2.1 Exhaustive search (Thuật toán vét cạn)

Created by TUNG DUC NGUYEN tung2.nguyen, last modified on 2021/05/12

1. Định nghĩa

Trong khoa học máy tính, vét cạn (brute-force search hoặc exhaustive search), còn được gọi là tạo và kiểm tra, là một kỹ thuật giải quyết vấn đề rất chung và mô hình thuật toán bao gồm liệt kê một cách có hệ thống tất cả các ứng viên có thể có cho giải pháp và kiểm tra xem mỗi ứng viên có đáp ứng yêu cầu của bài toán hay không.

Mặc dù vét cạn rất dễ thực hiện và luôn tìm ra giải pháp nếu nó tồn tại, nhưng chi phí thực hiện tỷ lệ thuận với số lương giải pháp. Các vấn đề trong thực tế có xu hướng tăng rất nhanh số lượng giải pháp khi quy mô của vấn đề tăng lên. Do đó, vét cạn thường được sử dụng khi kích thước hạn chế hoặc các kinh nghiệm về vấn đề có thể làm giảm số lượng giải pháp xuống mức chấp nhận được. Phương pháp này cũng được sử dụng khi tính đơn giản của việc thực hiện quan trọng hơn tốc độ.

Thuật toán

```
c ← first(P)
while c ≠ Λ do
    if valid(P,c) then
        output(P, c)
    c ← next(P, c)
end while
```

Xác định candidate solution c đầu tiên cho P

Kiểm tra xem c có phải là "null candidate" hay không

Kiểm tra xem c có phải là giải pháp hợp lệ của P hay không

In ra giải pháp hợp lệ c của P

Sinh ra candidate solution c tiếp theo.

Ví dụ: tìm số lớn nhất trong dãy số nguyên cho trước (-5, 2, -1, 0, 3, 1)

Các bước chạy:

› Click here to expand...

Step	Explanation
c ← first(P)	c = -5
while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = true
output(P, c)	max = -5
c ← next(P, c)	c = 2

while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = true
output(P, c)	max = 2
c ← next(P, c)	c = -1
while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = false
c ← next(P, c)	c = 0
while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = false
c ← next(P, c)	c = 3
while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = true
output(P, c)	max = 3
c ← next(P, c)	c =1
while c ≠ Λ do	c ≠ Λ = true
if valid(P,c) then	valid(P,c) = false
c ← next(P, c)	c = Λ
while c ≠ Λ do	c ≠ Λ = false

2.Tăng tốc độ vét cạn

2.1 Giảm không gian tìm kiếm

Một cách để tăng tốc thuật toán vét cạn là giảm không gian tìm kiếm, tức là tập hợp các giải pháp ứng viên, bằng cách sử dụng phương pháp kinh nghiệm. Việc một chút phân tích thường sẽ dẫn đến việc giảm đáng kể số lượng các khả năng và có thể biến một vấn đề khó thành một vấn đề tầm thường.

> Ví dụ: trong bài toán tám quân hậu, bài toán là đặt tám quân hậu trên một bàn cờ tiêu chuẩn sao cho không có quân hậu nào tấn công con nào khác. Vì mỗi quân hậu có thể được đặt vào bất kỳ ô nào trong số 64 ô vuông, về nguyên tắc có 64^8 = 281.474.976.710.656 khả năng cần xem xét. Tuy nhiên, bởi vì tất cả các quân hậu đều giống nhau, và không có hai quân hậu nào có thể được xếp vào cùng một hình vuông, nên tất cả các ứng cử viên đều có thể chọn một bộ 8 ô vuông từ tập tất cả 64 ô vuông; có nghĩa là 64 chọn 8 = 64! / (56! * 8!) = 4.426.165.368 khả năng - khoảng 1 / 60.000 so với ước tính trước đó. Hơn nữa, không có sự sắp xếp nào với hai quân hậu trên cùng một hàng hoặc cùng một cột có thể là một giải pháp. Do đó, chúng ta có thể hạn chế hơn nữa nhóm khả năng đối với những sắp xếp đó.

Trong một số trường hợp, phân tích có thể giảm các khả năng xuống thành tập hợp tất cả các khả năng hợp lệ; nghĩa là, có thể có một thuật toán liệt kê trực tiếp tất cả các giải pháp hợp lệ, mà không lãng phí thời gian với phần kiểm tra và tạo ra các khả năng không hợp lệ.

> Ví dụ: đối với bài toán "tìm tất cả các số nguyên từ 1 đến 1.000.000 chia hết cho 417", một giải pháp vét cạn sẽ tạo ra tất cả các số nguyên trong phạm vi, kiểm tra tính chia hết của từng số đó. Tuy nhiên, vấn đề đó có thể được giải quyết hiệu quả hơn nhiều bằng cách bắt đầu với 417 và liên tục thêm 417 cho đến khi con số vượt quá 1.000.000 - chỉ mất 2398 (= 1.000.000 ÷ 417) bước và không cần kiểm tra.

2.2 Sắp xếp lại không gian tìm kiếm

Trong các bài toán chỉ yêu cầu một giải pháp, thay vì tất cả các giải pháp, thời gian chạy dự kiến của vét cạn thường sẽ phụ thuộc vào thứ tự mà các khả năng được kiểm tra. Theo nguyên tắc chung, người ta nên kiểm tra những khả năng có triển vọng nhất trước. Và thường thì xác suất của một khả năng hợp lệ thường bị ảnh hưởng bởi các khả năng thất bại trước đó.

> Ví dụ: khi tìm kiếm ước riêng của một số ngẫu nhiên n, tốt hơn nên liệt kê các ước ứng viên theo thứ tự tăng dần, từ 2 đến n - 1, hơn là ngược lại - bởi vì xác suất n chia hết cho c là 1 / c.

3. Sự bùng nổ của các bài toán tổ hợp

Nhược điểm chính của phương pháp vét cạn là đối với nhiều bài toán trong thế giới thực, số lượng khả năng là rất lớn. Hãy xem xét các bài toán sau:

> **Bài toán 1**: Cho một danh sách n thành phố và khoảng cách giữa từng cặp thành phố, tìm con đường ngắn nhất có thể đến thăm mỗi thành phố chính xác một lần và quay trở lại thành phố ban đầu?
>
> https://www.spoj.com/PTIT/problems/BCTSP/
>
> **Bài toán 2**: Cho một danh sách n thành phố và khoảng cách giữa từng cặp thành phố, tìm con đường ngắn nhất có thể đến thăm chính xác m thành phố (m <= n), mỗi thành phố chính xác một lần và quay trở lại thành phố ban đầu?
>
> **Bài toán 3**: Cho một danh sách n thành phố và khoảng cách giữa từng cặp thành phố, tìm con đường ngắn nhất có thể đến thăm chính xác m thành phố (m <= n), mỗi thành phố chính xác một lần và không quay trở lại thành phố ban đầu?
>
> **Bài toán 4**: Cho một danh sách n thành phố và chi phí đi đến từng thành phố và giá trị văn hóa của từng thành phố, tìm cách đi sao cho giá trị văn hóa thu được là lớn nhất với một chi phí c cho trước?
>
> https://www.spoj.com/problems/KNAPSACK/

Bài toán 1 số lượng khả năng là **hoán vị của n**, bài toán 2 số lượng khả năng là **tổ hợp chập m của n** (hay còn gọi là các tập hợp con m phần tử), bài toán 3 là số lượng khả năng là **chỉnh hợp không lặp chập m của n** (hay còn gọi là tập hợp các tập con m phần tử có vị trí phần tử khác nhau), bài toán 4 số lượng khả năng là **hai mũ n**.

Với bài toán 1,2,3, số lượng khả năng máy tính thông thường chỉ xử lý được với kích cỡ 12 hoặc 13.

5	6	7	8	9	10	11	12	13	14	15	30	100
120	720	5040	40320	362880	4E+06	4E+07	5E+08	6E+09	9E+10	1E+12	3E+32	9E+157

Với bài toán số 4, số lượng khả năng máy tính thông thường chỉ xử lý được với kích cỡ 29 hoặc 30.

1	5	10	20	30	40	50	100	1000
2	32	1024	1048576	1.07E+09	1.1E+12	1.13E+15	1.27E+30	1.1E+301

4.Phương pháp sinh (generation)

Việc giải quyết bằng phương pháp vét can đòi hỏi cần có phương pháp sinh ra các khả năng. Phương pháp sinh có thể mô tả như sau:

```
<Xây dựng cấu hình đầu tiên>;
repeat
   <Đưa ra cấu hình đang có>;
   <Từ cấu hình đang có đưa ra cấu kình kế tiếp nếu còn>;
until <hết cầu hình>;
```

Việc từ cấu hình đang có đưa ra cấu hình kế tiếp là giải thuật chính của phương pháp sinh. Chúng ta xem xét việc đưa ra cấu hình tiếp như thế nào.

Với cơ sở lý thuyết tất cả các cấu hình (candidate) là độc nhất. Do đó tồn tại một thứ tự cho việc sắp xếp các cấu hình này và được gọi là **thứ tự từ điển**.

Thứ tự từ điển đối với các tập hợp có cùng độ dài:

Ví dụ: các tập con có ba phần tử có tính vị trí của tập hợp (1, 2, 3) theo thứ tự từ điển là:

1. (1, 2, 3)
2. (1, 3, 2)
3. (2, 1, 3)
4. (2, 3, 1)
5. (3, 1, 2)
6. (3, 2, 1)

Hai phần tử thứ i = (i1, i2, ..., in) và j = (j1, j2,...,jn), phần tử i nhỏ hơn phần tử j (i < j) nếu tồn tại một số nguyên dương k: 1≤ k < n để:

i1 = j1

i2 = j2

...

i(k-1) = j(k-1)

i(k) < j(k)

Thứ tự từ điển đối với các tập hợp không có cùng độ dài:

Ví dụ: các tập con của tập hợp (1, 2, 3) theo thứ tự từ điển là:

1. (Ø)
2. (1)
3. (1, 2)
4. (1, 2, 3)
5. (2)
6. (2, 3)
7. (3)

Bằng cách thêm các phần tử rỗng (Ø) dể độ dài của hai phần tử bằng nhau, ta xác định được thứ tự từ điển của các phần tử có độ dài không bằng nhau.

Dưới đây là các phương pháp sinh:

4.1Liệt kê các hoán vị (bài toán 1):

Ta sẽ lập chương trình liệt kê các hoán vị của {1,2,..,n} theo thứ tự từ điển.

Ví dụ với n = 4, ta phải liệt kê đủ 24 hoán vị:

1>1234	2>1243	3>1324	4>1342	5>1423	6>1432
7>2134	8>2143	9>2314	10>2341	11>2413	12>2431
13>3124	14>3142	15>3214	16>3241	17>3412	18>3421
19>4123	20>4132	21>4213	22>4231	23>4312	24>4321

Như vậy hoán vị đầu tiên sẽ là (1,2,..,n) và hoán vị cuối cùng là (n,...,2,1).

Hoán vị sẽ sinh phải lớn hơn hoán vị hiện tại, hơn thế nữa phải là hoán vị vừa đủ lớn hơn hoán vị hiện tại nghĩa là không thể có một hoán vị nào chen giữa chúng khi săp sếp thứ tự.

Giả sử hoán vị hiện tại là x = (3, 2, 6, 5, 4, 1), xét 4 phần tử cuối cùng, ta thấy chúng được sắp xếp giảm dần, điều đó có nghĩa là cho dù chúng ta có hoán vị 4 phần tử này thế nào, ta cũng được một hoán vị bé hơn hoán vị hiện tại.

Như vậy ta phải xét đến đến x2 = 2, thay nó bằng một giá trị khác. Vậy giá trị nào sẽ thay thế x2?

Giá trị thay thế x2 không thể là 1 vì nếu vậy sẽ được hoán vị nhỏ hơn.

Giá trị thay thế x2 không thể là 3 vì x1 đang có giá trị là 3 (phần tử sau không được chọn vào những giá trị mà phần tử trước đã chọn).

Các giá trị còn lại để thay thế x2 là 4,5,6. Vì cần một hoán vị **vừa đủ lớn hơn hiện tại** nên ta giá trị thay thế cho x2 là 4. Còn các giá trị (x3,x4,x5,x6) sẽ lấy trong tập {2, 6, 5 ,1}.

Cũng vì tính vừa đủ nên ta sẽ tìm cách biểu diễn nhỏ nhất của bốn số này gán cho x3,x4,x5,x6 tức là (1,2,5,6).

Vậy hoán vị mới sẽ là (3, 4, 1, 2, 5 ,6).

(3, 2, 6, 5, 4, 1) → (3, **4, 1, 2, 5 ,6**)

Qua ví dụ này ta nhận xét:

- **Đoạn cuối** của hoán vị được **sắp xếp giảm dần**.
- Số x5 = 4 là số nhỏ nhất trong đoạn cuối giảm dần thỏa mãn điều kiện lớn hơn x2 = 2. Nếu ta đổi chỗ x5 cho x2 thì ta vẫn được **đoạn cuối sắp xếp giảm dần**.
- Sau khi đổi chỗ x5 với x2, ta muốn biểu diễn giá trị nhỏ nhất của đoạn cuối thì chỉ cần **đảo ngược đoạn cuối**.

Trong trường hợp hoán vị hiện tại là (2, 1, 3, 4) thì hoán vị kế tiếp là (2, 1, 4, 3):

- Đoạn cuối giảm dần là (4).
- Giá trị nhỏ nhất lớn hơn x3 là 4 → đổi chỗ x3 và x4 ta được (2, 1, 4, 3) và có đoạn cuối giảm dần là (3).
- Đạo ngược đoạn cuối (3) → (3), cuối cùng ta có hoán vị kế tiếp là (2, 1, 4, 3).

Vậy kỹ thuật sinh hoán vị kế tiếp từ hoán vị hiện tại có thể xây dựng như sau:

1. *Xác định đoạn cuối giảm dần dài nhất, tìm chỉ số i của phần tử x(i) đứng liền trước đoạn đó. Điều đó đồng nghĩa với việc tìm từ sát vị trí cuối dãy lên đầu, gặp chỉ số i đầu tiên thỏa mãn x(i) < x(i+1). Nếu toàn dãy là giảm dần → đó là cấu hình cuối.*

```
i = n - 1;
while ((i > 0) && (x[i] > x[i+1])) --i;
```

2. *Trong đoạn cuối giảm dần, tìm phần tử x(k) sao cho thỏa mãn điều kiện x(k) > x(i). Do đoạn cuối giảm dần, ta cũng có thể tìm chỉ số k bằng cách tìm cuối dãy lên đầu và dừng khi gặp chỉ số k đầu tiên thỏa mãn x(k) > x(i) hoặc tìm kiếm nhị phân.*

```
k = n;
while (x[k] < x[i]) --k;
```

3. *Đổi chỗ x(k) và x(i), lật ngược thứ tự đoạn cuối giảm dần trở thành tăng dần.*

```
swap(x[i], x[k]);
a = i + 1;
b = n;
while (a < b) {
    swap(x[a], x[b]);
    ++a;
    --b;
}
```

4.2 Liệt kê các tập con k phần tử (bài toán 2):

Ta sẽ lập chương trình liệt kê các tập con k phần tử của tập {1, 2, ..., n} theo thứ tự từ điển.

Ví dụ: với n = 5, k = 3, ta phải liệt kê đầy đủ 10 tập con:

1> {1, 2, 3} 2> {1, 2, 4} 3> {1, 2, 5} 4> {1, 3, 4} 5> {1, 4, 5}

6> {1, 4, 5} 7> {2, 3, 4} 8> {2, 3, 5} 9> {2, 4, 5} 10> {3, 4, 5}

Như vậy tập con đầu tiên (cấu hình khởi tạo) là {1, 2, ..., k}.

Cấu hình kết thúc là {n - k + 1, n - k + 2, ..., n}.

Nhận xét:

- Các tập con có phần tử được sắp xếp theo thứ tự tăng dần → **giới hạn trên của các phần tử thứ i sẽ là x(i) = n - k + i;**
- Các tập con có phần tử được sắp xếp theo thứ tự tăng dần → **giới hạn dưới của các phần tử thứ i sẽ là x(i) = x(i-1) + 1;**
- Giới hạn trên của các phần tử thứ i sẽ là x(i) = n - k + i → tất cả phần tử đạt giới hạn thì quá trình sinh kết thúc.
- Dãy mới sinh ra phải tăng dần thỏa mãn **vừa đủ lớn** hơn dãy cũ.

Ví dụ: n = 9m k = 6. Cấu hình đang có x = {1, 2, 6, 7, 8, 9}.

Các phần tử x3 đến x6 đã đạt giới hạn trên nên ta không thể sinh bằng cách tăng một phần tử trong các số x6, x5, x4, x3 lên được → ta phải tăng x2 lên một đơn vị x2 = 3, cấu hình mới sẽ là {1, 3, 6, 7, 8, 9}.

Cấu hình {1, 3, 6, 7, 8, 9} đã thỏa mãn lớn hơn cầu hình hiện tại nhưng chưa thỏa mãn vừa đủ lớn, nên ta phải thay x3, x4, x5, x6 bằng các giới hạn dưới của nó, tức là:

x3 = x2 + 1 = 4

x4 = x3 + 1 = 5

x5 = x4 + 1 = 6

x6 = x5 + 1 = 7

Ta được cấu hình mới x = {1, **3**, 4, 5, 6, 7} là cấu hình kế tiếp.

Nếu muốn tìm tiếp, ta thấy x6 = 7 chưa đạt giới hạn trên, như vậy chỉ cần tăng x6 lên một đơn vị là được x = {1, 3, 4, 5, 6, **8**}.

Vậy kỹ thuật sinh tập con có thể xây dựng như sau:

1. Tìm từ cuối lên đầu dãy cho đến khi gặp phần tử x(i) chưa đạn giới hạn trên n - k + i.

```
i = n;
while ((i > 0) && (x[i] == n - k + i)) --i;
if (i == 0) continue;// lùi đến 0 nên cấu hình hiện tại là cấu hình kết thúc.
```

2. Tăng x(i) lên một đơn vị.

```
++x[i];
```

3. Đặt lại tất cả các phần tử phía sau bằng giới hạn dưới.

```
for (j = i + 1; j <= k; ++j) x[j] = x[j-1] + 1;
```

4.3 Liệt kê các tập con k phần tử có vị trí phần tử khác nhau (bài toán 3):

Nhận xét rằng việc liệt kê này bao gồm bài toán 2 và bài toán 1.

Do vậy kỹ thuật sinh là sử dụng kỹ thuật sinh các tập con (bài toán 2) và liệt kê các hoán vị trong tập con đó (bài toán 1) → Bài tập về nhà.

4.4 Liệt kê các dãy nhị phân (bài toán 4):

Một dãy nhị phân độ dài n là một dãy x =x1x2...x(n) trong đó x(i) € {0 , 1} với i: 1 ≤ i ≤ n.

Dễ thấy một dãy nhị phân độ dài n là biểu diễn của một giá trị nguyên p(x) nào đó nằm trong [0, 2^n). Do đó ta sẽ lập chương trình liệt kê các dãy nhị phân theo thứ tự từ điển của các số nguyên có thứ tự 0, 1, ..., (2^n)- 1.

Ví dụ: khi n = 3, các dãy nhị phân độ dài 3 được liệt kê như sau:

p(x)	0	1	2	3	4	5	6	7

x	000	001	010	011	100	101	110	111

Như vậy dãy đầu tiên sẽ là 00...0 và dãy cuối cùng sẽ là 11...1.

Nhận xét thấy rằng nếu dãy x = (x1, x2, ..., xn) không phải là dãy cuối cùng thì dãy kế tiếp sẽ được sinh bằng cách cộng thêm 1 (theo cơ số 2 có nhớ) vào dãy hiện tại.

Ví dụ: khi n = 8:

Dãy đang có:	10010000	Dãy đang có:	10010111
cộng thêm 1:	1	cộng thêm 1:	1
Dãy mới:	10010001	Dãy mới:	10011000

Như vậy kỹ thuật sinh: Xét từ cuối dãy về đầu, gặp số 0 đầu tiên thì thay nó bằng số 1 và đặt tất cả các phần tử phía sau vị trí đó bằng 0.

```
i = n;
while ((i > 0) && (x[i] == 1)) --i;
if (i > 0) {
   x[i] = 1;
   for (j = i + 1; j <=n; ++j) x[j] = 0;
}
```

5. Discussion question

1. Tại sao vét cạn luôn là phương pháp tiếp cận đầu tiên để giải bài toán?
2. Tăng tốc độ vét cạn bao gồm những chiến thuật nào?
3. Theo bạn, giới hạn của việc áp dụng vét cạn cho các bài toán tổ hợp là bao nhiêu và vì sao?
4. Lấy ví dụ trong thực tế (cuộc sống, công việc) bạn đã áp dụng thuật toán vét cạn. Cách bạn tăng tốc các xử lý.

6. Homework

2.1.0 Homework

7. Xem thêm:

2.1.2 Thuật toán quay lui (backtracking)

2.1.3 Kỹ thuật nhánh cận

No labels