NHSPC 109
AC sol(task[A: I])

吳邦一

2020 年 12 月 19 日

Summary

Task	
PA礦砂採集	Fractional knapsack
PB村莊與小徑	Shortest paths on dag
PC樣本解析	Set operation
PD包裝順序	構造題，List-scheduling
PE共同朋友	Binary matrix multiplication
PF歡樂外送點	Intersection of rectangle
PG矩陣相乘	Special matrix multiplication
PH跑跑遊戲場	構造題，元件構造
PI黑白機	具轉換延遲的兩台機器排程

PA礦砂採集

- 給N<=1000個物品的單價與數量，求M總重量內的最大價值。
- Fractional knapsack
  - Greedy
  - 依單價由大到小，一一裝到不能裝，剩下多少裝多少
  - (subtask2) sorting + O(N)

PB村莊與小徑

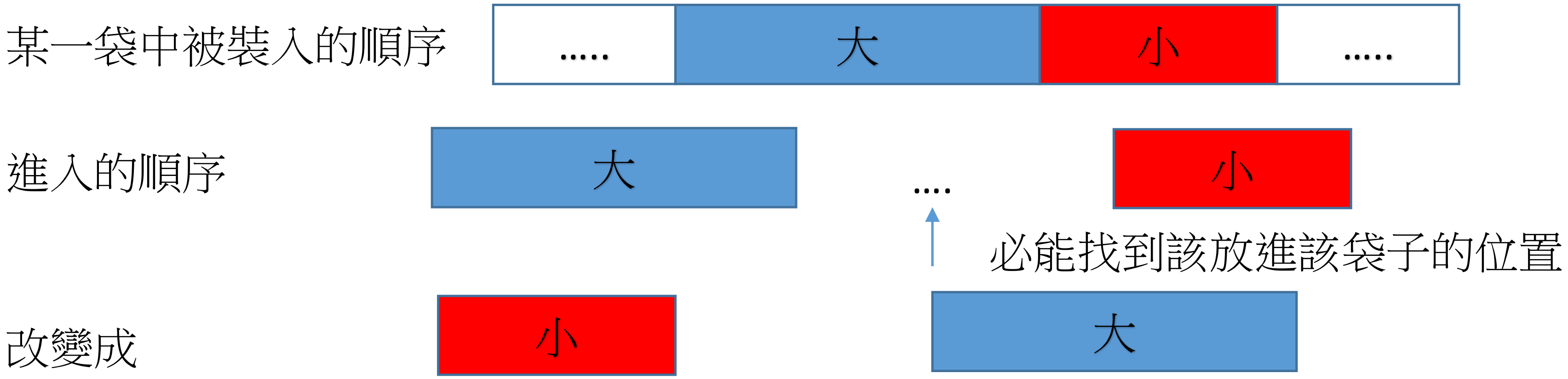
- 給一個dag，起點走得到所有點。求起點到所有點的最小距離和
- 依照topological sort從前往後算
  - 教科書基本題
  - O(n+m)

PC樣本解析

- 一個字串表示一個集合。給n<=18個集合S_i，以及另外一個集合X。保證S_i內部無部份交集，X至少與一個S_i部分交集。
  - 計算X與S_i的關係：不相交、包含、被包含、或部分交集。
  - 求X有幾種分割為(X1,X2)的方法，使得分割後與S均無部份交集的狀況
- 將集合轉換成整數或正規後的字串或[26]的陣列。依照定義計算就可以找出關係。
- 最後一個子題要求分割數，難?
  - 只有0種或2種：找出一個S_i與X部分交集，(X∩S_i, X-S_i)與(X-S_i, X∩S_i)是唯二可能的分割。

PD包裝順序

- N個水果依某種順序裝入M個袋子。
  - list-scheduling：每次放入目前最輕的袋子(相同放編號最小)
- 輸入一種包裝後的結果，構造出一種輸入的順序會被裝成給定的包裝方式，或輸出不可能。
- 若P是可能的包裝，S為其輸入順序，則一定存在另一種順序滿足以下條件：
  - 同一袋子內的水果被裝入的順序是由輕到重
- 將每一袋內的依重量排序，剩下就簡單了。
  - 記得檢查無解的狀況：未結束前，重量最輕的袋子已經無東西
  - M很大實需要Priority queue (PQ)

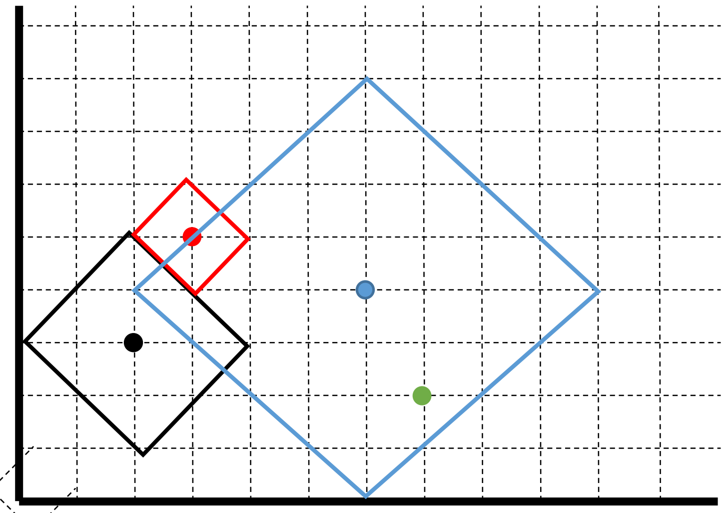


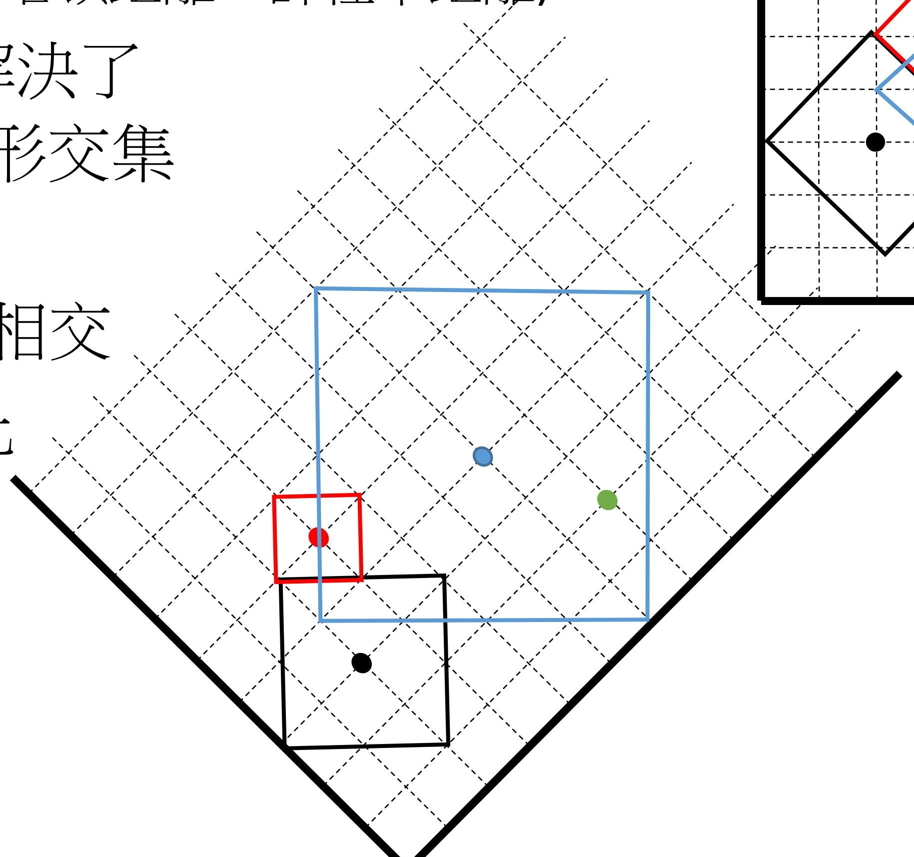
PE共同朋友

- 輸入N<=2500個人的朋友清單，計算有多少對人有共同朋友
  - 保證對稱關係，只算 i < j
- 圖中各點對的共同鄰居數
- 鄰接矩陣的「平方」就是答案：
  - CF(i, j) > 0 ⇔ row(i)*col(j) != 0
- But n=2500， O(n^3)？
- Bit-parallel, 且以AND取代乘法，檢查是否為0
  - 用C++的bitset，或
  - 自己將每64個裝入一個unsigned long long。
    - 或32個放入unsigned int

PF歡樂外送點

- 給格子點上n<=3e5家商店座標與服務半徑，以及各商店的加權分數。
  - 求服務範圍的最大(加權)重疊分數
  - 距離以X差值加Y差值計算
    - (L1-norm，曼哈頓距離，計程車距離)
- 將圖轉45度看就解決了
  - 平面上的正方形交集
  - 線段樹
- 子題1: 做n次線段相交
- 子題2: 線段樹退化成二維陣列$O(n^{1.5})$





PG矩陣相乘

- 給兩個N*N的方陣A與B，已知C=A×B最多只有2N個非零項，求C。
  - 模P運算，? < P <=5e7 是質數
  - 提示不可用太多mod，一次內積只用一個mod，N*P^2 < 2^63
- 子題
  - 子題1. C的每一列恰有一個非零項
  - 子題2. 每一列兩個
  - 子題3. 一列可能多個，總共2N個

第一子題，C的每一列恰有一個非零

- Let A[i] be the i–th row vector of A and B[j] the j-th column vector of B
- 分配率： A[i] * sum(B[j]) = sum(A[i] * B[j]) = x_i, the non-zero in i-th row of C
- Weighing j-th column by j:
  - A[i] * sum(j*B[j]) = sum(A[i] * (j*B[j])) = j*x_i
- 做2個內積可以找到 x_i 與 j*x_i ，然後可以算出 j 與 x_i (using inverse)
- Total complexity $O(n^2 + n \log P)$

第一子題，C的每一列恰有一個非零

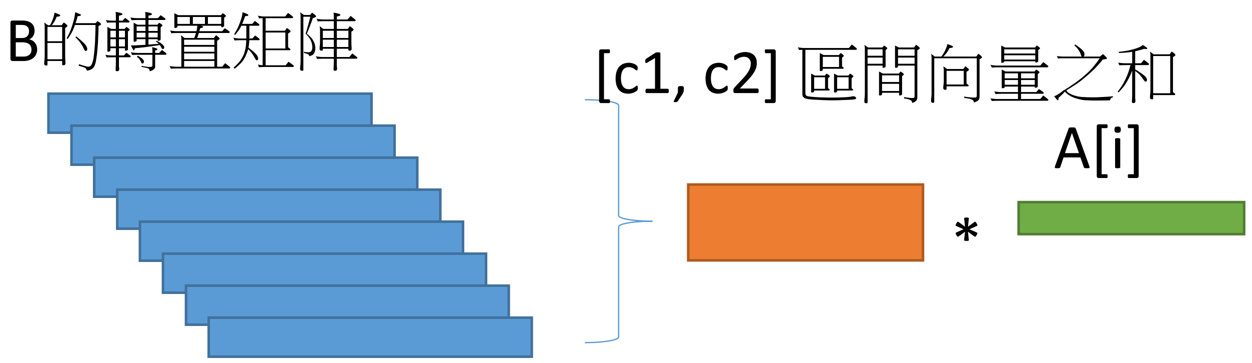
- Another approach
- 先看看網路上一個笑話
  - 如何找出一千個瓶子中的一瓶毒藥
- 對於一個列向量A[i]，要在N個行向量找出其中一個B[j]，使得 A[i]*B[j] ≠ 0
  - 找B[j]其實就是找毒藥
- 對於一個區間[c1,c2]，若 A[i]*sum(B[c1:c2]) ≠ 0，則毒藥在[c1:c2]中

二分搜 search(c1, c2) for subtask 1

- 利用前綴和可以快速求出任意[c1, c2] 區間之和向量B[c1:c2]，做一次內積可檢定毒藥是否在其中
- 只需要log(n)次向量內積，可以找出所對應的行
- 整體複雜度O(n^2log(n))

第二子題：每列恰好(至多)兩個

- 二分搜的大哥：二分繼續搜
- 二分繼續搜 search(c1, c2)
  - If (A[i]*sum(b[c1]:b[c2]) == 0) then return;
    if (c1 == c2) then found and return;
    mid = (c1+c2)/2;
    search(c1, mid);
    search(mid+1, c2);



= 0, 我看你沒有

≠ 0, 二分繼續搜
search(c1, mid);
search(mid+1,c2);

第二子題：每列恰好(至多)兩個

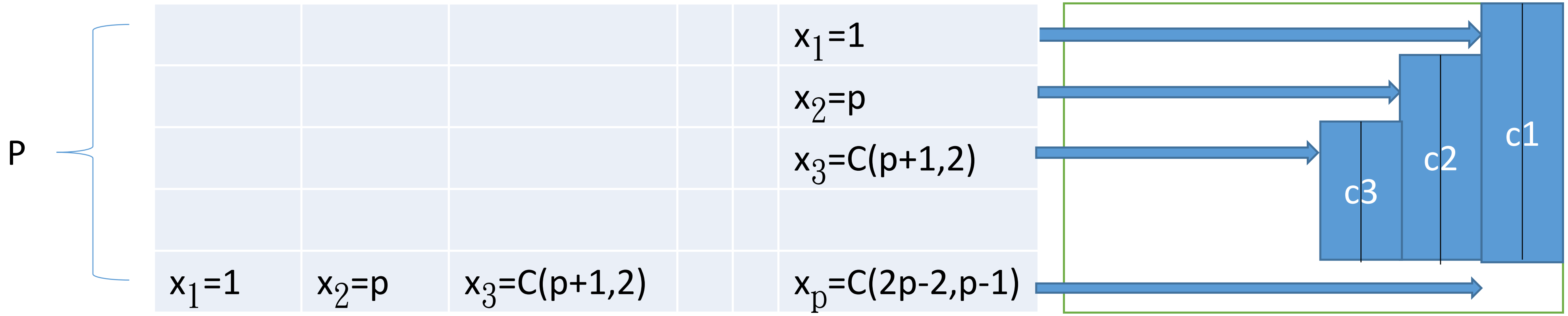
- 二分繼續搜用在第一子題沒問題。用在第二子題複雜度還是$O(n^2\log(n))$，因為每個被找出的位置只需要花log(n)次測試。此法成功的機率很大，但不能保證成功，因為：
  - 因為可能有壞人，讓以下狀況發生：
    $A[i]*(B[j]+B[k]) = 0$ 但$A[i]*B[j] \neq 0$且$A[i]*B[k] \neq 0$！
    有 1/P 的機會，有壞人就會這麼巧
  - 你看沒有的區間可能藏匿了兩個要找的對象
- 對抗壞人: 加權，將每個B[j]乘上j，再做一次
  - 若 $x + y = 0$ 且 $jx + ky = 0$，則 $x = y = 0$, unless $j \equiv k \pmod{P}$
- 將原始行向量搜一次，加權後再搜一次。若$A[i]*B[j] \neq 0$則壞人無法讓兩次都搜不到
  - 只要先搜位置，最後再做內積計算值就可以了

每列不定多少個，但總共2n個

- Randomized algorithm：
  - 每次加權用隨機數做加權
  - 二分繼續搜是單邊誤差的隨機演算法
    - 測到非零時不會錯，測到0時(false negative)有1/P的機率是錯的。
- 可以重複測試來降低錯誤的機率。計算順序很重要
- 比較好的計算順序：檢測區間時，針對所有列與所有加權
  - P>=37，False negative的機率是1/37，加權做五次錯誤的機率是1.5e-8，n=2800最多要做1400個檢測，失敗的機率大約是5萬分之一
  - 真的那麼倒楣？再submit一次，失敗的機率是25億分之一透了
- 比較差的計算順序：每加權一次做一次二分繼續搜，重複執行
  - 時間足夠做12次以上。
(其實只要能做5~6次，送幾次就會過，無人能擋 XD)。

PH跑跑遊戲場

- 給一個正整數 $T < 10^{18}$，建構一個N*M的格子遊戲場，控制相鄰兩格子之間通行與否，使得左上角到右下角的路徑數恰好是T。路徑只能往右往下。
  - 得分： (140 – N – M)*2，M+N 越小分數越高
- 第一想法，拉一個大方塊，$T = c_1x_1 + c_2x_2+\cdots$，



$p \leq 33$，N=p, M=3p-3，$N+M \leq 129$。

PH跑跑遊戲場

- 第二招，題目中的提示(有好好看題目嗎？)。
- 二進位轉換，江湖一點訣，說破了不值錢

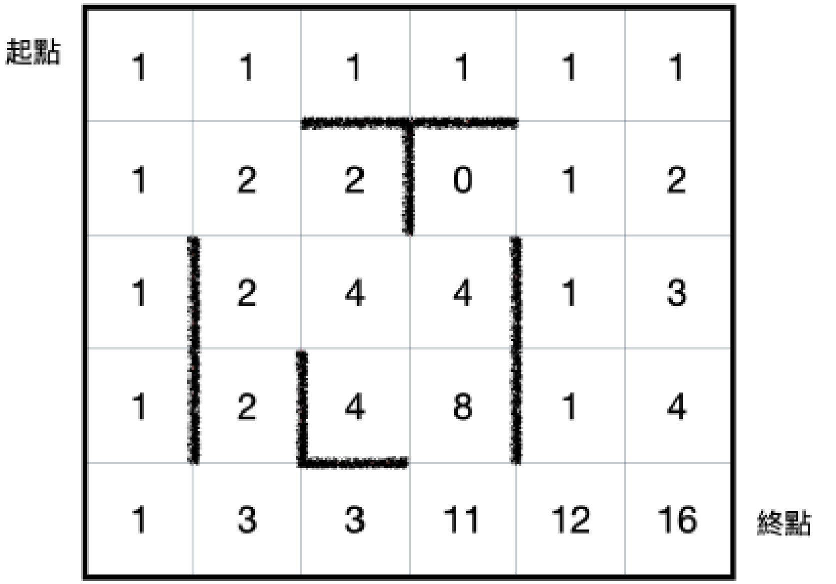
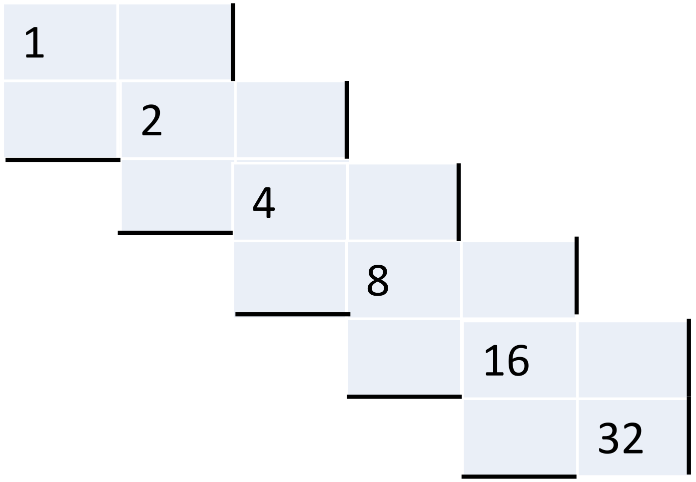
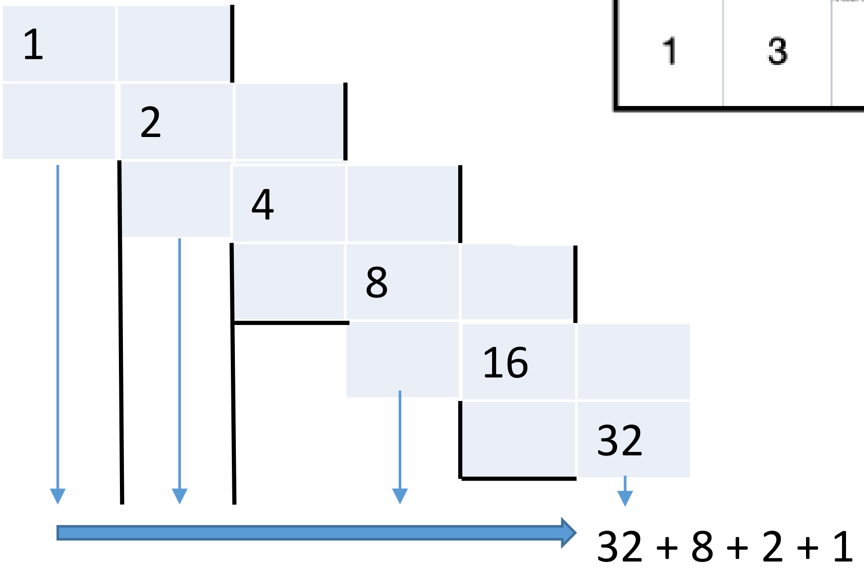


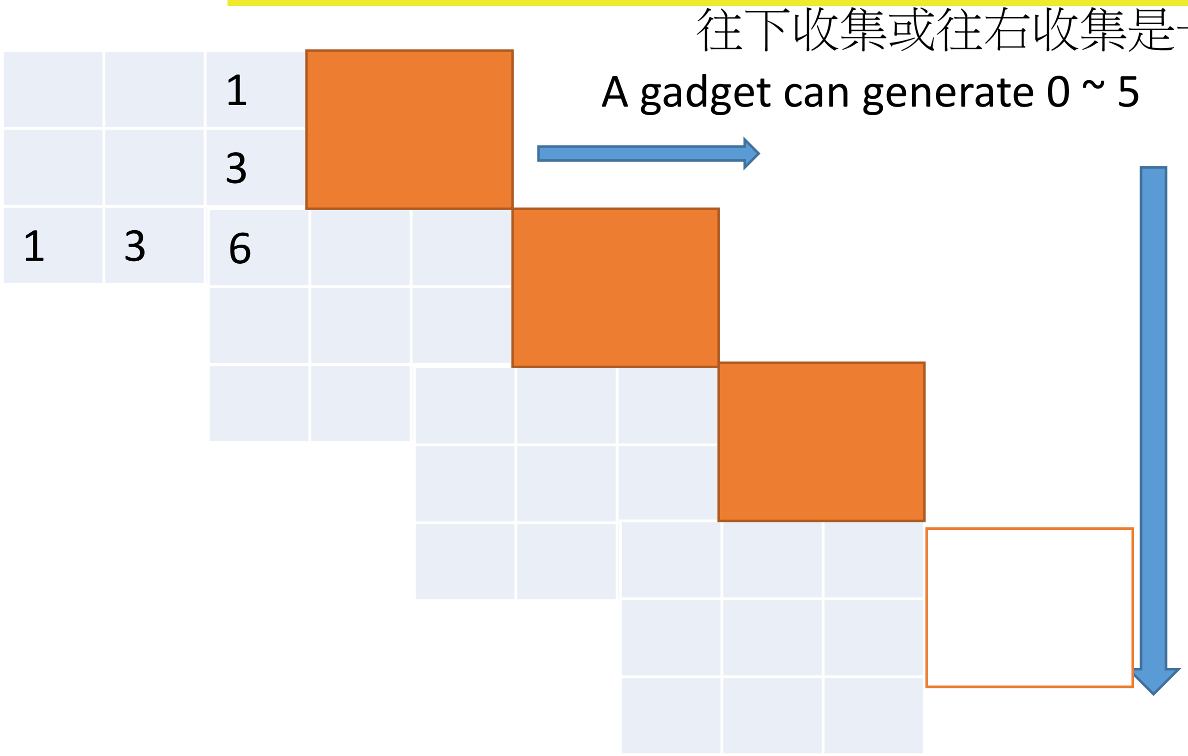
圖 2

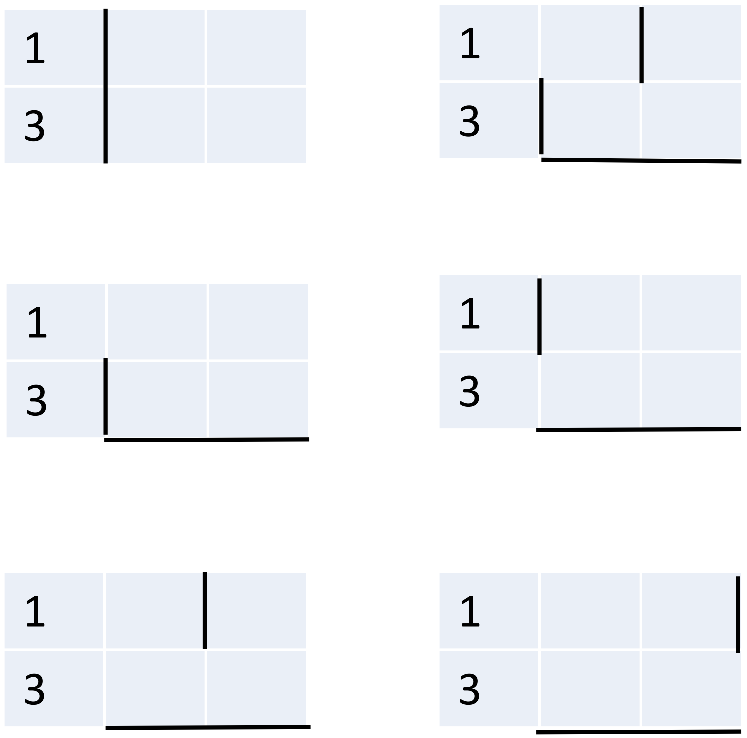




10^{18}是60 bit，此法需N=61, M=60，N+M $\leq$ 121。

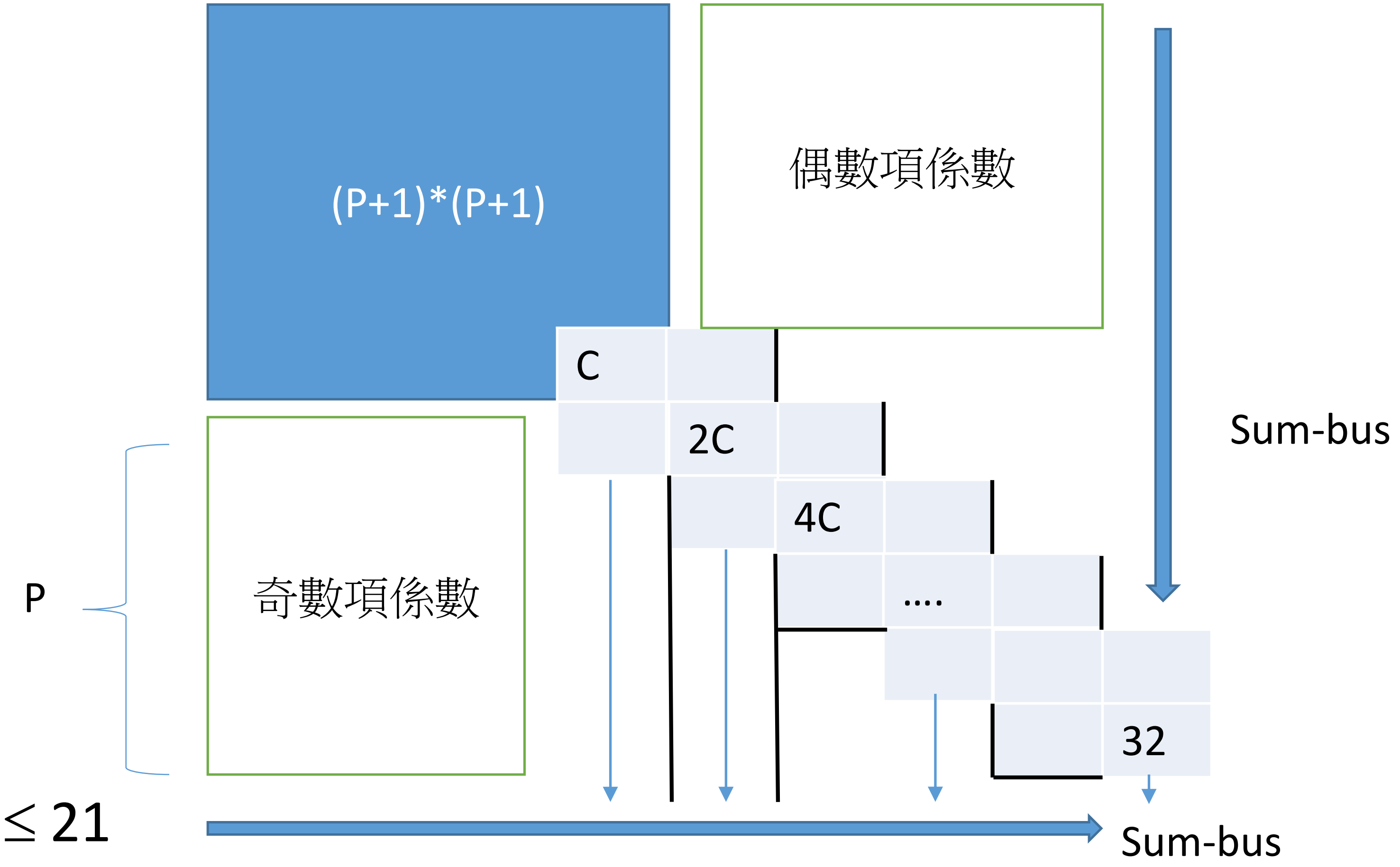
第三招：6進制





6進制需24位，N=24*2+1=49，列或行+3，N+M=101。
最後一個其實可以省下來, N+M=99

混合第一二招

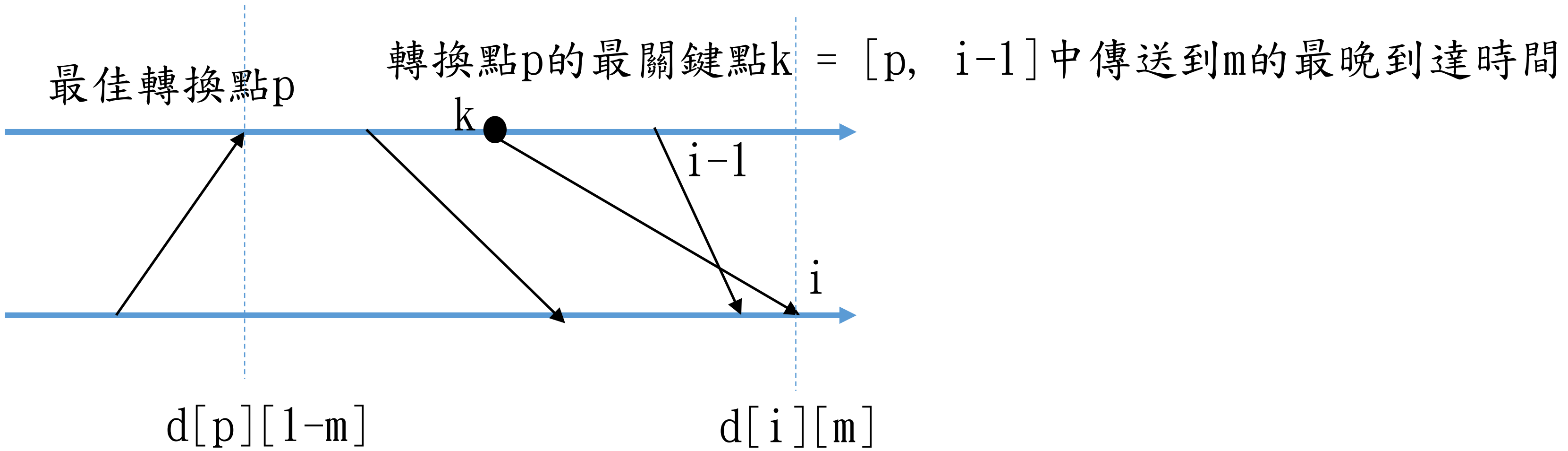


$C(2p,p)*2^p \leq 10^{18}$, $p \leq 21$

$N = M = 2p+2$ ，$N+M \leq 88$

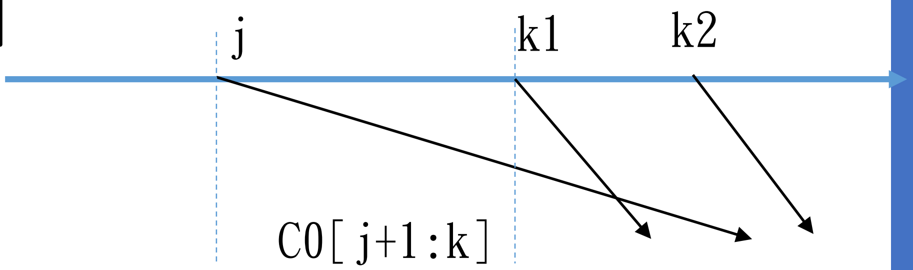
黑白機

- 對於每個 i 計算出在 i 點從黑機轉換到白機與白機轉黑機的最小完成時間
  - 0 -> 1 與 1 -> 0 是對稱的，做法相同。
  - 令 d[i][m]是 (i-1在 1-m 轉換到 i在m)的最小 i 結束時間
  - 最後答案是 min(d[i] + sum[i+1..n]}，嘗試每個 i是最後轉換點。
- 重點是如何算d[i][m]
- $d[i][m] = \min_{p<i}\left\{d[p][1-m] + \max_{p\le k<i}\{c[p+1:k][1-m] + t[k]\}\right\} + c[i][m]$
- P從i-1開始往前跑，直白一點就O(n^3)，記住並更新關鍵點，就O(n^2)



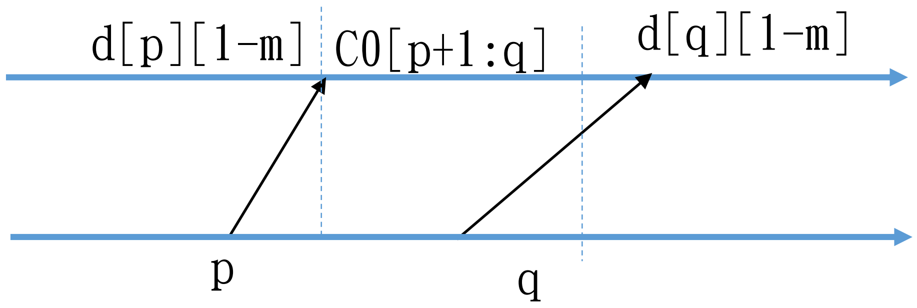
$$d[i][m] = \min_{p<i}\left\{d[p][1-m] + \max_{p\le k<i}\{c[p+1:k][1-m] + t[k]\}\right\} + c[i][m]$$

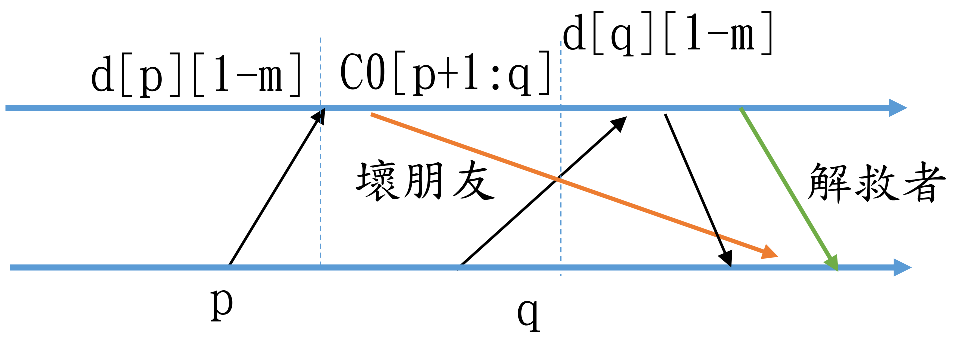
- 誰是關鍵點： j<k, t[j] + c0[j+1: k] <= t[k] 則 k 比較關鍵，否則 j 比較關鍵
  - 後者(k)較關鍵，則前者永遠無用，可砍掉
  - 前者(j)較關鍵，則未必；因為 j 不在某些轉移點範圍內 (j < p <= k)
- 有用的 (p1,k1), (p2,k2),…, => k值為遞減



$$d[i][m] = \min_{p<i}\left\{d[p][1-m] + \max_{p\le k<i}\{c[p+1{:}k][1-m] + t[k]\}\right\} + c[i][m]$$

- 轉移點誰較優，見右圖。
- p < q, 若後者(q)較優，則前者無用，可砍
  - 若 p 較優，未必解較好，因為 p 可能有壞朋友。但 p 不能死，只能先沉睡，因為可能出現解救者，有機會復活。
  - 出現更壞的延遲(解救者)，死豬不怕滾水燙，原來不能用 p 的理由(壞朋友)不再，用較好的 p 會得到更好的解





```
stack<Transfer> TP[2]; // (transfer point, max delay point)
int ans=min(psum[n][0], psum[n][1]);
for (i=2; i<=n; i++) {
  for (int m=0; m<2; m++) { // two machine
    // state from (1-m) -> m
    int new_p=i-1, new_k=i-1; // new transfer for i
    // pop useless delay, delay(j)+sum[j+1:k]<delay(k)
    while (!TP[m].empty() && less_delay(TP[m].top().k, new_k, 1-m)) {
        new_p = better_p(new_p, TP[m].top().p, 1-m); // possible better p
        TP[m].pop();
    }
    // if (new_p, new_k) is current best
    if (TP[m].empty() || delay({new_p,new_k},1-m)<delay(TP[m].top(),1-m)) {
        TP[m].push({new_p,new_k});
    }
    d[i][m]=delay(TP[m].top(),1-m) + c[i][m];
    ans = min(ans, d[i][m]+psum[n][m]-psum[i][m]);
 }
```

```cpp
// transfer point and its critical delay point
struct Transfer {
    int p,k;
};
// x<y, if x make less delay than y
bool less_delay(int x, int y, int m) {
    return t[x]<t[y]+psum[y][m]-psum[x][m];
}
// x>y, return y if y is a better transfer point
int better_p(int x, int y, int m) {
    if (d[y][m]+psum[x][m]-psum[y][m] < d[x][m])
        return y;
    return x;
}
// the delay for a transfer point and critical point
int delay(Transfer q,int m) {
    return d[q.p][m]+psum[q.k][m]-psum[q.p][m]+t[q.k];
}
```

Return 美好回憶 + 好朋友 + 經驗值；