Quy Hoạch Động Bitmask
BỒI DƯỠNG KIẾN THỨC NỀN HSG

Đỗ Phan Thuận

Email/Facebook: thuandp.sinhvien@gmail.com
Khoa Học Máy Tính
Đại Học Bách Khoa Hà Nội.

Ngày 8 tháng 7 năm 2023

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải cho từng dạng bài toán

- Duyệt toàn bộ
- Chia để trị
- Quy hoạch động
- Tham lam

Mỗi mô hình ứng dụng cho nhiều loại bài toán khác nhau

Quy hoạch động là gì?

- Là một mô hình giải bài
- Nhiều điểm tương đồng với hai phương pháp Chia để trị và Quay lui
- Nhắc lại Chia để trị:
  - Chia bài toán cha thành các bài toán con *độc lập*
  - Giải từng bài toán con (bằng đệ quy)
  - Kết hợp lời giải các bài toán con lại thành lời giải của bài toán cha
- Phương pháp quy hoạch động:
  - Chia bài toán cha thành các bài toán con *gối nhau*
  - Giải từng bài toán con (bằng đệ quy)
  - Kết hợp lời giải các bài toán con lại thành lời giải của bài toán cha
  - *Không tìm nhiều hơn một lần lời giải của cùng một bài toán*

Công thức Quy hoạch động

1. Tìm công thức quy hoạch động cho bài toán dựa trên các bài toán con
2. Cài đặt công thức quy hoạch động:
   Đơn giản là chuyển công thức thành hàm đệ quy
3. Lưu trữ kết quả các hàm đã tính toán

Nhận xét

Bước 1: tìm công thức quy hoạch động là bước khó nhất và quan trọng nhất. Bước 2 và 3 có thể áp dụng sơ đồ chung sau đây để thực hiện

Cài đặt Top-Down với Đệ quy có nhớ

```
map<problem, value> Memory;

value DP(problem P) {
    if (is_base_case(P))
        return base_case_value(P);

    if (Memory.find(P) != Memory.end())
        return Memory[P];

    value result = some value;
    for (problem Q in subproblems(P))
        result = Combine(result, DP(Q));

    Memory[P] = result;
    return result;
}
```

Bình luận

- Việc sử dụng hàm đệ quy để cài đặt công thức quy hoạch động là cách tiếp cận lập trình tự nhiên và đơn giản cho lập trình giải bài toán quy hoạch động, ta gọi đó là cách tiếp cận lập trình Top-Down, phù hợp với đa số người mới tiếp cận kỹ thuật Quy hoạch động
- Khi đã quen thuộc với các bài quy hoạch động ta có thể luyện tập phương pháp lập trình Bottom-Up, xây dựng dần lời giải từ các bài toán con đến các bài toán cha
- Các bước trên mới chỉ tìm ra được giá trị tối ưu của bài toán. Nếu phải đưa ra các phần tử trong lời giải tạo nên giá trị tối ưu của bài toán thì cần thực hiện thêm bước Truy vết. Bước Truy vết nên mô phỏng lại Bước 2 cài đặt đệ quy và tìm ra các phần tử của lời giải dựa trên thông tin các bài toán con đã được lưu trữ trong mảng memmory

2. Quy hoạch động trên bitmask
   - Bài toán người du lịch
   - Công thức Quy hoạch động
   - Cài đặt
   - Độ phức tạp
   - Truy vết

Quy hoạch động trên bitmask

- Có còn nhớ biểu diễn bitmask cho các tập con?
- Mỗi tập con của tập n phần tử được biểu diễn bởi một số nguyên trong khoảng $0, \ldots, 2^n - 1$
- Điều này có thể giúp thực hiện phương pháp quy hoạch động dễ dàng trên các tập con

Bài toán người du lịch

- Cho một đồ thi n đỉnh $\{0,1,\ldots,n-1\}$ và giá trị trọng số $C_{i,j}$ trên mỗi cặp đỉnh i,j. Hãy tìm một chu trình đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần sao cho tổng các trọng số trên chu trình đó là nhỏ nhất
- Đây là bài toán NP-khó, vì vậy không tồn tại thuật toán tất định thời gian đa thức nào hiện biết để giải bài toán này
- Thuật toán duyệt toàn bộ đơn giản duyệt qua toàn bộ các hoán vị các đỉnh cho độ phức tạp là $\mathcal{O}(n!)$, nhưng chỉ có thể chạy được đến $n \leq 11$
- Liệu có thể làm tốt hơn với phương pháp Quy hoạch động?

Bài toán người du lịch: Công thức Quy hoạch động

- Không mất tính tổng quát giả sử chu trình bắt đầu và kết thúc tại đỉnh 0
- Gọi $\mathrm{TSP}(i, S)$ là chi phí ít nhất để đi qua toàn bộ các đỉnh và quay trở lại đỉnh 0, nếu như hiện tại hành trình đang ở tại đỉnh i và người du lịch đã thăm tất cả các đỉnh trong tập S

- Bước cơ sở: $\mathrm{TSP}(i, \text{tập mọi đỉnh}) = C_{i,0}$
- Bước chuyển quy nạp: $\mathrm{TSP}(i, S) = \min_{j \notin S} \{ C_{i,j} + \mathrm{tsp}(j, S \cup \{j\}) \}$

Bài toán người du lịch: Cài đặt

```
1  const int N = 20;
2  const int INF = 100000000;
3  int C[N][N];
4  int iMem[N][1<<N];
5  memset(iMem, -1, sizeof(iMem));
6
7  int TSP(int i, int S) {
8      if (S == ((1 << N) - 1))  return C[i][0];
9      if (iMem[i][S] != -1)     return iMem[i][S];
10
11     int res = INF;
12     for (int j = 0; j < N; j++) {
13         if (S & (1 << j))
14             continue;
15         res = min(res, C[i][j] + TSP(j, S | (1 << j)));
16     }
17     iMem[i][S] = res;
18     return res;
19 }
```

Bài toán người du lịch: Cài đặt

- Kết quả tối ưu có thể được đưa ra như sau:

```
cout << TSP(0, 1<<0);
```

Bài toán người du lịch: Độ phức tạp tính toán

- Có $n \times 2^n$ khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(n)$
- Thời gian tính tổng cộng là $\mathcal{O}(n^2 \times 2^n)$
- Như vậy có thể tính nhanh được với n lên đến 20

Bài toán người du lịch: Độ phức tạp tính toán

- Có $n \times 2^n$ khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(n)$
- Thời gian tính tổng cộng là $\mathcal{O}(n^2 \times 2^n)$
- Như vậy có thể tính nhanh được với n lên đến 20

- Làm thế nào để đưa ra được chính xác hành trình của người du lịch?

Bài toán người du lịch - Truy vết bằng đệ quy

```
1  void Trace(int i, int S) {
2      cout << i << " ";
3      if (S == ((1 << N) - 1))  return;
4
5      int res = iMem[i][S];
6      for (int j = 0; j < N; j++) {
7        if (S & (1 << j))
8          continue;
9        if (res == C[i][j] + iMem[j][S | (1 << j)]) {
10         Trace(j, S | (1 << j));
11         break;
12       }
13     }
14 }
```

- Gọi `TSP(0, 1<<0);`
- Độ phức tạp hàm truy vết?

Bài toán người du lịch - Truy vết bằng đệ quy

```
1  void Trace(int i, int S) {
2      cout << i << " ";
3      if (S == ((1 << N) - 1))  return;
4
5      int res = iMem[i][S];
6      for (int j = 0; j < N; j++) {
7        if (S & (1 << j))
8          continue;
9        if (res == C[i][j] + iMem[j][S | (1 << j)]) {
10         Trace(j, S | (1 << j));
11         break;
12       }
13     }
14 }
```

- Gọi `TSP(0, 1<<0);`
- Độ phức tạp hàm truy vết? $\mathcal{O}(n^2)$

Bài toán người du lịch: Truy vết bằng vòng lặp

```
int ans = iMem[0][1];
cout << ans << endl;
stack<int> Stack;
Stack.push(0);
for (int i = 0, S = 1, k = 0; k < n-1; ++k) {
  for (int j = 0; j < n; ++j){
    if (!(S & (1 << j)) &&
      (iMem[i][S] == C[i][j] + iMem[j][S | (1 << j)])) {
        Stack.push(j);
        i = j;
        S = S | (1 << j);
    }
  }
}
while (!Stack.empty()) {
    cout << Stack.back() << " ";
    Stack.pop();
}
```

Bài toán Tổng trên tập con (SOS - Sum Over Subsets)

Cho một mảng A cố định gồm 2^N số nguyên, ta cần tính với $\forall x$ hàm $F(x) =$ Tổng của tất cả $A[i]$ sao cho $x\&i = i$, tức là i là một tập con của x.

$$F[mask] = \sum_{i \subseteq mask} A[i]$$

Phương pháp trực tiếp

```
for(int mask = 0;mask < (1<<N); ++mask){
    for(int i = 0;i < (1<<N); ++i){
        if((mask&i) == i){
            F[mask] += A[i];
        }
    }
}
```

- Độ phức tạp: $O(4^N)$

Phương pháp tốt hơn

```
// lặp qua tất cả các mask
for (int mask = 0; mask < (1<<n); mask++){
    F[mask] = A[0];
    // lặp qua tất cả các tập con của mask từ lớn về bé
    for(int i = mask; i > 0; i = (i-1) & mask){
        F[mask] += A[i];
    }
}
```

- Độ phức tạp: $O(3^N)$
- Lưu ý: đối với mỗi *mask*, ta chỉ lặp trên các tập con của nó. Do đó, nếu *mask* có K bit 1, ta chỉ thực hiện 2^K lần lặp. Ngoài ra, tổng số *mask* có K bit 1 là $\binom{N}{K}$. Do đó tổng số lần lặp $= \sum_{K=0}^{N} \binom{N}{K} 2^K = (1+2)^N = 3^N$.

Phương pháp quy hoạch động SOS

- Cải tiến bằng cách lặp tất cả các tập con của mặt nạ theo cách thông minh hơn.
- Ở phương pháp trước, một chỉ số $A[x]$ với x có K bit 0 được truy cập bởi 2^K $mask$, dẫn đến lặp lại về tính toán.
- Phương pháp trước không thiết lập bất kỳ mối quan hệ nào giữa các $A[x]$ đang được sử dụng bởi các $F[mask]$ khác nhau.
- Vì vậy, bằng cách nào đó, chúng ta cần thêm một trạng thái khác vào các mask này để tạo thành các nhóm tránh tính toán lại cùng nhóm.

Ý tưởng QHĐ SOS

- Gọi $S(mask) = \{x | x \subseteq mask\}$.
- Phân tập S thành các nhóm không giao nhau

$$S(mask, i) = \{x | x \subseteq mask \&\& mask \oplus x < 2^{i+1}\}$$

là tập các tập con của *mask* mà khác với *mask* ở $i + 1$ bit đầu tiên (từ bit thứ 0 đến bit thứ i). Ví dụ:
$S(\mathbf{101}1010, 3) = \{\mathbf{101}1010, \mathbf{101}0010, \mathbf{101}1000, \mathbf{101}0000\}$.
Điều này giúp ta biểu diễn bất kỳ tập hợp nào dưới dạng hợp của một số tập không giao nhau.

Thuật toán QHĐ SOS

Xét $S(mask, i)$, tập chứa tất cả các tập con của $mask$ chỉ khác ở $i+1$ bit đầu tiên:

- Bit thứ i của $mask$ bằng 0: Không tồn tại tập con nào của $mask$ có bit thứ i bằng 1. Do đó, các số trong tập hợp này chỉ có thể khác nhau ở i bit đầu tiên. Ta có:

$$S(mask, i) = S(mask, i-1)$$

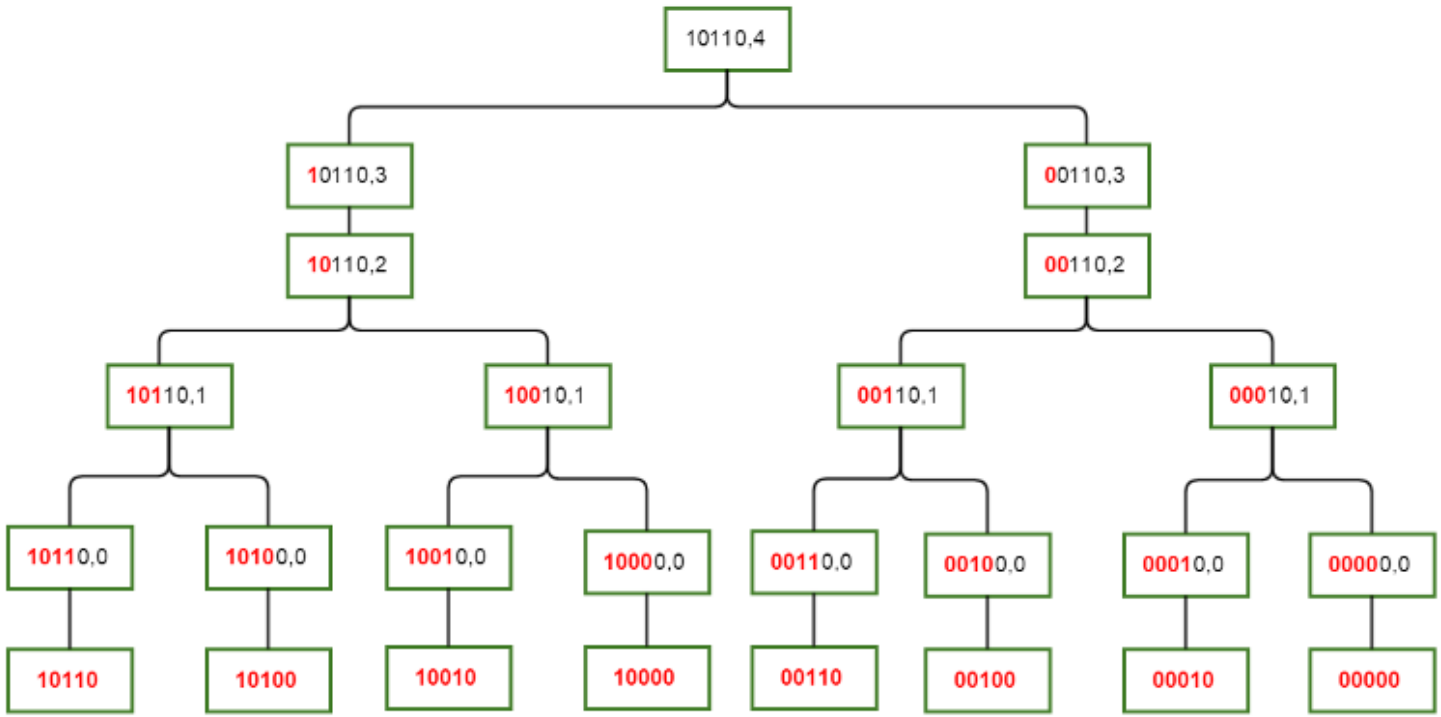
- Bit thứ i của $mask$ bằng 1: Các số thuộc $S(mask, i)$ có thể được chia thành hai tập hợp không giao nhau:
  - Tập thứ nhất chứa các số có bit thứ i là 1 và khác với $mask$ ở i bit tiếp theo.
  - Tập thứ hai chứa các số có bit thứ i là 0 và khác với $mask \oplus 2^i$ ở i bit tiếp theo. Ta có:

$$S(mask, i) = S(mask, i-1) \cup S(mask \oplus 2^i, i-1)$$

$$S(mask, i) = \begin{cases} S(mask, i-1) & \text{bit thứ } i \text{ bằng 0} \\ S(mask, i-1) \bigcup S(mask \oplus 2^i, i-1) & \text{bit thứ } i \text{ bằng 1} \end{cases}$$

Sơ đồ liên kết QHĐ SOS

Sơ đồ sau mô tả cách liên kết các tập $S(mask, i)$ với nhau. Các phần tử của một tập $S(mask, i)$ là các lá trong cây con của nó. Các tiền tố màu đỏ mô tả phần này của *mask* là chung cho tất cả các con của nó trong khi phần màu đen của *mask* có thể thay đổi.



Lưu ý: các mối quan hệ này tạo thành một đồ thị có hướng không chu trình và không nhất thiết phải là cây (trường hợp các giá trị khác nhau của *mask* và cùng một giá trị của i)

Code QHĐ SOS

```
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //xét riêng trường hợp cơ sở (các nút lá)
    for(int i = 0;i < N; ++i){
        if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
        else
            dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
}
//bộ nhớ được tối ưu, dễ dàng để code
for(int i = 0; i<(1<<N); ++i)
    F[i] = A[i];
for(int i = 0;i < N; ++i)
    for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
}
```

- Độ phức tạp: $O(N \cdot 2^N)$

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict