NP 難題演算法系列 · 第 03 篇

Subset Sum 與 Partition

Subset Sum / Partition — DP & Meet-in-the-Middle

「湊得出這個數嗎？」——最精瘦的 NP-Complete 問題，以及把 2^n 砍成 $2^{n/2}$ 的漂亮技巧

Contents

1 問題是什麼

白話比喻

你和朋友分帳。收據上有一串金額，你們想知道：能不能挑出其中幾筆，剛好加起來是 500 元？或者更常見的：能不能把全部金額分成**兩堆一樣多**，一人付一堆？前者是 Subset Sum，後者是 Partition。

定義 1.1 (Subset Sum)**.** 給定正整數多重集 $A = \{a_1, \ldots, a_n\}$ 與目標 T，是否存在 $S \subseteq A$ 使 $\sum_{a \in S} a = T$？

定義 1.2 (Partition)**.** 給定 A，能否分成 $A_1 \uplus A_2$ 使兩邊總和相等？等價於 Subset Sum 取 $T = \frac{1}{2}\sum a_i$（總和為奇數直接無解）。

1.1 難度定位

- 兩者都是 **NP-Complete**（Karp 21 問題成員；由 3-SAT 經 Exact Cover 歸約而來）。
- 和背包一樣是**弱 NP-Hard**：有 $O(nT)$ 偽多項式 DP。數字小就好解，數字大（例如 40 個 13 位數）才露出獠牙。
- Partition 是「**最容易被低估的 NP-Complete 問題**」——敘述只有一句話，卻是排程、公平分配等問題難度的源頭（本系列第 12 篇 Job Scheduling 的難度正是從它來的）。

2 演算法一：偽多項式 DP

2.1 可達性表

$$reach[s] = \text{「用目前處理過的數字，能否湊出總和}s\text{」}$$

初始 $reach[0] =$ 真。每加入一個數 a_i，把所有可達的 s 推進到 $s + a_i$（一維陣列需**由大到小**掃，避免同一數字用兩次）。時間 $O(nT)$、空間 $O(T)$。

2.2 還原方案

把布林表升級成 $from[s] =$「s 第一次變可達時用的數字索引」。回溯時從 T 一路減回 0：

2.3 手算範例

$A = \{3, 34, 4, 12, 5, 2\}$、$T = 9$。依序處理，列出每步後的可達集合（只列 ≤ 9）：

加入	可達集合（≤ 9）
—	$\{0\}$
3	$\{0, 3\}$
34	$\{0, 3\}$（34 太大，推不進 9 以內）
4	$\{0, 3, 4, 7\}$
12	不變
5	$\{0, 3, 4, 5, 7, 8, 9\}$ $\quad \leftarrow 9 = 4 + 5$ 出現！
2	（已可達，不需要）

答案：YES，$9 = 4 + 5$。

3 演算法二：Meet-in-the-Middle

3.1 動機

若數字大到 10^{12}，$O(nT)$ 的表根本開不出來。純枚舉 2^n 在 $n = 40$ 時是 10^{12} 也不行。MITM 的想法：**把 n 個數切成兩半，各自枚舉，然後「在中間會合」**。

3.2 做法

1. 左半 $n/2$ 個數：枚舉全部 $2^{n/2}$ 個子集和，存進清單 L。
2. 右半同理得 R，將 R **排序**。
3. 對每個 $\ell \in L$，在 R 中二分搜尋 $T - \ell$。找到即 YES。

$$T(n) = O\big(2^{n/2} \cdot n\big) \qquad \text{（排序與搜尋的 log 因子併入 } n\text{）}$$

$n = 40$：$2^{40} \approx 10^{12}$ 變成 $2^{20} \approx 10^{6}$——**平方根級的加速**，代價是 $O(2^{n/2})$ 記憶體。

實務技巧

MITM 是一個通用範式：雙向 BFS、密碼學的中途相遇攻擊（對 2DES 的攻擊讓它形同虛設）、$n \le 40$ 的各種子集枚舉題，都是同一招。看到「$n \approx 40$、值域巨大」就要條件反射想到它。

4 延伸問題：最平衡分割

Partition 無解時，退而求其次：把 A 分成兩堆使**差最小**。做法：對 $T = \lfloor \frac{\sum a_i}{2} \rfloor$ 建可達表，從 T 往下找第一個可達的 s，答案為 $\sum a_i - 2s$。這是排程（兩台機器最小化 makespan）的核心子程序。

5 完整 C++ 程式

包含：DP + 方案還原、MITM（含 long long 大值版本）、Partition、最平衡分割。編譯：

```
g++ -std=c++17 -O2 -Wall -Wextra -o subset_sum_partition subset_sum_partition.cpp
```

```cpp
// subset_sum_partition.cpp
// ------------------------------------------------------------------
// Subset Sum 與 Partition：
//   1. Subset Sum 偽多項式 DP O(n·T) + 還原子集
//   2. Meet-in-the-Middle O(2^(n/2))（值域大時的武器）
//   3. Partition（總和均分）= Subset Sum 特例
//   4. 最平衡分割：最小化兩組差
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o subset_sum_partition
    subset_sum_partition.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <random>
#include <chrono>

using namespace std;

// ---------- 1. DP + 還原 ----------
// from[s] = 使 s 第一次可達的「物品索引」，-1 表示不可達（s=0 例外）
pair<bool, vector<int>> subsetSumDP(const vector<int>& a, int target) {
    vector<int> from(target + 1, -2);          // -2 = 不可達
    from[0] = -1;                              // -1 = 起點
```

```cpp
    for (int i = 0; i < (int)a.size(); ++i)
        for (int s = target; s >= a[i]; --s)
            if (from[s] == -2 && from[s - a[i]] != -2)
                from[s] = i;
    if (from[target] == -2) return {false, {}};
    vector<int> chosen;                          // 還原
    int s = target;
    while (s > 0) {
        int i = from[s];
        chosen.push_back(i);
        s -= a[i];
    }
    reverse(chosen.begin(), chosen.end());
    return {true, chosen};
}

// ---------- 2. Meet-in-the-Middle ----------
bool subsetSumMITM(const vector<int>& a, long long target) {
    int n = (int)a.size(), h = n / 2;
    auto sums = [](const vector<int>& part) {
        int m = (int)part.size();
        vector<long long> out;
        out.reserve(1LL << m);
        for (int mask = 0; mask < (1 << m); ++mask) {
            long long s = 0;
            for (int i = 0; i < m; ++i)
                if (mask & (1 << i)) s += part[i];
            out.push_back(s);
        }
        return out;
    };
    vector<int> L(a.begin(), a.begin() + h), R(a.begin() + h, a.end());
    auto sl = sums(L), sr = sums(R);
    sort(sr.begin(), sr.end());
    for (long long s : sl)
        if (binary_search(sr.begin(), sr.end(), target - s)) return true;
    return false;
}

// ---------- 3. Partition：總和為偶數且一半可達 ----------
pair<bool, vector<int>> partitionEqual(const vector<int>& a) {
    int total = accumulate(a.begin(), a.end(), 0);
    if (total % 2 != 0) return {false, {}};
    return subsetSumDP(a, total / 2);
}

// ---------- 4. 最平衡分割：找 <= total/2 的最大可達和 ----------
int minPartitionDiff(const vector<int>& a) {
    int total = accumulate(a.begin(), a.end(), 0);
    int half = total / 2;
    vector<bool> dp(half + 1, false);
    dp[0] = true;
    for (int x : a)
        for (int s = half; s >= x; --s)
            if (dp[s - x]) dp[s] = true;
    for (int s = half; s >= 0; --s)
        if (dp[s]) return total - 2 * s;         // 兩組差 = total - 2s
    return total;
}
```

```cpp

static void printSet(const vector<int>& a, const vector<int>& idx) {
    int sum = 0;
    cout << "{";
    for (size_t k = 0; k < idx.size(); ++k) {
        cout << a[idx[k]] << (k + 1 < idx.size() ? "," : "");
        sum += a[idx[k]];
    }
    cout << "} (sum=" << sum << ")";
}

int main() {
    // ---- 範例 1：Subset Sum 基本 + 還原 ----
    cout << "=== Example 1: subset sum with reconstruction ===\n";
    vector<int> a = {3, 34, 4, 12, 5, 2};
    for (int t : {9, 11, 30}) {
        auto [ok, idx] = subsetSumDP(a, t);
        cout << "target=" << t << " -> " << (ok ? "YES  " : "no");
        if (ok) printSet(a, idx);
        cout << "\n";
    }
    cout << "\n";

    // ---- 範例 2：Partition ----
    cout << "=== Example 2: partition into two equal halves ===\n";
    vector<int> b = {1, 5, 11, 5};
    auto [okB, idxB] = partitionEqual(b);
    cout << "{1,5,11,5}: " << (okB ? "YES  side A = " : "no");
    if (okB) printSet(b, idxB);
    cout << "\n";
    vector<int> c = {1, 2, 3, 5};
    cout << "{1,2,3,5}: " << (partitionEqual(c).first ? "YES" : "no (odd
        total or unreachable)") << "\n\n";

    // ---- 範例 3：最平衡分割 ----
    cout << "=== Example 3: most balanced split (min difference) ===\n";
    vector<int> d = {8, 6, 5, 14, 13, 9, 12, 7};
    cout << "{8,6,5,14,13,9,12,7} total=" << accumulate(d.begin(),
        d.end(), 0)
          << " -> min diff = " << minPartitionDiff(d) << "\n";
    vector<int> e = {3, 1, 4, 2, 2, 1};
    cout << "{3,1,4,2,2,1} total=13 -> min diff = "
         << minPartitionDiff(e) << " (odd total, best split is 6/7)\n\n";

    // ---- 範例 4：值域大時 DP 失效，MITM 接手 ----
    cout << "=== Example 4: huge values -> meet-in-the-middle ===\n";
    mt19937 rng(7);
    uniform_int_distribution<long long> big(1e12, 9e12);
    vector<int> f36;
    vector<long long> raw;
    for (int i = 0; i < 36; ++i) raw.push_back(big(rng));
    // 挑其中 7 個當作可達目標
    long long target = raw[2] + raw[5] + raw[11] + raw[17] + raw[23] +
        raw[29] + raw[33];
    // MITM 直接吃 long long；DP 需要陣列大小 = target，根本開不出來
    vector<int> dummy;   // (DP 版此處不適用，只示範 MITM)
    auto t0 = chrono::steady_clock::now();
    // 轉存到 int 會溢位，MITM 這裡用 long long 版本
    {
```

```cpp
        // 重寫一份 long long 輸入的 MITM（沿用同邏輯）
        int n = (int)raw.size(), h = n / 2;
        auto sums = [](const vector<long long>& part) {
            int m = (int)part.size();
            vector<long long> out;
            out.reserve(1LL << m);
            for (int mask = 0; mask < (1 << m); ++mask) {
                long long s = 0;
                for (int i = 0; i < m; ++i)
                    if (mask & (1 << i)) s += part[i];
                out.push_back(s);
            }
            return out;
        };
        vector<long long> L(raw.begin(), raw.begin() + h),
                          R(raw.begin() + h, raw.end());
        auto sl = sums(L), sr = sums(R);
        sort(sr.begin(), sr.end());
        bool found = false;
        for (long long s : sl)
            if (binary_search(sr.begin(), sr.end(), target - s)) { found
               = true; break; }
        auto t1 = chrono::steady_clock::now();
        cout << "n=36, values ~1e12, target reachable? "
             << (found ? "YES" : "no") << "  ("
             << chrono::duration<double, milli>(t1 - t0).count() << " ms,
                "
             << "2^18 = 262144 sums per half)\n";
        cout << "(DP would need an array of ~" << target
             << " entries -> impossible; MITM wins)\n";
    }
    (void)f36; (void)dummy; (void)subsetSumMITM;
    return 0;
}
```

程式碼 1: subset_sum_partition.cpp — Subset Sum 與 Partition 完整實作

6 執行結果與解讀

```
=== Example 1: subset sum with reconstruction ===
target=9 -> YES  {4,5} (sum=9)
target=11 -> YES  {4,5,2} (sum=11)
target=30 -> no

=== Example 2: partition into two equal halves ===
{1,5,11,5}: YES  side A = {11} (sum=11)
{1,2,3,5}: no (odd total or unreachable)

=== Example 3: most balanced split (min difference) ===
{8,6,5,14,13,9,12,7} total=74 -> min diff = 0
{3,1,4,2,2,1} total=13 -> min diff = 1 (odd total, best split is 6/7)

=== Example 4: huge values -> meet-in-the-middle ===
n=36, values ~1e12, target reachable? YES  (65.8 ms, 2^18 = 262144 sums per half)
(DP would need an array of ~31040422420851 entries -> impossible; MITM wins)
```

筆記

Example 4 是兩種演算法分工的分水嶺展示：36 個約 10^{12} 的數，DP 需要 31 兆格的陣列（> 200 TB），直接出局；MITM 每半邊只枚舉 $2^{18} = 262144$ 個和，65 毫秒解決。**選演算法前先看數值範圍**，是弱 NP-Hard 問題的第一課。

7 實務應用

- **公平分配**：遺產、合夥拆帳、雲端負載對半分——全是 Partition 的變形。
- **對帳與審計**：「這批交易中哪幾筆加起來等於這個異常總額？」——會計舞弊偵測的常見子問題。
- **密碼學**：背包密碼系統與格密碼的安全性分析都圍繞 Subset Sum 的難度；低密度實例可用 LLL 格基化簡攻破。
- **排程**：兩台相同機器的工作分配 = 最平衡分割（見本系列第 12 篇）。

8 練習題

1. 手算：$A = \{7, 3, 2, 5, 8\}$、$T = 14$，畫出可達集合的演化表並還原方案。
2. 把 DP 的可達表改用 `std::bitset`，用位移一次推進整排：`reach |= reach << a[i]`。實測比逐格快多少。
3. 計數版本：湊出 T 的**方法數**有幾種？（把布林改成計數，注意溢位。）
4. MITM 目前回報「可不可行」，把它升級成能還原「左右各選了哪些」。
5. 挑戰：3-Partition（分成三堆等和，每堆恰 3 個數）是**強** NP-Complete——想想為什麼偽多項式 DP 救不了它。

9 小結

方法	時間	空間	適用時機
偽多項式 DP	$O(nT)$	$O(T)$	$T \lesssim 10^8$
bitset DP	$O(nT/64)$	$O(T/8)$ 位元組	同上，快 64 倍
Meet-in-the-Middle	$O(2^{n/2}n)$	$O(2^{n/2})$	$n \le 44$、值域任意大
暴力枚舉	$O(2^n)$	$O(n)$	$n \le 25$

延伸閱讀：Horowitz & Sahni (1974) 首次提出 MITM 解 Subset Sum；Garey & Johnson *Computers and Intractability* 的 Partition 條目；Howgrave-Graham & Joux (2010) 更快的 $O(2^{0.337n})$ 演算法。