主題: Branch-and-bound (III)

- 基礎問題一: 換零錢
- 基礎問題二: A.242 Stamps and Envelope Size
- Case Study: A.165 Stamps
- Case Study: H.92.1 找零錢的潔癖

基礎問題一: 換零錢

- 有 k 種整數幣值 $A=\{a_1, \ldots, a_k\}$，每一種幣值都有無限的供應量。
- 給一個數值 N，找出一個能湊成 N 且張數最少的方法。
- Example: A = {2, 3, 5, 8}, N = 12

  $2 + 2 + 8 = 12 \qquad \Rightarrow 3$ 張

A dynamic programming solution

- Optimal substructure
  - 湊出 N 的最少張方法:

    $$\boxed{a_j}\ \underbrace{\boxed{x_1}\ \boxed{x_2}\ \boxed{x_3}\ \cdots\ \boxed{x_q}}_{N-a_j\text{ 的最少方法}}$$

  - Example: A = {2, 3, 5, 8}, N = 12

    12 = 2 + 10 ⇒ 10 的最少方法 + 1 張
    12 = 3 + 9 ⇒ 9 的最少方法 + 1 張
    12 = 5 + 7 ⇒ 7 的最少方法 + 1 張
    12 = 8 + 4 ⇒ 4 的最少方法 + 1 張

- Recurrence
  - f[i]: 要湊出 i 塊錢所需的最少張數
  - $f[i] = \min_{1 \le j \le k} \{ f[i - a_j] \} + 1$
  - boundary conditions

    $$\begin{cases} f[0] = 0 \\ f[i] = \infty & \text{for } i < 0 \end{cases}$$

Example: A = {2, 3, 5, 8}

i	0	1	2	3	4	5	6	7	8	9	10	11	12
f[i-2]		∞	0	∞	1	1	2	1	2	2	1	3	2
f[i-3]		∞	∞	0	∞	1	1	2	1	2	2	1	3
f[i-5]		∞	∞	∞	∞	0	∞	1	1	2	1	2	2
f[i-8]		∞	∞	∞	∞	∞	∞	∞	0	∞	1	1	2
f[i]	0	∞	1	1	2	1	2	2	1	3	2	2	3

π (11) = 3 or 8
(for backtracking)

- Time complexity = O(k × N)

Pseudo code

```
int infinite = N+1;
int look_up(i) { if  i>=0 return (f[i]); return (infinite)}
f[0]=0;
for (i=1; i<=N; i++) {
    f[i] = infinite;
    for (j=1; j<=n; j++)
    {     x = look_up(i-a_j) + 1;
          if x < f[i] {  f[i] = x; π(i) =j  }
    }
}
```

f(i) = min{f(i-a_j)} + 1

for backtracking

- After the computation, f[N] is the solution

Another DP solution (incremental)

- $f_r[i]$: 用 $\{a_1, \ldots, a_r\}$要湊出 i 塊錢所需的最少張數
- Example: A = {2, 3, 5, 8}, N = 12

	0	1	2	3	4	5	6	7	8	9	10	11	12
f_1	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6
f_2	0	∞	1	1	2	2	2	3	3	3	4	4	4
f_3	0	∞	1	1	2	1	2	2	2	3	2	3	3
f_4	0	∞	1	1	2	1	2	2	1	3	2	2	3

- the answer is $f_k[N]$

Optimal substructure

- $f_r[i] = \min\{f_r[i - a_r] + 1, f_{r-1}[i]\}$

$f_r[i - a_r]$

$i - a_r$ 的最少方法，可使用 $\{a_1, a_2, \ldots, a_r\}$

$f_r[i] =$

a_r x_1 x_2 x_3 • • • x_q use a_r

$\neq a_r$ $\neq a_r$ $\neq a_r$ $\neq a_r$ • • • $\neq a_r$ not use a_r

i 的最少方法，只可使用 $\{a_1, a_2, \ldots, a_{r-1}\}$

$f_{r-1}[i]$

Example: A = {2, 3, 5, 8}

- $f_3[i] = \min \{f_3[i-5] + 1, f_2[i]\}$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
f_1	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7
f_2	0	∞	1	1	2	2	2	3	3	3	4	4	4	5	5

f_3	0	∞	1	1	2	1	2	2	2	3	2	3	3	3	4

7 = 5 + 2
10 = 5 + 5
11 = 5 + 3 + 3

The algorithm

- boundary conditions $\begin{cases} f_r[0] = 0 & \text{for } 0 \le r \le k \\ f_0[i] = \infty & \text{for } i \ne 0 \\ f_r[i] = \infty & \text{for } i < 0 \end{cases}$
- **for** r = 1 **to** k **do**
  compute $f_r[0..N]$ by using $f_{r-1}[0..N]$
- Time: O(k×N)
  - Each iteration takes O(N) time

基礎問題二:
A.242 Stamps and Envelope Size

- 給 k 種整數郵票的面額 $A=\{a_1, a_2, ..., a_k\}$以及一個信封可以貼的總郵票張數限制 S
- 找出最大的 N 值，滿足 1 ~ N 都可以用 S 張以內的郵票湊出，[1..N] 稱為 A 的 coverage
- $S \le 10$, $a_1 < a_2 < ... < a_k \le 100$
  $\Rightarrow k \le 100$, $N \le S \times a_k \le 1000$

Example

- Let A={1, 3} and S=3

	0	1	2	3	4	5	6	7	8	9	10	11	12
f[i]	0	1	2	1	2	3	2	3	4	3	4	5	4

$\Rightarrow$ The coverage is [1..7]

A DP solution

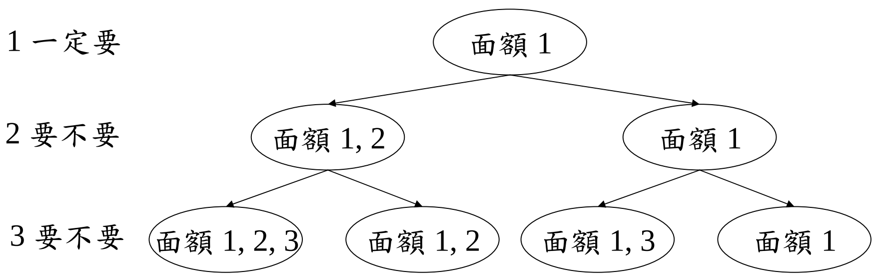
- Solution:
  Compute f[1], f[2], f[3], ..., until a value N with f[N+1] > S is found.
- Time: $O(k \times N) \approx O(10^5)$

Case Study: A.165 Stamps

- 給 h 與 k (h + k ≤ 9)， k 是面額種類， h 是總共可以貼的郵票張數限制，請決定用怎麼樣的面額，使得從 1 開始到 N 的所有數字都可以貼出來，並且 N 最大
- 例: k = 2, h = 3
  面額 = {1, 2}, 可貼出 = {1, 2, 3, 4, 5, 6}
  面額 = {1, 3}, 可貼出 = {1, 2, 3, 4, 5, 6, 7, 9}
  面額 = {1, 4}, 可貼出 = {1, 2, 3, 4, 5, 6, 8, 9, 12}
  ⇒ {1 , 3} 可以貼出最大的 N

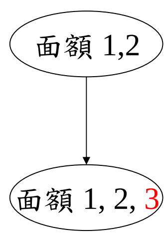
A B&B solution

- 暴法展開
  - 用暴力法展開所有可能的面額選擇
  - 若已知面額，則可用填表法來得知 N



Speedup the computation of N

- 在 recursive 過程中使用一陣列來記錄現在能貼的數字與最小使用張數，在加入新的面額 q 後可以很快得知新的結果



2×h

N	1	2	3	4	5	6
f	1	1	2	2	3	3

3×h

N	1	2	3	4	5	6	7	8	9
f ’	1	1	1	2	2	2	3	3	3

- f’[i] = min {f’[i – 3] + 1, f [i] } for i = q to q × h
- $O(qh \times g) \Rightarrow O(qh)$ (假設 q 是第 g 種面額)

- $f'[i] = \min \{f'[i - 3] + 1, f[i]\}$

f'[i -3]

i – 3 的最少方法，可使用 {1, 2, 3}

$f'[i] = \begin{cases} \boxed{3}\ \boxed{x_1}\ \boxed{x_2}\ \boxed{x_3}\ \cdots\ \boxed{x_q} & \text{sorted} \\ \boxed{1 \mid 2}\ \boxed{x_1}\ \boxed{x_2}\ \boxed{x_3}\ \cdots\ \boxed{x_q} & \text{sorted} \end{cases}$

i 的最少方法，只使用 {1, 2}

f [i]

How to stop recursion?

- 已經使用 k 種面額時
- Bound : 在決定面額 m 時，若先前決定的最大 N 為 m – 1，則面額 m 非使用不可
  - 例: 若在決定面額 6 時，之前所用的面額只能湊出 1, 2, 3, 4, 5，則面額 6 非用不可，不然往後因為面額都比 6 大，不可能湊出 6
  - cut (m, No), since it is impossible
- Other bounds ???

Case Study: H.92.1 找零錢的潔癖

- 給定所有幣值面額集合 A = {$a_1, a_2, \ldots, a_k$} (從小到大排序) ，$k \leq 50$，每種幣值可用張數不限
- 甲欠乙金額 N，甲可以給乙超過 N 的金額，若超過，乙必須找給甲剩餘的金額，請問甲乙共計鈔票張數最少的湊法
- N 與幣值面額不超過 2147483648

- 例: N = 9，A = { 1, 2, 4, 8 }
  - 甲 ⇒ 乙 8 * 1 + 1 * 1
  - 乙 ⇒ 甲
  - 2 張 + 0 張 = 2 張
- 例: N = 17，A = { 1, 5, 20 }
  - 甲 ⇒ 乙 20 * 1
  - 乙 ⇒ 甲 1 * 3
  - 1 張 + 3 張 = 4 張

Solution 1: DP

- f[i]: 要湊出 i 塊錢所需的最少張數
- m[i] = f[i] + f[i-N] //付 i 找 i-N 的最少張數
- best $= \min_{1 \le i \le \infty} \{m(i)\} = \min_{1 \le i \le x} \{m(i)\}$
- how large x is enough?
  - if best = L, then $x \ge \underline{L \times a_k}$ is large enough

at least L bills

Pseudo code

CPU or storage

increase table size

best =min {m[i]}

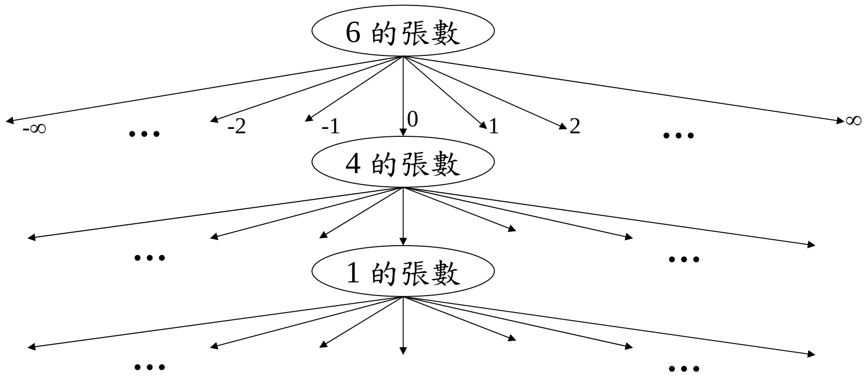
```
i=1; best = ∞;
for (L=1; L <= limitation; L++)
{  x = L *  a_k ;
   while (i <= x)
   {
      compute f(i);
      m = get_f(i)+get_f(i-N);
      if  (m < best) best = m;
      if (best <= L)  return (best); //solution found//
      i++;
   }
}
return (-1); //solution unknown//
```


- Time complexity: $O(k \times L \times a_k)$
- Problem: 因記憶體或 CPU time 的限制，不見得能找到答案
  - Example: 陣列可以開 100 萬，a_k 為 10 萬，則只有當 $L \le 10$ 才計算的出來
- 由題目無法判斷是否適用!

Solution 2: B&B — DFS

- Brute-force search: 產生每一種面額使用不同張數的所有組合
- 使用張數是正數代表甲給乙，負數代表乙找給甲
  - 每一種面額只會有一方使用
- 題目簡化成: 每張鈔票不限制使用量(正負皆可)，請求湊到 N 的最小張數為多少

圖例: A = {1, 4, 6}



- 問題: 每一個 node 有無限個 branches

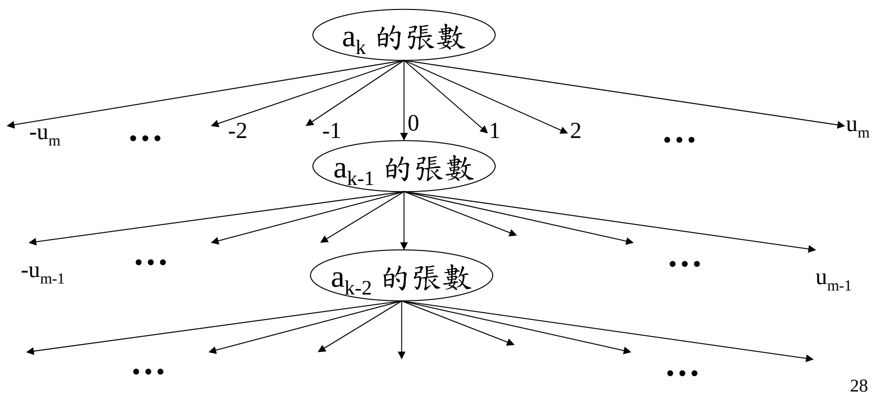
限制 branch 數

- 決定面額 a_i 的最大張數 u_i, $1 \le i \le k - 1$
  - 把 a_i 與其他比 a_i 大的面額取最小公倍數 g，則 $g/a_i - 1$ 是面額 a_i 最大張數的一個上限
    - Example: 假設有面額 6 跟 8，那面額 6 最多用 3 張，因為若在最佳解法裡，面額 6 使用超過 4 張，那每 4 張都可用 3 張 8 來取代，使用張數也會減少
  - 所有上限的最小值就是 u_i
    - Example: A = {10, 12, 25}，則面額 10 與 12 所得的上限為 5，與 25 所得的上限為 4，所以 u_1 = 4

- 最大面額 a_k 的上限 u_k
  - $u_k = (N + \Sigma_{1 \le i < k} \{a_i \times u_i\}) / a_k$
    - S = { 1 (5), 6 (3), 8 (4), 20 (?) }, N = 35
      (35 + 1 * 5 + 6 * 3 + 8*4) / 20 = 4.5
      面額 20 的上限為 4 張
  - 為什麼?

Brute-force search

- 計算最大張數 $U = \{u_1, u_2, \ldots, u_m\}$



How to stop recursion?

- Bound 1: 當展開幣值 a_i 的一個 branch 後，若發現把所有比 a_i 還小的幣值都以最大張數來計算，仍然湊不到 N，那就不用展開此 branch
  - 例: N = 35, A = {1, 4, 7, 10}, U = {3, 4, 6, 9}
    - 10 用 -2 張，7 用 5 張，$10\times -2 + 7\times 5 = 15$，那就算把 1 跟 4 都用到最大張數(共 19)也湊不到 35，此分支不可能

- Bound 2: 當展開幣值 a_i 的某 branch 時，若發現把所有比 a_i 還小的幣值都以負最大張數來計算，仍然超過 N，那就不用展開此 branch

- 加速 Bound 1 與 Bound 2 的計算
  - 填一個大小為 k – 1 的表格 P，P[i] 紀錄從 a_1 到 a_i 為止都用最大張數所能湊出的最大值
    - $P[i] = \sum_{1\le j\le i} a_j \times u_j$

1	2	3
3	19	61

A = {1 (3), 4 (4), 7 (6), 10 (9)}

- Bound 3: 目前已知最少的答案 b
  - 先用 DP 計算 f(N) 作為 initial bound
  - 假設目前正要考慮 a_i 的張數，且已經用了 x 張
  - C_e = ???

- Problem: 若面額種類 k 很大，或有某些面額的最大張數 u_i 過大，時間會太長
  - Example: u_i = 123456789
  - 面額 a_i 從 123456789 張到 -123456789 張，有太多 branch 要試

Solution 3: B&B — BFS+DFS

- for i =1, 2, 3, ... ,
  - Set initial bound b = i+1
  - DFS (i-level)

Remind: BFS + DFS 適用於預期 solution 不會太遠，但是用 DFS 又很可能會在找到第一個好的 bound 前，進入很深的 recursive ，甚至會不小心掉進 infinite recursion

Solution 4: DP + B&B

- 如果是一個標準的比賽問題描述，我們一定可以判斷出該選用 DP 或 B&B
- 但是這個題目完全無法判斷!
- Combination: 先用 DP 試試看，若發現找不出答案，再用 B&B

註: 如果 test cases 的 sizes 真如題目所述，這個題目是 NP-hard，唯一的處理方式是 B&B ，但時間會是 exponential