量とその価格を付加

```
      7 napsack DATA
0 0 0 0  0
1 0 0 2 16
2 0 0 3 19
3 0 0 4 23
```

```
4 0 0 5 28
1 1 0 4 32
2 1 0 5 35
2 2 0 6 38
3 1 0 6 39
3 2 0 7 42
4 1 0 7 44
1 1 1 6 48
2 1 1 7 51
index kg/price
```

- 出力が膨大な時は最適解に近い部分が直ぐにみられるよう列ソートを入れた
  `sort_fit=: 3 : '|. "1 /:~ |."1 y'`

2.4 幾つかの例題

ネットでいくつかの例題を集め、数え上げ法と比較してみよう。いずれも動的計画法を駆使した名作である。同時に動的計画法の的確さも実感することができる。

1. 例題２「病みつきになる動的計画法　その深淵に迫る」IT メディア Enterprize
   この例題は 1 品づつという制限が入っている。*napsack_one* には次の *1* 行を加えた
   `order=: 3 : ' /:~ "1 ~. "1 y '`

```
   /:~ DATA3
1 2
2 3
3 2
3 6
4 3
kg/price
limit 10kg
```

```
   _5 {.  10 napsack_one DATA3
0 0 1 4 5  8 11
0 0 3 4 5 10 11
0 0 2 4 5  9 12
0 1 2 3 4  9 13
0 1 2 4 5 10 14
```

同じ重量なら最適化で価格が安い方が選ばれることはない。

2. 例題３　チャリティーオークションへの出品。自分で運ぶので重さは *10kg*（片鱗懐古のブログ）

	kg	*yen*
カメラ三脚	3	7000
手提げ金庫	2	4000
ゲーム機	2	8000
高圧洗浄機	5	9000
木彫り置物	1	3000
電気ポット	1	5000

```
   DATA2
3 7000
2 4000
2 8000
5 9000
1 3000
1 5000
```

```
  /:~ DATA2
1 |1 3000
2 |1 5000
3 |2 4000
4 |2 8000
5 |3 7000
6 |5 9000
```

こちらを参照

```
 _5 {.  10 napsack_one DATA2
0 0 1 2 5 6 10 24000
0 0 0 4 5 6 10 24000
0 0 1 2 4 6  9 25000
0 0 2 3 4 6 10 26000
0 1 2 3 4 5  9 27000
```

高圧洗浄機は置いて行こう

References

久保幹雄「組み合わせ最適化とアルゴリズム」　共立出版　2000

スクリプトは次から DL できます。　japla.sakura.ne.jp

J 言語は次から入手できます。www.jsoftware.com