

Meet In The Middle (MITM)

Meet In The Middle (MITM)

Tác giả:

- Lê Minh Hoàng - Đại học Khoa học Tự nhiên, ĐHQG-HCM

Reviewer:

- Ngô Nhật Quang - The University of Texas at Dallas
- Phạm Công Minh - THPT chuyên Khoa học Tự Nhiên, ĐHQGHN

Giới thiệu

MITM là một kỹ thuật tìm kiếm được sử dụng khi đầu vào nhỏ nhưng không đủ nhỏ để có thể quay lui (backtracking). Trước khi tiếp tục về kỹ thuật MITM, chúng ta cần xem xét bài toán đơn giản sau:

CSES - Meet in the Middle [🔗](#)

Đề bài

Cho mảng t có N phần tử. Hỏi có bao nhiêu cách chọn tập con sao cho tổng bằng x .

Giới hạn:

- $1 \leq N \leq 40$
- $1 \leq x \leq 10^9$
- $1 \leq t_i \leq 10^9$

Thuật toán ngây thơ: Quay lui (Backtracking)

Ý tưởng

Ta duyệt qua tất cả các tập con có thể có rồi cập nhật kết quả bằng đệ quy (một cách khác để duyệt qua các tập con là sử dụng bitmask <insert bài bitmask>).

Cài đặt

```
1 | long long cnt;  
2 | // quay lui đến phần tử thứ i  
3 | // trong i-1 phần tử đầu, tổng các t[i] trong tập là sum  
4 | void Try(int i, int sum) {  
5 |
```

```

6      // tiếp tục quay lui với tập có sum > x là không cần thiết
7      if (sum > x) return;
8
9      if (i > n) {
10         if (sum == x) ++cnt;
11     }
12     else {
13         // không lấy phần tử thứ i
14         Try(i + 1, sum);
15         // lấy phần tử thứ i
16         Try(i + 1, sum + t[i]);
17     }
18 }
19 long long solve() {
20     cnt = 0;
21     Try(1, 0);
22     return cnt;
23 }

```

Thuật toán trên có độ phức tạp thời gian là $\mathcal{O}(2^N)$, không đủ nhanh để giải bài toán bởi vì 2^{40} khá lớn. Do đó, ta cần tìm một phương án tối ưu hơn.

Thuật toán tối ưu: kỹ thuật MITM

Kỹ thuật MITM được mô tả như sau:

- Đặt $K = N/2$
- Chia N phần tử thành 2 tập:
 - Tập X bao gồm K phần tử đầu tiên.
 - Tập Y bao gồm tất cả phần tử còn lại.
- Quay lui ở tập X và lưu tổng của tất cả tập con vào mảng A . Tương tự, quay lui ở tập Y và lưu tổng của tất cả tập con vào mảng B . Do đó, kích thước tối đa của mỗi mảng A và B là 2^K .
- Bây giờ, ta cần kết hợp 2 mảng A và B :
 - Cách đơn giản nhất là lặp qua từng phần tử của A , với mỗi phần tử, ta duyệt qua tất cả phần tử của B . Độ phức tạp là $\mathcal{O}(2^K \times 2^K) = \mathcal{O}(2^N)$ (không đủ nhanh)
 - Để tối ưu, ta sắp xếp mảng B trước. Sau đó, lặp qua từng phần tử của A , với mỗi phần tử, ta sử dụng [tìm kiếm nhị phân](#) \square trong mảng B . Độ phức tạp là $\mathcal{O}(2^K \times \log_2 2^K) = \mathcal{O}(2^K K)$
 - Một cách tối ưu khác là ta sắp xếp cả 2 mảng A và B trước, sau đó sử dụng [kỹ thuật hai con trỏ](#) \square . Độ phức tạp là $\mathcal{O}(\text{sort algorithm}) + \mathcal{O}(2^K)$

Cài đặt (sử dụng tìm kiếm nhị phân)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 40 + 2;
4  int n, x;
5

```

```

6   int t[N];
7   vector<int> A, B;
8
9   void TryX(int i, int sum) {
10      if (sum > x) return;
11
12      if (i > n / 2) A.push_back(sum);
13      else {
14          TryX(i + 1, sum);
15          TryX(i + 1, sum + t[i]);
16      }
17  }
18  void TryY(int i, int sum) {
19      if (sum > x) return;
20
21      if (i > n) B.push_back(sum);
22      else {
23          TryY(i + 1, sum);
24          TryY(i + 1, sum + t[i]);
25      }
26  }
27
28  int main() {
29      cin >> n >> x;
30      for (int i = 1; i <= n; ++i) cin >> t[i];
31
32      // Quay lui 2 tập X và Y
33      TryX(1, 0);
34      TryY(n / 2 + 1, 0);
35
36      // Sắp xếp mảng B
37      sort(B.begin(), B.end());
38
39      // Lặp qua mảng A và tìm kiếm nhị phân:
40      // - Đếm số lượng phần tử trong B có giá trị bằng x - A[i]
41      long long cnt = 0;
42      for (int sum : A) {
43          cnt += upper_bound(B.begin(), B.end(), x - sum)
44                - lower_bound(B.begin(), B.end(), x - sum);
45      }
46      cout << cnt << '\n';
47  }

```

Cài đặt (sử dụng kỹ thuật hai con trỏ)

```

1   // Quay lui 2 tập X và Y
2   TryX(1, 0);
3   TryY(n / 2 + 1, 0);
4
5   // Sắp xếp mảng A và B

```

```

0  sort(A.begin(), A.end(), greater<int>());
7  sort(B.begin(), B.end());
8
9  // Sử dụng kỹ thuật 2 con trỏ
10 long long cnt = 0;
11 for (int i = 0, j1 = 0, j2 = 0; i < A.size(); ++i) {
12     int s = x - A[i]; // cần đếm lượng B[j] thoả B[j] = s
13     while (j1 < B.size() && B[j1] < s) ++j1;
14     while (j2 < B.size() && B[j2] <= s) ++j2;
15     cnt += j2 - j1;
16 }
17 cout << cnt << '\n';

```

Ứng dụng

Bài toán 1: VNOJ - Cái túi 1 [↗](#)

Có N cục vàng, mỗi cục vàng có trọng lượng W_i và giá trị V_i . Bạn có một cái túi có tải trọng tối đa là M . Hỏi tổng giá trị vàng lớn nhất có thể thu được mà không làm rách túi.

Giới hạn:

- $1 \leq N \leq 40$
- $1 \leq M \leq 10^9$
- $1 \leq W_i, V_i \leq 10^8$

Ý tưởng

Áp dụng MITM, ta tách N cục vàng thành 2 tập X và Y , tập X chứa $N/2$ cục vàng đầu tiên và tập Y chứa phần còn lại.

Bây giờ, quay lui cho với mỗi tập X và Y , ta được 2 tập A và B chứa các cặp (tổng trọng lượng $sumW$, tổng giá trị $sumV$) của các tập con.

Để kết hợp 2 tập A và B , ta cần giải quyết bài toán con: Với mỗi cặp $(sumW_i, sumV_i)$ của tập A , ta cần tìm một cặp $(sumW_j, sumV_j)$ trong tập B sao cho $sumW_j \leq M - sumW_i$ và $sumV_j$ là lớn nhất.

Để giải bài toán con này, gợi ý là sắp xếp lại mảng B theo thứ tự tăng dần của $sumW_j$ và đặt $maxSumV_j = max(sumV_1, \dots, sumV_j)$ (phần này có thể tính nhanh bằng [mảng cộng dồn](#) [↗](#)).

Cài đặt

```

#include <bits/stdc++.h>
using namespace std;
const int N = 40 + 2, MaxSize = (1 << 20) + 10;
int n, m;
int w[N], v[N];

long long sumVA[MaxSize];

```

```

9  int sumWA[MaxSize];
10 int sizeA;
11
12 pair<int, long long> B[MaxSize];
13 int sizeB;
14 int sumWB[MaxSize];
15 long long maxSumVB[MaxSize];
16
17 void TryX(int i, int sumW, long long sumV) {
18     if (sumW > m) return;
19     if (i > n / 2) {
20         ++sizeA;
21         sumWA[sizeA] = sumW;
22         sumVA[sizeA] = sumV;
23         return;
24     }
25     TryX(i + 1, sumW, sumV);
26     TryX(i + 1, sumW + w[i], sumV + v[i]);
27 }
28
29 void TryY(int i, int sumW, long long sumV) {
30     if (sumW > m) return;
31     if (i > n) {
32         ++sizeB;
33         B[sizeB].first = sumW;
34         B[sizeB].second = sumV;
35         return;
36     }
37     TryY(i + 1, sumW, sumV);
38     TryY(i + 1, sumW + w[i], sumV + v[i]);
39 }
40
41 int main() {
42     cin >> n >> m;
43     for (int i = 1; i <= n; ++i) cin >> w[i] >> v[i];
44
45     TryX(1, 0, 0);
46     TryY(n / 2 + 1, 0, 0);
47     sort(B + 1, B + sizeB + 1);
48     for (int i = 1; i <= sizeB; ++i) {
49         sumWB[i] = B[i].first;
50         maxSumVB[i] = max(maxSumVB[i - 1], B[i].second);
51     }
52
53     long long maxValue = 0;
54     for (int i = 1; i <= sizeA; ++i) {
55         int j = upper_bound(sumWB + 1, sumWB + sizeB + 1, m - sumWA[i]) - sumWB;
56         maxValue = max(maxValue, sumVA[i] + maxSumVB[j]);
57     }
58     cout << maxValue;
59 }

```

Bài toán 2

Cho mảng a gồm n số nguyên, đếm số lượng dãy con tăng có độ dài 3.

Giới hạn:

- $1 \leq n \leq 2000$
- $1 \leq a_i \leq 10^9$

Ý tưởng

Đặt a_i, a_j, a_k ($i < j < k$) ứng với một dãy con tăng có độ dài 3.

Theo cách làm ngây thơ, với mỗi i , ta đếm số cặp (j, k) thoả mãn trong $\mathcal{O}(n^2)$, tổng độ phức tạp thời gian sẽ là $\mathcal{O}(n^3)$.

Ta có thể ứng dụng "middle" như sau: thay vì xét i đầu tiên, ta xét j đầu tiên.

Với mỗi j , ta đếm số lượng $i < j$ thoả $a_i < a_j$ và $k > j$ thoả $a_k > a_j$ trong $\mathcal{O}(n)$, tổng độ phức tạp thời gian lúc này sẽ là $\mathcal{O}(n^2)$.

Cài đặt

```

1  for (int j = 0; j < n; ++j) {
2      int smaller = 0, bigger = 0;
3      for (int i = 0; i < j; ++i) {
4          if (a[i] < a[j]) ++smaller;
5      }
6      for (int k = j + 1; k < n; ++k) {
7          if (a[k] > a[j]) ++bigger;
8      }
9      answer += smaller * bigger;
10 }
```

Bài toán 3: CSES - Sum of Four Values

Cho mảng a gồm n số nguyên và số nguyên x . Ta cần tìm 4 vị trí **phân biệt** sao cho tổng giá trị ở 4 vị trí đó bằng x .

Giới hạn:

- $1 \leq n \leq 1000$
- $1 \leq x, a_i \leq 10^9$

Ý tưởng

Đặt i, j, k, l ($i < j < k < l$) là 4 vị trí thoả mãn $a_i + a_j + a_k + a_l = x$.

Thuật toán ngây thơ của bài toán này là sử dụng 4 vòng lặp lồng nhau với độ phức tạp $\mathcal{O}(n^4)$.

```

1 | for (int i = 1; i <= n; ++i)
2 |     for (int j = i + 1; j <= n; ++j)
3 |         for (int k = j + 1; k <= n; ++k)
4 |             for (int l = k + 1; l <= n; ++l)
5 |                 if (a[i] + a[j] + a[k] + a[l] == x) { ... }

```

Ta có nhận xét: trong vòng lặp thứ 2 (biến j), ta đang giải bài toán: tìm 2 vị trí phân biệt **lớn hơn** j sao cho tổng giá trị của 2 vị trí đó bằng $x - a_i - a_j$.

Ta có thể giải bài toán này trước bằng cách:

- Duyệt qua tất cả cặp 2 vị trí (đặt 2 vị trí này là k và l , $k < l$), với mỗi cặp vị trí, ta có một tổng $a_k + a_l$.
- Với mỗi giá trị tổng, có thể có nhiều cặp vị trí khác nhau thoả mãn, ta chỉ cần lưu lại cặp có k lớn nhất (vì k càng lớn thì càng có nhiều j thoả mãn).

Cài đặt

Sử dụng `std::map` để lưu cặp vị trí của mỗi giá trị tổng.

```

#include <bits/stdc++.h>
using namespace std;
const int N = 1000 + 3;
int n, x;
int a[N];

int main() {
    cin >> n >> x;
    for (int i = 1; i <= n; ++i) cin >> a[i];

    // preprocess
    map<int, pair<int, int>> mp;
    for (int i = 1; i <= n; ++i)
        for (int j = i + 1; j <= n; ++j)
            mp[a[i] + a[j]] = make_pair(i, j);

    // solve
    for (int i = 1; i <= n; ++i)
        for (int j = i + 1; j <= n; ++j) {
            // thay vì 2 vòng for, bây giờ ta chỉ cần
            // truy vấn trên std::map
            int X = x - a[i] - a[j];
            if (mp.count(X)) {
                pair<int, int> arr = mp[X];
                if (j < arr.first) {
                    cout << i << ' ' << j << ' ' << arr.first << ' ' << arr.second;
                    return 0;
                }
            }
        }
}

```

```

31 |         }
32 |
33 |     cout << "IMPOSSIBLE";
    | }

```

Phân tích

Độ phức tạp tiền xử lý: $\mathcal{O}(n^2 \log(n^2))$

Độ phức tạp truy vấn: $\mathcal{O}(\log(n^2))$

Có $\mathcal{O}(n^2)$ truy vấn, vì thế, tổng độ phức tạp thời gian là: $\mathcal{O}(n^2 \log(n^2))$

Bài toán 4: Kattis - Playlist [↗](#)

Cho đồ thị có hướng n đỉnh ($n \leq 100$) và **bậc ngoài** của mỗi đỉnh không quá 40. Tất cả đỉnh đều được tô màu. Tìm một đường đi độ dài 9 sao cho 9 đỉnh trong đường đi có màu phân biệt. Nếu có nhiều cách chọn, in ra bất kỳ, ngược lại, in ra "fail".

Giới hạn thời gian là rất lớn (12 giây).

Ý tưởng

Tương tự [Bài toán 2](#), ta có thể ứng dụng "middle" như sau:

- ▶ Đặt đỉnh thứ 5 trong đường đi là u .
- ▶ Với mỗi u :
 - ▶ Ta có tập A gồm các đường đi độ dài 4 ứng với 4 đỉnh 1, 2, 3, 4 thoả mãn các màu là phân biệt và khác màu của u (bằng DFS hoặc 4 vòng for từ u **trong đồ thị ngược**)
 - ▶ Ta có tập B gồm các đường đi độ dài 4 ứng với 4 đỉnh 6, 7, 8, 9 thoả mãn các màu là phân biệt và khác màu của u (bằng DFS hoặc 4 vòng for từ u)
 - ▶ Độ phức tạp thời gian: $\mathcal{O}(40^4)$
- ▶ Bây giờ, để kết hợp 2 tập, ta cần giải bài toán:
 - ▶ Với mỗi đường đi độ dài 4 trong A , đặt là X , ta kiểm tra xem có tồn tại đường đi độ dài 4 trong B sao cho màu của 8 đỉnh là phân biệt.
- ▶ Ta có thể giải bài toán này bằng [Bao hàm - loại trừ](#) [↗](#) :

Số đường đi độ dài 4 trong B có màu phân biệt với $X = |B| - (\text{số đường đi trùng ít nhất 1 màu}) + (\text{số đường đi trùng ít nhất 2 màu}) - (\text{số đường đi trùng ít nhất 3 màu}) + (\text{số đường đi trùng cả 4 màu})$
- ▶ Độ phức tạp thời gian: $\mathcal{O}(2^4 \times 40^4)$
- ▶ Độ phức tạp thời gian của thuật toán là $\mathcal{O}(N \times 2^4 \times 40^4)$

Cài đặt

```

#include <bits/stdc++.h>
using namespace std;

const int N = 100 + 2;
int n;

```



```

int c[N];
vector<int> g[2][N];

int cntbit[16];
void init() {
    for (int msk = 1; msk < 16; ++msk) {
        cntbit[msk] = cntbit[msk >> 1] + (msk & 1);
    }
}

void readData() {
    cin >> n;
    map<string, int> artist;
    for (int i = 1; i <= n; ++i) {
        string name;
        cin >> name;
        c[i] = artist.count(name) ? artist[name] : (artist[name] = artist.size);

        int k, to;
        cin >> k;
        while (k--) {
            cin >> to;
            g[0][i].push_back(to);
            g[1][to].push_back(i);
        }
    }
}

vector<int> getAns(vector<int> res) {
    set<int> s;
    for (int u : res) s.insert(c[u]);

    for (int v0 : g[0][res.back()]) {
        if (s.count(c[v0])) continue;
        s.insert(c[v0]);
        for (int v1 : g[0][v0]) {
            if (s.count(c[v1])) continue;
            s.insert(c[v1]);
            for (int v2 : g[0][v1]) {
                if (s.count(c[v2])) continue;
                s.insert(c[v2]);
                for (int v3 : g[0][v2]) {
                    if (s.count(c[v3])) continue;

                    res.push_back(v0);
                    res.push_back(v1);
                    res.push_back(v2);
                    res.push_back(v3);
                    return res;
                }
            }
        }
        s.erase(c[v2]);
    }
}

```

```

        }
        s.erase(c[v1]);
    }
    s.erase(c[v0]);
}
return {};
}

int cnt[N * N * N * N];
int getHash(const array<int, 4> &a, int msk) {
    int hsh = 0;
    for (int i = 0; i < 4; ++i) {
        if (msk >> i & 1) hsh = hsh * N + c[a[i]];
    }
    return hsh;
}

vector<int> solve(int u) {
    vector<int> sav(1, 0);

    for (int v0 : g[0][u]) {
        if (c[v0] == c[u]) continue;
        for (int v1 : g[0][v0]) {
            if (c[v1] == c[v0] || c[v1] == c[u]) continue;
            for (int v2 : g[0][v1]) {
                if (c[v2] == c[v1] || c[v2] == c[v0] || c[v2] == c[u]) continue;
                for (int v3 : g[0][v2]) {
                    if (c[v3] == c[v2] || c[v3] == c[v1] || c[v3] == c[v0] || c[v3] == c[u]) continue;

                    array<int, 4> a = { c[v0], c[v1], c[v2], c[v3] };
                    sort(a.begin(), a.end());
                    for (int msk = 0; msk < 16; ++msk) {
                        int hsh = getHash(a, msk);
                        ++cnt[hsh];
                        sav.push_back(hsh);
                    }
                }
            }
        }
    }

    for (int v0 : g[1][u]) {
        if (c[v0] == c[u]) continue;
        for (int v1 : g[1][v0]) {
            if (c[v1] == c[v0] || c[v1] == c[u]) continue;
            for (int v2 : g[1][v1]) {
                if (c[v2] == c[v1] || c[v2] == c[v0] || c[v2] == c[u]) continue;
                for (int v3 : g[1][v2]) {
                    if (c[v3] == c[v2] || c[v3] == c[v1] || c[v3] == c[v0] || c[v3] == c[u]) continue;

                    array<int, 4> a = { c[v0], c[v1], c[v2], c[v3] };
                    sort(a.begin(), a.end());








```

```

109         int sum = 0;
110         for (int msk = 0; msk < 16; ++msk) {
111             int hsh = getHash(a, msk);
112             sum += cnt[hsh] * (cntbit[msk] & 1 ? -1 : 1);
113         }
114         if (sum > 0) {
115             vector<int> res = { v3, v2, v1, v0, u };
116             return getAns(res);
117         }
118     }
119 }
120 }
121 }
122
123 for (int x : sav) cnt[x] = 0;
124 return vector<int>();
125 }
126
127 void solve() {
128     for (int i = 1; i <= n; ++i) {
129         vector<int> vec = solve(i);
130         if (!vec.empty()) {
131             for (int x : vec) cout << x << ' ';
132             return;
133         }
134     }
135     cout << "fail";
136 }
137
138 int main() {
139     init();
140     readData();
141     solve();
142 }

```

Bài tập áp dụng khác

- [VNOJ - Chia vàng](#) 
- [VNOJ - Phân tập](#) 
- [VNOJ - Bipalindrome](#) 
- [Codeforces - 888E \(Maximum Subsequence\)](#) 
- [Codeforces - 1006F \(Xor-Paths\)](#) 
- [Codeforces - 995E \(Number Clicker\)](#) 
- [Atcoder - ARC135F \(Delete 1, 4, 7, ...\)](#) 

Được cung cấp bởi [Wiki.js](#)