Analysis and Design of Algorithms

Lecture 11,12

Backtracking Method

Lecturer: Ha Dai Duong
duonghd@mta.edu.vn

Nội dung

1. Lược đồ chung
2. Bài toán 8 hậu
3. Bài toán ngựa đi tuần
4. Trò chơi Sudoku
5. **Liệt kê các hoán vị**
6. Liệt kê dãy nhị phân độ dài N
7. Duyệt đồ thị

Bài toán

- Có N đối tượng (được đánh số từ 1 đến N), hãy liệt kê tất cả các hoán vị có thể của N đối tượng đó.
- Bài toán trên có thể qui về bài toán: Liệt kê tất cả các hoán vị của N số nguyên đầu tiên.
- Ví dụ: Các hoán vị của 3 số 1,2,3:

  123,132,213,231,312,321 (thứ tự từ điển)

  321,312,231,213,132,123 (thứ tự TD ngược)

Bài toán liệt kê

- Bài toán liệt kê: có thể tiếp cận theo cách liệt kê (phương pháp sinh - Generating) các khả năng ứng với mỗi thành phần của vector phương án (tìm hiểu sau)

Thứ tự từ điển (từ bé đến lớn)

123,132,213,231,312,321

Thứ tự từ điển ngược (từ lớn đến bé)

321,312,231,213,132,123

Ý tưởng thuật toán

Ý tưởng (**Thử và Sai**)

1. Cần xếp các số từ 1-N vào N vị trí (khác nhau từng đôi một)
2. Giả sử đã xếp được đến vị trí thứ i-1.
3. Tìm 1 giá trị thích hợp (chưa được dùng) trong khoảng từ 1 đến N cho vị trí thứ i. Lặp lại bước 3 khi i<N.

Phương án nghiệm

- Bộ gồm N số từ 1 đến N khác nhau từng đôi một.

Ứng viên

- Các giá trị từ 1 đến N

Tính hợp lệ

- x[i] nhận giá trị J nếu J chưa được dùng cho các x[1] đến x[i-1]

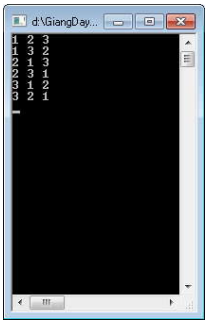
Cài đặt

- Dùng mảng b[j], j=1..N để dánh dấu giá trị j đã được dùng hay chưa.
  - b[j] = 0 : j đã được dùng
  - b[j] = 1 : j chưa được dùng


```
Try( i)≡
{
        for ( j = 1; j <= n; j++)
                if ( b[j])
                {
                        a[i] = j;
                        b[j] = 0;        // Ghi nhận trạng thái mới
                        if (i < n)
                                Try(i+1);
                        else
                                Xuất();
                        b[j] = True;     // Trả lại trạng thái cũ
                }
}
```

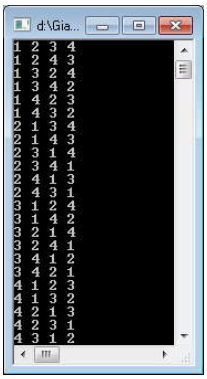

Minh họa

- Với n=3



Minh họa

- Với n=4



Nội dung

1. Lược đồ chung
2. Bài toán 8 hậu
3. Bài toán ngựa đi tuần
4. Trò chơi Sudoku
5. Liệt kê các hoán vị
6. **Liệt kê dãy nhị phân độ dài N**
7. Duyệt đồ thị

Bài toán

- Bài toán 1: Liệt kê tất cả các số nhị phân có độ dài N. Ví dụ với N=3, ta có các (8) liệt kê sau:

  000, 001, 010, 011, 100, 101, 110, 111.

- Bài toán 2: Liệt kê tập tất cả các tập con của tập có N phần tử.

Nhận xét: Bài toán 2 có thể chuyển về bài toán 1.

Ý tưởng thuật toán

Ta có thể sử dụng sơ đồ tìm tất cả các lời giải của bài toán. Hàm Try(i) xác định x_i, trong đó x_i chỉ có 1 trong 2 giá trị là 0 hay 1. Các giá trị này mặc nhiên được chấp nhận mà không cần phải thoả mãn điều kiện gì.

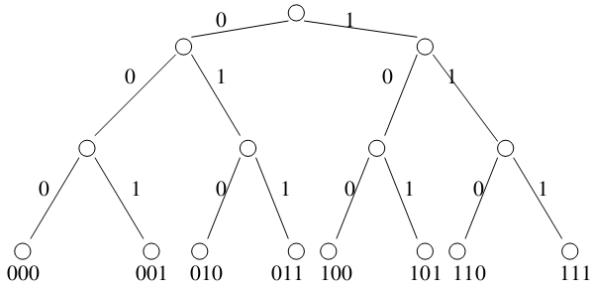
Phương án nghiệm

- Dãy N các giá trị 0, 1 (đảm bảo không có 2 nghiệm nào trùng nhau)

Cài đặt

```
Try ( i) ≡
        for (j = 0; j <= 1; j++)
        {
                x[i]  = j;
                if (i < n )
                        Try (i+1);
                else
                        Xuất(x);
        }
```

Minh họa



Nội dung

1. Lược đồ chung
2. Bài toán 8 hậu
3. Bài toán ngựa đi tuần
4. Trò chơi Sudoku
5. Liệt kê các hoán vị
6. Liệt kê dãy nhị phân độ dài N
7. **Duyệt đồ thị**

Bài toán

G = (V, U) là đơn đồ thị (có hướng hoặc vô hướng). V = {1,. ., n} là tập các đỉnh, U là tập cạnh (cung). Với s, t ∈ V, tìm tất cả các đường đi từ s đến t.

Các thuật toán tìm kiếm cơ bản :

Thuật toán DFS : Tìm kiếm theo chiều sâu.

Thuật toán BFS : Tìm kiếm theo chiều rộng.

Tìm kiếm theo chiều sâu DFS (Depth First Search)

- Ý tưởng

Thuật toán DFS tiến hành tìm kiếm trong đồ thị theo chiều sâu. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước. Đỉnh được thăm càng muộn sẽ càng sớm được duyệt xong (cơ chế LIFO – Vào Sau Ra Trước). Nên thuật toán có thể tổ chức bằng một thủ tục đệ quy quay lui.

Mô tả

```
Input  G = (V,U), s, t
Output  Tất cả các đường đi từ s đến t (nếu có).
DFS ( int  s)  ≡
        for ( u = 1; u <= n; u++)
        {
                if  (chấp nhận được)
                 {
                        Ghi nhận nó;
                        if  (u ≠ t)
                                DFS(u);
                        else
                                In đường đi;
                        bỏ việc ghi nhận;
                }
        }
```


Cài đặt

Ta cần mô tả dữ liệu đồ thị và các mệnh đề được phát biểu trong mô hình.

Ma trận sẽ được biểu diễn bằng ma trận kề :

$$a_{ij} = \begin{cases} 1; (i,j) \in U \\ 0; (i,j) \notin U \end{cases}$$

Ghi nhận đỉnh được thăm để tránh trùng lặp khi quay lui bằng cách đánh dấu. Ta sữ dụng một mảng một chiều Daxet[] với qui ước :

Daxet[u] = 1 , u đã được thăm.

Daxet[u] = 0 , u chưa được thăm.

Mảng Daxet[] lúc đầu khởi động bằng 0 tất cả.

Điều kiện chấp nhận được cho đỉnh u chính là u kề với v (a_{vu} = 1) và u chưa được thăm (Daxet[u] = 0).

Để ghi nhận các đỉnh trong đường đi, ta dùng một mảng một chiều Truoc[ ], với qui ước :

Truoc[u] = v ⇔ v là đỉnh đứng trước đỉnh u, và u kề với v

Ta khởi động mảng Truoc[ ] bằng 0 tất cả.

Cài đặt

Input $G = (a_{ij})_{nxn}$, s, t
Output Tất cả các đường đi từ s đến t (nếu có).

```
void DFS( s) ≡
        int u;
        daxet[s] = 1;
        for( u = 1;u <= n; u++)
        {
                if( a[s][u] && !daxet[u])
                {
                        Truoc[u] = s;
                        if ( u == t )
                                Xuat_duongdi();
                        else
                                DFS(u);
                        daxet[u] = 0;
                }
        }
```


Cài đặt

Mảng truoc[ ] lưu trử các đỉnh trên đường đi,

Nếu kết thúc thuật toán, Daxet[t] = 0 (Truoc[t] = 0) thì không có đường đi từ s đến t.

Trong trường hợp tồn tại đường đi, xuất đường đi chính là xuất mảng Truoc[ ]. Thao tác này có thể viết như sau :

```
Xuat_duongdi()≡
        cout<<t<<"<--";
        j = t;
        while ( truoc[j] != s)
        {
                cout<<truoc[j]<<"<--";
                j = truoc[j];
        }
        cout<<s<<endl;
```


Minh họa

Với đồ thị có hướng cho bởi ma trận kề :

```
7
0 0 0 1 0 1 1
0 0 1 1 0 0 0
0 1 0 1 0 1 0
1 0 1 0 0 0 0
0 0 0 0 1 1 0
1 0 0 0 1 0 0
1 0 0 1 0 0 0
```

Đường đi từ 1 đến 4

```
7
0 0 0 1 0 1 1
0 0 1 1 0 0 0
0 1 0 1 0 1 0
1 0 1 0 0 0 0
0 0 0 0 1 1 0
1 0 0 0 1 0 0
1 0 0 1 0 0 0
```

s = 1, t = 4

4 ← 1

4 ← 7 ← 1

Đường đi từ 2 đến 5

```
7
0 0 0 1 0 1 1
0 0 1 1 0 0 0
0 1 0 1 0 1 0
1 0 1 0 0 0 0
0 0 0 0 1 1 0
1 0 0 0 1 0 0
1 0 0 1 0 0 0
```

s = 2, t = 5

5 ← 6 ← 1 ← 4 ← 3 ← 2

5 ← 6 ← 3 ← 2

5 ← 6 ← 1 ← 4 ← 2

5 ← 6 ← 3 ← 4 ← 2

Tìm kiếm theo chiều rộng BFS (Breadth First Search)

- Ý tưởng

Thuật toán BFS tiến hành tìm kiếm trên đồ thị theo chiều rộng. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước theo từng mức kề. Đỉnh được thăm càng sớm thì sẽ càng sớm được duyệt xong (cơ chế FIFO – Vào Trước Ra Trước).

Input G = (V,E),
s, t ∈ V;
Output
Đường đi từ s đến t.
Mô tả :

- Bước 0 : $A_0 = \{s\}$.
- Bước 1 : $A_1 = \{x \in V \setminus A_0 : (s,x) \in E\}$.
- Bước 2 : $A_2 = \{x \in V \setminus [A_0 \cup A_1] : \exists y \in A_1, (y,x) \in E\}$.
- . . .
- Bước i : $A_i = \{x \in V \setminus \bigcup_{k=0}^{i-1} A_k : \exists y \in A_{i-1}, (y,x) \in E\}$.
- …

Thuật toán có không quá n bước lặp; một trong hai trường hợp sau xảy ra :

- Nếu với mọi i, t ∉ A_i : không có đường đi từ s đến t;
- Ngược lại, t ∈ A(m) với m nào đó. Khi đó tồn tại đường đi từ s tới t, và đó là một đường đi ngắn nhất từ s đến t.

Trong trường hợp này, ta xác định được các đỉnh trên đường đi bằng cách quay ngược lại từ t đến các đỉnh trước t trong từng các tập trước cho đến khi gặp s.

Cài đặt

Trong thuật toán BFS, đỉnh được thăm càng sớm sẽ càng sớm trở thành duyệt xong, nên các đỉnh được thăm sẽ được lưu trữ trong hàng đợi queue. Một đỉnh sẽ trở thành duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề của nó .

Ta dùng một mảng logic Daxet[ ] để đánh dấu các đỉnh được thăm, mảng này được khởi động bằng 0 tất cả để chỉ rằng lúc đầu chưa đỉnh nào được thăm.

Một mảng truoc[ ] để lưu trữ các đỉnh nằm trên đường đi ngắn nhất cần tìm (nếu có), với ý nghĩa Truoc[i] là đỉnh đứng trước đỉnh i trong đường đi. Mảng Truoc[ ] được khởi động bằng 0 tất cả để chỉ rằng lúc đầu chưa có đỉnh nào.

Đồ thị G được biểu diễn bằng ma trận kề a= $(a_{uv})_{nxn}$

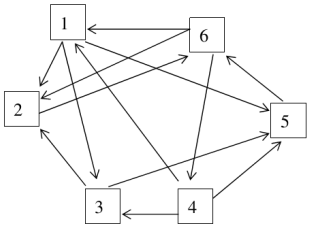
trong đó : $a_{uv} = \begin{cases} 1; (u,v) \in E; \\ 0; (u,v) \notin E; \end{cases}$

Hàng đợi queue ta cài đặt bằng mảng . Thuật toán được cài đặt như sau :


```
BFS(s) ≡
        int u, j, dauQ = 1, cuoiQ = 1;
        queue[cuoiQ] = s;
        Daxet[s] = 1;
        while ( dauQ <= cuoiQ)
        {
                u = queue[dauQ];
                dauQ++;
                for ( j = 1; j <= n; j++)
                        if( a[u][j] == 1 && !Daxet[j] )
                        {
                                cuoiQ++;
                                queue[cuoiQ] = j;
                                Daxet[j] = 1;
                                Truoc[j] = u;
                        }
        }
```

Minh họa

Cho đơn đồ thị có hướng :

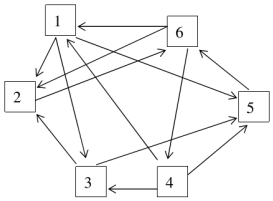


Bài tập

1. Liệt kê các hoán vị của tập 4 phần tử (theo thuật toán mục 5)
2. Liệt kê tất cả các dãy nhị phân có độ dài 5 (theo thuật toán mục 6)

Bài tập

Cho đồ thị có hướng



3. Thực hiện từng bước thuật toán DFS trên đồ thị để tìm đường đi từ đỉnh 1 đến đỉnh 4.
4. Thực hiện từng bước thuật toán BFS trên đồ thị để tìm đường đi từ đỉnh 2 đến đỉnh 5.

Bài tập

5. Cài đặt thuật toán giải bài toán người du lịch (dựa trên thuật toán liệt kê các hoán vị) theo phương pháp quay lui. Đánh giá độ phức tạp thuật toán bằng lý thuyết, bằng thực nghiệm và so sánh.
6. Cài đặt thuật toán tìm đường đi trên đồ thị, thuật toán DFS, theo phương pháp quay lui . Đánh giá độ phức tạp thuật toán bằng lý thuyết, bằng thực nghiệm và so sánh.

Bài tập

7. Cài đặt thuật toán tìm đường đi trên đồ thị, thuật toán BFS, theo phương pháp quay lui . Đánh giá độ phức tạp thuật toán bằng lý thuyết, bằng thực nghiệm và so sánh.