



25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

THUẬT TOÁN ỨNG DỤNG

Tư Duy Thuật Toán Và CTDL + Kỹ Năng Lập Trình

1 Giới thiệu chung

2 Các kỹ năng cơ bản cần rèn luyện

3 Dạng bài toán Ad Hoc

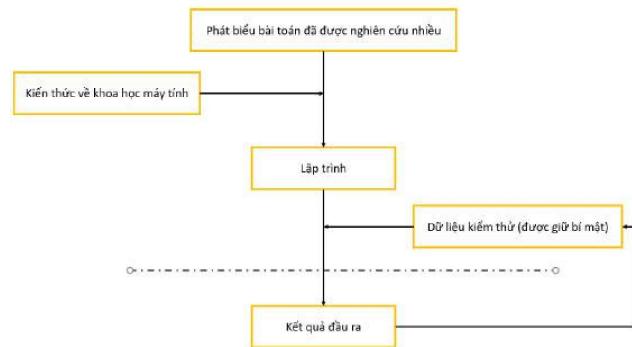
1 Giới thiệu chung

- Mô hình tổng quát
- Mục tiêu
- Phương pháp tiếp cận
- Tài liệu tham khảo
- Các chủ đề
- Mẫu đề bài
- Bài toán ví dụ – ALICEADD
- Hệ thống chấm điểm
- Các phản hồi

2 Các kỹ năng cơ bản cần rèn luyện

3 Dạng bài toán Ad Hoc

Mô hình Bài tập lập trình Thuật toán ứng dụng



Yêu tố chính : giải bài toán đúng nhanh nhất có thể!

Mục tiêu

- Đề bài yêu cầu lập trình giải quyết một bài toán có nội dung ứng dụng thực tế, các vấn đề bao gồm:
 - mô hình hoá bài toán,
 - tìm lời giải hiệu quả sử dụng các thuật toán và cấu trúc dữ liệu,
 - chuyển lời giải thành chương trình và chạy thử nghiệm,
 - làm càng nhanh càng tốt dưới áp lực thời gian và độ chính xác,
 - và phải làm đúng: chương trình không sinh lỗi, kết quả đúng trong thời gian và bộ nhớ hạn chế.
- Mục tiêu của khóa học này là thực hành giải quyết những vấn đề trên.

Phương pháp tiếp cận

- Học những dạng bài phổ biến khác nhau
- Chỉ ra những ứng dụng của các thuật toán và cấu trúc dữ liệu bạn biết từ
 - ▶ khóa học cơ bản về các thuật toán
 - ▶ khóa học cơ bản về cấu trúc dữ liệu
- Học các dạng thuật toán và cấu trúc dữ liệu phổ biến khác
- Học một số lý thuyết toán/tin học hay dùng
- Thực hành giải bài toán
- Thực hành lập trình
- Thực hành nữa
- .. và thực hành mãi



Các chủ đề dự kiến

Thứ tự	Chủ đề
Buổi 1	Giới thiệu
Buổi 2	Cấu trúc dữ liệu và thư viện
Buổi 3	Thực hành
Buổi 4	Kỹ thuật đệ quy và nhánh cận
Buổi 5	Chia để trị
Buổi 6	Qui hoạch động
Buổi 7	Thực hành
Buổi 8	Qui hoạch động
Buổi 9	Kiểm tra giữa kỳ
Buổi 10	Đồ thị
Buổi 11	Thực hành
Buổi 12	Đồ thị
Buổi 13	Xử lý xâu và thực hành
Buổi 14	Thuật toán tham lam
Buổi 15	Thực hành
	Lớp bài toán NP-đầy đủ



Bài toán ví dụ – ALICEADD

Mô tả bài toán

Alice có a cái kẹo, Bob cho Alice thêm b cái kẹo. Hỏi Alice có tất cả bao nhiêu cái kẹo?

Mô tả dữ liệu vào

- Dòng đầu chứa một số nguyên không âm T là số bộ dữ liệu ($T \leq 10$).
- Mỗi dòng trong số T dòng tiếp theo chứa hai số nguyên không âm a và b cách nhau bởi dấu cách ($a, b \leq 10^{18}$).

Mô tả kết quả ra

Gồm T dòng là kết quả cho T bộ dữ liệu theo thứ tự đầu vào.



Tài liệu tham khảo

- Competitive Programming. Steven Halim <http://libgen.io/ads.php?md5=f6f195012783a8b3c8bb7628882a51b7>
- Slides bài giảng Phân tích và thiết kế thuật toán. Nguyễn Đức Nghĩa
- Algorithm design. Jon Kleinberg and Éva Tardos.
- Introduction to Algorithms. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein.
- Bài giảng Chuyên đề Lê Minh Hoàng
- Competitive Programming Course at Reykjavík University



Mẫu đề bài

- Mẫu chuẩn bài toán trong hầu hết các kỳ thi bao gồm:
 - ▶ Mô tả bài toán
 - ▶ Mô tả định dạng dữ liệu vào
 - ▶ Mô tả định dạng kết quả ra
 - ▶ Ví dụ Dữ liệu vào/Kết quả ra
 - ▶ Giới hạn thời gian theo giây
 - ▶ Giới hạn bộ nhớ theo bytes/megabytes
 - ▶ Giới hạn kích thước các tham số đầu vào
- Yêu cầu viết chương trình giải bài toán đúng càng nhiều bộ dữ liệu càng tốt
 - ▶ Mặc định là dữ liệu vào đúng, không cần kiểm tra tính đúng đắn
 - ▶ Chương trình không được chạy quá giới hạn thời gian và giới hạn bộ nhớ



Bài toán ví dụ – ALICEADD

Ví dụ dữ liệu vào	Dữ liệu kết quả ra
2	8
3 5	5
4 1	



Lời giải ví dụ

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         int a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

Lời giải ví dụ

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         int a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng với mọi bộ dữ liệu test không?

Lời giải ví dụ

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         int a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng với mọi bộ dữ liệu test không?
- Điều gì xảy ra nếu $a = b = 10^{18}$?

Lời giải ví dụ

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         int a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng với mọi bộ dữ liệu test không?
- Điều gì xảy ra nếu $a = b = 10^{18}$? Kết quả phép cộng sẽ bị tràn số lớn.

Lời giải ví dụ

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         int a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng với mọi bộ dữ liệu test không? KHÔNG!
- Điều gì xảy ra nếu $a = b = 10^{18}$? Kết quả phép cộng sẽ bị tràn số lớn.

Lời giải ví dụ

- Khi $a = b = 10^{18}$, kết quả phải là 2×10^{18}

Lời giải ví dụ

- Khi $a = b = 10^{18}$, kết quả phải là 2×10^{18}
- Giá trị này quá lớn với biến nguyên 32-bit, nên bị tràn số



Lời giải ví dụ

- Khi $a = b = 10^{18}$, kết quả phải là 2×10^{18}
- Giá trị này quá lớn với biến nguyên 32-bit, nên bị tràn số
- Sử dụng biến nguyên 64-bit sẽ cho lời giải đúng



Lời giải đúng

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         long long a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```



Lời giải đúng

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         long long a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng không?



Lời giải đúng

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         long long a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng không? **DÚNG!**



Lời giải đúng

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         long long a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11    return 0;
12 }
```

- Lời giải này có đúng không? **DÚNG!**

- Làm thế nào nếu giá trị a, b lớn nữa?



Lời giải đúng

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int T;
5     cin >> T;
6     for (int i = 0; i < T; i++) {
7         long long a, b;
8         cin >> a >> b;
9         cout << a + b << endl;
10    }
11 }
12 }
```

- Lời giải này có đúng không? **DÚNG!**
- Làm thế nào nếu giá trị a, b lớn nữa? **XỬ LÝ SỐ LỚN!**



Hệ thống chấm điểm trực tuyến tự động

- Một số server phổ biến: codeforces.com, spoj.com, ru.kattis.com, ...
- Khi nộp bài giải lên hệ thống chấm sẽ phản hồi kết quả ngay
- Hầu hết có thể nộp bài giải với các ngôn ngữ lập trình phổ biến:
 - ▶ C/C++
 - ▶ Java
 - ▶ Python
 - ▶ Pascal
 - ▶ ...

DHBKHN sử dụng *server codeforces riêng* để giúp sinh viên thực hành giải bài trực tuyến và sử dụng hệ thống cms kết hợp với hệ thống check trùng code tự động để đánh giá kiểm tra môn học



1 Giới thiệu chung

2 Các kỹ năng cơ bản cần rèn luyện

- A. Kỹ năng đọc đề
- B. Kỹ năng phân loại bài toán
- C. Kỹ năng phân tích thuật toán
- D. Kỹ năng làm chủ ngôn ngữ lập trình
- E. Kỹ năng đặt tên biến
- F. Kỹ năng test chương trình
- G. Kỹ năng gõ nhanh
- I. Kỹ năng thực hành

3 Dạng bài toán Ad Hoc



Hệ thống chấm điểm trực tuyến tự động

- Một số server phổ biến: codeforces.com, spoj.com, ru.kattis.com, ...
- Khi nộp bài giải lên hệ thống chấm sẽ phản hồi kết quả ngay
- Hầu hết có thể nộp bài giải với các ngôn ngữ lập trình phổ biến:
 - ▶ C/C++
 - ▶ Java
 - ▶ Python
 - ▶ Pascal
 - ▶ ...



Một số phản hồi chính

- Các phản hồi từ hệ thống chấm điểm thường không thật chi tiết. Một số phản hồi thường gặp:
 - ▶ Kết quả đúng (Accepted)
 - ▶ Kết quả sai (Wrong Answer)
 - ▶ Chương trình dịch lỗi (Compile Error)
 - ▶ Chương trình chạy sinh lỗi (Run time Error)
 - ▶ Quá thời gian cho phép (Time Limit Exceeded)
 - ▶ Quá bộ nhớ cho phép (Memory Limit Exceeded)
- Một số server cho phản hồi chi tiết đến từng test.



A. Kỹ năng đọc đề

- Kiến thức cơ sở của bài toán (Background): Đây là phần thể hiện ứng dụng của bài toán
- Các dữ kiện và yêu cầu của bài toán
- Mô tả khuôn dạng dữ liệu vào và ra
- Ví dụ khuôn dạng vào ra: thường chỉ là những test đơn giản
- Các hạn chế (kích thước dữ liệu kiểm thử, thời gian, bộ nhớ) cho các test của bài toán



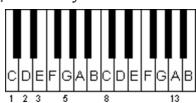
Phần Background có quan trọng không?

• KHÔNG?

- ▶ Ví dụ: "Hãy lập trình sắp xếp n số thực. Hạn chế bộ nhớ: 3MB. Hạn chế thời gian: 1 giây. Hạn chế kích thước đầu vào $n \leq 10^6$. Hạn chế giá trị số thực trong 4 byte với định dạng đầu vào có chính xác hai chữ số sau dấu phẩy động".
- ▶ Quá ngắn gọn và dễ hiểu!

• Tin học: BUỘC PHẢI CÓ!!!

- ▶ Ví dụ: Bản vanxơ Fibonacci là một bản nhạc mà giai điệu của nó bắt nguồn từ một trong những dãy số nổi tiếng nhất trong Lý thuyết số - dãy số Fibonacci. Hai số đầu tiên của dãy là số 1 và số 2, các số tiếp theo được xác định bằng tổng của hai số liên tiếp ngay trước nó trong dãy. Bản vanxơ Fibonacci thu được bằng việc chuyển dãy số Fibonacci thành dãy các nốt nhạc theo qui tắc chuyển một số nguyên dương thành nốt nhạc sau đây:



[Ấn vào đây để nghe bản nhạc]

- ▶ Tạo hứng cho người đọc giải bài
- ▶ Một chủ đề ứng dụng của KHM

B. Kỹ năng phân loại bài toán

- Thực hành phân loại nhanh các dạng bài toán
- Có thể là kết hợp của nhiều dạng khác nhau
- Một số dạng bài hay gặp:

Loại	Loại hép
Ad Hoc	Trực tiếp, Mô phỏng
Duyệt toàn bộ	Lặp, Quay lui, Nhánh cận
Chia để trị	Cỗ điển, Cải tiến
Giảm để trị	Cỗ điển, Cải tiến
Tham lam	Cỗ điển, Cải tiến
Quy hoạch động	Cỗ điển, Cải tiến
Đồ thị	Cỗ điển, Cải tiến
Đồ thị đặc biệt	Cỗ điển, Cải tiến
Toán học	Cỗ điển, Cải tiến
Xử lý xâu	Cỗ điển, Cải tiến
Tính toán hình học	Cỗ điển, Cải tiến
Một số loại thuật toán đặc thù	Cỗ điển, Cải tiến

C. Kỹ năng phân tích thuật toán

- Lời giải bài toán phải đủ nhanh và không sử dụng quá nhiều bộ nhớ
- Lời giải nên càng đơn giản càng tốt
- Sử dụng phương pháp Phân Tích Thuật Toán để xác định xem lời giải đưa ra có thỏa mãn giới hạn thời gian và bộ nhớ không
- Thông thường tính: 5×10^8 phép tính trong một giây
- Ví dụ cần sắp xếp $n \leq 10^6$ số nguyên chạy trong 1 giây
 - ▶ Liệu có thể sử dụng sắp xếp nối bọt, dễ cài đặt, và có độ phức tạp $O(n^2)$ được không?
 - ▶ Sử dụng một thuật toán sắp xếp nhanh, khó cài đặt hơn, và có độ phức tạp $O(n \log n)$?
- Nếu sắp xếp $n \leq 10^3$ số nguyên chạy trong 1 giây
 - ▶ Bây giờ có thể sử dụng sắp xếp nối bọt được không?
- Luôn nhớ hãy sử dụng giải pháp đơn giản nhất thỏa mãn giới hạn thời gian và bộ nhớ

- Hãy thực hành cách ước lượng xấp xỉ trong đầu

• Mẹo:

- ▶ $2^{10} \approx 10^3$
- ▶ $\log_2 10 \approx 3$

- Trường hợp tìm ra lời giải mà bạn không chắc là nó đúng thì:

- ▶ Hãy tìm cách chứng minh nó!
- ▶ Thủ tính đúng sai với các bộ kiểm thử có kích thước nhỏ
- ▶ Ngay cả khi không tìm được cách chứng minh hoặc phản bác nó thì điều thu được là hiểu sâu thêm bài toán

D. Kỹ năng làm chủ ngôn ngữ lập trình

- Hãy thành thạo ngôn ngữ lập trình như lòng bàn tay
- Bao gồm cả các thư viện có sẵn
 - ▶ Thư viện STL của C++ (Standard Template Library)
 - ▶ Thư viện của Java (Java Class Library)
- Nếu đã có sẵn trong thư viện chuẩn thì không cần phải lập trình lại: cài đặt nhanh, không có bug
- Hạn chế: khả năng tùy biến của thư viện không linh hoạt. Rất nhiều phần kỹ thuật xử lý thuật toán không thể gọi trực tiếp thư viện mà phải tùy biến đi hoặc phải cài đặt lại

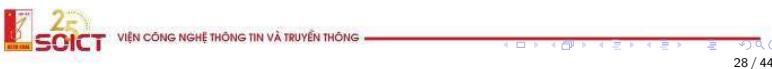
E. Kỹ năng đặt tên biến

- Tên hàm viết hoa chữ cái đầu. Ví dụ: `void ReadInp()`
- Tên hằng số viết in hoa. Ví dụ: `const int MAX=100000;`
- Tên mảng bắt đầu bằng chữ cái đầu của kiểu giá trị: `int → i`, tiếp theo là viết hoa chữ cái đầu. Ví dụ: `int iCost[MAX], bool bMark[1001], string sLine[100];`
- Tên biến đơn viết thường. Ví dụ: `int i, u, sum, res, ans;`
- Tên biến nên càng giống với tên được cho trong bài toán càng tốt
- ...



G. Kỹ năng gõ nhanh

- Hãy trở thành thợ gõ nhanh và chính xác hơn
- Đừng để việc gõ lời giải là hạn chế cho việc giải bài
- Khi đánh máy không nhìn vào bàn phím, mắt nhìn vào màn hình kiểm tra luôn tính đúng đắn của việc gõ, trong lúc đó đầu vẫn có thể suy nghĩ song song các vấn đề tiếp theo
- Ví dụ: sử dụng TypeRacer là một cách luyện tập hiệu quả và thú vị: <http://play.typeracer.com/>



Các bài toán Ad Hoc

F. Kỹ năng test/debug chương trình

- Phải test để chắc chắn kết quả bài giải là đúng và thỏa mãn giới hạn thời gian và bộ nhớ, hoặc ít ra là biết lời giải chưa đúng
- Cố gắng phản biện bài giải bằng cách tìm ra phản ví dụ (một dữ liệu vào mà bài giải trả kết quả ra sai, hoặc mất quá nhiều thời gian để tìm ra kết quả)
- Test các bộ dữ liệu biến và dữ liệu lớn, ...
- Viết một thuật toán trực tiếp đơn giản để kiểm tra kết quả chương trình của mình có đúng không với những trường hợp kích thước đầu vào nhỏ



I. Thực hành, thực hành nữa và thực hành mãi

- Càng thực hành nhiều thì càng hoàn thiện các kỹ năng giải bài và lập trình
- Rất nhiều các trang chấm bài trực tuyến giúp bạn giải các bài toán của những kỳ thi trong quá khứ
- Một số trang giải bài trực tuyến thường xuyên tổ chức các cuộc thi đấu lập trình: Codeforces, TopCoder, Codechef, Kattis, ...

Lập trình viên thi đấu cũng như những vận động viên thể thao, cần phải rèn luyện thường xuyên để giữ được cảm hứng và phát triển các kỹ năng lập trình giải bài!



1 Giới thiệu chung

2 Các kỹ năng cơ bản cần rèn luyện

3 Dạng bài toán Ad Hoc

- Bài toán Ad Hoc là gì
- Bài toán: Cắt giảm cửa hàng
- Bài toán: Gõ SMS



Loại bài toán Ad Hoc

- Là dạng bài toán đơn giản nhất
- Thường làm đúng như mô tả bài toán yêu cầu
- Trực tiếp hoặc Mô phỏng
- Giới hạn thời gian thường không quan trọng
- Đôi khi mô tả dài dòng khó hiểu
- Đôi khi một số test biên lừa
- Một số bài toán phức hợp có thể khó lập trình

Bài toán: Cắt giảm cửa hàng

Đại dịch COVID19 trên toàn thế giới khiến người dân rất hạn chế ra khỏi nhà, điều này đã đẩy rất nhiều công ty phải phá sản và rất nhiều công ty khác phải cắt giảm bớt các hệ thống cửa hàng, văn phòng, sa thải nhân viên, giảm lương thưởng,...

Công ty XTEC cũng không phải ngoại lệ. Họ có 4 cửa hàng và quyết định đóng cửa tối đa 2 trong số đó có lợi nhuận âm thấp nhất năm 2019.

Yêu cầu: Cho trước lợi nhuận năm 2019 của 4 cửa hàng, hãy đưa ra tổng lợi nhuận âm của những cửa hàng phải đóng cửa.

Bài toán: Cắt giảm cửa hàng

Dữ liệu vào

- Dòng đầu tiên chứa một số nguyên T ($T < 20$) là số lượng trường hợp test.
- Mỗi dòng trong số T dòng sau chứa 4 số nguyên phân biệt biểu diễn lợi nhuận năm 2019 của 4 cửa hàng. Tất cả các số nguyên ở đây nằm trong khoảng $[-10000, 10000]$.

Kết quả ra

Mỗi test ghi trên một dòng một số duy nhất là tổng lợi nhuận âm của các cửa hàng phải cắt bỏ.

Bài toán: Cắt giảm cửa hàng

Ví dụ dữ liệu vào	Ví dụ kết quả ra
3 -1000 2000 3000 -4000 3000 -2500 1500 100 -1500 -1200 -1800 -1900	-5000 -2500 -2700

Lời giải bài toán: Cắt giảm cửa hàng

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int main() {
4     ios_base::sync_with_stdio(0);
5     cin.tie(0); cout.tie(0);
6
7     int iProfit[4];
8     int T; cin >> T;
9     for (int i = 0; i < T; i++) {
10         cin >> iProfit[1] >> iProfit[2];
11         cin >> iProfit[3] >> iProfit[4];
12
13         sort(iProfit, iProfit + 4);
14         int sum=0;
15         if (iProfit[1]<0) sum+=iProfit[1];
16         if (iProfit[2]<0) sum+=iProfit[2];
17         cout << sum << endl;
18     }
19     return 0;
20 }
```

Bài toán: Gõ SMS

Điện thoại di động (DTDD) trở nên một phần không thể thiếu trong cuộc sống hiện đại. Ngoài việc gọi, DTDD có thể gửi tin nhắn mà người ta quen gọi là SMS. Không như bàn phím máy tính, đa phần DTDD hạn chế số phím. Để có thể gõ được tất cả các ký tự trong bảng chữ cái, nhiều ký tự sẽ được hiển thị trên cùng một phím. Vì vậy, để gõ một số ký tự, một phím sẽ phải được ấn liên tục đến khi ký tự cần tìm hiển thị trên màn hình.

Cho một đoạn văn bản, hãy tính số lần gõ phím để hiển thị được đoạn văn bản.

Bài toán: Gõ SMS

Bài toán giả thiết rằng các phím được sắp xếp như sau.

	abc	def
ghi	jkl	mno
pqrs	tuv	wxyz
	<SP>	

Trong bảng trên mỗi ô biểu diễn một phím. <SP> biểu diễn phím space. Để hiển thị ký tự 'a' thì sẽ phải bấm phím tương ứng 1 lần, nhưng để hiển thị ký tự 'b' của cùng phím đó thì sẽ phải bấm liên tục 2 lần và đối với phím 'c' là 3 lần.

Tương tự, bấm 1 lần cho 'd', hai lần cho 'e' và 3 lần cho 'f'. Các ký tự khác cũng được làm tương tự. Lưu ý là để ra 1 khoảng trống thì cần bấm 1 lần phím space.



Bài toán: Gõ SMS

Ví dụ dữ liệu vào	Ví dụ kết quả ra
2 welcome to ulab good luck and have fun	Case #1: 29 Case #2: 41



Lời giải bài toán: Gõ SMS

```
1 // Mỗi trường hợp Test được thực hiện ở đây
2 string sLine;
3 getline(cin, sLine);
4 int res = 0;
5 for (int i = 0; i < sLine.size(); i++) {
6     int cur;
7     for (int j = 0; j < 12; j++) {
8         for (int k = 0; k < sKey[j].size(); k++) {
9             if (sLine[i] == sKey[j][k]) {
10                 cur = k + 1;
11             }
12         }
13     }
14     res += cur;
15 }
cout<<"Case #<<"<<t + 1<<": "<<res);
```



Bài toán: Gõ SMS

Dữ liệu vào

Dòng đầu tiên là một số nguyên T là số lượng trường hợp test. T dòng tiếp theo mỗi dòng chỉ chứa các khoảng trắng và các ký tự in thường. Mỗi dòng chứa ít nhất 1 và tối đa 100 ký tự.

Kết quả ra

Mỗi trường hợp test đầu vào tương ứng với một dòng ở kết quả ra. Mỗi dòng bắt đầu bởi thứ tự trường hợp test và sau đó là một số biểu thị số lần bấm phím cho ra văn bản tương ứng. Xem ví dụ kết quả ra để thấy định dạng chuẩn xác.



Lời giải bài toán: Gõ SMS

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 string sKey[12] = {
4     "",      "abc",   "def",
5     "ghi",   "jkl",   "mno",
6     "pqrs",  "tuv",   "wxyz",
7     " ",      " ",     ""
8 };
9 int main() {
10     ios_base::sync_with_stdio(0);
11     cin.tie(0); cout.tie(0);
12     int T; cin >>T;
13     for (int t = 0; t < T; t++) {
14         // Mỗi trường hợp Test được thực hiện ở đây
15     }
16     return 0;
17 }
```



BÀI TẬP THỰC HÀNH

- ALICEADD
- SUBSEQMAX



Thank you for
your attentions!



25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Cấu trúc dữ liệu và Thư viện THUẬT TOÁN ỨNG DỤNG

- ① Các kiểu dữ liệu cơ bản
- ② Số nguyên lớn
- ③ Thư viện CTDL và Thuật toán
- ④ Biểu diễn tập hợp bằng Bitmask
- ⑤ Một số ứng dụng của CTDL
- ⑥ Cấu trúc dữ liệu mở
- ⑦ Biểu diễn đồ thị

Các kiểu dữ liệu cơ bản

- Các kiểu dữ liệu phải biết:
 - ▶ bool: biến bun (boolean) (true/false)
 - ▶ char: biến nguyên 8-bit (thường được sử dụng để biểu diễn các ký tự ASCII)
 - ▶ short: biến nguyên 16-bit
 - ▶ int/long: biến nguyên 32-bit
 - ▶ long long: biến nguyên 64-bit
 - ▶ float: biến thực 32-bit
 - ▶ double: biến thực 64-bit
 - ▶ long double: biến thực 128-bit
 - ▶ string: biến xâu ký tự

Các kiểu dữ liệu cơ bản

Loại	Số Byte	Giá trị nhỏ nhất	Giá trị lớn nhất
bool	1		
char	1	-128	127
short	2	-32768	32767
int/long	4	-2148364748	2147483647
long long	8	-9223372036854775808	9223372036854775807
	n	-2^{8n-1}	$2^{8n-1} - 1$

Loại	Số Byte	Giá trị nhỏ nhất	Giá trị lớn nhất
unsigned char	1	0	255
unsigned short	2	0	65535
unsigned int	4	0	4294967295
unsigned long long	8	0	18446744073709551615
	n	0	$2^{8n} - 1$

Loại	Số Byte	Giá trị nhỏ nhất	Giá trị lớn nhất
float	4	$\approx -3.4 \times 10^{-38}$	$\approx 3.4 \times 10^{-38}$
double	8	$\approx -1.7 \times 10^{-308}$	$\approx 1.7 \times 10^{-308}$

Số nguyên lớn

- Làm thế nào để tính toán với số nguyên cực lớn, nghĩa là không thể lưu trữ bằng kiểu long long
- Ý tưởng đơn giản: Lưu số nguyên dưới dạng string
- Tuy nhiên làm thế nào để tính toán số học giữa hai số nguyên?
- Có thể dùng thuật toán giống như phương pháp tính bậc tiểu học: tính từng chữ số, từng phần, có lưu phần nhớ

1 Các kiểu dữ liệu cơ bản

2 Số nguyên lớn

3 Thư viện CSDL và Thuật toán

4 Biểu diễn tập hợp bằng Bitmask

5 Một số ứng dụng của CSDL

6 Cấu trúc dữ liệu mở

7 Biểu diễn đồ thị



Tầm quan trọng của cấu trúc dữ liệu

- Dữ liệu cần được biểu diễn theo cách thuận lợi để thực hiện hiệu quả các toán tử thông dụng:
 - ▶ Truy vấn
 - ▶ Chèn
 - ▶ Xóa
 - ▶ Cập nhật
- Dữ liệu còn cần được biểu diễn theo cách phức tạp hơn:
 - ▶ Làm thế nào để biểu diễn số nguyên lớn?
 - ▶ Làm thế nào để biểu diễn đồ thị?
- Các cấu trúc dữ liệu cơ bản và nâng cao giúp chúng ta thực hiện được những điều này

1 Các kiểu dữ liệu cơ bản

2 Số nguyên lớn

3 Thư viện CSDL và Thuật toán

4 Biểu diễn tập hợp bằng Bitmask

5 Một số ứng dụng của CSDL

6 Cấu trúc dữ liệu mở

7 Biểu diễn đồ thị



Các cấu trúc dữ liệu thông dụng

- Mảng tĩnh
- Mảng động
- Danh sách liên kết
- Ngăn xếp
- Hàng đợi
- Hàng đợi ưu tiên
- Hàng đợi hai đầu
- Tập hợp
- Ánh xạ

Các cấu trúc dữ liệu thông dụng

- Mảng tĩnh - int Arr[10]
- Mảng động - vector<int>
- Danh sách liên kết - list<int>
- Ngăn xếp - stack<int>
- Hàng đợi - queue<int>
- Hàng đợi ưu tiên - priority_queue<int>
- Hàng đợi hai đầu - deque<int>
- Tập hợp - set<int>
- Ánh xạ - map<int, int>, sử dụng cây cân bằng đỏ đen



Các cấu trúc dữ liệu thông dụng

- Mảng tĩnh - `int Arr[10]`
- Mảng động - `vector<int>`
- Danh sách liên kết - `list<int>`
- Ngăn xếp - `stack<int>`
- Hàng đợi - `queue<int>`
- Hàng đợi ưu tiên - `priority_queue<int>`
- Hàng đợi hai đầu - `deque<int>`
- Tập hợp - `set<int>`
- Ánh xạ - `map<int, int>`, sử dụng cây cân bằng đỏ đen
- Thông thường nên sử dụng thư viện chuẩn
 - Gần như chắc chắn chạy nhanh và không lỗi
 - Giảm bớt việc viết code
- Nhiều khi vẫn cần tự viết code thay vì dùng thư viện chuẩn
 - Khi muốn kiểm soát linh hoạt
 - Khi muốn tùy biến/hiệu chỉnh cấu trúc dữ liệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

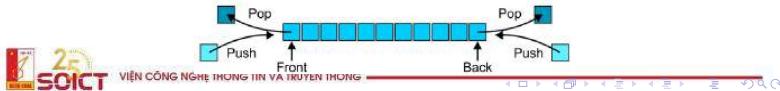
Deque - Hàng đợi hai đầu

- Deque=Double-Ended Queue: là CTDL có tính chất của cả Stack và Queue, nghĩa là cho phép thêm và xóa ở cả hai đầu

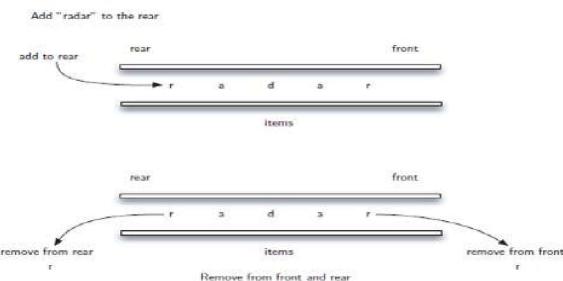
```
#include <deque>
deque<string> myDeque;
```

- hỗ trợ tất cả các phương thức của kiểu `vector` và `list` bao gồm cả chỉ số và con trỏ (iterator)

- `size()` trả về kích thước của deque
- `front()` trả về phần tử đầu tiên của deque
- `back()` trả về phần tử cuối cùng của deque
- `push_front()` thêm phần tử mới vào đầu của deque
- `push_end()` thêm phần tử mới vào cuối của deque
- `pop_front()` xóa phần tử đầu của deque
- `pop_end()` xóa phần tử cuối của deque



Deque - Kiểm tra chuỗi Palindrome



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Tùy biến kiểu `priority_queue<int>`

Trong nhiều trường hợp không thể dùng trực tiếp kiểu `priority_queue` mà cần tùy biến lại để cài đặt thuật toán. Ví dụ:

```
class Plane{ //Tuy_Bien_Priority_Queue_Min
public: int fuel
public: Plane(int q){(*this).fuel=fuel;}
friend ostream& operator<<(ostream& os, const Plane& p){
os<<p.fuel<<endl; return os;
}
bool operator>(const Plane& p) const{
    return fuel>p.fuel;
}
};

typedef priority_queue<Plane, vector<Plane>, greater<Plane> > PQPlane;
int main(){
    vector<Plane> vP;
    vP.push_back(Plane(4)); vP.push_back(Plane(7));
    vP.push_back(Plane(3)); vP.push_back(Plane(9));
    PQPlane PQ(vP.begin(),vP.end());
    while(!PQ.empty()) { cout<<PQ.top(); PQ.pop(); }
    return 0;
}
```



Sắp xếp và Tìm kiếm

- Các toán tử thông dụng nhất:
 - Sắp xếp một mảng - `sort(arr.begin(), arr.end())`
 - Tìm kiếm trên một mảng chưa sắp xếp - `find(arr.begin(), arr.end(), x)`
 - Tìm kiếm trên một mảng đã sắp xếp - `lower_bound(arr.begin(), arr.end(), x)`
- Thông thường nên sử dụng thư viện chuẩn
- Có lúc cần phiên bản khác của tìm kiếm nhị phân nhưng bình thường `lower_bound` là đủ
- hơn 90% sinh viên tự viết chương trình tìm kiếm nhị phân lần đầu cho kết quả sai



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

- 1 Các kiểu dữ liệu cơ bản
- 2 Số nguyên lớn
- 3 Thư viện CTDL và Thuật toán
- 4 Biểu diễn tập hợp bằng Bitmask
- 5 Một số ứng dụng của CTDL
- 6 Cấu trúc dữ liệu mở
- 7 Biểu diễn đồ thị



Biểu diễn tập hợp

- Cho một số lượng nhỏ ($n \leq 30$) phần tử
- Gán nhãn bởi các số nguyên $0, 1, \dots, n - 1$
- Biểu diễn tập hợp các phần tử này bởi một biến nguyên 32-bit
- Phần tử thứ i trong tập được biểu diễn bởi số nguyên x nếu bit thứ i của x là 1
- Ví dụ:
 - Cho tập hợp $\{0, 3, 4\}$
 - `int x = (1<<0) | (1<<3) | (1<<4);`



Biểu diễn tập hợp

- Kiểm tra một phần tử xuất hiện trong tập hợp:

```
1 if (x & (1<<i)) {  
2     // yes  
3 } else {  
4     // no  
5 }
```



- Các kiểu dữ liệu cơ bản
- Số nguyên lớn
- Thư viện CTDL và Thuật toán
- Biểu diễn tập hợp bằng Bitmask
- Một số ứng dụng của CTDL
- Cấu trúc dữ liệu mở
- Biểu diễn đồ thị



Biểu diễn tập hợp

- Tập rỗng: 0
- Tập có một phần tử: $1 << i$
- Tập vũ trụ (nghĩa là tất cả các phần tử): $(1 << i) - 1$
- Hợp hai tập: $x | y$
- Giao hai tập: $x & y$
- Phần bù một tập: $x & ((1 << i) - 1)$



Biểu diễn tập hợp

- Tại sao nên làm như vậy mà không dùng `set<int>`?
- Biểu diễn đỡ tốn khá nhiều bộ nhớ (nén 32,64,128 lần)
- Tất cả các tập con của tập n phần tử này có thể biểu diễn bởi các số nguyên trong khoảng $0 \dots 2^n - 1$
- Dễ dàng lặp qua tất cả các tập con
- Dễ dàng sử dụng một tập hợp như một chỉ số của một mảng



Ứng dụng của Mảng và Danh sách liên kết

- Ứng dụng trong trường hợp có quá nhiều dữ liệu để liệt kê
- Phần lớn các bài toán cần lưu trữ dữ liệu, thường là được lưu trong mảng hoặc danh sách liên kết



Ứng dụng của Ngăn xếp

- Xử lý các sự kiện theo trình tự vào-sau-ra-trước
- Khử đệ quy
- Tìm kiếm theo chiều sâu trên đồ thị
- Đảo ngược chuỗi
- Kiểm tra dãy ngoặc
- ...

Ứng dụng của Hàng đợi

- Xử lý các sự kiện theo trình tự vào-trước-ra-trước
- Tìm kiếm theo chiều rộng trên đồ thị
- Tìm đường đi qua ít cạnh nhất
- Thuật toán loang
- ...

Ứng dụng của Hàng đợi ưu tiên

- Xử lý các sự kiện theo trình tự ưu tiên giá trị sử dụng tốt nhất
- Tìm đường đi ngắn nhất trên đồ thị
- Cây khung nhỏ nhất theo thuật toán Prim
- Một số thuật toán tham lam
- ...

Ứng dụng của kiểu Ánh xạ

- Gắn một giá trị với một khóa
- Bảng tần xuất
- Mảng lưu trữ khi thực hiện thuật toán Quy hoạch động
- ...

1 Các kiểu dữ liệu cơ bản

2 Số nguyên lớn

3 Thư viện CTDL và Thuật toán

4 Biểu diễn tập hợp bằng Bitmask

5 Một số ứng dụng của CTDL

6 Cấu trúc dữ liệu mở

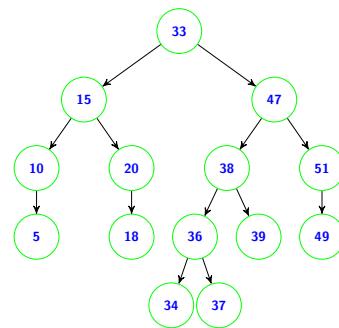
7 Biểu diễn đồ thị

Cấu trúc dữ liệu mở (Augmenting Data Structures)

- Nhiều khi cần lưu trữ thêm thông tin trong cấu trúc dữ liệu đang sử dụng để có thêm tính năng cho thuật toán
- Thông thường thì không làm được điều này với các cấu trúc dữ liệu trong thư viện chuẩn
- Cần tự cài đặt để có thể tùy biến
- Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

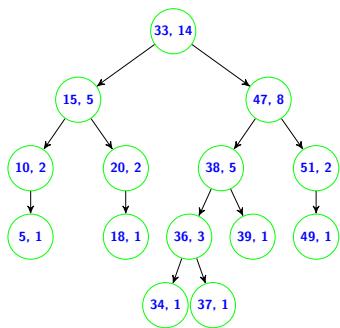
Cây nhị phân tìm kiếm mở

- Thiết lập một Cây nhị phân tìm kiếm mở và muốn thực hiện hiệu quả:
 - Đếm số lượng phần tử $< x$
 - Tìm phần tử lớn thứ k
- Phương pháp trực tiếp là duyệt qua tất cả các đỉnh: $\mathcal{O}(n)$



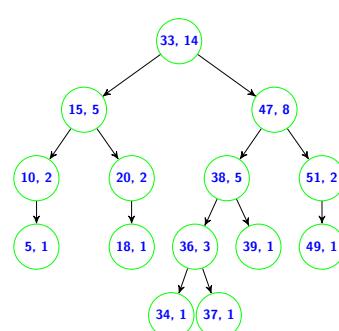
Cây nhị phân tìm kiếm mở

- Tư tưởng: Tại mỗi nút lưu kích thước cây con của nó
- Thông tin lưu trữ này sẽ được cập nhật khi thêm/xóa các phần tử mà không ảnh hưởng đến độ phức tạp chung của thuật toán



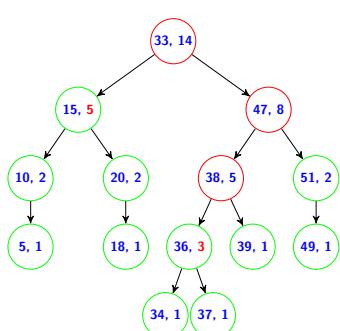
Cây nhị phân tìm kiếm mở

- Tính số lượng phần tử < 38
 - Tìm vị trí 38 trên cây
 - Đếm số đỉnh duyệt qua mà nhỏ hơn 38
 - Khi duyệt đến một đỉnh mà tiếp theo sẽ phải duyệt sang phải, lấy kích thước cây con trái và cộng vào biến đếm cần tính



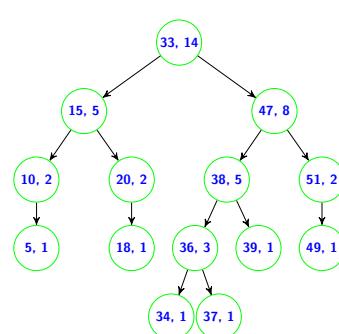
Cây nhị phân tìm kiếm mở

- Tính số lượng phần tử < 38
 - Tìm vị trí 38 trên cây
 - Đếm số đỉnh duyệt qua mà nhỏ hơn 38
 - Khi duyệt đến một đỉnh mà tiếp theo sẽ phải duyệt sang phải, lấy kích thước cây con trái và cộng vào biến đếm cần tính
- Độ phức tạp $\mathcal{O}(\log n)$



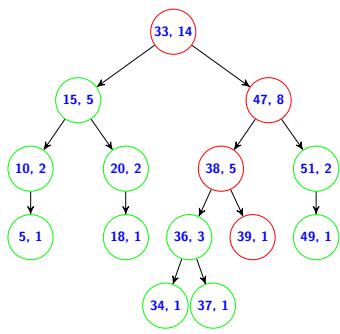
Cây nhị phân tìm kiếm mở

- Tìm phần tử lớn thứ k
 - Tại một đỉnh mà cây con trái của nó có kích thước là m
 - Nếu $k = m + 1$, thu được phần tử cần tìm
 - Nếu $k \leq m$, tìm phần tử lớn thứ k trong cây con trái
 - Nếu $k > m + 1$, tìm phần tử lớn thứ $k - m - 1$ trong cây con phải



Cây nhị phân tìm kiếm mở

- Tìm phần tử lớn thứ k
 - Tại một đỉnh mà cây con trái của nó có kích thước là m
 - Nếu $k = m + 1$, thu được phần tử cần tìm
 - Nếu $k \leq m$, tìm phần tử lớn thứ k trong cây con trái
 - Nếu $k > m + 1$, tìm phần tử lớn thứ $k - m - 1$ trong cây con phải
- Ví dụ: $k = 11$



1 Các kiểu dữ liệu cơ bản

2 Số nguyên lớn

3 Thư viện CSDL và Thuật toán

4 Biểu diễn tập hợp bằng Bitmask

5 Một số ứng dụng của CSDL

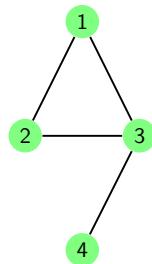
6 Cấu trúc dữ liệu mở

7 Biểu diễn đồ thị

Danh sách kề

1: 2, 3
2: 1, 3
3: 1, 2, 4
4: 3

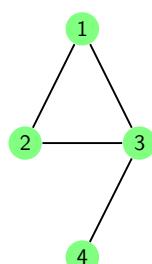
```
vector<int> Adj[5];
Adj[1].push_back(2);
Adj[1].push_back(3);
Adj[2].push_back(1);
Adj[2].push_back(3);
Adj[3].push_back(1);
Adj[3].push_back(2);
Adj[3].push_back(4);
Adj[4].push_back(3);
```



Danh sách cạnh

(1,2)
(1,3)
(2,3)
(3,4)

```
vector<pair<int, int>> Edges;
Edges.push_back(make_pair(1,2));
Edges.push_back(make_pair(1,3));
Edges.push_back(make_pair(2,3));
Edges.push_back(make_pair(3,4));
```



Biểu diễn đồ thị

• Có nhiều dạng đồ thị:

- ▶ Có hướng vs. Vô hướng
- ▶ Có trọng số vs. Không trọng số
- ▶ Đơn đồ thị vs. Đa đồ thị

• Có nhiều cách biểu diễn đồ thị

• Một số đồ thị đặc biệt (như Cây) có cách biểu diễn đặc biệt

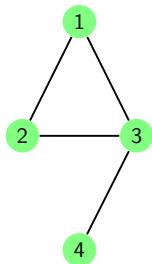
• Chủ yếu sử dụng các biểu diễn chung:

- ① Danh sách kề
- ② Ma trận kề
- ③ Danh sách cạnh

Ma trận kề

0	1	1	0
1	0	1	0
1	1	0	1
0	0	1	0

```
bool Adj[5][5];
Adj[1][2] = true;
Adj[1][3] = true;
Adj[2][1] = true;
Adj[2][3] = true;
Adj[3][1] = true;
Adj[3][2] = true;
Adj[3][4] = true;
Adj[4][3] = true;
```



Hiệu quả

	Danh sách kề	Ma trận kề	Danh sách cạnh
Lưu trữ	$\mathcal{O}(V + E)$	$\mathcal{O}(V ^2)$	$\mathcal{O}(E)$
Thêm đỉnh	$\mathcal{O}(1)$	$\mathcal{O}(V ^2)$	$\mathcal{O}(1)$
Thêm cạnh	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Xóa đỉnh	$\mathcal{O}(E)$	$\mathcal{O}(V ^2)$	$\mathcal{O}(E)$
Xóa cạnh	$\mathcal{O}(E)$	$\mathcal{O}(1)$	$\mathcal{O}(E)$
Truy vấn: u, v có kề nhau không?	$\mathcal{O}(V)$	$\mathcal{O}(1)$	$\mathcal{O}(E)$

- Các cách biểu diễn khác nhau hiệu quả tùy tình huống sử dụng
- Cải tiến cách biểu diễn tùy thuộc vào bài toán
- Có thể cùng lúc sử dụng nhiều cách biểu diễn

Thank you for
your attentions!



25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Đệ Qui và Nhánh Cận THUẬT TOÁN ỨNG DỤNG

- 1 Giới thiệu
- 2 Quay lui
- 3 Nhánh và Cận

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán

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

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

- 1 Giới thiệu
 - Đệ qui
 - Mô hình chung của đệ qui
 - Đệ qui đối với các mô hình giải bài
 - Duyệt toàn bộ
- 2 Quay lui
- 3 Nhánh và Cận

Đệ qui là gì

Trong thực tế ta thường gặp những đối tượng bao gồm chính nó hoặc được định nghĩa dưới dạng của chính nó. Ta nói các đối tượng đó được xác định một cách đệ qui



Kỹ thuật đệ qui

Kỹ thuật đệ qui là kỹ thuật tự gọi đến chính mình với đầu vào kích thước thường là nhỏ hơn

Việc phát triển kỹ thuật đệ qui là thuận tiện khi cần xử lý với các đối tượng được định nghĩa đệ qui (chẳng hạn: tập hợp, hàm, cây, ...)



Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

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



Đệ qui là gì

Trong thực tế ta thường gặp những đối tượng bao gồm chính nó hoặc được định nghĩa dưới dạng của chính nó. Ta nói các đối tượng đó được xác định một cách đệ qui

Đệ qui và qui nạp

Đệ qui và qui nạp toán học có những nét tương đồng và là bổ sung cho nhau. Định nghĩa đệ qui thường giúp cho chứng minh bằng qui nạp các tính chất của các đối tượng được định nghĩa đệ qui. Ngược lại, các chứng minh bằng qui nạp toán học thường là cơ sở để xây dựng các thuật toán đệ qui để giải quyết nhiều bài toán:

- (1) Bước cơ sở qui nạp —> giống như bước cơ sở trong định nghĩa đệ qui
- (2) Bước chuyển qui nạp —> giống như bước đệ qui

Mô hình chung của đệ qui

```
1 void Try(input) {  
2     if (<Kich_Thuoc_Dau_Vao_Du_Nho>) {  
3         do <Buoc_Co_So>  
4             //Tra_Ve_KQ_Truong_Hop_Co_So  
5     } else { //Buoc de qui  
6         foreach(<Bai_Toan_Con_Trong_CTDQ>)  
7             call Try(new_input);  
8         Combine(<Loi_Giai_Cac_Bai_Toan_Con>);  
9         return solution;  
10    }  
11 }
```

- Độ phức tạp hàm đệ qui có thể được tính tiêm cận đơn giản bởi số lượng lời gọi đệ qui nhân với độ phức tạp tối đa của một lời gọi đệ qui

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

Duyệt toàn bộ

Duyệt toàn bộ đa phần phải sử dụng kỹ thuật đệ qui
(Một phương pháp ít phổ biến hơn là phương pháp sinh kế tiếp)

- Chia để trị
- Qui hoạch động
- Tham lam

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ
- **Chia để trị**
- Qui hoạch động
- Tham lam



Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ
- Chia để trị
- **Qui hoạch động**
- Tham lam



Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ

Duyệt toàn bộ đa phần phải sử dụng kỹ thuật đê qui
(Một phương pháp ít phổ biến hơn là phương pháp sinh kế tiếp)

- Chia để trị

Các thuật toán được phát triển dựa trên phương pháp chia để trị thông thường
được mô tả dưới dạng kỹ thuật đê qui

- Qui hoạch động

Các thuật toán được phát triển dựa trên phương pháp qui hoạch động trả nén
sáng sủa hơn khi được mô tả dưới dạng kỹ thuật đê qui

- **Tham lam**: Các thuật toán tham lam có thể cài đặt theo kỹ thuật đê qui



Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ

Duyệt toàn bộ đa phần phải sử dụng kỹ thuật đê qui
(Một phương pháp ít phổ biến hơn là phương pháp sinh kế tiếp)

- **Chia để trị**

Các thuật toán được phát triển dựa trên phương pháp chia để trị thông thường
được mô tả dưới dạng kỹ thuật đê qui

- Qui hoạch động
- Tham lam



Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ

Duyệt toàn bộ đa phần phải sử dụng kỹ thuật đê qui
(Một phương pháp ít phổ biến hơn là phương pháp sinh kế tiếp)

- **Chia để trị**

Các thuật toán được phát triển dựa trên phương pháp chia để trị thông thường
được mô tả dưới dạng kỹ thuật đê qui

- **Qui hoạch động**

Các thuật toán được phát triển dựa trên phương pháp qui hoạch động trả nén
sáng sủa hơn khi được mô tả dưới dạng kỹ thuật đê qui



- **Phương pháp vạn năng Duyệt toàn bộ** (Brute force – Exhaustive search)
 - ▶ bài toán yêu cầu tìm một hoặc nhiều đối tượng có đặc tính riêng (*loại bài toán*)

▶ áp dụng mô hình *Duyệt toàn bộ*: duyệt qua tất cả các đối tượng, với mỗi đối tượng, kiểm tra xem nó có đặc tính cần tìm không:

- ★ nếu có, dừng lại;
- ★ nếu không, tiếp tục tìm



Duyệt toàn bộ

- Cho một tập hữu hạn các phần tử
- Yêu cầu tìm một phần tử trong tập thỏa mãn một số ràng buộc
 - hoặc tìm **tất cả** các phần tử trong tập thỏa mãn một số ràng buộc



Duyệt toàn bộ

- Cho một tập hữu hạn các phần tử
- Yêu cầu tìm một phần tử trong tập thỏa mãn một số ràng buộc
 - hoặc tìm **tất cả** các phần tử trong tập thỏa mãn một số ràng buộc
- Đơn giản! Chỉ cần duyệt qua tất cả các phần tử trong tập, với mỗi phần tử thì kiểm tra xem nó có thỏa mãn các ràng buộc không
- Tất nhiên là cách này không hiệu quả do có thể dẫn đến bùng nổ tổ hợp
- Nhưng nhớ là nên tìm cách giải đơn giản nhất mà chạy trong giới hạn thời gian
- Duyệt toàn bộ luôn là mô hình giải bài đầu tiên nên nghĩ đến khi giải một bài toán



Phân tích thời gian tính khấu trừ (Amortized time)

Đối với các thuật toán duyệt toàn bộ, khi số lượng lời giải lên đến hàm mũ, ta cần đánh giá hiệu quả của thuật toán thông qua thời gian tính khấu trừ

Thời gian tính khấu trừ là thời gian tính trung bình toàn bộ thuật toán mà một câu hình lời giải được liệt kê ra. Như vậy đại lượng này được ước lượng bởi tổng số bước chạy của thuật toán chia cho tổng số câu hình lời giải của bài toán:

$$\text{Thời gian tính khấu trừ} = \frac{\#bước}{\#\text{lời giải}}$$

Thuật toán duyệt toàn bộ được gọi là hiệu quả nếu thời gian tính khấu trừ là hằng số $\mathcal{O}(1)$ (Constant Amortized Time – CAT)



Thuật toán quay lui

- Tư tưởng chính của thuật toán quay lui là xây dựng dần các thành phần của câu hình lời giải S bằng cách **thử tất cả các khả năng có thể**, xuất phát từ trạng thái rỗng của lời giải
- Mô tả:** giả thiết câu hình lời giải được mô tả bởi một bộ gồm n thành phần x_1, x_2, \dots, x_n . Giả sử đã xác định được $i - 1$ thành phần x_1, x_2, \dots, x_{i-1} (gọi là **lời giải bộ phận cấp $i - 1$**). Bây giờ cần xác định thành phần x_i bằng cách thử tất cả các ứng viên có thể có nhờ các luật chuyển. Với mỗi ứng viên c , kiểm tra xem c có chấp nhận được hay không, xảy ra 2 khả năng:
 - ① nếu chấp nhận c thì xác định x_i theo c ; nếu $i = n$ thì ghi nhận lời giải mới, trái lại tiếp hành việc xác định x_{i+1}
 - ② nếu không có khả năng nào cho x_i thì quay lại bước trước để xác định lại x_{i-1}
- Lưu ý:** ghi nhớ tại mỗi bước những khả năng nào đã thử để tránh trùng lặp. Các thông tin này cần được lưu trữ theo cơ cấu stack (vào sau ra trước - LIFO)



1 Giới thiệu

2 Quay lui

- Sơ đồ chung
- Liệt kê nhị phân
- Liệt kê hoán vị
- Liệt kê tổ hợp
- Bài toán chia kẹo
- Bài toán xếp hậu

3 Nhánh và Cận

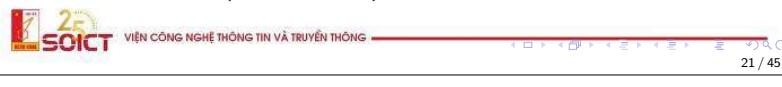


Sơ đồ chung

Bước xác định x_i có thể được diễn tả qua thủ tục được tổ chức đệ qui dưới đây:

```
1 void Try(int i) {
2     foreach(<Ung_Vien_Duoc_Chap_Nhan_c>) {
3         Update(<Cac_Bien_Trang_Thai>);
4         x[i] -- c;
5         if (i==n) <Ghi_Nhan_Mot_Loi_Giai> ;
6         else Try(i+1);
7         <Tra_Cac_Bien_Ve_Trang_Thai_Cu>;
8     }
9 }
```

Quá trình tìm kiếm lời giải theo thuật toán quay lui có thể được mô tả bởi cây tìm kiếm lời giải (Vẽ cây đệ qui)



Liệt kê nhị phân

- ❶ Liệt kê các xâu nhị phân độ dài n

```
1 void Try(int k) {  
2     for (int i=0; i<=1; i++) {  
3         A[k] = i;  
4         if (k==n) <Ghi_Nhan_Mot_Cau_Hinh>;  
5         else Try(k+1);  
6     }  
7 }
```

BÀI TẬP: Phân tích độ phức tạp khâu trừ của thuật toán!



Liệt kê hoán vị

- ❷ Liệt kê các hoán vị của n phần tử

```
1 void Try(int k) {  
2     for (int i=1; i<=n; i++) {  
3         if (!bMark[i]) {  
4             A[k] = i;  
5             bMark[i] = true;  
6             if (k==n) <Ghi_Nhan_Mot_Cau_Hinh>;  
7             else Try(k+1);  
8             bMark[i] = 0;  
9         }  
10    }  
11 }
```

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khâu trừ của thuật toán



Bài toán chia kẹo

- ❸ Liệt kê tất cả các cách chia m kẹo cho n em bé sao cho em bé nào cũng có kẹo

- ▶ Dưa về bài toán liệt kê tất cả các nghiệm nguyên dương của phương trình tuyến tính

$$x_1 + x_2 + \cdots + x_n = m$$

với $(a_i)_{1 \leq i \leq n}$ và m và các số nguyên dương

- ▶ Lời giải bộ phân $(x_1, x_2, \dots, x_{k-1})$
- ▶ $f = \sum_{i=1}^{k-1} x_i$, tổng số kẹo đã chia
- ▶ $p = n - k$, số lượng em bé còn phải chia
- ▶ $m_0 = m - f - p$, số lượng kẹo tối đa có thể chia cho em k
- ▶ Úng viên x_k và $\{v \in \mathbb{Z} \mid 1 \leq v \leq m\}$



Liệt kê nhị phân: CAT

```
1 void Try(int k) {  
2     for (int i=0; i<=1; i++) {  
3         A[k] = i;  
4         if (k==n) <Ghi_Nhan_Mot_Cau_Hinh>;  
5         else Try(k+1);  
6     }  
7 }
```

$$\begin{aligned} \text{Thời gian tính khâu trừ} &= \frac{\#bước}{\#lời_giải} \\ &= \frac{O(1) \times \#lời_gọi_đề_quí}{\#nút_cây_đề_quí} = \frac{O(1) \times \#nút_cây_đề_quí}{\#xâu_nhi_phân} = O(1) \end{aligned}$$



Liệt kê tổ hợp

- ❹ Liệt kê các tổ hợp chập m của n phần tử $\{1, 2, \dots, n\}$

```
1 void Try(int k) {  
2     for (int i= A[k-1]+1; i<=n-m+k; i++) {  
3         A[k] = i;  
4         if (k==m) <Ghi_Nhan_Mot_Cau_Hinh>;  
5         else Try(k+1);  
6     }  
7 }
```

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khâu trừ của thuật toán



Code bài toán chia kẹo

```
1 void Try(int k) {  
2     if (k==n) {  
3         x[k] = m0 - f;  
4         return <Mot_Loi_Giai>  
5     }  
6     m0 = m - f - (n - k);  
7     for (int v = 1; v <= m0; ++v) {  
8         x[k] = v;  
9         f = f + v;  
10        Try(k+1);  
11        f = f - v;  
12    }  
13 }
```

- Gọi Try(1);

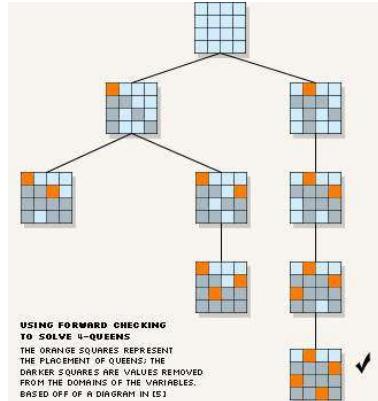
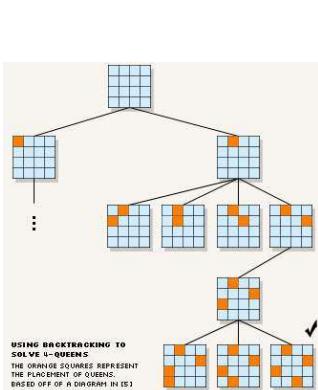


Code bài toán chia kẹo

```

1 void Try(int k) {
2     if (k==n) {
3         x[k] = m0 - f;
4         return <Mot_Loi_Giai>
5     }
6     m0 = m - f - (n - k);
7     for (int v = 1; v <= m0; ++v) {
8         x[k] = v;
9         f = f + v;
10        Try(k+1);
11        f = f - v;
12    }
13 }
```

- Gọi Try(1);
- Số lượng lời giải: $(m-1)_{n-1}$

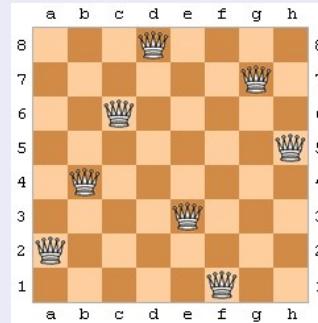


Một số bài toán ví dụ

- Mã đi tuần: Cho bàn cờ $n \times n$ và một quân mã xuất phát tại vị trí (i, j) . Hãy di chuyển quân mã trên bàn cờ sao cho có thể đi được toàn bộ các ô trên bàn cờ mà mỗi ô chỉ được qua 1 lần. Liệt kê tất cả khả năng có thể
- Bài toán Khoảng cách Hamming
<http://uva.onlinejudge.org/external/7/729.html>

Bài toán xếp hậu

Liệt kê tất cả các cách xếp n quân Hậu trên bàn cờ $n \times n$ sao cho chúng không ăn được lẫn nhau.



Code bài toán xếp hậu

```

1 void Try(int i) {
2     for (int j = 1; j <= n; ++j)
3         if (bCol[j] && bDiag1[i+j] && bDiag2[i-j]) {
4             iRes[i] = j; //Chap_Nhan_j
5             //Ghi_Nhan_Trang_Thai_Moi
6             bCol[j] = false;
7             bDiag1[i+j] = false; bDiag2[i-j] = false;
8             if (i==n) <Ghi_Nhan_Mot_Ket_Qua>;
9             else Try(i+1);
10            //Tra_Lai_Trang_Thai_Cu
11            bCol[j] = true;
12            bDiag1[i+j] = true; bDiag2[i-j] = true;
13        }
14 }
```

n	3	4	7	8	9	10	11	12	13	14	15
H_n	0	2	40	92	352	724	2680	14,200	73,712	365,596	2,279,184
U_n	0	1	6	12	46	92	341	1,781	9,233	45,752	285,053

1 Giới thiệu

2 Quay lui

3 Nhánh và Cận

- Bài toán tối ưu tổ hợp
- Mô hình thuật toán nhánh cận
- Bài toán người du lịch

Bài toán tối ưu tổ hợp

Chọn trong số tất cả các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất.

Dạng tổng quát

$$F(x) \rightarrow \min(\max)$$

- $x \in D$ được gọi là một phương án,
- D gọi là tập các phương án của bài toán (thỏa mãn một số tính chất cho trước),
- hàm $F(x)$ gọi là hàm mục tiêu của bài toán,
- phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu, khi đó $f^* = F(x^*)$ được gọi là giá trị tối ưu của bài toán.



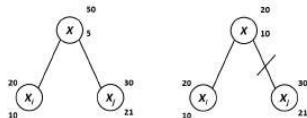
Một số bài toán ứng dụng

- Bài toán người du lịch
- Bài toán cái túi
- Bài toán định tuyến xe (VRP)
- Bài toán lập lịch (Scheduling)
- Bài toán xếp thời khóa biểu (Timetabling)
- Bài toán đóng thùng (Bin Packing)
- Bài toán phân bổ tài nguyên (Resource allocations)
- ...



Thuật toán nhánh cận (TTNC)

- Là một phương pháp giải chủ yếu của bài toán tối ưu tổ hợp.
- Phân hoạch các phương án của bài toán thành 2 hay nhiều tập con được biểu diễn như các nút trên cây tìm kiếm.
- Tìm cách đánh giá cận nhằm loại bỏ những nhánh của cây tìm kiếm mà ta biết chắc là không chứa phương án tối ưu.
- Tinh huống tồi nhất vẫn phải duyệt toàn bộ.



- Áp dụng TTNC trong trường hợp có thể tìm được một hàm G thỏa mãn:
$$G(a_1, a_2, \dots, a_k) \leq \min\{F(x) : x \in D, x_i = a_i, i = 1, 2, \dots, k\}$$
với mọi lời giải bộ phận (a_1, a_2, \dots, a_k) , và với mọi $k = 1, 2, \dots$
- G được gọi là **hàm cận dưới**. Giá trị $G(a_1, a_2, \dots, a_k)$ là cận dưới của phương án bộ phận (a_1, a_2, \dots, a_k) .

Cắt nhánh

- Gọi \bar{f} là giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã duyệt, ký hiệu $\bar{f} = F(\bar{x})$. Ta gọi \bar{x} là **phương án tốt nhất hiện có**, còn \bar{f} là **kỉ lục**.
- Nếu: $G(a_1, a_2, \dots, a_k) > \bar{f}$
 $\Rightarrow \bar{f} < G(a_1, a_2, \dots, a_k) \leq \min\{F(x) : x \in D, x_i = a_i, i = 1, 2, \dots, k\}$
 \Rightarrow tập con các phương án của bài toán $D(a_1, a_2, \dots, a_k)$ chắc chắn không chứa phương án tối ưu.
 \Rightarrow không cần phải phát triển phương án bộ phận (a_1, a_2, \dots, a_k)
 \Rightarrow loại bỏ các phương án trong tập $D(a_1, a_2, \dots, a_k)$ khỏi quá trình tìm kiếm.



Mô hình bài toán MIN tổng quát

Bài toán

$$\min\{F(x) : x \in D\}$$

- D là tập hữu hạn phần tử:
$$D = \{x = (x_1, x_2, \dots, x_n) \in A_1 \times A_2 \times \dots \times A_n ; x \text{ thoả mãn tính chất } P\},$$
- A_1, A_2, \dots, A_n là các tập hữu hạn,
- P là tính chất cho trên tích đề các $A_1 \times A_2 \times \dots \times A_n$.

Nhánh cận

- Sử dụng thuật toán quay lui để xây dựng dần các thành phần của phương án.
- Gọi một bộ phận gồm k thành phần (a_1, a_2, \dots, a_k) xuất hiện trong quá trình thực hiện thuật toán sẽ được gọi là **phương án bộ phận cấp k** .



```
1 void Try(int k) {
2     //Phat_Trien_Phuong_An_Bo_Phan_(a[1], ..., a[k])
3     //_Theo_Thuat_Toan_Quay_Lui_Co_Kiem_Tra_Can_Duo
4     foreach (<a[k]>_Thoa_Man_De_Bai)
5         if (<Chap_Nhan_a[k]>) {
6             iRes[k] = a[k];
7             if (k == n) <Cap_Nhat_Ki_Luc>;
8             else if (G(a[1], ..., a[k]) < f0)
9                 Try(i+1);
10        }
11 }
```

- Khởi tạo giá trị kỉ lục $f0 = +\infty$ hoặc nếu biết một phương án x nào đó có thể khởi tạo $f0 = f(x)$
- Gọi Try(1);
- Sau khi kết thúc đệ qui, nếu $f0 < +\infty$ thì $f0$ là giá trị tối ưu của bài toán, nếu không bài toán không có phương án nào thỏa mãn điều kiện đề bài



Nhận xét

- Nếu không có điều kiện cắt nhánh if ($G(a[1], \dots, a[k]) < f_0$) thì thủ tục Try sẽ liệt kê toàn bộ các phương án của bài toán \Rightarrow thuật toán duyệt toàn bộ.
- Việc xây dựng hàm G phụ thuộc vào từng bài toán cụ thể.
- Việc tính giá trị của G phải đơn giản hơn việc giải bài toán gốc.
- Giá trị của $G(a[1], \dots, a[k])$ phải sát với giá trị tối ưu của bài toán gốc.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Cố định thành phố xuất phát là T_1 . Bài toán Người du lịch được đưa về bài toán: Tìm cực tiểu của hàm

$$F(x_2, x_3, \dots, x_n) = C[1, x_2] + C[x_2, x_3] + \dots + C[x_{n-1}, x_n] + C[x_n, 1] \rightarrow \min$$

Gọi:

$$c_{\min} = \min \{C[i, j], i, j = 1, 2, \dots, n, i \neq j\}$$

là chi phí đi lại nhỏ nhất giữa các thành phố.

Giả sử ta đang có phương án bộ phận (u_1, u_2, \dots, u_k) tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \rightarrow T(u_2) \rightarrow \dots \rightarrow T(u_{k-1}) \rightarrow T(u_k)$$

với chi phí phải trả theo hành trình bộ phận này là

$$\sigma = C[1, u_2] + C[u_2, u_3] + \dots + C[u_{k-1}, u_k].$$

Do chi phí phải trả cho việc đi qua mỗi một trong số $n - k + 1$ đoạn đường còn lại đều không nhiều hơn c_{\min} \Rightarrow **cận dưới** cho phương án bộ phận (u_1, u_2, \dots, u_k) :

$$G(u_1, u_2, \dots, u_k) = \sigma + (n - k + 1)c_{\min}$$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

```

1 void Try(int k) { //Tham_Thanh_Pho_Thu_k
2     for (int v=1; v <= n; ++v)
3         if (!bVisited[v]) {
4             iRes[k] = v;
5             bVisited[k] = true;
6             f = f + C[iRes[k-1]][v];
7             if (k == n) {
8                 if (f+C[v][iRes[1]]<f0)
9                     f0 = f + C[v][iRes[1]];
10            }
11            else {
12                g = f + (n - k + 1) * cmin;
13                if (g < f0)
14                    Try(k+1);
15            }
16            f = f + C[iRes[k-1]][v];
17            bVisited[k] = false;
18        }
19    }

```

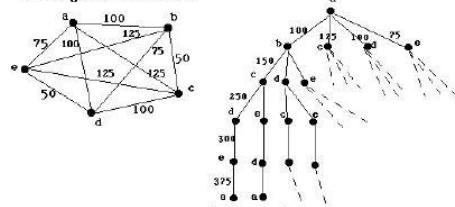


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TTNC giải bài toán Người du lịch



An Instance of the Traveling Salesman Problem



Search Space

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Code bài toán TSP

```

1 int main() {
2     for (int v=1; v <= n; ++v)
3         bVisited[v] = false;
4     f0 = INFINITY; //Khai_Tao_Gia_Tri_Ki_Luc
5     f = 0;
6     iRes[1] = 1; //Co_Dinh_1_La_TP_Xuat_Phát
7     bVisited[1] = true;
8     Try(2);
9     return f0;
10}

```


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÀI TẬP THỰC HÀNH

- TSP
- KNAPSAC
- TAXI
- CVRPOPT
- BCA

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thank you for
your attentions!



www.solict.hust.edu.vn/ fb.com/groups/solict

Chia Đề Trị THUẬT TOÁN ỨNG DỤNG

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

- Duyệt toàn bộ

Duyệt toàn bộ đa phần phải sử dụng kỹ thuật đệ quy
(Một phương pháp ít phổ biến hơn là phương pháp sinh kế tiếp)

- Chia để trị

Các thuật toán được phát triển dựa trên phương pháp chia để trị thông thường
được mô tả dưới dạng kỹ thuật đệ qui

- Qui hoạch động

- Tham lam



25 YEARS ANNIVERSARY
SOLICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

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**
- Qui hoạch động
- Tham lam

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

- Chia để trị
- Giảm để trị
- Một số loại chia để trị thông dụng khác

Chia để trị

Chia để trị là một mô hình giải bài theo hướng làm dễ bài toán đi bằng cách chia thành các phần nhỏ hơn và xử lý từng phần một

Thông thường làm theo 3 bước chính:

- ① CHIA: chia bài toán thành một hay nhiều bài toán con - thường hay chia một nửa hoặc gần một nửa
- ② XỬ LÝ: giải đệ qui mỗi bài toán con - mỗi bài toán cần giải trả về
- ③ KẾT HỢP: kết hợp lời giải các bài toán con thành lời giải bài toán ban đầu

1 Chia để trị

- Mô hình chia để trị
- Một số bài toán cơ bản và ứng dụng
- Phân tích độ phức tạp thuật toán chia để trị
- Sắp xếp trộn
- Đoạn con có tổng lớn nhất

2 Giảm để trị

3 Một số loại chia để trị thông dụng khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Mô hình chung của chia để trị

```
1 void DC(n) {  
2     if n <= n0 {  
3         [Giai_Bai_Toan_Con_Mot_Cach_Truc_Tiep];  
4     } else {  
5         [Chia_Bai_Toan_Thanh_a_Bai_Toan_Con_Kich_Thuoc_n/b];  
6         [foreach Moi_Bai_Toan_Trong_a_Bai_Toan_Con] {  
7             call DC(n/b);  
8         }  
9         [Tong_Hop_Loi_Giai_Cua_a_Bai_Toan_Con];  
10        return solution;  
11    }  
12 }
```

- n_0 là kích thước nhỏ nhất của bài toán con (bước neo đệ qui), được giải trực tiếp
- a là số lượng bài toán con cần giải
- b liên quan đến kích thước của bài toán con được chia



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Ứng dụng của thuật toán chia để trị

- Giải các bài toán khó: bằng cách chia nhỏ thành cách bài toán nhỏ dễ giải hơn và kết hợp các lời giải bài toán nhỏ lại thành lời giải bài toán ban đầu
- Tính toán song song: tính toán trên nhiều máy tính, nhiều vi xử lý, tính toán trên dàn/lưới máy tính. Trong trường hợp này độ phức tạp chỉ phí truyền thông giữa các phần tính toán là rất quan trọng
- Truy cập bộ nhớ: bài toán được chia nhỏ đến khi có thể giải trực tiếp trên bộ nhớ đệm sẽ cho thời gian thực hiện nhanh hơn nhiều so với việc truy cập sử dụng bộ nhớ chính
- Xử lý dữ liệu: dữ liệu lớn được chia thành cách phần nhỏ để lưu trữ và xử lý dữ liệu
- ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Một số bài toán chia để trị cơ bản

- Sắp xếp nhanh (Quick sort)
- Sắp xếp trộn (Merge sort)
- Thuật toán Karatsuba nhân nhanh số lớn
- Thuật toán Strassen nhân ma trận
- Rất nhiều thuật toán trong tính toán hình học
 - ▶ Bao lồi (Convex hull)
 - ▶ Cặp điểm gần nhất (Closest pair of points)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Phân tích độ phức tạp thuật toán chia để trị

- Được mô tả bởi một công thức truy hồi
- Gọi $T(n)$ là thời gian tính toán của bài toán kích thước n
- $$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq n_c \\ aT(n/b) + D(n) + C(n) & \text{if } n \geq n_c, \end{cases}$$
với
 - a : số lượng bài toán con
 - n/b : kích thước mỗi bài toán con
 - $D(n)$: chi phí việc chia nhỏ bài toán
 - $C(n)$: chi phí việc kết hợp kết quả các bài toán con



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Ví dụ

```
1 void Solve(int n) {  
2     if (n == 0)  
3         return;  
4  
5     Solve(n/2);  
6     Solve(n/2);  
7  
8     for (int i = 0; i < n; i++) {  
9         //Mot_So_Cau_Lenh_Don  
10    }  
11}
```

- $T(n) = 2T(n/2) + n$



Dịnh lý thợ rút gọn

$T(n) = aT(n/b) + cn^k$, với a, b, c, k là các hằng số dương, và $a \geq 1, b \geq 2$, ta có

$T(n) =$

- $\mathcal{O}(n^{\log_b a})$, nếu $a > b^k$,
- $\mathcal{O}(n^k \log n)$, nếu $a = b^k$,
- $\mathcal{O}(n^k)$, nếu $a < b^k$,



Hàm trộn

- Hàm trộn là hàm thiết yếu trong thuật toán sắp xếp trộn
- Giả sử các dãy con được lưu trữ trong mảng A . Hai dãy con $A[p \dots q]$ và $A[q+1 \dots r]$ đã được sắp xếp
- $\text{Merge}(A, p, q, r)$ sẽ trộn 2 dãy con thành dãy kết quả $A[p \dots r]$
- $\text{Merge}(A, p, q, r)$ tốn thời gian $\Theta(r - p + 1)$

```
1 Merge_Sort(A, p, r){  
2     if (p < r) {  
3         q=(p+r)/2  
4  
5         Merge_Sort(A, p, q)  
6         Merge_Sort(A, q+1, r)  
7         Merge(A, p, q, r)}  
8 }
```

Gọi hàm $\text{Merge_Sort}(A, 1, n)$ (với $n = \text{length}(A)$)



Chia để trị: Độ phức tạp thuật toán

- Nhưng làm thế nào để giải được công thức truy hồi này?
- Thường đơn giản nhất là sử dụng định lý thợ để giải
 - ▶ Định lý thợ cho phép đưa ra lời giải cho công thức đệ qui dạng $T(n) = aT(n/b) + f(n)$ theo ký pháp hàm tiệm cận
 - ▶ Đa phần các thuật toán chia để trị thông dụng có công thức truy hồi theo mẫu này
- Định lý thợ cho biết $T(n) = 2T(n/2) + n$ có thời gian tính $O(n \log n)$
- Nên thuộc định lý thợ
- Phương pháp cây đệ qui cũng rất hữu ích để giải công thức truy hồi



Sắp xếp trộn

- **CHIA:** chia dãy n phần tử thành 2 dãy con mỗi dãy $n/2$ phần tử
- **XỬ LÝ:** sắp xếp mỗi dãy con sử dụng lời gọi đệ qui thuật toán sắp xếp trộn, đến khi độ dài dãy là 1 thì dừng
- **KẾT HỢP** trộn 2 dãy con đã được sắp xếp lại thành dãy kết quả

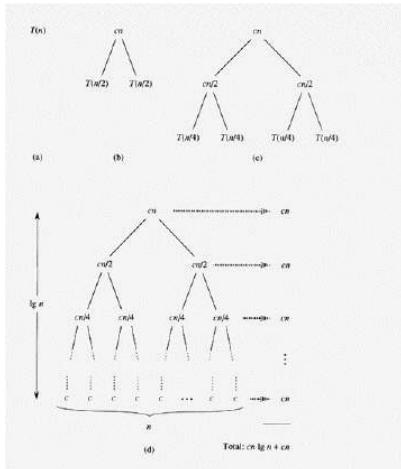


Phân tích độ phức tạp thuật toán Sắp xếp trộn

- **CHIA:** $D(n) = \Theta(1)$
- **XỬ LÝ:** $a = 2, b = 2$, so $2T(n/2)$
- **KẾT HỢP:** $C(n) = \Theta(n)$
- $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$
- $T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$
- $T(n) = \mathcal{O}(n \log n)$ theo Định lý thợ hoặc Phương pháp Cây đệ qui



Cây đệ qui phân tích độ phức tạp Merge_Sort



Đoạn con có tổng lớn nhất

- Cho một mảng số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

Đoạn con có tổng lớn nhất

- Cho một mảng số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

- Tổng của đoạn có trọng số lớn nhất trong mảng là 8

Đoạn con có tổng lớn nhất

- Cho một mảng số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong mảng có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

- Tổng của đoạn có trọng số lớn nhất trong mảng là 8

- Cách giải thế nào?

- Phương pháp trực tiếp thử tất cả gần $\approx n^2$ khoảng, và tính trọng số mỗi đoạn, cho độ phức tạp $O(n^3)$
- Ta có thể xử lý kỹ thuật bởi một "mẹo" lưu trữ cố định trong vòng lặp để giảm độ phức tạp về $O(n^2)$
- Liệu có thể làm tốt hơn với phương pháp Chia để trị?

Đoạn con có tổng lớn nhất

- CHIA:** chia dãy n phần tử thành 2 dãy con tại điểm giữa $mid = \lfloor (n+1)/2 \rfloor$ phần tử, ký hiệu là A_L và A_R
- XỬ LÝ:** Tính đoạn con có tổng lớn nhất của mỗi nửa một cách đệ quy. Gọi w_L và w_R là trọng số của các đoạn con có tổng lớn nhất trong A_L và A_R tương ứng
- KẾT HỢP** ký hiệu trọng số của đoạn con lớn nhất mà nằm đè lên điểm chia ở giữa là w_M . Kết quả cần tìm sẽ là $\max(w_L, w_R, w_M)$
 - w_M được tính bằng tổng độ dài đoạn con có tổng lớn nhất nửa bên trái mà kết thúc tại mid và độ dài đoạn con có tổng lớn nhất nửa bên phải mà bắt đầu tại $mid + 1$

Đoạn con có tổng lớn nhất: Code

```

1 int SubSeqMax(int i, int j){
2     if (i == j) return a[i];
3     int mid = (i+j)/2;
4     int wL = SubSeqMax(i,mid);
5     int wR = SubSeqMax(mid+1,j);
6     int maxLM = MaxLeftMid(i,mid);
7     int maxRM = MaxRightMid(mid+1,j);
8     int wM = maxL + maxR;
9     return max(max(wL,wR),wM);
10}

```

- Gọi `SubSeqMax(1,n);`
- Độ phức tạp: $O(n \log n)$ (giống như bài toán Sắp xếp trộn)

Đoạn con có tổng lớn nhất: Code

```
1 int MaxLeftMid(int i, int j){  
2     int maxLM = a[j];  
3     int s = 0;  
4     for(int k = j; k >= i; k--){  
5         s += a[k];  
6         maxLM = max(maxLM,s);  
7     }  
8     return maxLM;  
9 }  
  
10 int MaxRightMid(int i, int j){  
11     int maxRM = a[i];  
12     int s = 0;  
13     for(int k = i; k <= j; k++){  
14         s += a[k];  
15         maxRM = max(maxRM,s);  
16     }  
17     return maxRM;  
18 }  
19 }
```



22 / 48

BÀI TẬP THỰC HÀNH

- SUBSEQMAX
- CLOPAIR



48 / 48

1 Chia để trị

2 Giảm để trị

- Tìm kiếm nhị phân
- Tìm kiếm nhị phân trên các số nguyên
- Tìm kiếm nhị phân trên các số thực
- Tìm kiếm nhị phân câu trả lời

3 Một số loại chia để trị thông dụng khác



24 / 48

Tìm kiếm nhị phân

- Cho một mảng n phần tử $A[1], A[2], \dots, A[n]$ đã được sắp xếp, hãy kiểm tra xem mảng có chứa phần tử x không
- Thuật toán:
 - 1 Trường hợp biên: mảng rỗng, trả lời KHÔNG
 - 2 So sánh x với phần tử ở vị trí giữa mảng
 - 3 Nếu bằng, tìm thấy x và trả lời CÓ
 - 4 Nếu nhỏ hơn, x chắc chắn nằm bên nửa trái mảng
 - * Tìm kiếm nhị phân (đệ qui) tiếp nửa trái mảng
 - 5 Nếu lớn hơn, x chắc chắn nằm bên nửa phải mảng
 - * Tìm kiếm nhị phân (đệ qui) tiếp nửa phải mảng



26 / 48

Giảm để trị (Decrease and conquer)

- Đôi khi không cần chia bài toán thành nhiều bài toán con, mà chỉ giảm về một bài toán con kích thước nhỏ hơn
- Thường gọi là **Giảm để trị**
- Ví dụ thông dụng nhất là **Tìm kiếm nhị phân**



48 / 48

Tìm kiếm nhị phân

```
1 bool Binary_Search(const vector<int> &A, int lo, int hi, int x){  
2     if (lo > hi)  
3         return false;  
4     int mid = (lo + hi) / 2;  
5     if (A[mid] == x)  
6         return true;  
7     if (x < A[mid])  
8         return Binary_Search(A, lo, mid - 1, x);  
9     if (x > A[mid])  
10        return Binary_Search(A, mid + 1, hi, x);  
11 }
```

- Gọi `Binary_Search(A, 1, n, x);`
- Độ phức tạp:
 - $T(n) = T(n/2) + 1$
 - $\mathcal{O}(\log n)$



27 / 48

Tìm kiếm nhị phân trên các số nguyên

- Đây có lẽ là ứng dụng phổ biến nhất của tìm kiếm nhị phân
- Cụ thể, cho hàm $P : \{0, \dots, n-1\} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ thỏa mãn nếu $P(i) = \text{TRUE}$, thì $P(j) = \text{TRUE}$ với mọi $j > i$
- Yêu cầu tìm chỉ số j nhỏ nhất sao cho $P(j) = \text{TRUE}$

i	0	1	\dots	$j-1$	j	$j+1$	\dots	$n-2$	$n-1$
$P(i)$	FALSE	FALSE	\dots	FALSE	TRUE	TRUE	\dots	TRUE	TRUE

- Có thể thực hiện trong $\mathcal{O}(\log(n) \times f)$, với f là giá của việc đánh giá hàm P

Tìm kiếm nhị phân trên các số nguyên

```
1 int lo = 0, hi = n - 1;
2 while (lo < hi) {
3     int mid = (lo + hi) / 2;
4     if (P(mid)) {
5         hi = mid;
6     } else {
7         lo = mid + 1;
8     }
9 }
10 if (lo == hi && P(lo)) {
11     cout<<"Chi so nho nhat tim duoc la "<< lo;
12 } else {
13     cout<<"Khong ton tai phan tu "<< x;
14 }
```

Tìm kiếm nhị phân trên các số nguyên

- Tìm vị trí của x trong mảng đã sắp xếp A

```
1 bool p(int i) {
2     return A[i] >= x;
3 }
```

Tìm kiếm nhị phân trên các số thực

```
1 double EPS = 1e-10,
2     lo = -1000.0,
3     hi = 1000.0;
4 while (hi - lo > EPS) {
5     double mid = (lo + hi) / 2.0;
6     if (P(mid)) {
7         hi = mid;
8     } else {
9         lo = mid;
10    }
11 }
cout << lo;
```

Tìm kiếm nhị phân trên các số thực

- Đây là phiên bản tổng quát hơn của tìm kiếm nhị phân
- Cho hàm $P : [lo, hi] \rightarrow \{\text{TRUE}, \text{FALSE}\}$ thỏa mãn nếu $P(i) = \text{TRUE}$, thì $P(j) = \text{TRUE}$ với mọi $j > i$
- Yêu cầu tìm số thực nhỏ nhất j sao cho $P(j) = \text{TRUE}$
- Do làm việc với số thực, khoảng $[lo, hi]$ có thể bị chia vô hạn lần mà không dừng ở một số thực cụ thể
- Thay vào đó có thể tìm một số thực j' rất sát với lời giải đúng j , sai số trong khoảng $EPS = 2^{-30}$
- Có thể làm được trong thời gian $\mathcal{O}(\log(\frac{hi-lo}{EPS}))$ tương tự cách làm tìm kiếm nhị phân trên mảng

Tìm kiếm nhị phân trên các số thực

- Có nhiều ứng dụng thú vị
- Tìm căn bậc hai của x

```
1 bool P(double j) {
2     return j*j >= x;
3 }
```

- Tìm nghiệm của hàm $F(x)$

```
1 bool P(double x) {
2     return F(x) >= 0.0;
3 }
```

- Đây cũng được gọi là phương pháp chia đôi trong phương pháp tính (Bisection method)

Tìm kiếm nhị phân câu trả lời

- Một số bài toán có thể khó tìm ra lời giải tối ưu một cách trực tiếp,
- Mặt khác, dễ dàng kiểm tra một số x nào đó có phải là lời giải không
- Phương pháp sử dụng tìm kiếm nhị phân để tìm lời giải nhỏ nhất hoặc lớn nhất của một bài toán
- Chỉ áp dụng được khi bài toán có tính chất tìm kiếm nhị phân: nếu i là một lời giải, thì tất cả $j > i$ cũng là lời giải
- $P(i)$ kiểm tra nếu i là một lời giải, thì có thể áp dụng một cách đơn giản tìm kiếm nhị phân trên P để nhận được lời giải nhỏ nhất hoặc lớn nhất



34 / 48

BÀI TẬP THỰC HÀNH

- AGGCOW
- BOOK1
- PIE
- EKO



35 / 48

Một số loại chia để trị thông dụng khác

1 Chia để trị

2 Giảm để trị

3 Một số loại chia để trị thông dụng khác

- Nhi phân hàm mũ
- Chuỗi Fibonacci



36 / 48

Nhi phân hàm mũ (Binary exponentiation)

- Yêu cầu tính x^n , với x, n là các số nguyên
- Giả thiết ta không biết phương thức pow trong thư viện
- Phương pháp trực tiếp:

```
1 int Pow(int x, int n) {
2     int res = 1;
3     for (int i = 0; i < n; i++) {
4         res = res * x;
5     }
6
7     return res;
8 }
```

- Độ phức tạp $\mathcal{O}(n)$, tuy nhiên với n lớn thì sao?



38 / 48

Nhi phân hàm mũ

- Hãy sử dụng chia để trị
- Để ý 3 đẳng thức sau:
 - $x^0 = 1$
 - $x^n = x \times x^{n-1}$
 - $x^n = x^{n/2} \times x^{n/2}$
- Hoặc theo ngôn ngữ hàm:
 - $Pow(x, 0) = 1$
 - $Pow(x, n) = x \times Pow(x, n - 1)$
 - $Pow(x, n) = Pow(x, n/2) \times Pow(x, n/2)$
- $Pow(x, n/2)$ được sử dụng 2 lần, nhưng ta chỉ cần tính 1 lần:
 - $Pow(x, n) = Pow(x, n/2)^2$



39 / 48

Nhị phân hàm mũ

- Hãy sử dụng các đẳng thức đó để tìm câu trả lời theo cách đệ qui

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     return x * Pow(x, n - 1);  
4 }
```

Nhị phân hàm mũ

- Hãy sử dụng các đẳng thức đó để tìm câu trả lời theo cách đệ qui

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     return x * Pow(x, n - 1);  
4 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n - 1)$
- $\mathcal{O}(n)$

Nhị phân hàm mũ

- Để ý đẳng thức thứ 3:

- $n/2$ không là số nguyên khi n lẻ, vì vậy chỉ sử dụng nó khi n chẵn

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     if (n % 2 != 0) return x * pow(x, n - 1);  
4     int res = Pow(x, n/2);  
5     return res * res;  
6 }
```

- Độ phức tạp?

Nhị phân hàm mũ

- Hãy sử dụng các đẳng thức đó để tìm câu trả lời theo cách đệ qui

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     return x * Pow(x, n - 1);  
4 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n - 1)$

Nhị phân hàm mũ

- Hãy sử dụng các đẳng thức đó để tìm câu trả lời theo cách đệ qui

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     return x * Pow(x, n - 1);  
4 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n - 1)$
- $\mathcal{O}(n)$
- Vẫn chậm như trước...

Nhị phân hàm mũ

- Để ý đẳng thức thứ 3:

- $n/2$ không là số nguyên khi n lẻ, vì vậy chỉ sử dụng nó khi n chẵn

```
1 int Pow(int x, int n) {  
2     if (n == 0) return 1;  
3     if (n % 2 != 0) return x * pow(x, n - 1);  
4     int res = Pow(x, n/2);  
5     return res * res;  
6 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n - 1)$ nếu n lẻ
- $T(n) = 1 + T(n/2)$ nếu n chẵn

Nhị phân hàm mũ

- Để ý đẳng thức thứ 3:

► $n/2$ không là số nguyên khi n lẻ, vì vậy chỉ sử dụng nó khi n chẵn

```
1 int Pow(int x, int n) {
2     if (n == 0) return 1;
3     if (n % 2 != 0) return x * pow(x, n - 1);
4     int res = Pow(x, n/2);
5     return res * res;
6 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n-1)$ nếu n lẻ
- $T(n) = 1 + T(n/2)$ nếu n chẵn
- Do $n-1$ chẵn khi n lẻ:
- $T(n) = 1 + 1 + T((n-1)/2)$ nếu n lẻ



Nhị phân hàm mũ

- Để ý đẳng thức thứ 3:

► $n/2$ không là số nguyên khi n lẻ, vì vậy chỉ sử dụng nó khi n chẵn

```
1 int Pow(int x, int n) {
2     if (n == 0) return 1;
3     if (n % 2 != 0) return x * pow(x, n - 1);
4     int res = Pow(x, n/2);
5     return res * res;
6 }
```

- Độ phức tạp?

- $T(n) = 1 + T(n-1)$ nếu n lẻ
- $T(n) = 1 + T(n/2)$ nếu n chẵn
- Do $n-1$ chẵn khi n lẻ:
- $T(n) = 1 + 1 + T((n-1)/2)$ nếu n lẻ

$\mathcal{O}(\log n)$

Thuật toán tối ưu!



Nhị phân hàm mũ

- Để ý là x không nhất thiết là số nguyên và $*$ không nhất thiết là phép nhân số nguyên...

- Cũng dùng được cho:

- Tính x^n , với x là số thực và $*$ là phép nhân số thực
- Tính A^n , với A là một ma trận và $*$ là phép nhân ma trận
- Tính $x^n \pmod m$, với x là một số nguyên và $*$ là phép nhân số nguyên lấy mod m
- Tính $x * x * \dots * x$, với x là bất kỳ loại phần tử gì và $*$ là một toán tử phù hợp

- Tất cả có thể giải trong $\mathcal{O}(\log(n) \times f)$, với f là giá để thực hiện một toán tử $*$



Số Fibonacci

- Thứ với một cặp xâu, và đặt $+$ là phép toán ghép xâu:

- $G_1 = A$
- $G_2 = B$
- $G_n = G_{n-2} + G_{n-1}$

- Ta thu được dãy các xâu:

- A
- B
- AB
- BAB
- $ABBAB$
- $BABABAB$
- $ABBABBABABBAB$
- $BABABBABABBABABBAB$
- \dots



Số Fibonacci

- Nhắc lại dãy Fibonacci được định nghĩa như sau:

- $Fib_1 = 1$
- $Fib_2 = 1$
- $Fib_n = Fib_{n-2} + Fib_{n-1}$

- Ta có dãy $1, 1, 2, 3, 5, 8, 13, 21, \dots$

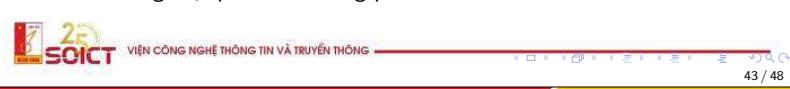
- Có rất nhiều biến thể của dãy Fibonacci

- Một kiểu là cùng công thức nhưng bắt đầu bởi các số khác, ví dụ:

- $F_1 = 5$
- $F_2 = 4$
- $F_n = F_{n-2} + F_{n-1}$

- Ta có dãy $5, 4, 9, 13, 22, 35, 57, \dots$

- Với những loại phần tử không phải số thì sao?



Số Fibonacci

- g_n dài bao nhiêu?

- $\text{len}(G_1) = 1$
- $\text{len}(G_2) = 1$
- $\text{len}(G_n) = \text{len}(G_{n-2}) + \text{len}(G_{n-1})$

- Trông quen thuộc?

- $\text{len}(G_n) = Fib_n$

- Vì vậy các xâu trở nên rất lớn rất nhanh

- $\text{len}(G_{10}) = 55$
- $\text{len}(G_{100}) = 354224848179261915075$
- $\text{len}(G_{1000}) =$

434665576869374564356885276750406258025646605173717

804024817290895365554179490518904038798400792551692

959225930803226347752096896232398733224711616429964

409065331879382989696499285160037044761377951668492

28875



Sô Fibonacci

- Nhiệm vụ: Hãy tính ký tự thứ i trong G_n



Sô Fibonacci

- Nhiệm vụ: Hãy tính ký tự thứ i trong G_n
- Dễ dàng thực hiện trong $\mathcal{O}(\text{len}(n))$, nhưng sẽ cực kỳ chậm với n lớn



Sô Fibonacci

- Nhiệm vụ: Hãy tính ký tự thứ i trong G_n
- Dễ dàng thực hiện trong $\mathcal{O}(\text{len}(n))$, nhưng sẽ cực kỳ chậm với n lớn
- Có thể giải trong $\mathcal{O}(n)$ sử dụng chia để trị



BÀI TẬP THỰC HÀNH

- FIBWORDS
- TRIPLE



Thank you for
your attentions!



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Qui Hoạch Động

THUẬT TOÁN ỨNG DỤNG



- Trong chiến tranh thế giới thứ 2, các ngành khoa học cơ bản ở Mỹ không được đầu tư để dành toàn bộ nguồn lực cho chiến đấu, chỉ những kết quả khoa học ứng dụng trực tiếp cho chiến trường mới được cấp kinh phí nghiên cứu, ví dụ: qui hoạch tuyến tính với bài toán khẩu phần ăn cho binh sĩ.
- Nhà toán học Bellman thời kỳ đó nghiên cứu ra phương pháp 'multistage decision processes' (quá trình ra quyết định thông qua nhiều lớp) trong lĩnh vực lập kế hoạch (planning). Tuy nhiên từ 'planning' không phù hợp vào thời kỳ đó nên ông đã thay bằng từ 'programming' (lập trình) thời thượng hơn khi mà máy tính là đầu tiên của quân đội Mỹ ra đời. Tiếp theo ông thay từ 'multistage' bằng từ 'dynamic' nghe hay hơn thể hiện sự gối nhau về thời gian. Thuật ngữ 'dynamic programming' ra đời từ đó.
- Dynamic programming mang tính kỹ thuật lập trình nhiều hơn là tính mô hình dạng bài toán (như qui hoạch tuyến tính), tuy nhiên từ dịch ra 'Qui Hoạch Động' nghe hay và thuận hơn từ 'Lập Trình Động'.

1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

4 Đổi tiền

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất

7 Qui hoạch động trên bitmask

RICHARD BELLMAN ON THE BIRTH OF DYNAMIC PROGRAMMING

STUART DREYFUS
University of California, Berkeley, CA, Berkeley, California 94720, dreyfus@econberkeley.edu



What follows concerns events from the summer of 1950 when Richard Bellman first became interested in multistage decision processes, until 1955. Although Bellman died on March 19, 1984, the story will be told in his own words since he left behind an entertaining and informative autobiography, *Eve of the Hurricane* (World Scientific, Princeton, New Jersey, 1986), which his publisher, Princeton University Press, has generously approved extensive excerpts.

During the summer of 1949 Bellman, a tenured associate professor of mathematics at Stanford University with a developing interest in analysis, number theory, was consulting for the second summer at the RAND Corporation in Santa Monica. He had graduated from Princeton in 1946 at the age of 25, despite various war-related activities during World War II—including being assigned by the Army to the Manhattan Project in Los Alamos. He had already exhibited outstanding ability both in pure mathematics and in applied operations research. After the war, he was offered a successful conventional academic career, Bellman, during the period under consideration, cast his lot instead with the kind of applied mathematics later to be known as operations research. In those days applied mathematics was seen as directly seconding the needs of the mathematical frontier. No one to enjoy controversy, when invited to speak at various university mathematics department seminars, Bellman delighted in justifying his choice of applied over pure mathematics as being motivated by the real world's greater challenges and mathematical demands.

what RAND was interested in. He suggested that I work on multistage decision processes. I started following that suggestion" (p. 137).

CHOICE OF THE NAME DYNAMIC PROGRAMMING

"I spent the summer of 1950 at RAND. My first task was to find a name for multistage decision processes. An interesting question is, 'Where did the name, dynamic programming, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense. He was also my personal mathematician. He had a fear of the word, research. I'm using the term lightly; I'm using it precisely. His face would suffice, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt then, about the term, dynamic programming. Consider the two words. Assume of a successful conventional academic career, Bellman, during the period under consideration, cast his lot instead with the kind of applied mathematics later to be known as operations research. In those days applied mathematics was seen as directly seconding the needs of the mathematical frontier. No one to enjoy controversy, when invited to speak at various university mathematics department seminars, Bellman delighted in justifying his choice of applied over pure mathematics as being motivated by the real world's greater challenges and mathematical demands.

Hình: R.E.Bellman (1920-1984)

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
- Qui hoạch động**
- Tham lam

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

1 Sơ đồ Qui hoạch động

- Qui hoạch động là gì?
- Công thức Qui hoạch động
- Cài đặt Top-Down với Đề qui có nhớ

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

4 Đổi tiền

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất

Qui 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 qui)
 - ▶ Kết hợp lời giải các bài toán con lại thành lời giải của bài toán cha
- Phương pháp qui hoạch động:
 - ▶ Chia bài toán cha thành các bài toán con *gối nhau*
 - ▶ Giải từng bài toán con (bằng đệ qui)
 - ▶ 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 Qui hoạch động

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

Nhận xét

Bước 1: tìm công thức qui 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 Dệ qui có nhớ

```
1 map<problem, value> Memory;
2
3 value DP(problem P) {
4     if (is_base_case(P))
5         return base_case_value(P);
6
7     if (Memory.find(P) != Memory.end())
8         return Memory[P];
9
10    value result = some value;
11    for (problem Q in subproblems(P))
12        result = Combine(result, DP(Q));
13
14    Memory[P] = result;
15    return result;
16}
```



Bình luận

- Việc sử dụng hàm đệ qui để cài đặt công thức qui hoạch động là cách tiếp cận lập trình tự nhiên và đơn giản cho lập trình giải bài toán qui hoạch động, ta gọi đó là cách tiếp cận lập trình Top-Down, phù hợp với đa số người mới tiếp cận kỹ thuật Qui hoạch động
- Khi đã quen thuộc với các bài qui 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 qui và tìm ra các phần tử của lời giải dựa trên thông tin các bài toán con đã được lưu trữ trong mảng memory



1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

- Công thức Qui hoạch động
- Dệ qui không nhớ
- Dệ qui có nhớ
- Độ phức tạp

3 Đoạn con có tổng lớn nhất

4 Dổi tiền

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất



Tính số Fibonacci

Hai số đầu tiên của dãy Fibonacci là 1 và 1. Tất cả các số khác của dãy được tính bằng tổng của hai số ngay trước nó trong dãy

- Yêu cầu: Tính số Fibonacci thứ n
- Thủ giải bài toán bằng phương pháp Qui hoạch động

- ④ Tìm công thức truy hồi:

$$\text{Fib}(1) = 1$$

$$\text{Fib}(2) = 1$$

$$\text{Fib}(n) = \text{Fib}(n - 2) + \text{Fib}(n - 1)$$



Tính số Fibonacci

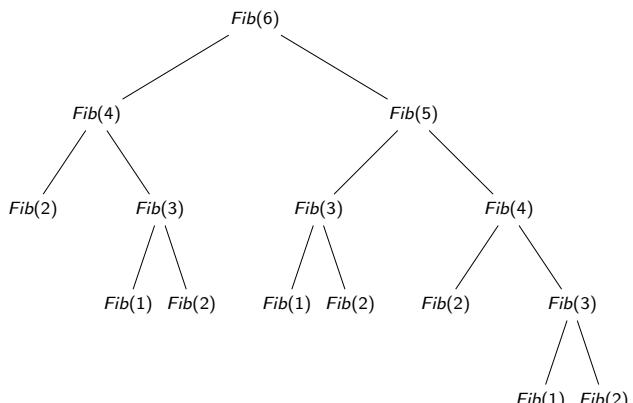
- 2. Cài đặt công thức qui hoạch động

```
1 int Fib(int n) {
2     if (n <= 2)
3         return 1;
4
5     int res = Fib(n - 2) + Fib(n - 1);
6
7     return res;
8 }
```



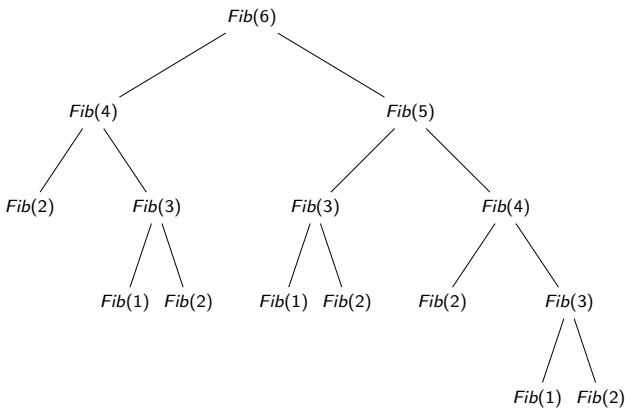
Tính số Fibonacci

- Độ phức tạp là bao nhiêu?



Tính số Fibonacci

- Độ phức tạp là bao nhiêu? Hàm mũ, gần như $\mathcal{O}(2^n)$



Tính số Fibonacci

- Lưu trữ kết quả các hàm đã tính

```
1 map<int, int> Mem;
2
3 int fibonacci(int n) {
4     if (n <= 2)
5         return 1;
6
7     if (Mem.find(n) != Mem.end())
8         return Mem[n];
9
10    int res = Fib(n - 2) + Fib(n - 1);
11
12    Mem[n] = res;
13    return res;
14 }
```


Dãy Fibonacci

```
1 int iMem[1001];
2 for (int i = 1; i <= 1000; i++)
3     iMem[i] = -1;
4
5 int Fib(int n) {
6     if (n <= 2)
7         return 1;
8     if (iMem[n] != -1)
9         return iMem[n];
10
11    int res = Fib(n - 2) + Fib(n - 1);
12    iMem[n] = res;
13    return res;
14 }
```

- Bây giờ độ phức tạp là bao nhiêu?



Tính số Fibonacci: Độ phức tạp

- Ta có n khả năng đầu vào input cho hàm đệ quy: 1, 2, ..., n .
- Với mỗi input:
 - hoặc là kết quả được tính và lưu trữ lại
 - hoặc là lấy luôn ra từ bộ nhớ nếu như trước đây đã được tính
- Mỗi input sẽ được tính tối đa một lần
- Thời gian tính là $\mathcal{O}(n \times f)$, với f là thời gian tính toán của hàm với một input, với giả thiết là kết quả đã tính trước đây sẽ được lấy trực tiếp từ bộ nhớ, chỉ trong $\mathcal{O}(1)$
- Do ta chỉ tồn một lượng hằng số phép tính đôi với một input của hàm, nên $f = \mathcal{O}(1)$
- Thời gian tính tổng cộng là $\mathcal{O}(n)$

1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

- Bài toán
- Công thức Qui hoạch động
- Cài đặt
- Độ phức tạp
- Truy vết

4 Đổi tiền

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất



7 Qui hoạch động trên bitmask

19 / 67

Đoạn con có tổng lớn nhất

- Cho một dãy số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong dãy có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

- Tổng của đoạn có trọng số lớn nhất trong dãy là 8



20 / 67

Đoạn con có tổng lớn nhất: Công thức sai

- Bước đầu tiên là đi tìm công thức Qui hoạch động
- Gọi $\text{MaxSum}(i)$ là trọng số của đoạn có trọng số lớn nhất giới hạn trong đoạn $1, \dots, i$
- Bước cơ sở: $\text{MaxSum}(1) = \max(0, A[1])$
- Bước chuyển qui nạp: $\text{MaxSum}(i)$ có liên hệ gì với $\text{MaxSum}(i - 1)$?
- Liệu có thể kết hợp lời giải của các bài toán con có kích thước bé hơn i thành lời giải bài toán có kích thước bằng i ?
- Câu trả lời không hoàn toàn hiển nhiên ...



21 / 67

Đoạn con có tổng lớn nhất

- Cho một dãy số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong dãy có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----



20 / 67

Đoạn con có tổng lớn nhất

- Cho một dãy số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm một đoạn trong dãy có trọng số lớn nhất, nghĩa là tổng các số trong đoạn là lớn nhất

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

- Tổng của đoạn có trọng số lớn nhất trong dãy là 8

Nhớ lại:

- Phương pháp Chia để trị cho độ phức tạp $\mathcal{O}(n \log n)$
- Liệu có thể làm tốt hơn với phương pháp Qui hoạch động?



20 / 67

Đoạn con có tổng lớn nhất: Công thức

Hãy thay đổi hàm mục tiêu:

- Gọi $\text{MaxSum}(i)$ là trọng số đoạn có trọng số lớn nhất giới hạn bởi $1, \dots, i$, mà phải kết thúc tại i
- Bước cơ sở: $\text{MaxSum}(1) = A[1]$
- Bước chuyển qui nạp:
$$\text{MaxSum}(i) = \max(A[i], A[i] + \text{MaxSum}(i - 1))$$
- Vậy kết quả cuối cùng chính là: $\max_{1 \leq i \leq n} \{ \text{MaxSum}(i) \}$



22 / 67

Đoạn con có tổng lớn nhất: Cài đặt

- Bước tiếp theo là cài đặt công thức Qui hoạch động

```
1 int A[1001];  
2  
3 int MaxSum(int i) {  
4     if (i == 1)  
5         return A[i];  
6  
7     int res = max(A[i], A[i] + MaxSum(i - 1));  
8     return res;  
9 }
```



Đoạn con có tổng lớn nhất: Cài đặt

- Bước tiếp theo là cài đặt công thức Qui hoạch động

```
1 int A[1001];  
2  
3 int MaxSum(int i) {  
4     if (i == 1)  
5         return A[i];  
6  
7     int res = max(A[i], A[i] + MaxSum(i - 1));  
8     return res;  
9 }
```

- Không cần sử dụng mảng nhớ Memory, vì sao?



Đoạn con có tổng lớn nhất: Cài đặt

- Bước tiếp theo là cài đặt công thức Qui hoạch động

```
1 int A[1001];  
2  
3 int MaxSum(int i) {  
4     if (i == 1)  
5         return A[i];  
6  
7     int res = max(A[i], A[i] + MaxSum(i - 1));  
8     return res;  
9 }
```

- Không cần sử dụng mảng nhớ Memory, vì sao?
- Nhưng vẫn cần mảng nhớ Memory phục vụ cho bước truy vết!



Đoạn con có tổng lớn nhất: Cài đặt

```
1 int A[1001];  
2 int iMem[1001];  
3 bool bMark[1001];  
4 memset(bMark, 0, sizeof(bMark));  
5  
6 int MaxSum(int i) {  
7     if (i == 1)  
8         return A[i];  
9     if (bMark[i])  
10        return iMem[i];  
11  
12     int res = max(A[i], A[i] + MaxSum(i - 1));  
13     iMem[i] = res;  
14     bMark[i] = true;  
15     return res;  
16 }
```



Đoạn con có tổng lớn nhất: Kết quả

- Thủ tục chính chỉ cần gọi đệ qui một lần cho $\text{MaxSum}(n)$, hàm đệ qui sẽ tính toàn bộ các giá trị của $\text{MaxSum}(i), 1 \leq i \leq n$
- Kết quả bài toán là giá trị lớn nhất trong các giá trị $\text{MaxSum}(i)$ đã được lưu trữ trong $iMem[i]$ sau quá trình gọi đệ qui

```
1 int ans = 0;  
2 for (int i = 0; i < n; i++) {  
3     ans = max(ans, iMem[i]);  
4 }  
5 cout << ans;
```

- Nếu bài toán yêu cầu tìm đoạn có trọng số lớn nhất trong nhiều dãy khác nhau, thì hãy nhớ xóa bộ nhớ khi kết thúc tính toán ở mỗi mảng



Đoạn con có tổng lớn nhất: Độ phức tạp

- Có n khả năng đầu vào input cho hàm đệ qui
- Mỗi input được tính trong $\mathcal{O}(1)$
- Thời gian tính toán tổng cộng là $\mathcal{O}(n)$



Đoạn con có tổng lớn nhất: Độ phức tạp

- Có n khả năng đầu vào input cho hàm đệ qui
- Mỗi input được tính trong $\mathcal{O}(1)$
- Thời gian tính toán tổng cộng là $\mathcal{O}(n)$
- Làm thế nào để biết chính xác một đoạn nào trong dãy tạo ra giá trị tổng lớn nhất tìm được?

Đoạn con có tổng lớn nhất: Truy vết bằng đệ qui

- Làm giống như hàm đệ qui chính và sử dụng mảng iMem đã có để truy vết ngược lại

```
1 void Trace(int i) {
2     if (i != 1 && iMem[i] == A[i] + iMem[i-1]) {
3         Trace(i - 1);
4     }
5     cout << A[i] << " ";
6 }
```

- Độ phức tạp hàm truy vết? $\mathcal{O}(n)$

1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

4 Đổi tiền

- Bài toán
- Công thức Qui hoạch động
- Cài đặt
- Độ phức tạp
- Truy vết

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất

Đoạn con có tổng lớn nhất: Truy vết bằng đệ qui

- Làm giống như hàm đệ qui chính và sử dụng mảng iMem đã có để truy vết ngược lại

```
1 void Trace(int i) {
2     if (i != 1 && iMem[i] == A[i] + iMem[i-1]) {
3         Trace(i - 1);
4     }
5     cout << A[i] << " ";
6 }
```

- Độ phức tạp hàm truy vết?

Đoạn con có tổng lớn nhất: Truy vết bằng vòng lặp

```
1 int ans = 0, pos = -1;
2 for (int i = 0; i < n; i++) {
3     ans = max(res, iMem[i]);
4     if (ans == iMem[i]) pos = i;
5 }
6
7 cout << ans << endl;
8 int first = pos, last = pos, sum = A[first];
9 while (sum != res){
10     --first;
11     sum += A[first];
12 }
13 cout << first << " " << last;
```

Đổi tiền

- Cho trước một tập các đồng tiền mệnh giá D_1, D_2, \dots, D_n , và một mệnh giá x . Hãy tìm số lượng ít nhất các đồng tiền để đổi cho mệnh giá x ?
- Giống bài toán cái túi?
- Tồn tại thuật toán tham lam cho bài Đổi tiền này?

Đổi tiền

- Cho trước một tập các đồng tiền mệnh giá D_1, D_2, \dots, D_n , và một mệnh giá x . Hãy tìm số lượng ít nhất các đồng tiền để đổi cho mệnh giá x ?
- Giống bài toán cái túi?
- Tồn tại thuật toán tham lam cho bài Đổi tiền này?
- Bài toán cái túi đã học trong môn Toán rắc rối được giải bằng thuật toán duyệt nhánh cận. Còn thuật toán tham lam không hề chắc chắn đưa ra lời giải tối ưu, thậm chí nhiều trường hợp còn không đưa ra được lời giải...
- Hãy thử sử dụng phương pháp Qui hoạch động!
- Cuối cùng đưa ra nhận xét giữa các phương pháp khác nhau tiếp cận giải bài toán này



30 / 67

Đổi tiền: Cài đặt

```
1 int INF = 100000;
2 int D[11];
3
4 int MinCoin(int i, int x) {
5     if (x < 0) return INF;
6     if (x == 0) return 0;
7     if (i == 0) return INF;
8
9     int res = INF;
10    res = min(res, 1 + MinCoin(i, x - D[i]));
11    res = min(res, MinCoin(i - 1, x));
12
13    return res;
14 }
```



32 / 67

Đổi tiền: Độ phức tạp

- Độ phức tạp?



34 / 67

Đổi tiền: Công thức Qui hoạch động

Bước đầu tiên: xây dựng công thức Qui hoạch động

- Gọi $\text{MinCoin}(i, x)$ là số lượng tiền ít nhất cần để đổi mệnh giá x nếu chỉ được phép sử dụng các đồng tiền mệnh giá D_1, \dots, D_i

Các bước cơ sở:

- $\text{MinCoin}(i, x) = \infty$ nếu $x < 0$
- $\text{MinCoin}(i, 0) = 0$
- $\text{MinCoin}(0, x) = \infty$

Bước chuyển qui nạp:

$$\text{MinCoin}(i, x) = \min \begin{cases} 1 + \text{MinCoin}(i, x - D_i) \\ \text{MinCoin}(i - 1, x) \end{cases}$$



31 / 67

Đổi tiền: Cài đặt

```
1 int INF = 100000;
2 int D[11];
3 int iMem[11][10001];
4 memset(iMem, -1, sizeof(iMem));
5
6 int MinCoin(int i, int x) {
7     if (x < 0) return INF;
8     if (x == 0) return 0;
9     if (i == 0) return INF;
10
11    if (iMem[i][x] != -1) return iMem[i][x];
12    int res = INF;
13    res = min(res, 1 + MinCoin(i, x - D[i]));
14    res = min(res, MinCoin(i - 1, x));
15    iMem[i][x] = res;
16    return res;
17 }
```



33 / 67

Đổi tiền: Độ phức tạp

- Độ phức tạp?
- Số lượng khả năng đầu vào input là $n \times x$
- Mỗi input được xử lý trong $\mathcal{O}(1)$, giả thiết mỗi lời gọi đệ quy thực hiện trong thời gian hằng số
- Thời gian tính toán tổng cộng là $\mathcal{O}(n \times x)$



34 / 67

Dổi tiền: Độ phức tạp

- Độ phức tạp?
- Số lượng khả năng đầu vào input là $n \times x$
- Mỗi input được xử lý trong $\mathcal{O}(1)$, giả thiết mỗi lời gọi đệ qui thực hiện trong thời gian hằng số
- Thời gian tính toán tổng cộng là $\mathcal{O}(n \times x)$
- Làm thế nào để xác định được những đồng tiền nào cho phương án tối ưu?
- Hãy truy vết ngược lại quá trình đệ qui

Dổi tiền: Truy vết bằng đệ qui

```
1 void Trace(int i, int x) {  
2     if (x < 0) return;  
3     if (x == 0) return;  
4     if (i == 0) return;  
5  
6     int res = INF;  
7     if (iMem[i][x] == 1 + iMem[i][x - D[i]]){  
8         cout << D[i] << " ";  
9         Trace(i, x - D[i]);  
10    } else {  
11        Trace(i-1, x);  
12    }  
13 }
```

- Gọi `Trace(n, x);`
- Độ phức tạp hàm truy vết? $\mathcal{O}(\max(n, x))$

1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

4 Đổi tiền

5 Dãy con tăng dài nhất

- Bài toán
- Công thức Qui hoạch động
- Cài đặt
- Độ phức tạp
- Truy vết

6 Dãy con chung dài nhất

Dổi tiền: Truy vết bằng đệ qui

```
1 void Trace(int i, int x) {  
2     if (x < 0) return;  
3     if (x == 0) return;  
4     if (i == 0) return;  
5  
6     int res = INF;  
7     if (iMem[i][x] == 1 + iMem[i][x - D[i]]){  
8         cout << D[i] << " ";  
9         Trace(i, x - D[i]);  
10    } else {  
11        Trace(i-1, x);  
12    }  
13 }
```

- Gọi `Trace(n, x);`
- Độ phức tạp hàm truy vết?

Dổi tiền: Truy vết bằng vòng lặp

```
1 int ans = iMem[n][x];  
2 cout << ans << endl;  
3 for (int i = n, k = 0; k < ans; ++k) {  
4     if (iMem[i][x] == 1 + iMem[i][x-D[i]]){  
5         cout << D[i] << " ";  
6         x -= D[i];  
7     } else {  
8         --i;  
9     }  
10 }
```

Dãy con tăng dài nhất

- Cho một dãy n số nguyên $A[1], A[2], \dots, A[n]$, hãy tìm độ dài của dãy con tăng dài nhất?
- Định nghĩa: Nếu xoá đi 0 phần tử hoặc một số phần tử của dãy A thì sẽ thu được một dãy con của A
- Ví dụ: $a = [2, 0, 6, 1, 2, 9]$
- $[2, 6, 9]$ là một dãy con
- $[2, 2]$ là một dãy con
- $[2, 0, 6, 1, 2, 9]$ là một dãy con
- $[]$ là một dãy con
- $[9, 0]$ không là một dãy con
- $[7]$ không là một dãy con

Dãy con tăng dài nhất

- Một dãy con tăng của A là một dãy con của A sao cho các phần tử là tăng chẵn từ trái sang phải
- [2, 6, 9] và [1, 2, 9] là hai dãy con tăng của $A = [2, 0, 6, 1, 2, 9]$
- Làm thế nào để tính độ dài dãy con tăng dài nhất?
- Có 2^n dãy con, phương pháp đơn giản nhất là duyệt qua toàn bộ các dãy này
- Thuật toán cho độ phức tạp $\mathcal{O}(n \times 2^n)$, chỉ có thể chạy nhanh được ra kết quả với $n \leq 23$
- Hãy thử phương pháp Qui hoạch động !



Dãy con tăng dài nhất: Công thức Qui hoạch động

- Gọi $LIS(i)$ là độ dài dãy con tăng dài nhất của mảng $A[1], \dots, a[i]$
- Bước cơ sở: $LIS(1) = 1$
- Bước chuyển qui nạp cho $LIS(i)$?



Dãy con tăng dài nhất: Công thức Qui hoạch động

- Gọi $LIS(i)$ là độ dài dãy con tăng dài nhất của mảng $A[1], \dots, a[i]$
 - Bước cơ sở: $LIS(1) = 1$
 - Bước chuyển qui nạp cho $LIS(i)$?
- Nếu đặt hàm mục tiêu như vậy sẽ gặp phải vấn đề giống như bài toán dãy con có tổng lớn nhất ở trên, hãy thay đổi một chút hàm mục tiêu



Dãy con tăng dài nhất: Công thức Qui hoạch động

- Gọi $LIS(i)$ là độ dài dãy con tăng dài nhất của mảng $A[1], \dots, A[i]$, mà kết thúc tại i
- Bước cơ sở: không cần thiết
- Bước chuyển qui nạp:
 $LIS(i) = \max(1, \max_{j \text{ s.t. } A[j] < A[i]} \{1 + LIS(j)\})$



Dãy con tăng dài nhất: Cài đặt

```
1 int A[1001];
2 int iMem[1001];
3 memset(iMem, -1, sizeof(iMem));
4
5 int LIS(int i) {
6     if (iMem[i] != -1)
7         return iMem[i];
8
9     int res = 1;
10    for (int j = 1; j < i; ++j) {
11        if (A[j] < A[i]) {
12            res = max(res, 1 + LIS(j));
13        }
14    }
15    iMem[i] = res;
16    return res;
17}
```



Dãy con tăng dài nhất: Cài đặt

- Độ dài dãy con tăng dài nhất chính là giá trị lớn nhất trong các giá trị $LIS(i)$:

```
1 int ans = 0, pos = 0;
2 for (int i = 1; i <= n; i++) {
3     ans = max(ans, iMem[i]);
4     if (ans == iMem[i]) pos = i;
5 }
6
7 cout << ans;
```



Dãy con tăng dài nhất: Độ phức tạp tính toán

- Có n khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(n)$
- Thời gian tính tổng cộng là $\mathcal{O}(n^2)$
- Có thể chạy được đến $n \leq 10\,000$, tốt hơn rất nhiều so với phương pháp duyệt toàn bộ!
- Áp dụng cấu trúc cây phân đoạn vào phương pháp trên sẽ cải tiến độ phức tạp thành $\mathcal{O}(n \log n)$
- Phương pháp cải tiến khác là đặt công thức Qui hoạch động mới kết hợp với phương pháp chặt nhị phân cũng cho độ phức tạp $\mathcal{O}(n \log n)$
- Truy vết?

Dãy con tăng dài nhất: Truy vết bằng đệ qui

```
1 void Trace(int i) {  
2     int res = 1;  
3     for (int j = 1; j < i; j++) {  
4         if (A[j] < A[i] && iMem[i] == 1 + iMem[j]) {  
5             Trace(j);  
6             break;  
7         }  
8     }  
9     cout << i << " ";  
10}
```

- Gọi `Trace(pos);`
- Độ phức tạp hàm truy vết? vẫn là $\mathcal{O}(n^2)$
- Có thể cải tiến thành $\mathcal{O}(n)$ bằng cách sử dụng một mảng nhớ lưu tại mỗi vị trí i vị trí j làm tạo ra giá trị $\max(LIS(i))$ để giảm bớt vòng lặp `for` trong hàm truy vết này

Dãy con tăng dài nhất: Truy vết bằng đệ qui cải tiến

```
1 void Trace(int i) {  
2     int res = 1;  
3     for (int j = i-1; j >= 1; j++) {  
4         if (A[j] < A[i] && iMem[i] == 1 + iMem[j]) {  
5             Trace(j);  
6             break;  
7         }  
8     }  
9     cout << i << " ";  
10}
```

- Gọi `Trace(pos);`
- Độ phức tạp hàm truy vết? $\mathcal{O}(n)$

Dãy con tăng dài nhất: Truy vết bằng đệ qui

```
1 void Trace(int i) {  
2     int res = 1;  
3     for (int j = 1; j < i; j++) {  
4         if (A[j] < A[i] && iMem[i] == 1 + iMem[j]) {  
5             Trace(j);  
6             break;  
7         }  
8     }  
9     cout << i << " ";  
10}
```

- Gọi `Trace(pos);`
- Độ phức tạp hàm truy vết?

Dãy con tăng dài nhất: Truy vết bằng đệ qui cải tiến

```
1 void Trace(int i) {  
2     int res = 1;  
3     for (int j = i-1; j >= 1; j++) {  
4         if (A[j] < A[i] && iMem[i] == 1 + iMem[j]) {  
5             Trace(j);  
6             break;  
7         }  
8     }  
9     cout << i << " ";  
10}
```

- Gọi `Trace(pos);`
- Độ phức tạp hàm truy vết?

Dãy con tăng dài nhất: Truy vết bằng vòng lặp

```
1 stack<int> S;  
2 for (int i = pos, k = 0; k < ans; ++k) {  
3     S.push(i);  
4     for (int j = 1; j < i; ++j){  
5         if (A[j] < A[i] && iMem[j]+1 == iMem[i]) {  
6             i = j;  
7             break;  
8         }  
9     }  
10    while (!S.empty()){  
11        cout << S.back() << " ";  
12        S.pop();  
13    }  
14}
```

1 Sơ đồ Qui hoạch động

2 Tính số Fibonacci

3 Đoạn con có tổng lớn nhất

4 Đổi tiền

5 Dãy con tăng dài nhất

6 Dãy con chung dài nhất

- Bài toán
- Công thức Qui hoạch động
- Cài đặt
- Độ phức tạp
- Truy vết

Dãy con chung dài nhất: Công thức Qui hoạch động

- Gọi $LCS(i, j)$ là độ dài dãy con chung dài nhất của $X[1], \dots, X[i]$ và $Y[1], \dots, Y[j]$

- Bước cơ sở:

- $LCS(0, j) = 0$
- Bước cơ sở: $LCS(i, 0) = 0$

- Bước chuyển qui nạp:

$$LCS(i, j) = \max \begin{cases} LCS(i, j - 1) \\ LCS(i - 1, j) \\ 1 + LCS(i - 1, j - 1) & \text{nếu } X[i] = Y[j] \end{cases}$$

Dãy con chung dài nhất: Ví dụ

iMem	j	0	1	2	3	4	5
i		$Y[j]$	<u>b</u>	<u>d</u>	<u>c</u>	<u>a</u>	<u>b</u>
0	$X[i]$	0	0	0	0	0	0
1	<u>a</u>	0	0	0	0	1	1
2	<u>b</u>	0	1 \rightarrow	1 \rightarrow	1	1	2
3	<u>c</u>	0	1	1	2 \rightarrow	2 \rightarrow	2
4	<u>b</u>	0	1	1	2	2	3

$$LCS(i, j) = \max \begin{cases} LCS(i, j - 1) \\ LCS(i - 1, j) \\ 1 + LCS(i - 1, j - 1) & \text{nếu } X[i] = Y[j] \end{cases}$$

Dãy con chung dài nhất

- Cho hai xâu (hoặc hai mảng số nguyên) n phần tử $X[1], \dots, X[n]$ và $Y[1], \dots, Y[m]$, hãy tìm độ dài của dãy con chung dài nhất của hai xâu

- $X = "abcb"$

- $Y = "bdcab"$

- Dãy con chung dài nhất của X và Y , "bcb", có độ dài 3

Dãy con chung dài nhất: Cài đặt

```
1 string X = "abcb",
2         Y = "bdcab";
3 int iMem[1001][1001];
4 memset(iMem, -1, sizeof(iMem));
5
6 int LCS(int i, int j) {
7     if (i == 0 || j == 0) return 0;
8     if (iMem[i][j] != -1) return iMem[i][j];
9
10    int res = 0;
11    res = max(res, LCS(i, j - 1));
12    res = max(res, LCS(i - 1, j));
13    if (X[i] == Y[j]) {
14        res = max(res, 1 + LCS(i - 1, j - 1));
15    }
16    iMem[i][j] = res;
17    return res;
18 }
```

Dãy con chung dài nhất: Độ phức tạp tính toán

- Có n khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(1)$
- Thời gian tính tổng cộng là $\mathcal{O}(n \times m)$

Dãy con chung dài nhất: Độ phức tạp tính toán

- Có n khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(1)$
- Thời gian tính tổng cộng là $\mathcal{O}(n \times m)$
- Làm thế nào để biết chính xác những phần tử nào thuộc dãy con chung dài nhất?

Dãy con chung dài nhất: Truy vết bằng đệ qui

```
1 void Trace(int i, int j) {
2     if (i == 0 || j == 0) return;
3
4     if (iMem[i][j] == iMem[i-1][j]) {
5         Trace(i-1, j);
6         return;
7     }
8     if (iMem[i][j] == iMem[i][j-1]) {
9         Trace(i, j-1);
10    return;
11}
12 if (X[i] == Y[j] && iMem[i][j] == 1 + iMem[i-1][j-1]) {
13     Trace(i-1, j-1);
14     cout << A[i] << " ";
15 }
16 }
```

- Độ phức tạp hàm truy vết?



Dãy con chung dài nhất: Truy vết bằng đệ qui

```
1 void Trace(int i, int j) {
2     if (i == 0 || j == 0) return;
3
4     if (iMem[i][j] == iMem[i-1][j]) {
5         Trace(i-1, j);
6         return;
7     }
8     if (iMem[i][j] == iMem[i][j-1]) {
9         Trace(i, j-1);
10    return;
11}
12 if (X[i] == Y[j] && iMem[i][j] == 1 + iMem[i-1][j-1]) {
13     Trace(i-1, j-1);
14     cout << A[i] << " ";
15 }
16 }
```

- Độ phức tạp hàm truy vết? $\mathcal{O}(n + m)$

Dãy con chung dài nhất - Truy vết bằng vòng lặp

```
1 int ans = iMem[n][m];
2 cout << ans << endl;
3 stack<int> S;
4 for (int i = n, j = m, k = 0; k < ans; ++k) {
5     if (X[i] == Y[j] && iMem[i][j] == 1 + iMem[i-1][j-1]){
6         S.push(X[i]);
7         --i; --j; continue;
8     }
9     if (iMem[i][j] == iMem[i-1][j]){
10        --i; continue;
11    }
12     if (iMem[i][j] == iMem[i][j-1]){
13        --j; continue;
14    }
15 }
16 while (!S.empty()) {
17     cout << S.back() << " ";
18     S.pop();
19 }
```



7 Qui hoạch động trên bitmask

- Bài toán người du lịch
- Công thức Qui hoạch động
- Cài đặt
- Độ phức tạp
- Truy vết

Qui 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 $0, \dots, 2^n - 1$
- Điều này có thể giúp thực hiện phương pháp qui hoạch động dễ dàng trên các tập con



Bài toán người du lịch



Applying the Traveling Salesman Problem to Business Analytics

Published on April 2, 2019

"The history and evolution of solutions to the Traveling Salesman Problem can provide us with some valuable concepts for business analytics and algorithm development"

"Just as the traveling salesman makes his journey, new analytical requirements arise that require a journey into the development of solutions for them. As the data science and business analytics landscape evolves with new solutions, we can learn from the history of these journeys and apply the same concepts to our ongoing development"



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán người du lịch (hay người bán hàng)



Applying the Traveling Salesman Problem to Business Analytics

Published on April 2, 2019

Bài toán người du lịch là bài toán NP-khó kinh điển nhưng có rất nhiều ứng dụng trong thực tế, đặc biệt ngày nay với ứng dụng của khoa học dữ liệu và phân tích tài chính.

Mỗi khi một hành trình của người bán hàng kết thúc, các dữ liệu sẽ được phân tích bởi các thuật toán trong khoa học dữ liệu, áp dụng vào ngành phân tích tài chính, từ đó có thể 'học máy' các kết quả lịch sử để áp dụng vào kế hoạch phát triển tiếp theo.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán người du lịch

- Cho một đồ thị n đỉnh $\{0, 1, \dots, n - 1\}$ 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 $O(n!)$, nhưng chỉ có thể chạy được đến $n \leq 11$
- Liệu có thể làm tốt hơn với phương pháp Qui hoạch động?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

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 }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán người du lịch: Công thức Qui hoạch động

- Không mất tính tổng quát giả sử chu trình bắt đầu và kết thúc tại đỉnh 0
- Gọi $TSP(i, S)$ cách sử dụng ít chi phí 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: $TSP(i, \text{tập mọi đỉnh}) = C_{i,0}$
- Bước chuyển qui nạp: $TSP(i, S) = \min_{j \notin S} \{ C_{i,j} + \text{tsp}(j, S \cup \{j\}) \}$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán người du lịch: Cài đặt

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

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



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

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

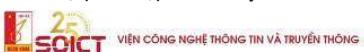
- Có $n \times 2^n$ khả năng cho đầu vào input
- Mỗi input được tính trong thời gian $\mathcal{O}(n)$
- Thời gian tính tổng cộng là $\mathcal{O}(n^2 \times 2^n)$
- Như vậy có thể tính nhanh được với n lên đến 20
- Làm thế nào để đưa ra được chính xác hành trình của người du lịch?



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}
```

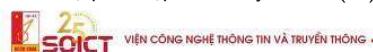
- Gọi TSP(0, 1<0);
- Độ phức tạp hàm truy vết?



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

```
1 int ans = iMem[0][1];  
2 cout << ans << endl;  
3 stack<int> Stack;  
4 Stack.push(0);  
5 for (int i = 0, S = 1, k = 0; k < n-1; ++k) {  
6     for (int j = 0; j < n; ++j){  
7         if ((S & (1 << j)) &&  
8             (iMem[i][S] == C[i][j] + iMem[j][S | (1 << j)])) {  
9                 Stack.push(j);  
10                i = j;  
11                S = S | (1 << j);  
12            }  
13        }  
14    }  
15    while (!Stack.empty()) {  
16        cout << Stack.back() << " ";  
17        Stack.pop();  
18    }
```



Thank you for
your attentions!





25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



25
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật Toán Cơ Bản Trên

Đồ Thị Không Trọng Sô

THUẬT TOÁN ỨNG DỤNG

1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

1 Cơ bản về đồ thị

- Khái niệm
- Biểu diễn đồ thị
- Một số tính chất cơ bản

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

Đồ thị là gì?

Đồ thị là gì?

1 Đỉnh

- Giao cửa các con đường
- Máy tính
- Các sàn trong nhà
- Sân bay
- Các đối tượng

1

2

3

4



25
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

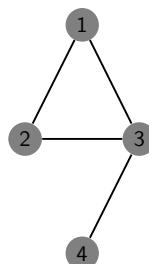


5 / 55

Đồ thị là gì?

Dịnh

- Giao của các con đường
- Máy tính
- Các sàn trong nhà
- Sân bay
- Các đối tượng

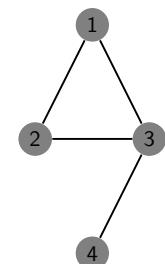


Cạnh

- Con đường
- Dây mạng
- Thang bộ và thang máy
- Đường bay trực tiếp
- Mối quan hệ giữa các đối tượng

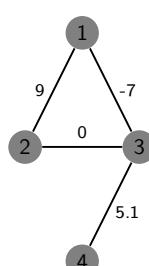
Các loại cạnh

Không có trọng số



Các loại cạnh

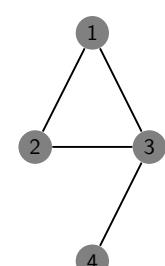
Không có trọng số hoặc Có trọng số



Các loại cạnh

Không có trọng số hoặc Có trọng số

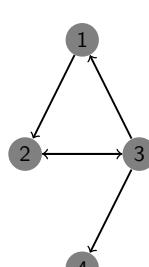
Vô hướng



Các loại cạnh

Không có trọng số hoặc Có trọng số

Vô hướng hoặc Có hướng

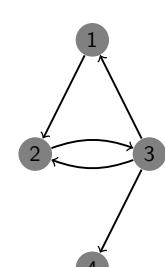


Các loại cạnh

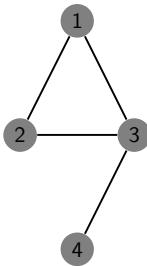
Không có trọng số hoặc Có trọng số

Vô hướng hoặc Có hướng

- Trong trường hợp đồ thị có hướng, cạnh có hướng thường được gọi là cung chỉ tính định hướng của cung giữa hai đỉnh đầu mút

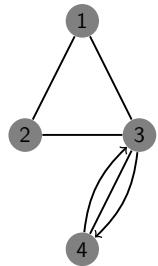


Đa đồ thị



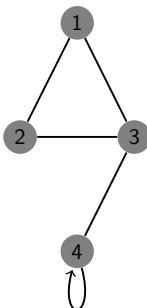
Đa đồ thị

- Cạnh lặp



Đa đồ thị

- Cạnh lặp
- Khuỷu



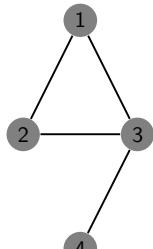
- 1 Cơ bản về đồ thị
 - Khái niệm
 - Biểu diễn đồ thị
 - Một số tính chất cơ bản

- 2 Tìm kiếm theo chiều sâu và ứng dụng - DFS
- 3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

Danh sách kề

```
1: 2, 3  
2: 1, 3  
3: 1, 2, 4  
4: 3
```

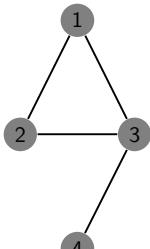
```
vector<int> Adj[5];  
Adj[1].push_back(2);  
Adj[1].push_back(3);  
Adj[2].push_back(1);  
Adj[2].push_back(3);  
Adj[3].push_back(1);  
Adj[3].push_back(2);  
Adj[3].push_back(4);  
Adj[4].push_back(3);
```



Ma trận kề

```
0 1 1 0  
1 0 1 0  
1 1 0 1  
0 0 1 0
```

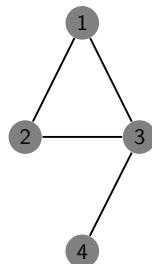
```
bool Adj[5][5];  
Adj[1][2] = true;  
Adj[1][3] = true;  
Adj[2][1] = true;  
Adj[2][3] = true;  
Adj[3][1] = true;  
Adj[3][2] = true;  
Adj[3][4] = true;  
Adj[4][3] = true;
```



Danh sách cạnh

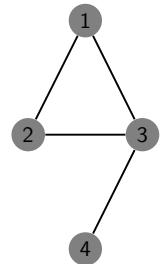
(1, 2)
(1, 3)
(2, 3)
(3, 4)

```
vector<pair<int, int>> Edges;
Edges.push_back(make_pair(1, 2));
Edges.push_back(make_pair(1, 3));
Edges.push_back(make_pair(2, 3));
Edges.push_back(make_pair(3, 4));
```



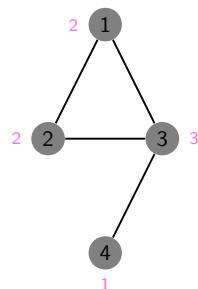
Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh v, ký hiệu $\text{Deg}(v)$:
 - Số lượng cạnh kề với v
 - Số lượng đỉnh kề với v



Một số tính chất trên đỉnh (đồ thị vô hướng)

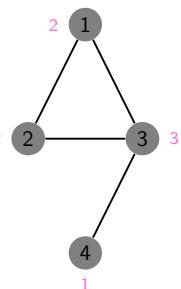
- Bậc của một đỉnh v, ký hiệu $\text{Deg}(v)$:
 - Số lượng cạnh kề với v
 - Số lượng đỉnh kề với v



Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh v, ký hiệu $\text{Deg}(v)$:
 - Số lượng cạnh kề với v
 - Số lượng đỉnh kề với v
- Bổ đề về những cái bắt tay (Handsaking theorem)

$$\sum_{v \in V} \text{Deg}(v) = 2|E|$$

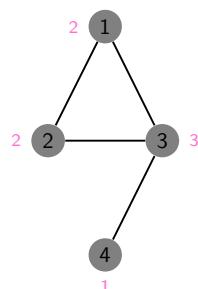


Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh v, ký hiệu $\text{Deg}(v)$:
 - Số lượng cạnh kề với v
 - Số lượng đỉnh kề với v
- Bổ đề về những cái bắt tay (Handsaking theorem)

$$\sum_{v \in V} \text{Deg}(v) = 2|E|$$

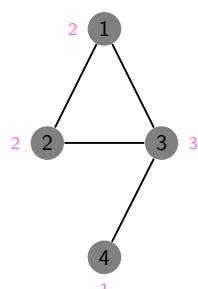
$$2 + 2 + 3 + 1 = 2 \times 4$$



Một số tính chất trên đỉnh (đồ thị vô hướng)

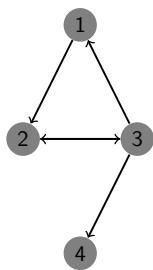
1: 2, 3
2: 1, 3
3: 1, 2, 4
4: 3

`Adj[1].size() // 2`
`Adj[2].size() // 2`
`Adj[3].size() // 3`
`Adj[4].size() // 1`



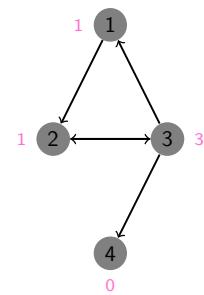
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh



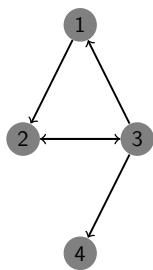
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh



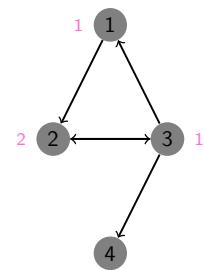
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
 - ▶ Số lượng cạnh đi vào đỉnh



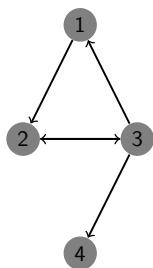
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
 - ▶ Số lượng cạnh đi vào đỉnh



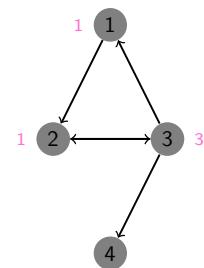
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
 - ▶ Số lượng cạnh đi vào đỉnh



Một số tính chất trên đỉnh (đồ thị có hướng)

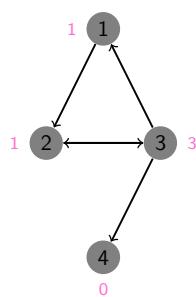
- Bán bậc ra của một đỉnh
 - ▶ Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
 - ▶ Số lượng cạnh đi vào đỉnh



Danh sách kề (có hướng)

1: 2
2: 3
3: 1, 2, 4
4:

```
Adj[1].size() // 1
Adj[2].size() // 1
Adj[3].size() // 3
Adj[4].size() // 0
```

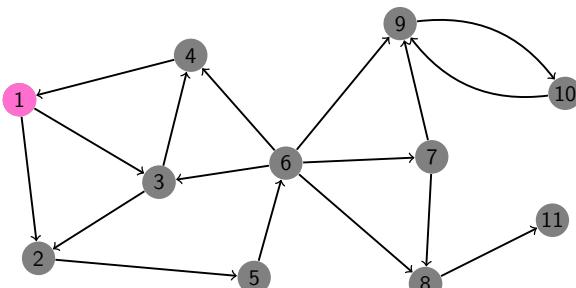


Tìm kiếm theo chiều sâu - DFS

- Cho đồ thị $G = (V, E)$ (có hướng hoặc vô hướng) và hai đỉnh u và v , hỏi có tồn tại một đường đi từ u đến v ?
- Thuật toán tìm kiếm theo chiều sâu đưa ra được đường đi như vậy nếu tồn tại
- Thuật toán duyệt đồ thị ưu tiên theo chiều sâu (LIFO - Last In First Out), bắt đầu từ đỉnh xuất phát u
- Thực chất ta không cần chỉ rõ đỉnh v , vì ta có thể để thuật toán thăm tất cả các đỉnh có thể đến được từ u (mà vẫn cùng độ phức tạp)
- Vậy độ phức tạp của thuật toán là bao nhiêu?
- Mỗi đỉnh được thăm đúng một lần, và mỗi cạnh cũng được duyệt qua đúng một lần (nếu đến được)
- $O(n + m)$



Tìm kiếm theo chiều sâu: Ví dụ



Stack:	1
Visited	1 F F F F F F F F F F



1 Cơ bản về đồ thị

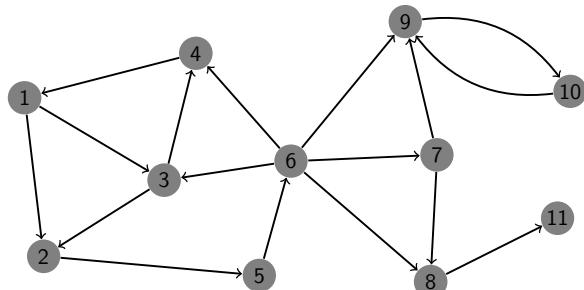
2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS



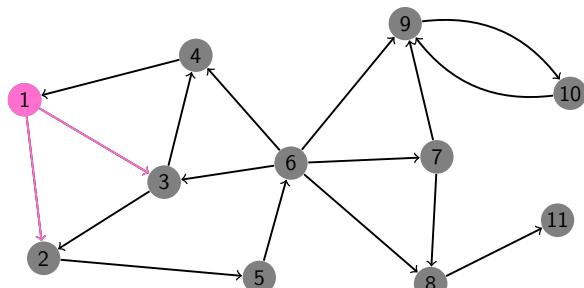
Tìm kiếm theo chiều sâu: Ví dụ



Stack:	
Visited	1 F F F F F F F F F F



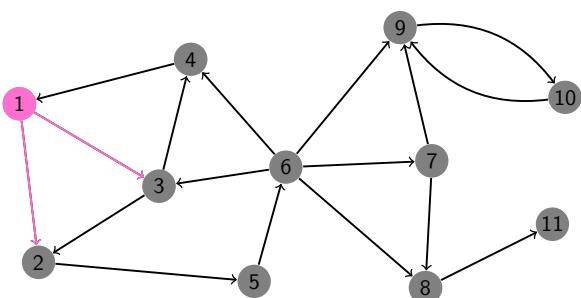
Tìm kiếm theo chiều sâu: Ví dụ



Stack:	1
Visited	1 F F F F F F F F F F

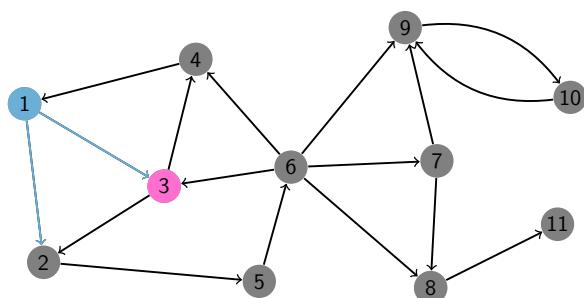


Tìm kiếm theo chiều sâu: Ví dụ



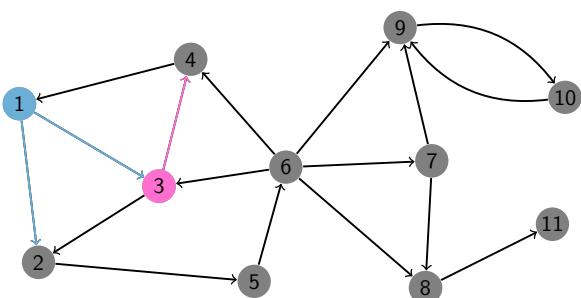
Stack: 1 | 3 2
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ



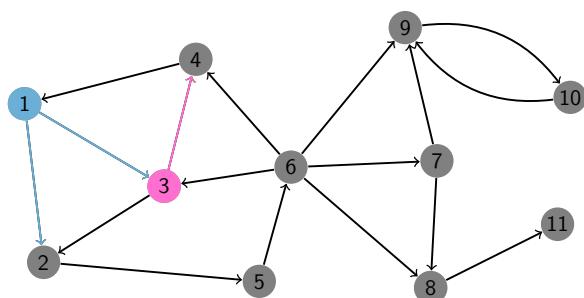
Stack: 3 | 2
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ



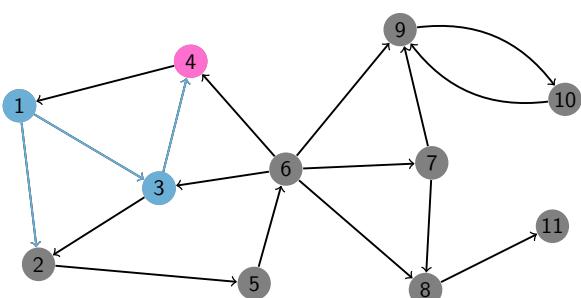
Stack: 3 | 2
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ



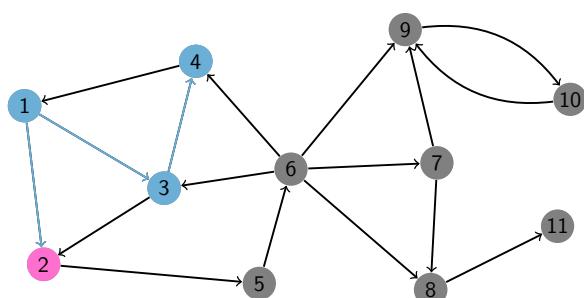
Stack: 3 | 4 2
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ



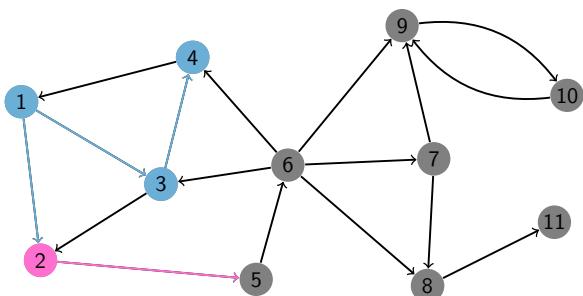
Stack: 4 | 2
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ



Stack: 2 |
 Visited | 1 2 3 4 5 6 7 8 9 10 11
 T F F F F F F F F F F

Tìm kiếm theo chiều sâu: Ví dụ

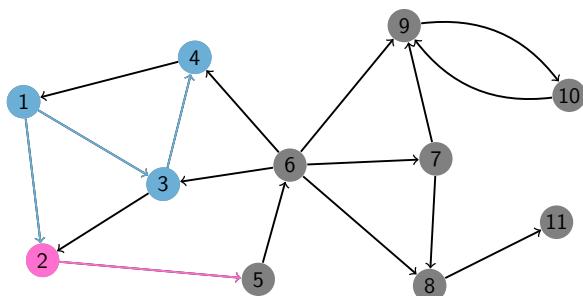


Stack: 2 |

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	F	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

Tìm kiếm theo chiều sâu: Ví dụ

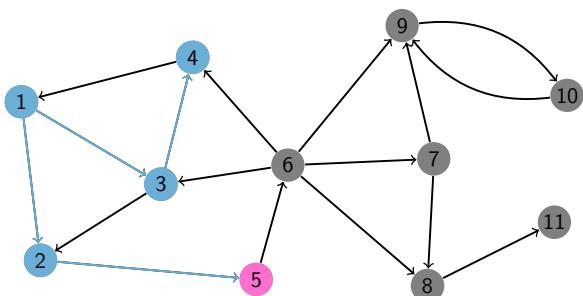


Stack: 2 | 5

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	T	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

Tìm kiếm theo chiều sâu: Ví dụ

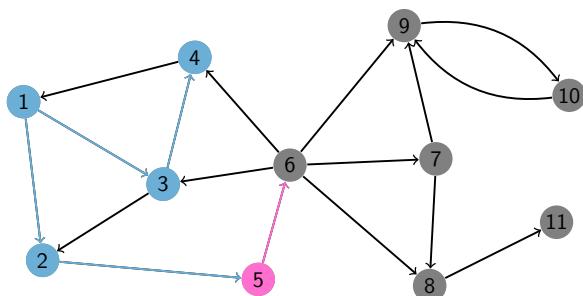


Stack: 5 |

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	T	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

Tìm kiếm theo chiều sâu: Ví dụ

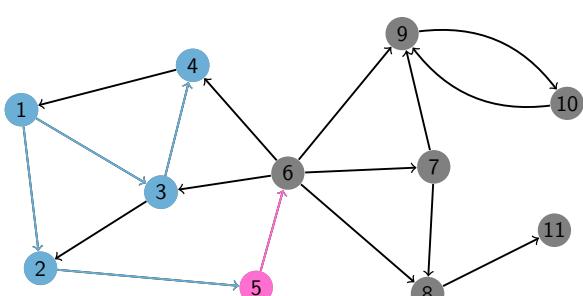


Stack: 5 |

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	T	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

Tìm kiếm theo chiều sâu: Ví dụ

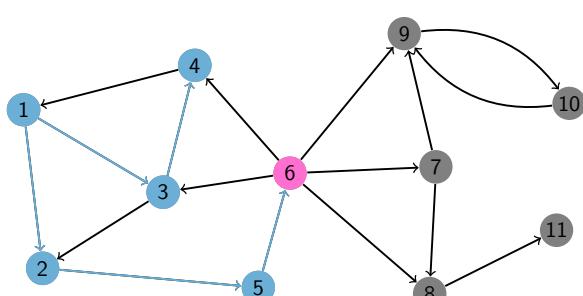


Stack: 5 | 6

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	T	T	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

Tìm kiếm theo chiều sâu: Ví dụ

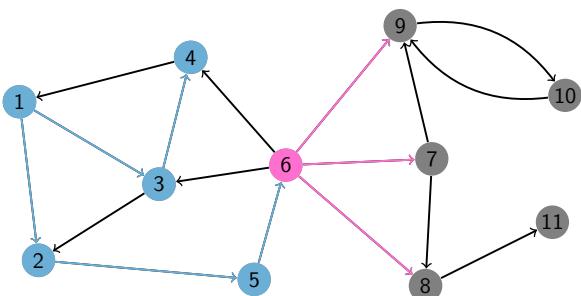


Stack: 6 |

	1	2	3	4	5	6	7	8	9	10	11
--	---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	T	T	T	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---	---

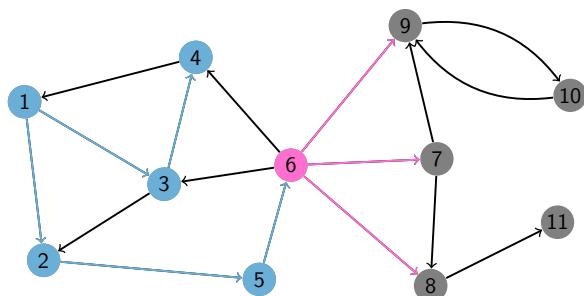
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 6 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	F	F	F	F	F	F

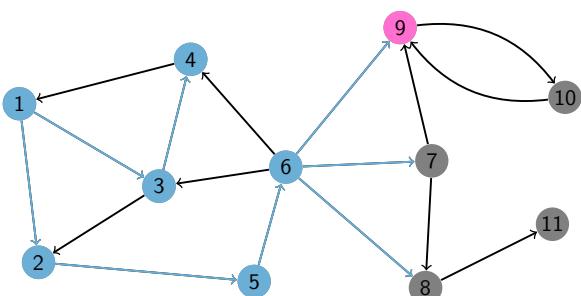
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 6 | 9 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	F	F	F

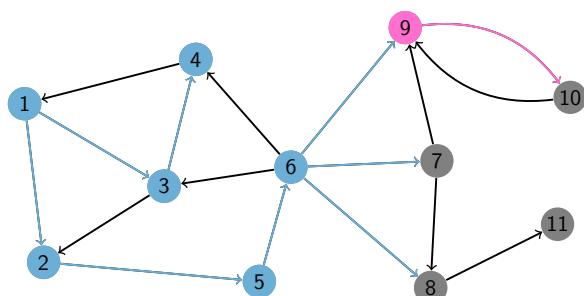
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 9 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	F	F	F

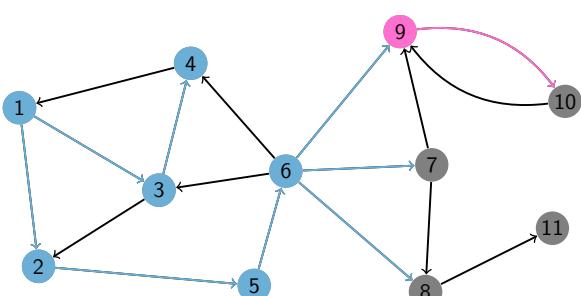
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 9 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	F	F	F

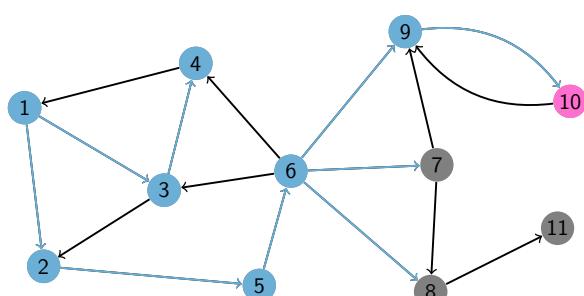
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 9 | 10 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	F

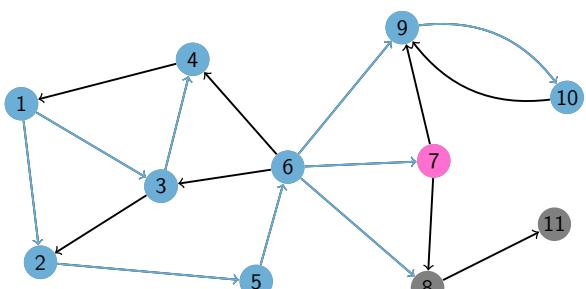
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 10 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	F

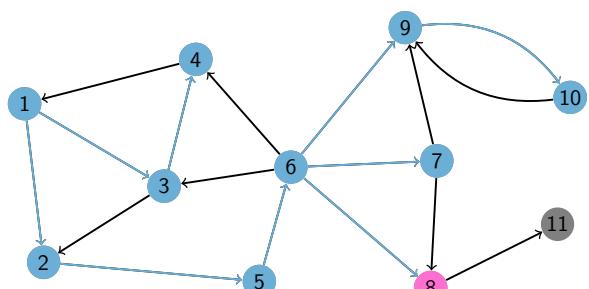
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 7 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	

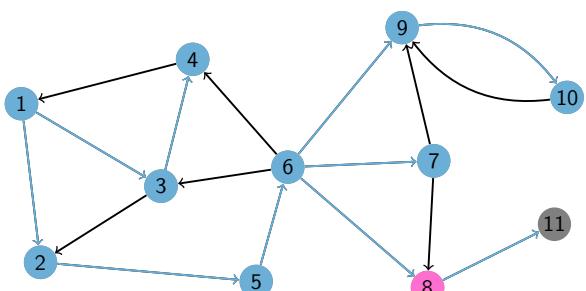
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 8 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	

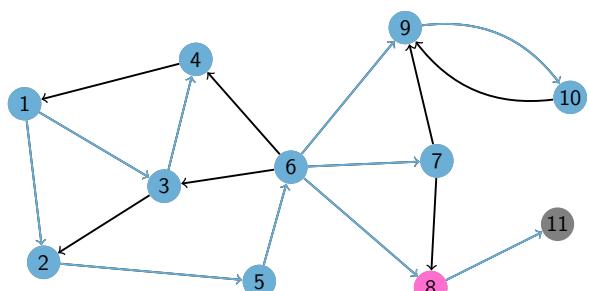
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 8 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	

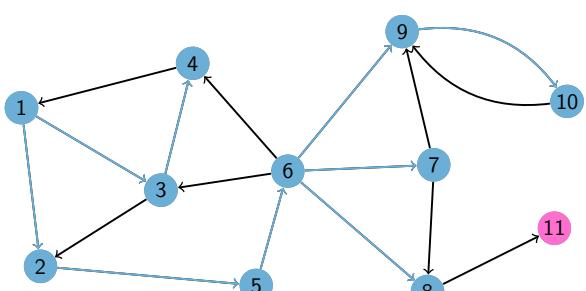
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 8 | 11

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

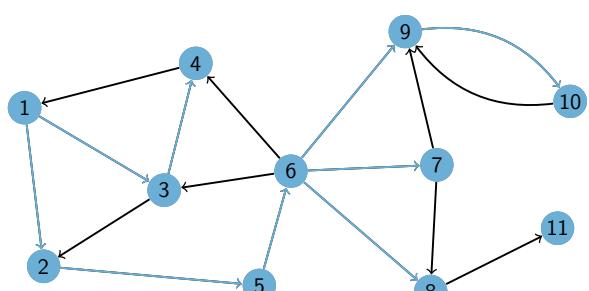
Tìm kiếm theo chiều sâu: Ví dụ



Stack: 11 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

Tìm kiếm theo chiều sâu: Ví dụ



Stack: |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

Tìm kiếm theo chiều sâu: Code

```
1 vector<int> Adj[1001];
2 vector<bool> bVisited(1001, false);
3
4 void DFS(int u) {
5     if (bVisited[u])
6         return;
7
8     bVisited[u] = true;
9
10    for (int i = 0; i < Adj[u].size(); ++i) {
11        int v = Adj[u][i];
12        DFS(v);
13    }
14 }
```

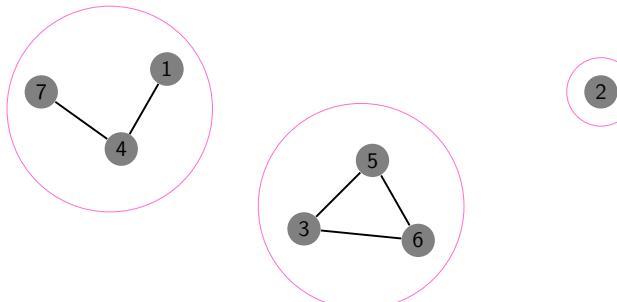


Thành phần liên thông: Bài toán

- Một đồ thị vô hướng có thể phân chia thành các thành phần liên thông (TPLT)
- Một TPLT là một tập con tối đa các đỉnh sao cho giữa hai đỉnh bất kỳ trong tập đều có đường đi giữa chúng
- Bài toán có thể được giải quyết bởi một số thuật toán cơ bản: Cấu trúc dữ liệu các tập không giao nhau (Union-Find), Tìm kiếm theo chiều sâu (DFS) hoặc Tìm kiếm theo chiều rộng (BFS)



Thành phần liên thông: Ví dụ



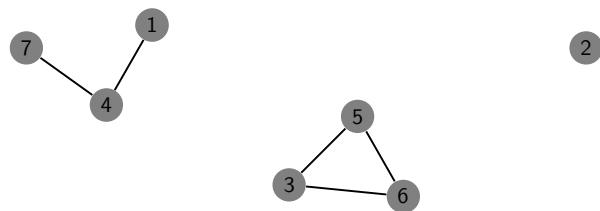
1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

Thành phần liên thông: Ví dụ



Thành phần liên thông

- Cũng có thể sử dụng tìm kiếm theo chiều sâu để tìm các TPLT
- Lấy một đỉnh bất kỳ và gọi thuật toán tìm kiếm theo chiều sâu xuất phát từ đỉnh đó
- Tất cả các đỉnh đến được từ đỉnh xuất phát đó đều thuộc cùng một TPLT
- Lặp lại quá trình trên đến khi thu được toàn bộ các TPLT
- Độ phức tạp $\mathcal{O}(n + m)$

Thành phần liên thông: Code

```
1 vector<int> Adj[1001];
2 vector<int> iComponent(1001, -1);
3
4 void Find_Component(int cur_comp, int u) {
5     if (iComponent[u] != -1) return;
6
7     iComponent[u] = cur_comp;
8
9     for (int i = 0; i < Adj[u].size(); ++i) {
10        int v = Adj[u][i];
11        Find_Component(cur_comp, v);
12    }
13}
14 int num_comp = 0;
15 for (int u = 1; u <= n; ++u) {
16     if (iComponent[u] == -1) {
17         Find_Component(num_comp, u);
18         num_comp++;
19     }
20 }
```



Cây DFS: Giới thiệu

- Khi gọi DFS từ một đỉnh nào đó, các vết tìm kiếm tạo thành một cây
- Khi duyệt từ một đỉnh đến một đỉnh khác chưa được thăm, cạnh duyệt qua đó gọi là *cạnh xuôi* (*forward edge*)
- Khi duyệt từ một đỉnh đến một đỉnh đã thăm trước đó rồi, cạnh duyệt qua đó gọi là *cạnh ngược* (*backward edge*)
- Ngoài ra còn khái niệm cạnh *cạnh vòng* (*cross edge*) trên đồ thị có hướng
- Tập các cạnh xuôi tạo thành một cây DFS

xem ví dụ



Phân tích cây DFS

- Hãy quan sát kỹ hơn cây DFS
- Đầu tiên, đánh số các đỉnh theo thứ tự duyệt của thuật toán DFS trong mảng *num*
- Với mỗi đỉnh *u*, cần tính *Low[u]* là chỉ số của đỉnh nhỏ nhất mà *u* có thể chạm được (bởi một cạnh ngược) khi tiếp tục duyệt theo cây con có gốc tại *u*
- Khởi tạo thì *Low[u] = Num[u]*, *Low[u]* sẽ thay đổi khi có một cạnh ngược.
- Những thông số này rất hữu ích cho các bài toán quan trọng: tìm khớp/cầu, tìm thành phần liên thông mạnh, ...



1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

Cây DFS: Giới thiệu

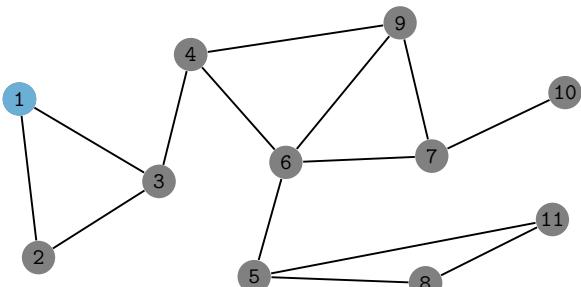
- Cây từ các cạnh xuôi, cùng với các cạnh ngược, chứa đựng rất nhiều thông tin về đồ thị ban đầu
- Ví dụ: một cạnh ngược luôn nằm trên một chu trình của đồ thị ban đầu
- Nếu không có cạnh ngược thì không có chu trình trong đồ thị ban đầu (nghĩa là loại đồ thị không có chu trình - acyclic graph)

Phân tích cây DFS: Code

```
1 const int n = 1001;
2 vector<int> Adj[n];
3 vector<int> Low(n), Num(n, -1);
4 int curnum = 0;
5 void AnalyzeDFS(int u, int p) {
6     Low[u] = Num[u] = ++curnum;
7     for (int i = 0; i < Adj[u].size(); ++i) {
8         int v = Adj[u][i];
9         if (v == p) continue; // Khong_Xet_Dinh_Ngay_Truoc
10        if (Num[v] == -1) {
11            Analyze(v, u);
12            Low[u] = min(Low[u], Low[v]);
13        } else {
14            Low[u] = min(Low[u], Num[v]);
15        }
16    }
17 }
```

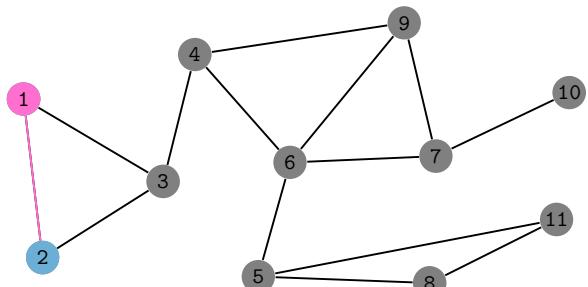
- Gọi AnalyzeDFS(*u*, -1) với mọi *u* chưa được thăm (*Num[u] == -1*)

Phân tích cây DFS: Ví dụ



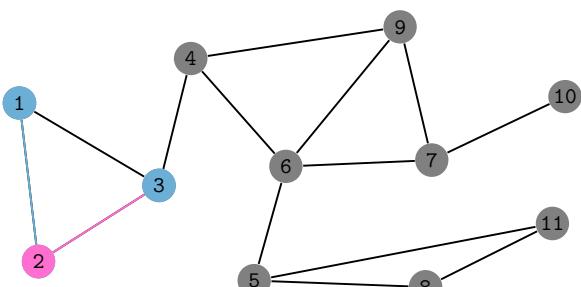
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



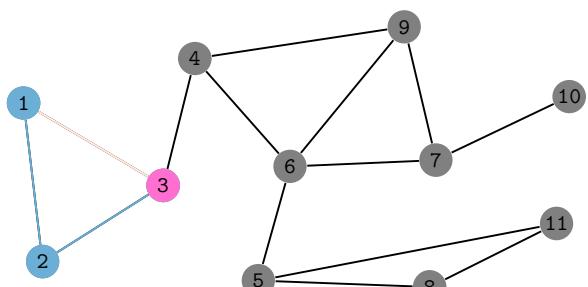
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



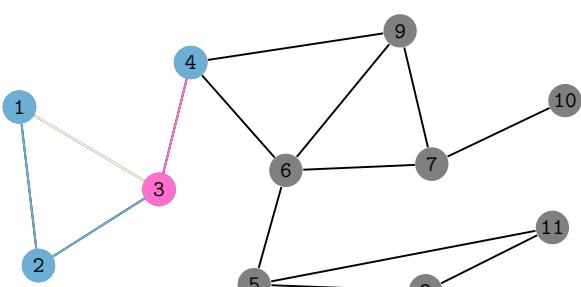
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



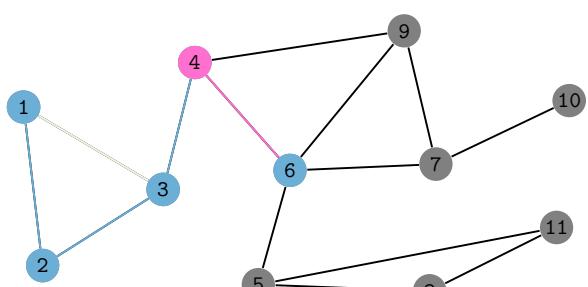
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



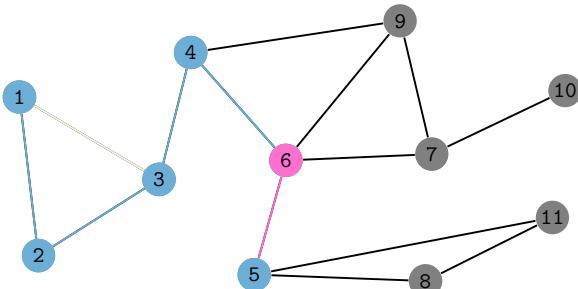
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



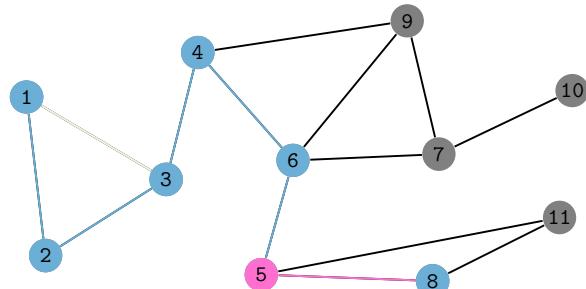
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



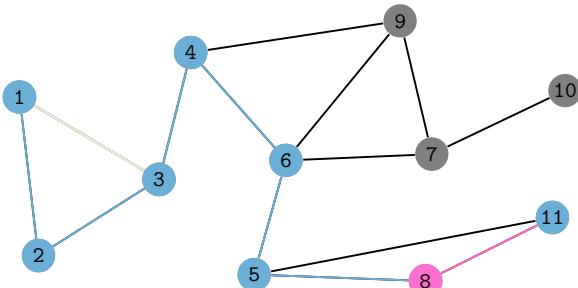
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



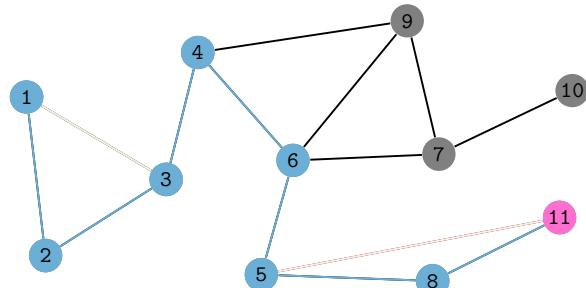
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



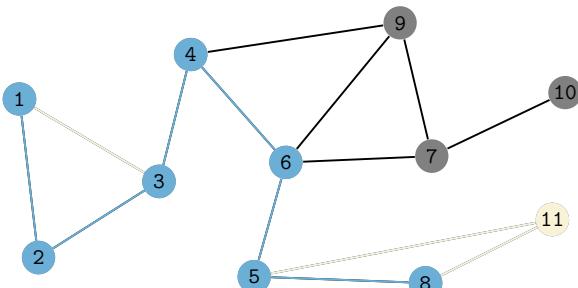
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



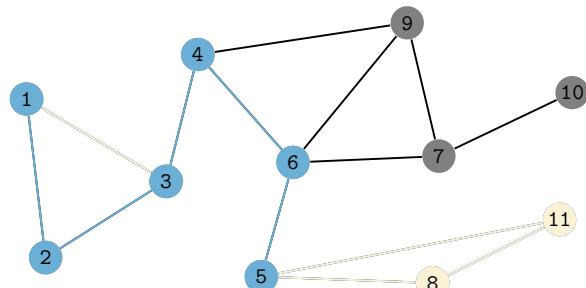
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



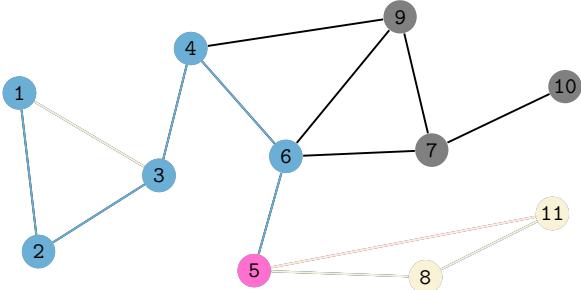
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



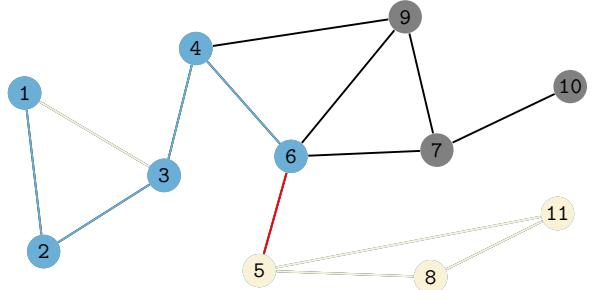
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



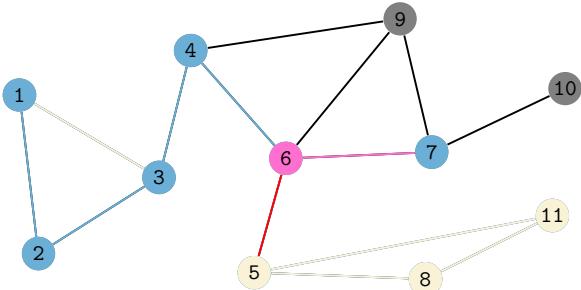
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



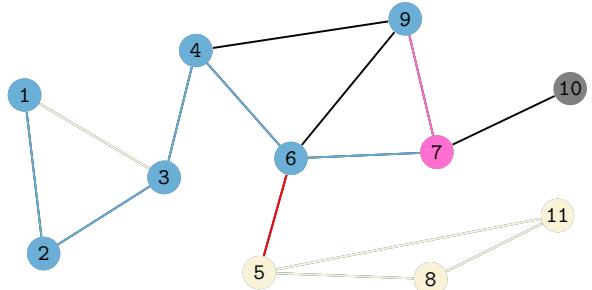
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



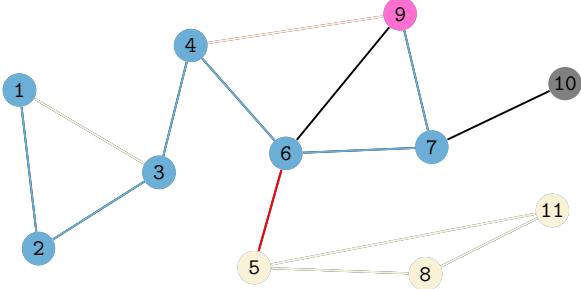
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



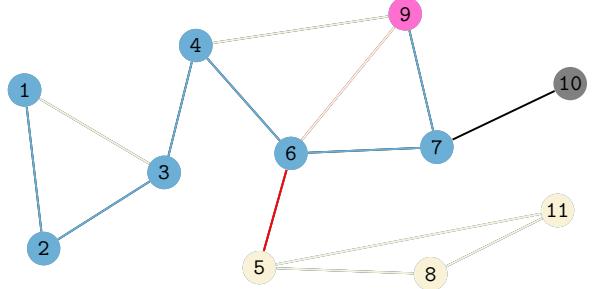
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



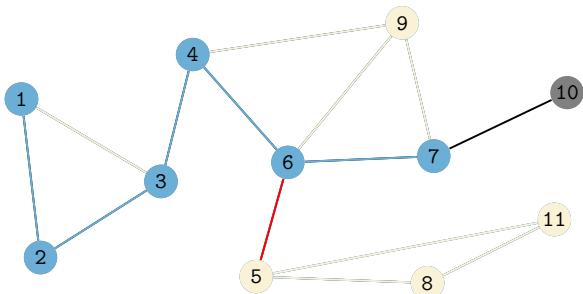
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



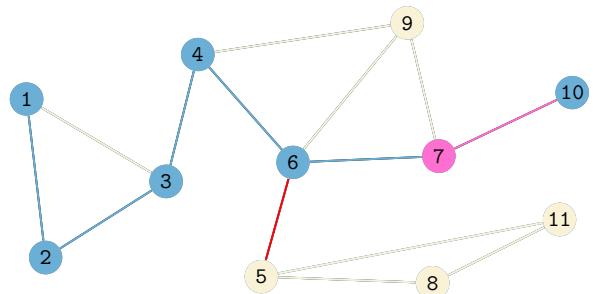
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



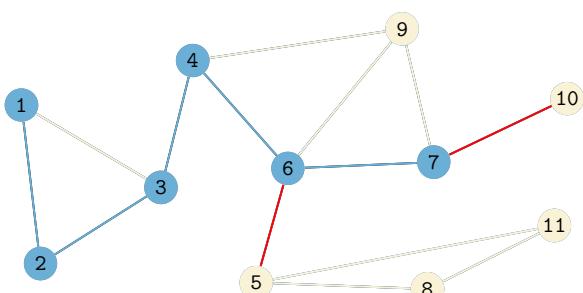
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



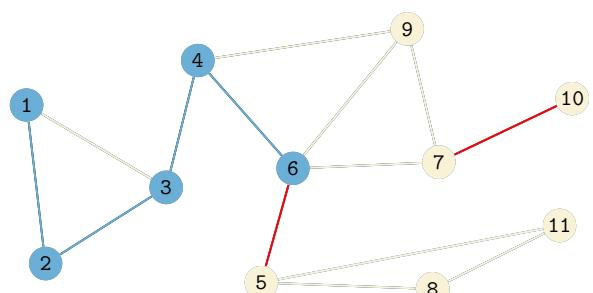
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



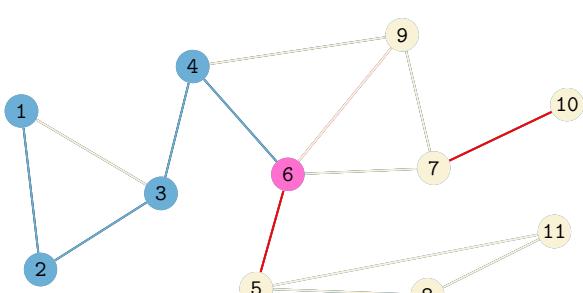
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



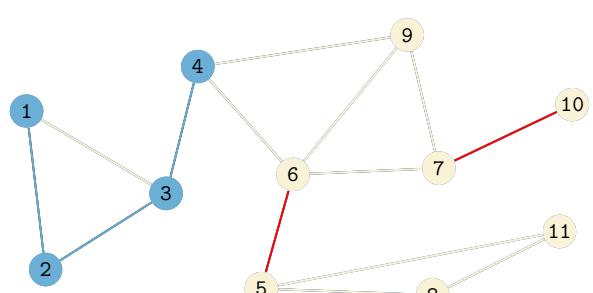
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



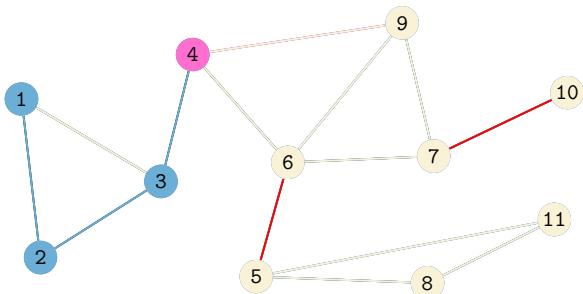
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



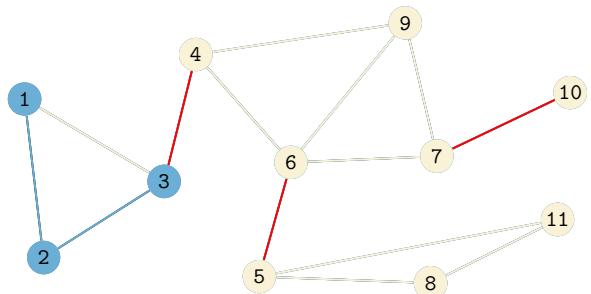
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



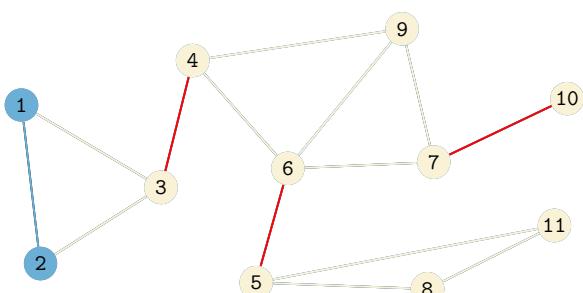
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



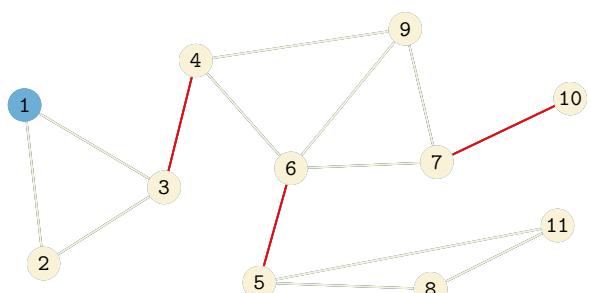
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



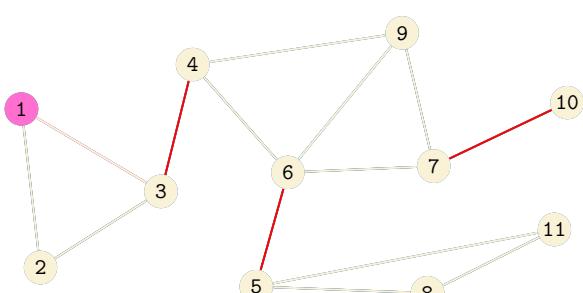
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



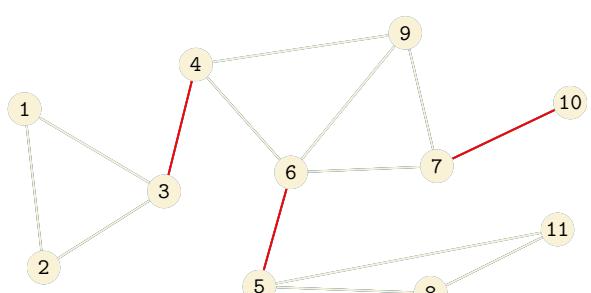
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Phân tích cây DFS: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

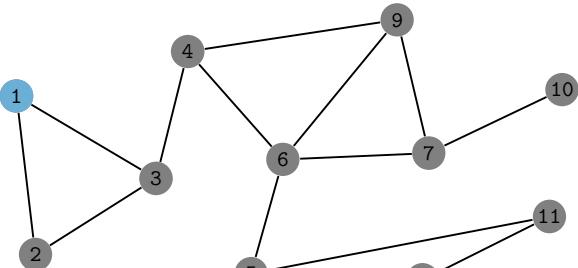
Phân tích cây DFS

- Độ phức tạp chỉ là $\mathcal{O}(n + m)$, do chỉ gọi một lần DFS
- Bây giờ hãy xem một số ứng dụng của thuật toán này

Cầu: Giới thiệu

- Cho đồ thị không trọng số $G = V, E$)
- Không mất tính tổng quát, giả sử G liên thông (nghĩa là G là một TPLT lớn)
- Tìm một cạnh mà nếu loại bỏ cạnh đó ra khỏi G thì G mất tính liên thông
- Thuật toán trực tiếp: Thử loại bỏ từng cạnh một, và tính số TPLT thu được
- Cách này thiêu hiệu quả: $\mathcal{O}(m(n + m))$

Cầu: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

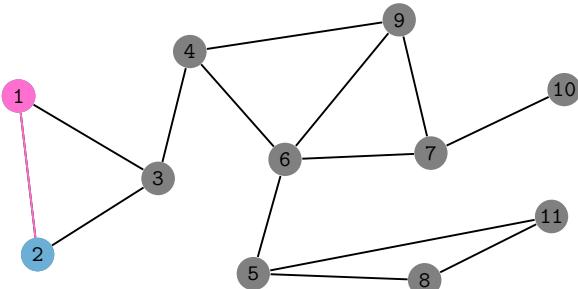
- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

Cầu: Giới thiệu

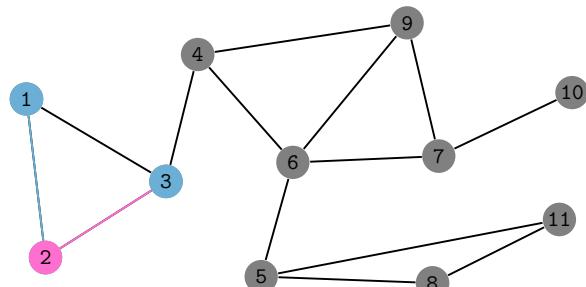
- Bây giờ quan sát các giá trị tính được từ cây DFS
- Nhận thấy rằng một cạnh xuôi (u, v) là cầu khi và chỉ khi $\text{low}[v] > \text{Num}[u]$
- Như vậy chỉ cần mở rộng thuật toán phân tích cây DFS ở trên để đưa ra toàn bộ cầu
- Độ phức tạp thuật toán chỉ là $\mathcal{O}(n + m)$ với chỉ một lần gọi DFS đối với mỗi TPLT!

Cầu: Ví dụ



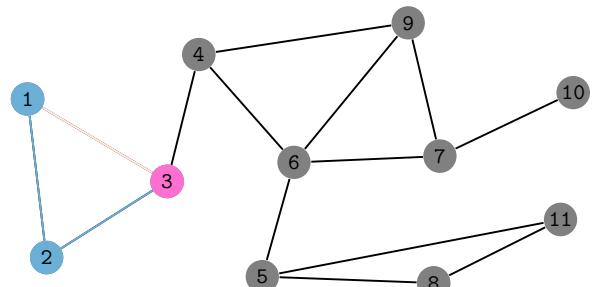
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



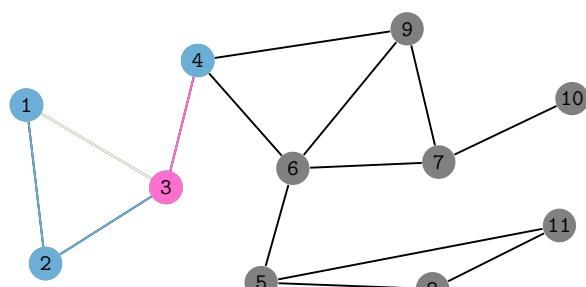
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



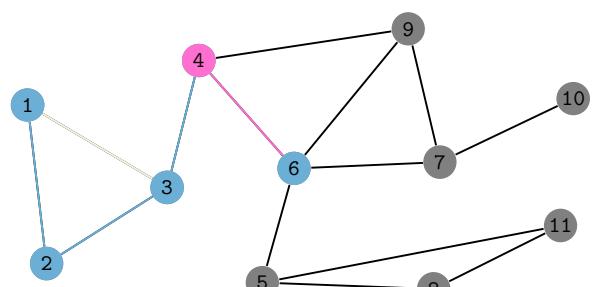
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



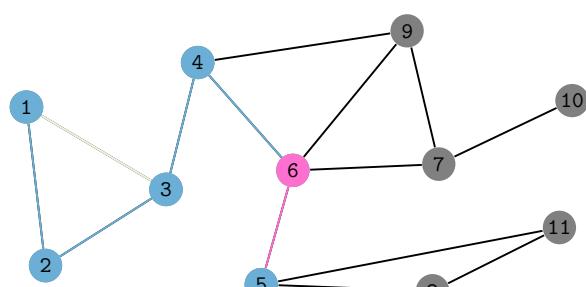
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



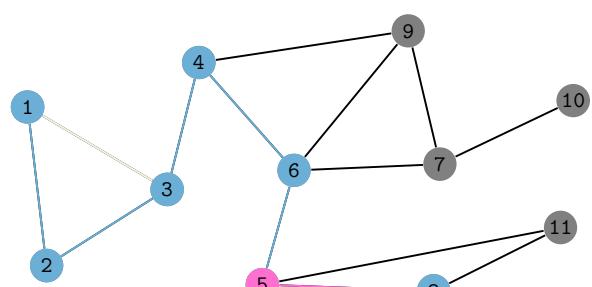
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



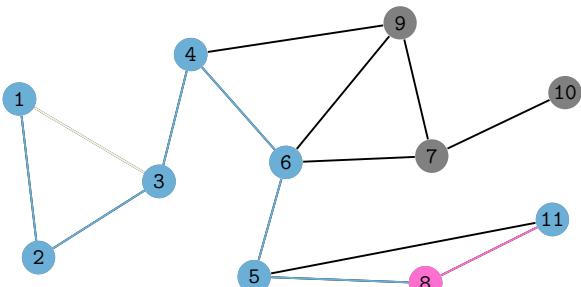
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ

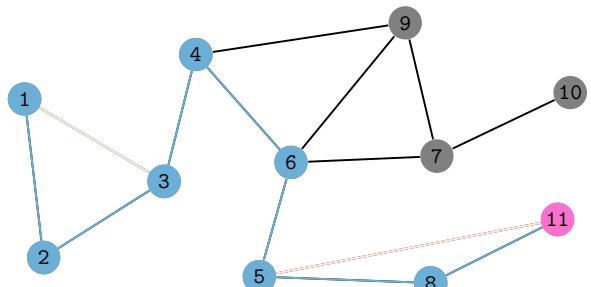


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ

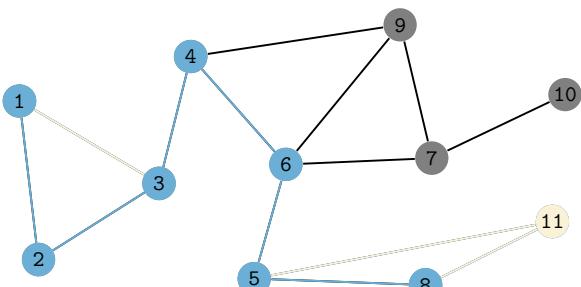


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ

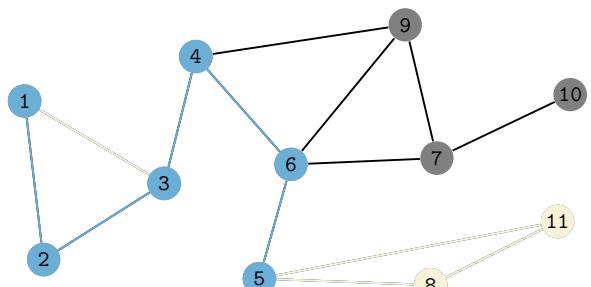


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ

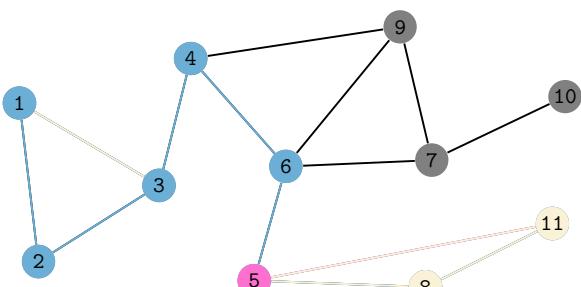


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

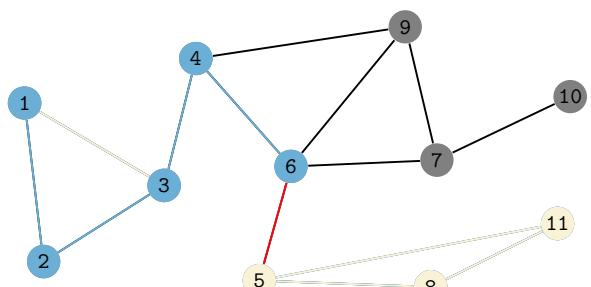
Câu: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

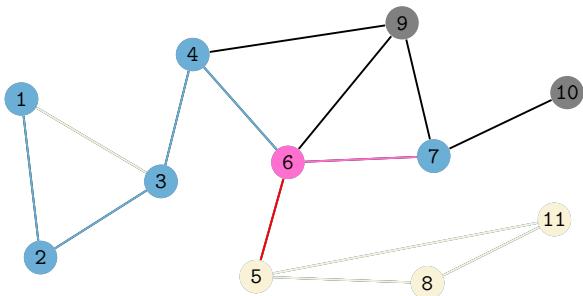


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



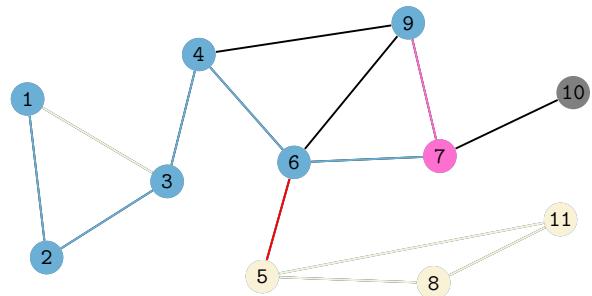
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ



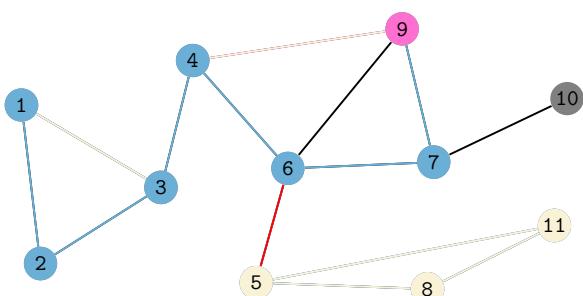
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



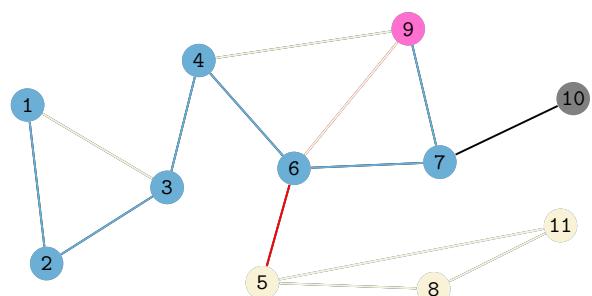
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



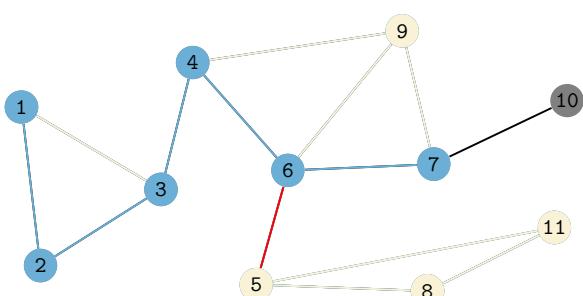
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



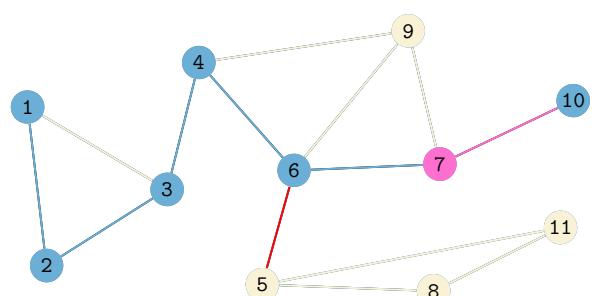
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



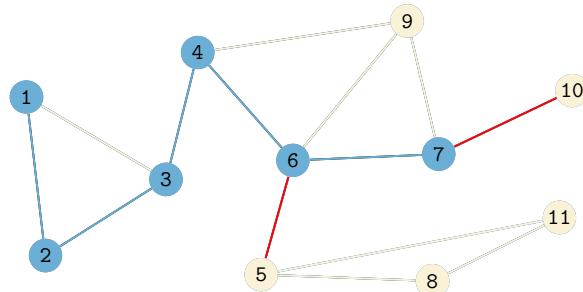
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



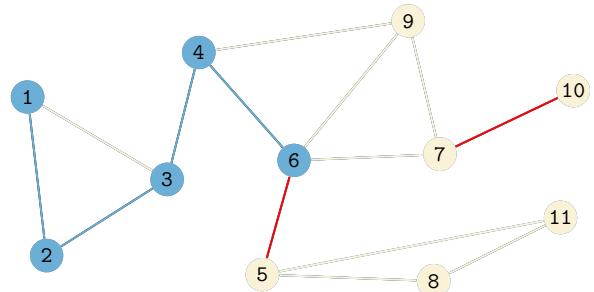
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



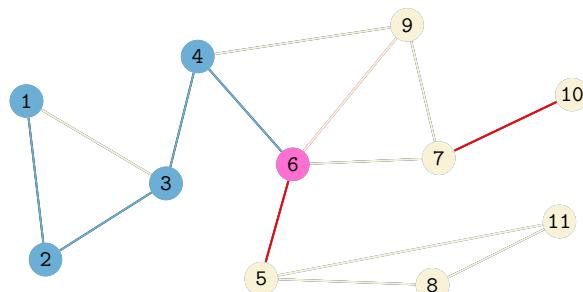
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



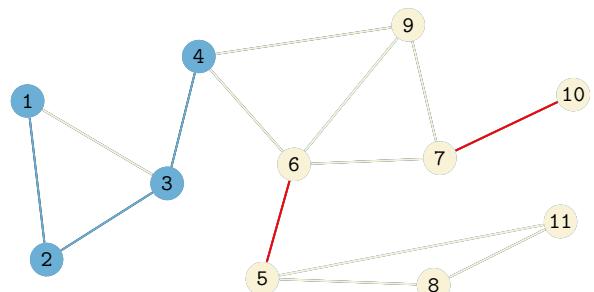
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ



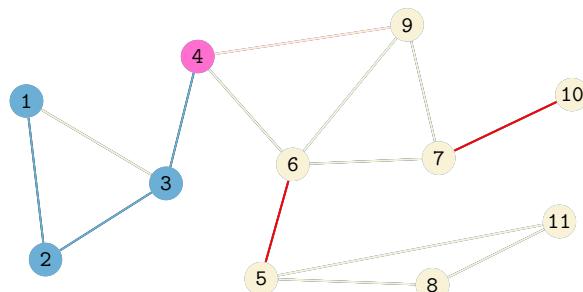
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ

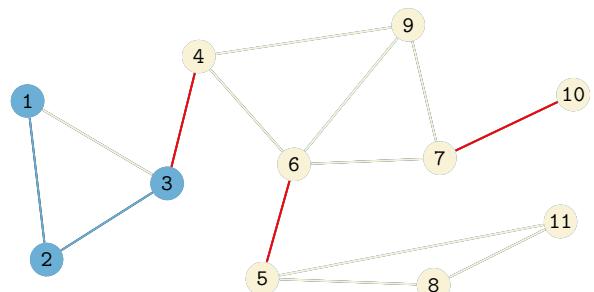


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ

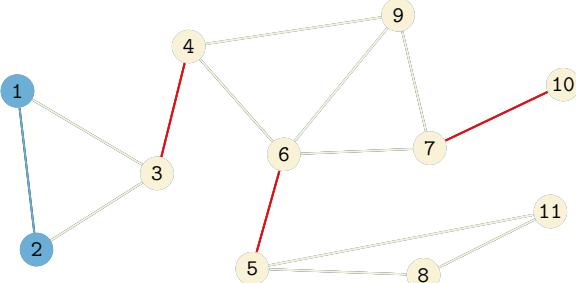


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

Câu: Ví dụ

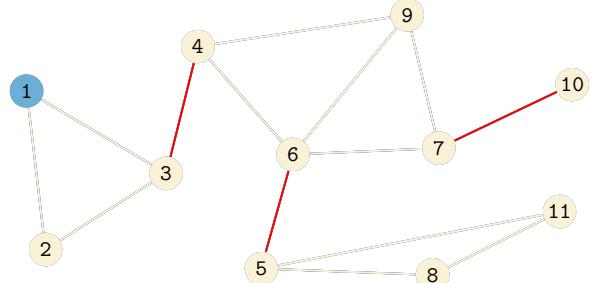


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

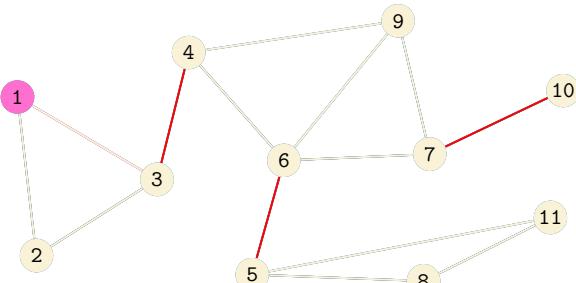
Câu: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ

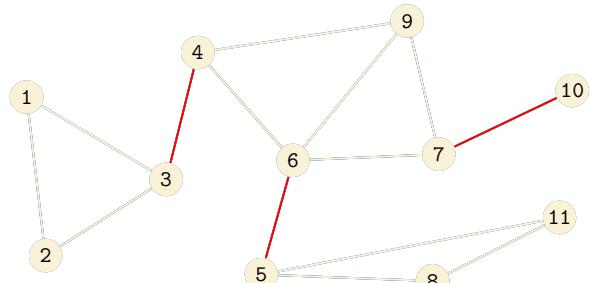


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Câu: Code

```

1 const int n = 1001;
2 vector<int> Adj[n], Low(n), Num(n, -1);
3 int curnum = 0;
4 vector<pair<int, int>> iiBridges;
5
6 void Find_Bridges(int u, int p) {
7     Low[u] = Num[u] = ++curnum;
8     for (int i = 0; i < Adj[u].size(); ++i) {
9         int v = Adj[u][i];
10        if (v == p) continue;
11        if (Num[v] == -1) {
12            Find_Bridges(v, u);
13            Low[u] = min(Low[u], Low[v]);
14        } else {
15            Low[u] = min(Low[u], Num[v]);
16        }
17        if (Low[v] > Num[u])
18            iiBridges.push_back(make_pair(u, v));
19    }
20}

```

- Gọi `Find_Bridges(u, -1)` với mọi u chưa được thăm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

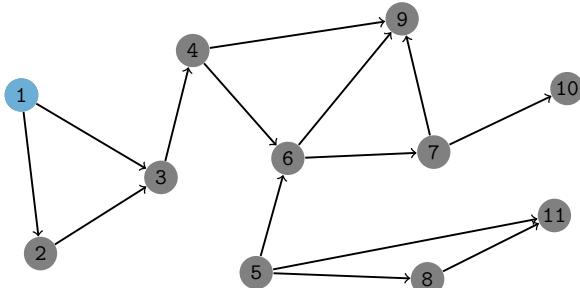
3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TPLT mạnh: Giới thiệu

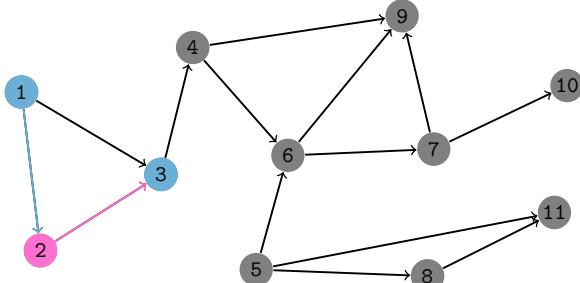
- Ta đã biết cách tìm các TPLT trên đồ thị vô hướng
- Thế trên đồ thị có hướng thì sao?
- Các TPLT này có một chút khác biệt trên đồ thị có hướng, bởi vì nếu v đến được từ u thì không có nghĩa là u đến được từ v
- Tuy vậy, định nghĩa vẫn tương tự
- Một TPLT mạnh là một tập con tối đa các đỉnh sao cho giữa hai đỉnh bất kỳ trong tập luôn có đường đi từ đỉnh này đến đỉnh kia và ngược lại

TPLT mạnh: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ

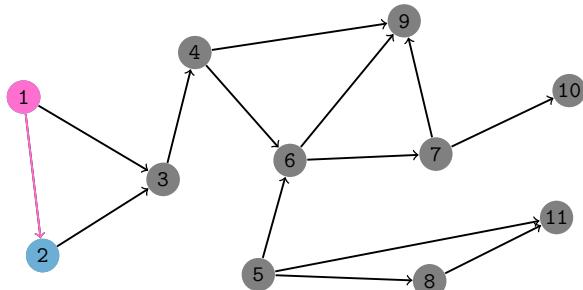


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Giới thiệu

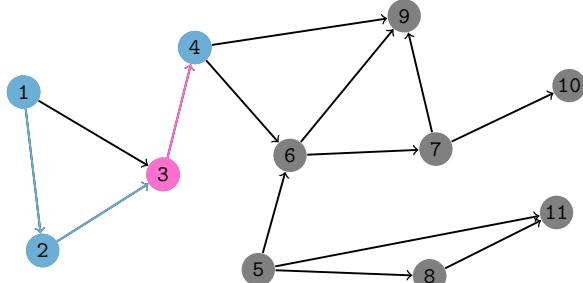
- Thuật toán tìm các TPLT ở trên không áp dụng được
- Thay vào đó ta có thể sử dụng cây DFS để tìm các TPLT mạnh này
- xem ví dụ

TPLT mạnh: Ví dụ



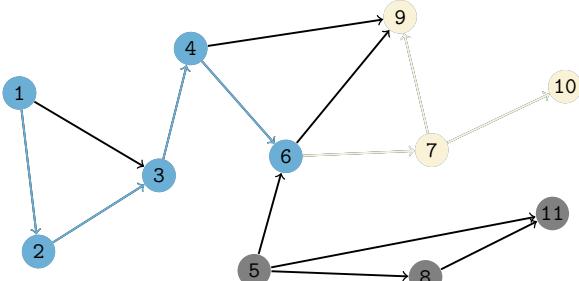
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



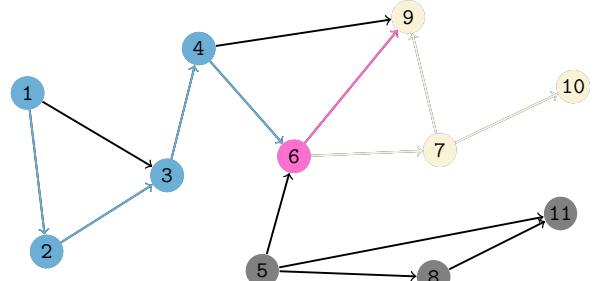
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



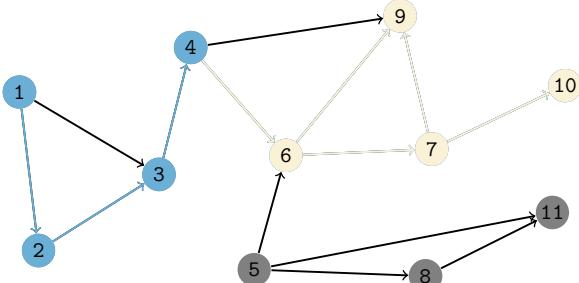
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



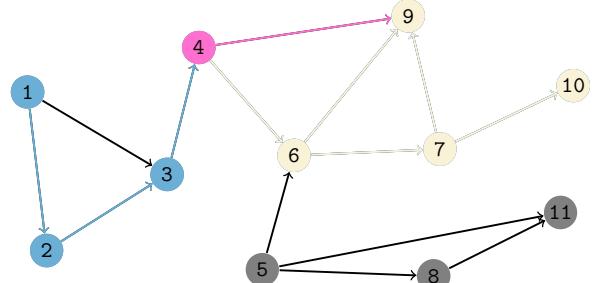
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



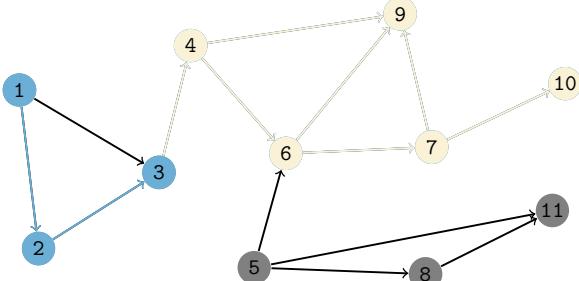
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



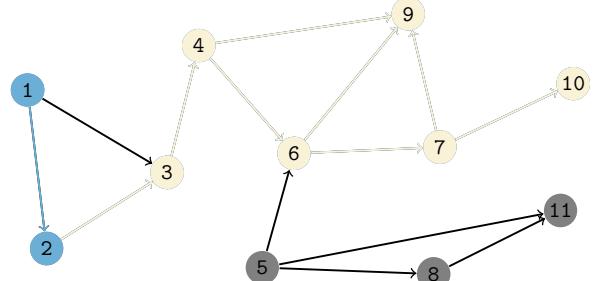
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



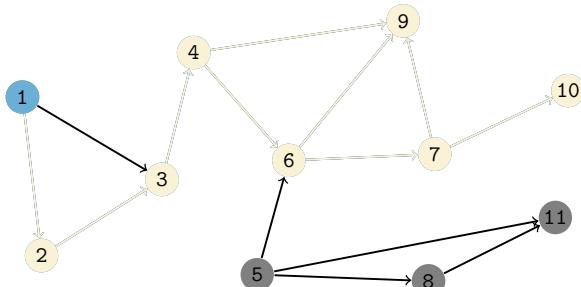
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



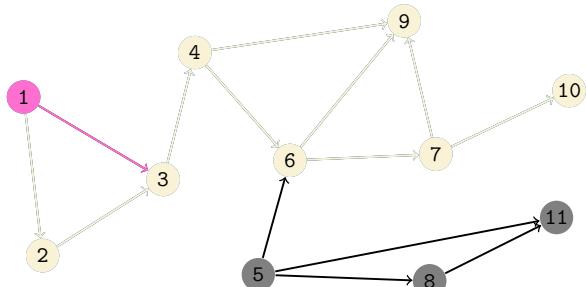
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



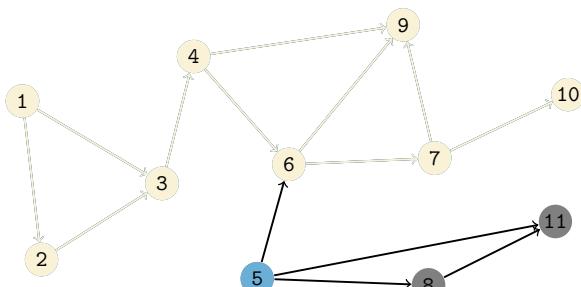
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



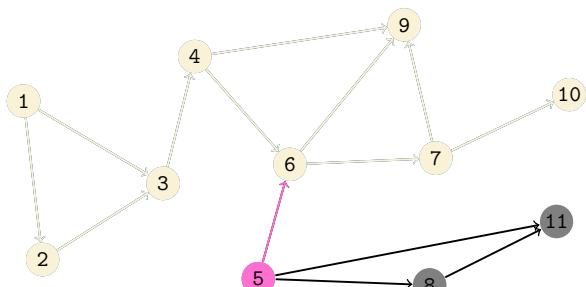
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



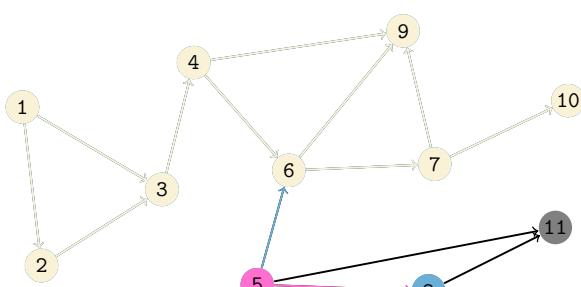
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ

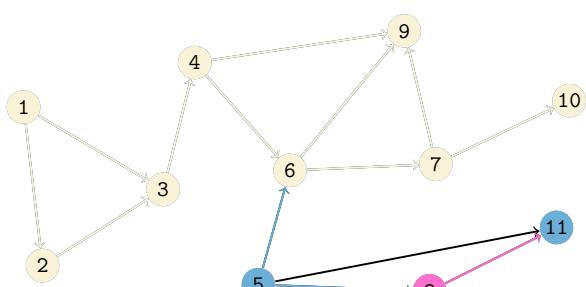


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ

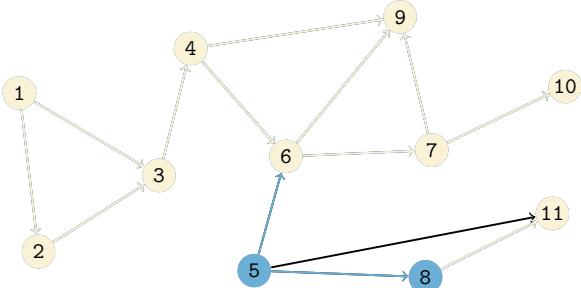


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6



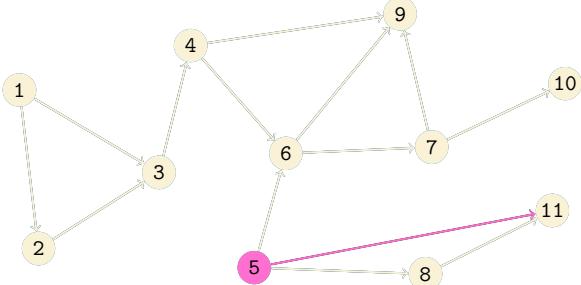
i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ

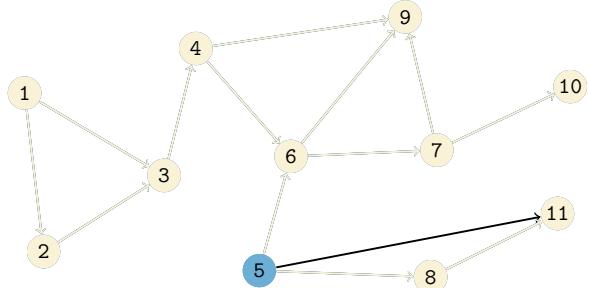


i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh

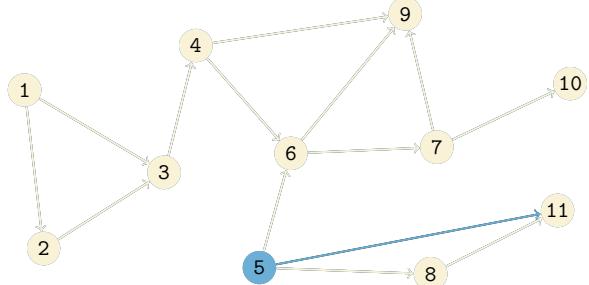
- Sau quá trình phân tích cây tại đỉnh u mà $\text{Low}[u] = \text{Num}[u]$ thì ta tìm được một thành phần liên thông mạnh theo quá trình duyệt cây từ u
- Sử dụng mảng $bConnect[]$ dùng để kiểm tra xem đỉnh v có còn được “kết nối” trong đồ thị hay không? Nếu phát hiện ra một thành phần liên thông mạnh, và một đỉnh v có trong thành phần liên thông mạnh đó, thì ta loại đỉnh v này ra khỏi đồ thị bằng câu lệnh $bConnect[v] = 0$, điều này là quan trọng vì để tránh gây ảnh hưởng đến việc tính mảng $\text{Low}[]$ của những đỉnh khác vẫn còn nằm trong đồ thị

TPLT mạnh: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
num[i]	1	2	3	4	6	5	9	7	10	11	8
low[i]	1	1	1	4	6	4	4	6	4	11	6

TPLT mạnh: Code

```
1 vector<int> Adj[10001];
2 vector<int> Low(10001), Num(10001, -1);
3 vector<bool> bConnect(10001, false);
4 int curnum = 0;
5
6 stack<int> iComp;
7
8 void SCC(int u) {
9     // SCC code...
10 }
11
12 int main() {
13     for (int i = 0; i < n; ++i)
14         if (Num[i] == -1)
15             SCC(i);
16
17     return;
18 }
```

TPLT mạnh: Code

```
1 void SCC(int u) {
2     iComp.push(u);
3     bConnect[u] = true;
4     Low[u] = Num[u] = ++curnum;
5
6     for (int i = 0; i < Adj[u].size(); ++i) {
7         int v = Adj[u][i];
8         if (Num[v] == -1) {
9             SCC(v);
10            Low[u] = min(Low[u], Low[v]);
11        } else if (iConnect[v]) {
12            Low[u] = min(Low[u], Num[v]);
13        }
14    }
15
16    if (Num[u] == Low[u]) {
17        cout << "TPLT: ";
18        while (true) {
19            int cur = iComp.top();
20            iComp.pop();
21            bConnect[cur] = false;
22            cout << cur << " ";
23            if (cur == u) break;
24        }
25        cout << endl;
26    }
27}
```



TPLT mạnh

- Độ phức tạp thuật toán?
- Cơ bản là chỉ dùng thuật toán phân tích cây DFS (có độ phức tạp $\mathcal{O}(n + m)$), cộng thêm một vòng lặp để xây dựng TPLT mạnh
- Do mỗi đỉnh chỉ thuộc một TPLT, độ phức tạp vẫn chỉ là $\mathcal{O}(n + m)$
- Thuật toán được biết đến với tên gọi thuật toán Tarjan

Sắp xếp topo: Bài toán

- Có n công việc
- Mỗi công việc i có một danh sách các công việc cần phải hoàn thành trước khi bắt đầu công việc i
- Hãy tìm một trình tự mà ta có thể thực hiện toàn bộ các công việc
- Có thể biểu diễn trên một đồ thị có hướng
 - Mỗi công việc là một đỉnh của đồ thị
 - Nếu công việc j phải hoàn thành trước công việc i , thì thêm một cạnh có hướng từ đỉnh j đến đỉnh i
- Lưu ý là không thể có một trình tự thỏa mãn nếu như đồ thị có chu trình
- Có thể sửa đổi thuật toán DFS để đưa ra một trình tự thỏa mãn trong thời gian $\mathcal{O}(n + m)$, hoặc đưa ra không có trình tự thỏa mãn

1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

- DFS
- Thành phần liên thông
- Cây DFS
- Cầu
- TPLT mạnh
- Sắp xếp topo

3 Tìm kiếm theo chiều rộng và ứng dụng - BFS



Sắp xếp topo: code

```
1 vector<int> Adj[1001];
2 vector<bool> bVisited(1001, false);
3 vector<int> iOrder;
4
5 void Topo_Sort(int u) {
6     if (bVisited[u]) return;
7
8     bVisited[u] = true;
9     for (int i = 0; i < Adj[u].size(); ++i) {
10        int v = Adj[u][i];
11        Topo_Sort(v);
12    }
13    iOrder.push_back(u);
14}
```

- Gọi Topo_Sort(u) với mọi đỉnh u chưa được thăm



1 Cơ bản về đồ thị

2 Tìm kiếm theo chiều sâu và ứng dụng - DFS

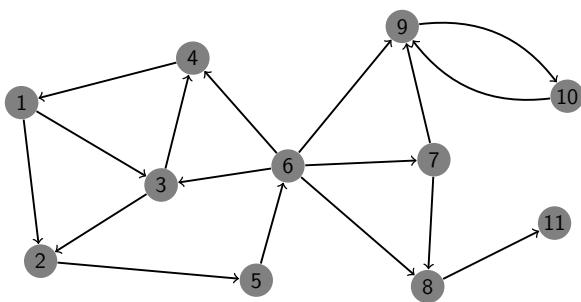
3 Tìm kiếm theo chiều rộng và ứng dụng - BFS

- Tìm kiếm theo chiều rộng - BFS
- Đường đi ngắn nhất trên đồ thị không trọng số

Tìm kiếm theo chiều rộng - BFS

- Thuật toán tìm kiếm theo chiều rộng chỉ khác DFS ở trình tự thăm các đỉnh
- BFS thăm đỉnh theo trình tự FIFO (First In First Out)
- BFS cho trình tự thăm tăng dần theo khoảng cách từ đỉnh xuất phát

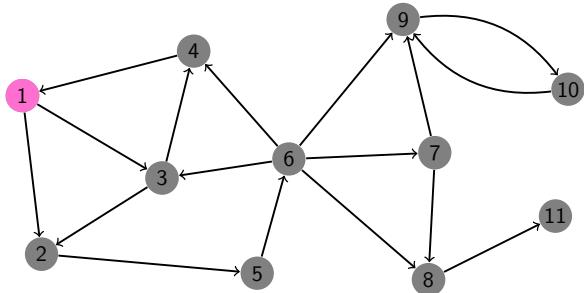
BFS: Ví dụ



Queue:

1	2	3	4	5	6	7	8	9	10	11
F	F	F	F	F	F	F	F	F	F	F

BFS: Ví dụ

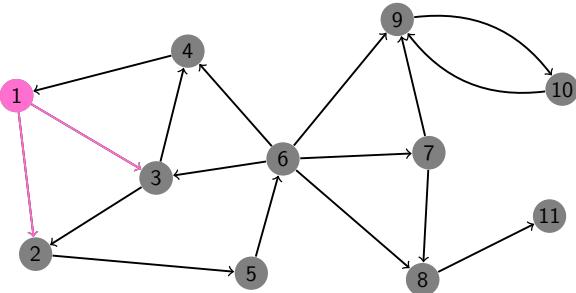


Queue: 1

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

Visited	T	F	F	F	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---

BFS: Ví dụ

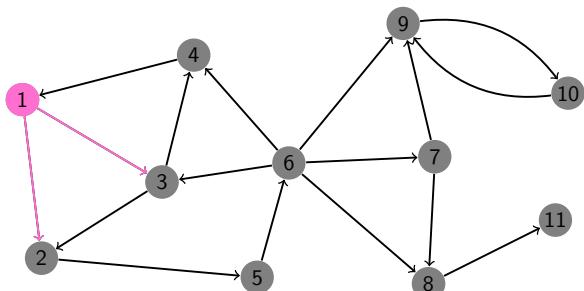


Queue: 1

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

Visited	T	F	F	F	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---

BFS: Ví dụ

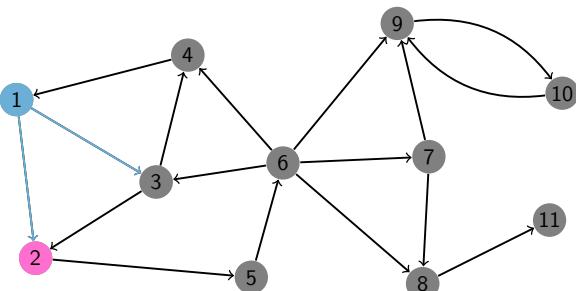


Queue: 1 2 3

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	F	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---

BFS: Ví dụ

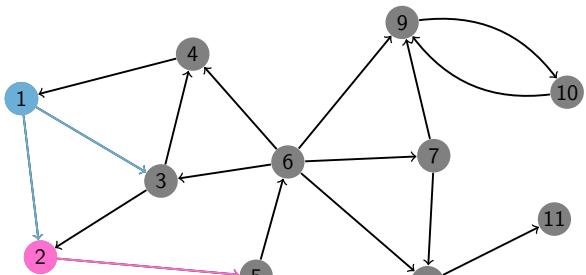


Queue: 2 3

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

Visited	T	T	T	F	F	F	F	F	F	F
---------	---	---	---	---	---	---	---	---	---	---

BFS: Ví dụ



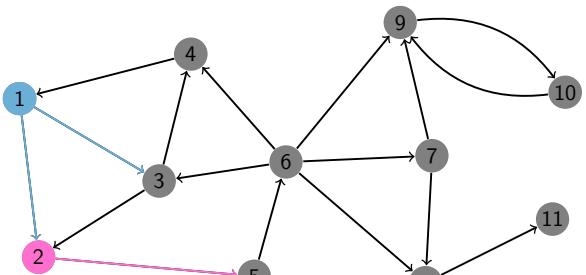
Queue: 2 3

Visited	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	F	F	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



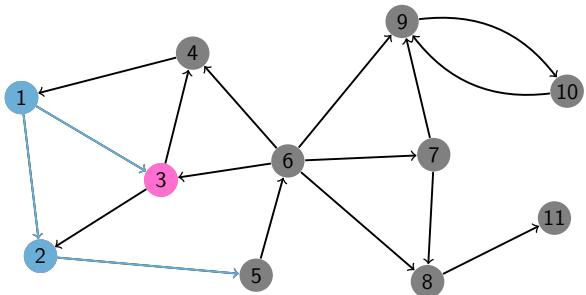
Queue: 2 3 5

Visited	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	F	T	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



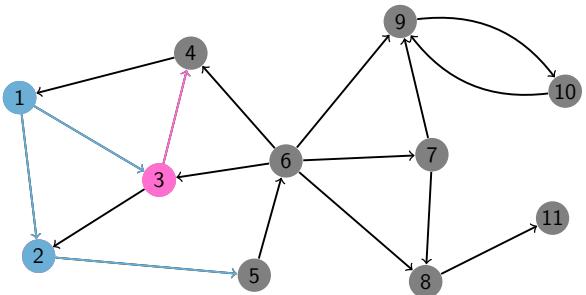
Queue: 3 5

Visited	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	F	T	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



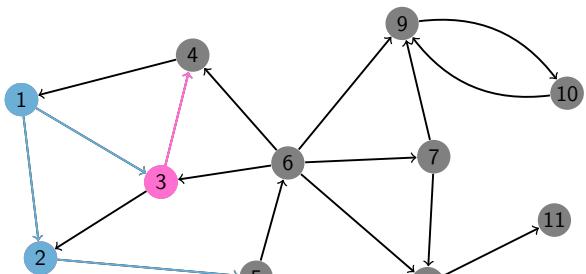
Queue: 3 5

Visited	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	F	T	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



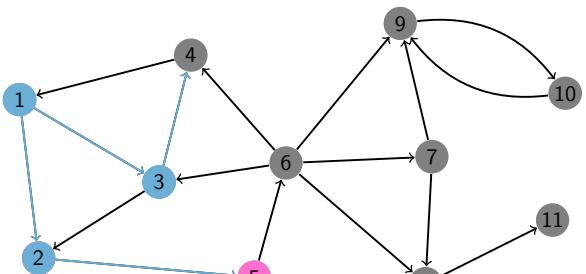
Queue: 3 5 4

Visited	1	2	3	4	5	6	7	8	9	10	11
	T	T	T	T	T	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



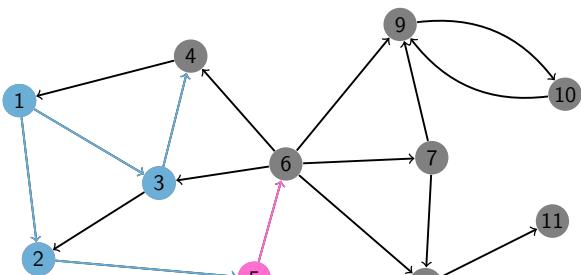
Queue: 5 4

Visited	1	2	3	4	T	F	F	F	F	F	F
	T	T	T	T	T	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



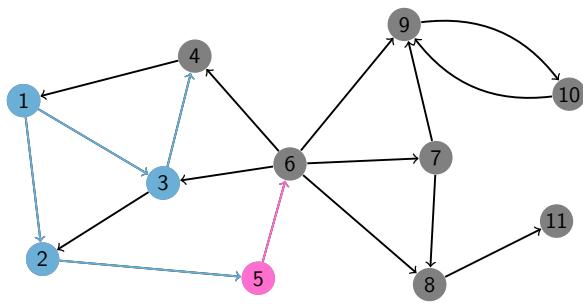
Queue: 5 4

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	F	F	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



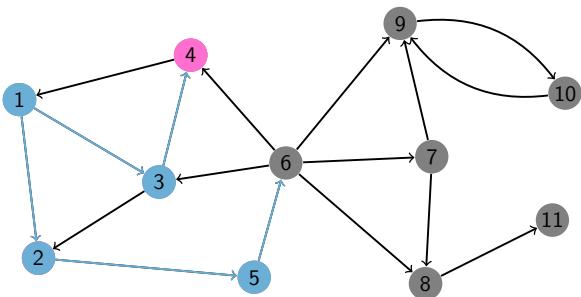
Queue: 5 4 6

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



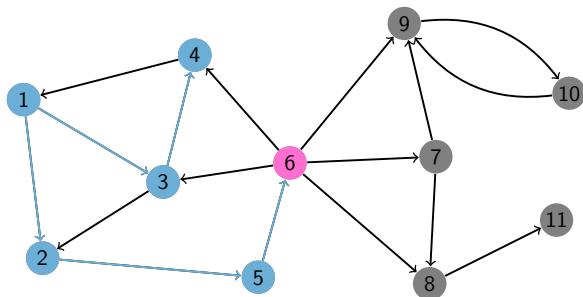
Queue: 4 6

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



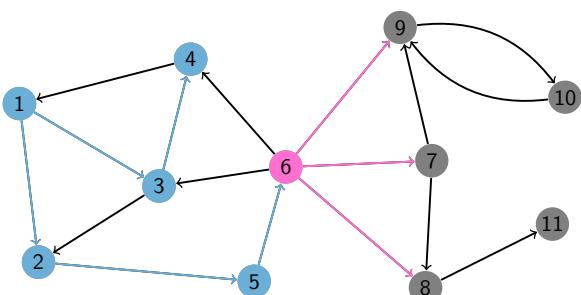
Queue: 6

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



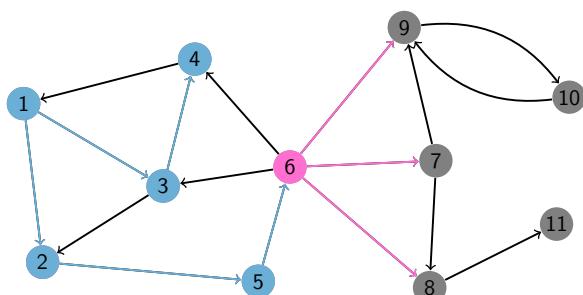
Queue: 6

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



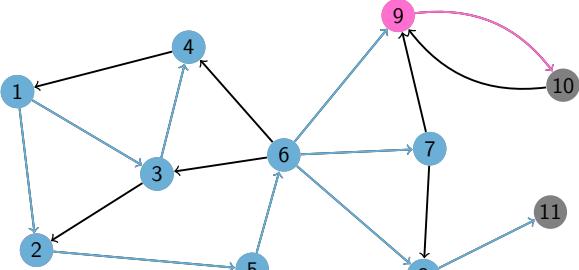
Queue: 6 7 8 9

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	F	F	F



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BFS: Ví dụ



Queue: 9 11 10

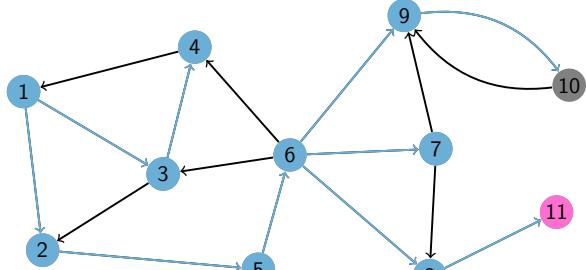
	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50 / 55

BFS: Ví dụ



Queue: 11 10

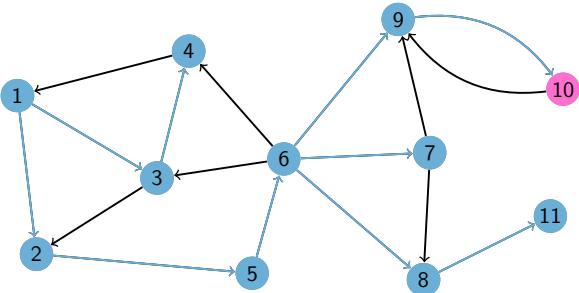
	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50 / 55

BFS: Ví dụ



Queue: 10

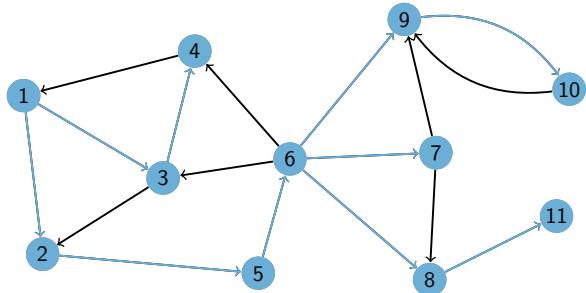
	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50 / 55

BFS: Ví dụ



Queue:

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50 / 55

BFS

```

1 vector<int> Adj[1001];
2 vector<bool> bVisited(1001, false);
3 queue<int> Q;
4 Q.push(start);
5 bVisited[start] = true;

6
7 while (!Q.empty()) {
8     int u = Q.front(); Q.pop();

9     for (int i = 0; i < Adj[u].size(); ++i) {
10        int v = Adj[u][i];
11        if (!bVisited[v]) {
12            Q.push(v);
13            bVisited[v] = true;
14        }
15    }
16 }
}

```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51 / 55

① Cơ bản về đồ thị

② Tìm kiếm theo chiều sâu và ứng dụng - DFS

③ Tìm kiếm theo chiều rộng và ứng dụng - BFS

- Tìm kiếm theo chiều rộng - BFS
- Đường đi ngắn nhất trên đồ thị không trọng số



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52 / 55

Dường đi ngắn nhất trên đồ thị không trọng số

- Cho đồ thị không trọng số $G = (V, E)$, hãy tìm đường đi ngắn nhất từ đỉnh A đến đỉnh B
- Có nghĩa là tìm đường đi từ a đến b đi qua ít cạnh nhất
- BFS duyệt các đỉnh theo trình tự tăng dần khoảng cách từ đỉnh xuất phát
- Vì vậy chỉ cần gọi một lần BFS từ đỉnh a đến khi tìm thấy b
- Hoặc duyệt qua toàn bộ G , và ta có đường đi ngắn nhất từ a đến tất cả các đỉnh khác
- Độ phức tạp: $\mathcal{O}(n + m)$



Dường đi ngắn nhất trên đồ thị không trọng số

```
1 vector<int> Adj[1001];
2 vector<int> iDist(1001, -1);
3 queue<int> Q;
4 Q.push(a);
5 iDist[a] = 0;
6
7 while (!Q.empty()) {
8     int u = Q.front(); Q.pop();
9     for (int i = 0; i < Adj[u].size(); ++i) {
10         int v = Adj[u][i];
11         if (iDist[v] == -1) {
12             Q.push(v);
13             iDist[v] = 1 + iDist[u];
14         }
15     }
16 }
17 cout << iDist[b] << endl;
```



Thank you for
your attentions!

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



Đồ Thị Nâng Cao THUẬT TOÁN ỨNG DỤNG

- Đồ thị có trọng số
- Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND
- Cây khung nhỏ nhất – MST
- Dường đi ngắn nhất
- Một số bài toán kinh điển trên đồ thị
- Một số đồ thị đặc biệt



1 Đồ thị có trọng số

2 Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3 Cây khung nhỏ nhất - MST

4 Đường đi ngắn nhất

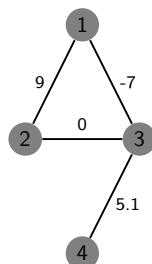
5 Một số bài toán kinh điển trên đồ thị

6 Một số đồ thị đặc biệt



Đồ thị có trọng số

```
struct edge {  
    int u, v;  
    int weight;  
  
    edge(int _u, int _v, int _w) {  
        u = _u;  
        v = _v;  
        weight = _w;  
    }  
};
```



1 Đồ thị có trọng số

2 Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3 Cây khung nhỏ nhất - MST

4 Đường đi ngắn nhất

5 Một số bài toán kinh điển trên đồ thị

6 Một số đồ thị đặc biệt



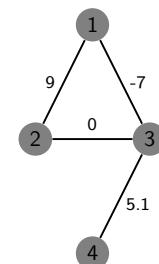
Đồ thị có trọng số

- Trong phần này ta xét đồ thị mà mỗi cạnh của nó có trọng số đi kèm
 - giá trị trên cạnh
 - trọng số trên cạnh
 - ví dụ: khoảng cách của con đường, chi phí truyền thông giữa hai nút mạng, ...
- Biểu diễn đồ thị có trọng số bởi danh sách kề mở rộng



Đồ thị có trọng số

```
vector<edge> Adj[4];  
  
Adj[1].push_back(edge(1, 2, 9));  
Adj[1].push_back(edge(1, 3, -7));  
  
Adj[2].push_back(edge(2, 1, 9));  
Adj[2].push_back(edge(2, 3, 0));  
  
Adj[3].push_back(edge(3, 1, -7));  
Adj[3].push_back(edge(3, 2, 0));  
Adj[3].push_back(edge(3, 4, 5.1));  
  
Adj[4].push_back(edge(4, 3, 5.1));
```



Union-Find

- Cho n phần tử
- Cần quản lý vào các tập không giao nhau
- Mỗi phần tử ở trong đúng 1 tập
- $Items = \{1, 2, 3, 4, 5, 6\}$
- $Collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $Collections = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$
- Hai toán tử hiệu quả: $Find(x)$ và $Union(x, y)$.



Union-Find

- $Items = \{1, 2, 3, 4, 5, 6\}$
- $Collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $\text{Find}(x)$ trả về phần tử đại diện của tập chứa x
 - ▶ $\text{Find}(1) = 1$
 - ▶ $\text{Find}(4) = 1$
 - ▶ $\text{Find}(3) = 5$
 - ▶ $\text{Find}(5) = 5$
 - ▶ $\text{Find}(6) = 5$
 - ▶ $\text{Find}(2) = 2$
- a và b thuộc cùng một tập khi và chỉ khi $\text{Find}(a) == \text{Find}(b)$



Cài đặt Union-Find

- Hợp nhất nhanh với kỹ thuật nén đường (Quick Union with path compression)
- Cực kỳ dễ cài đặt
- Cực kỳ hiệu quả

```
1 struct Union_Find {
2     vector<int> iParent;
3     Union_Find(int n) {
4         iParent = vector<int>(n);
5         for (int i = 0; i < n; ++i) {
6             iParent[i] = i;
7         }
8     }
9     // toan tu Find va Union
10};
```



Ứng dụng của Union-Find

- Union-Find quản lý các tập không giao nhau
- Xử lý từng loại các tập không giao nhau tùy thời điểm
- Các bài toán ứng dụng quen thuộc thường trên đồ thị



Union-Find

- $Items = \{1, 2, 3, 4, 5, 6\}$
- $Collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $\text{Union}(x, y)$ trộn tập chứa x và tập chứa y vào nhau
 - ▶ $\text{union}(4, 2)$
 - ▶ $Collections = \{1, 2, 4\}, \{3, 5, 6\}$
 - ▶ $\text{union}(3, 6)$
 - ▶ $Collections = \{1, 2, 4\}, \{3, 5, 6\}$
 - ▶ $\text{union}(2, 6)$
 - ▶ $Collections = \{1, 2, 3, 4, 5, 6\}$

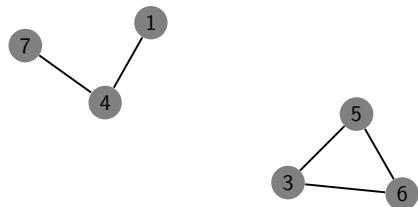


Cài đặt Union-Find

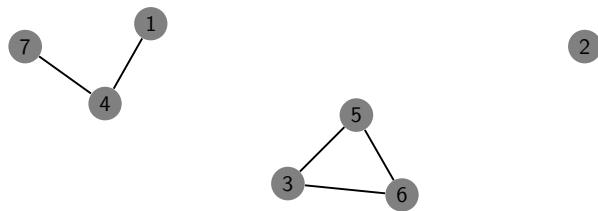
```
1 // toan tu Find va Union
2
3 int Find(int x) {
4     if (iParent[x] == x) {
5         return x;
6     } else {
7         iParent[x] = Find(iParent[x]); //Nen parent
8         return iParent[x];
9     }
10}
11
12 void Unite(int x, int y) {
13     iParent[Find(x)] = Find(y);
14}
```



Union-Find trên đồ thị

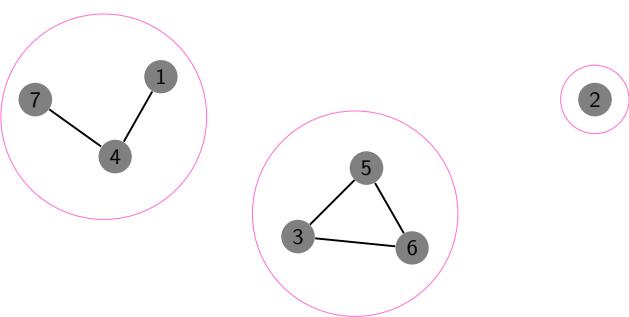


Union-Find trên đồ thị



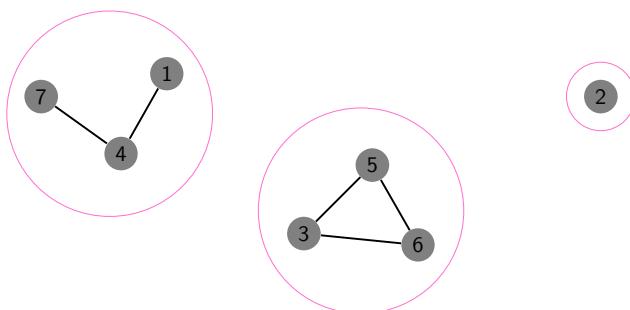
- $items = \{1, 2, 3, 4, 5, 6, 7\}$

Union-Find trên đồ thị



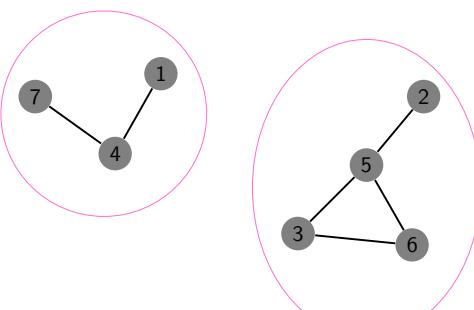
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$

Union-Find trên đồ thị



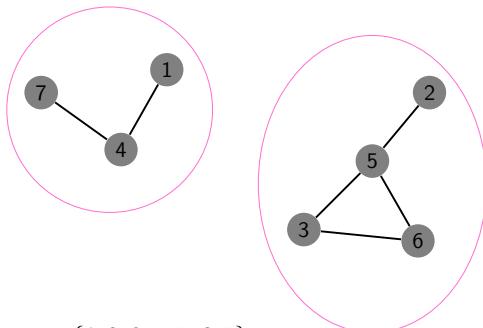
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$
- $union(2, 5)$

Union-find trên đồ thị



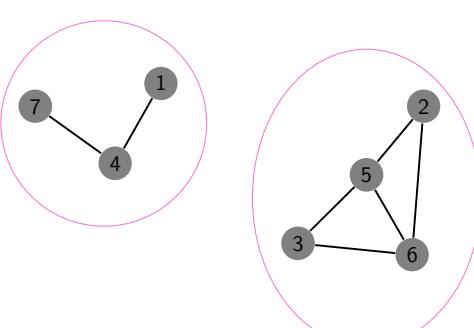
- $Items = \{1, 2, 3, 4, 5, 6, 7\}$
- $Collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

Union-Find trên đồ thị



- $Items = \{1, 2, 3, 4, 5, 6, 7\}$
- $Collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$
- $Union(6, 2)$

Union-Find trên đồ thị



- $Items = \{1, 2, 3, 4, 5, 6, 7\}$
- $Collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

1 Đồ thị có trọng số

2 Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3 Cây khung nhỏ nhất - MST

- Kruskal
- Prim

4 Đường đi ngắn nhất

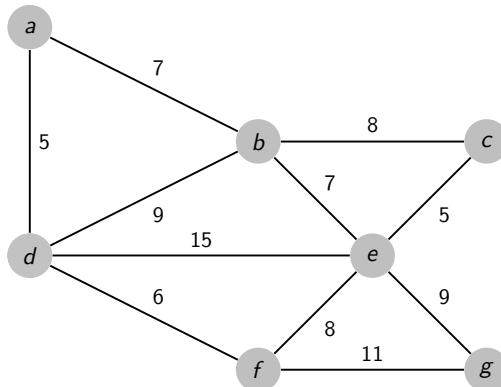
5 Một số bài toán kinh điển trên đồ thị

6 Một số đồ thị đặc biệt

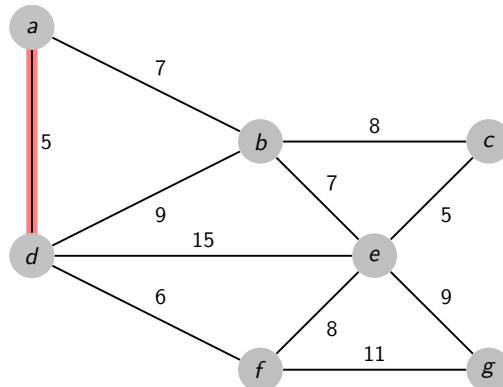
Cây khung nhỏ nhất - MST

- Cho đồ thị vô hướng có trọng số $G = (V, E)$
- $T = (V, F)$ với $F \subset E$ gọi là cây khung của G nếu như T là một cây (nghĩa là T không chứa chu trình và liên thông)
- Trọng số của T là tổng trọng số các cạnh thuộc F
- Bài toán đặt ra là tìm T có trọng số nhỏ nhất

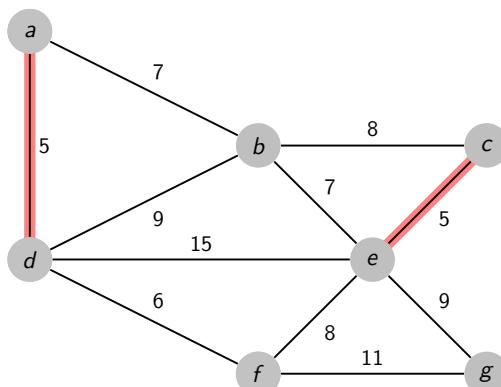
Demo thuật toán Kruskal



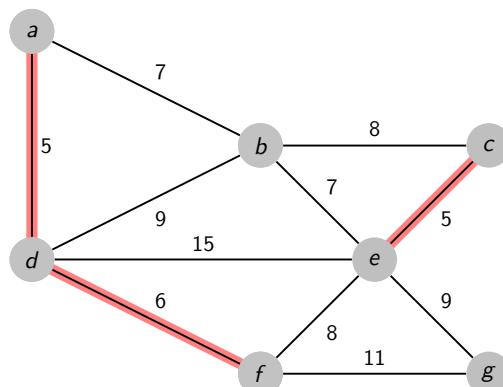
Demo thuật toán Kruskal



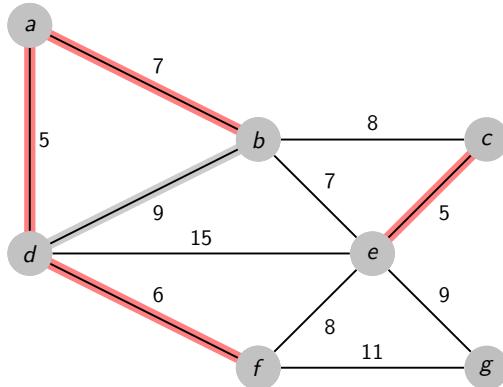
Demo thuật toán Kruskal



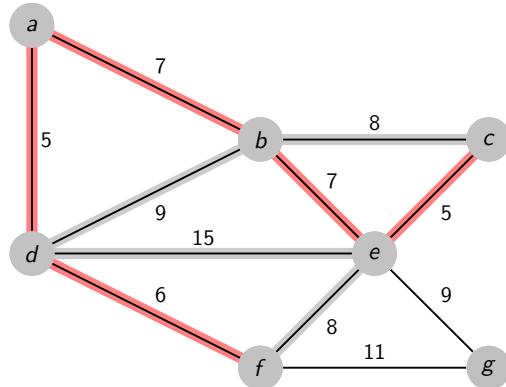
Demo thuật toán Kruskal



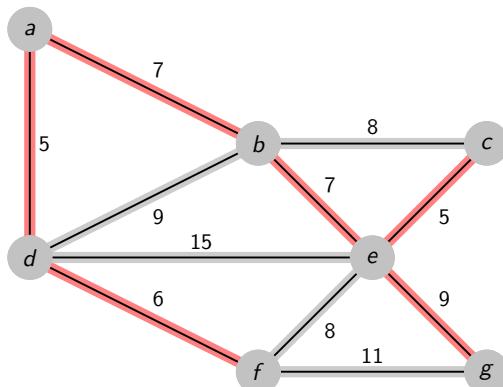
Demo thuật toán Kruskal



Demo thuật toán Kruskal



Demo thuật toán Kruskal



Hai vấn đề quan trọng khi cài đặt thuật toán Kruskal:

- ➊ Làm thế nào để xét được các cạnh từ cạnh có trọng số nhỏ tới cạnh có trọng số lớn? Ta có thể thực hiện bằng cách **sắp xếp danh sách cạnh** theo thứ tự không giảm của trọng số, sau đó duyệt từ đầu đến cuối danh sách cạnh \Rightarrow Có thể sử dụng thuật toán HeapSort cho phép chọn lần lượt các cạnh từ cạnh trọng số nhỏ nhất tới cạnh trọng số lớn nhất ra khỏi Heap.
- ➋ Làm thế nào kiểm tra xem việc thêm một cạnh có tạo thành chu trình đơn trong T hay không? Để ý rằng các cạnh trong T ở các bước sẽ tạo thành một rừng (đồ thị không có chu trình đơn). Vì vậy muốn thêm một cạnh (u, v) vào T mà không tạo thành chu trình đơn thì (u, v) phải **nối hai cây khác nhau** của rừng T , bởi nếu u, v thuộc cùng một cây thì sẽ tạo thành chu trình đơn trong cây đó. Ban đầu, khởi tạo rừng T gồm n cây, mỗi cây chỉ gồm đúng một đỉnh \Rightarrow mỗi khi xét đến cạnh nối hai cây khác nhau của rừng T thì ta sẽ kết nạp cạnh đó vào $T \Rightarrow$ hợp nhất hai cây đó lại thành một cây \Rightarrow Sử dụng kỹ thuật Union-Find

Thuật toán Kruskal

- Bài toán có thể giải bằng thuật toán tham lam sau
- Khởi tạo $F = \emptyset$
- Duyệt lần lượt các cạnh của đồ thị theo chiều tăng dần của trọng số
- Kết nạp vào T nếu như cạnh đó không tạo ra chu trình với các cạnh đã có trong T (có thể sử dụng Union-Find để lưu vết kiểm tra chu trình)
- Khi duyệt qua hết một lượt các cạnh ta sẽ thu được cây khung nhỏ nhất hoặc xác định không tồn tại cây khung của đồ thị
- Độ phức tạp $O(|E| \log |V|)$

Thuật toán Kruskal: Code

```

1  bool Edge_Cmp(const edge &a, const edge &b) {
2      return a.weight < b.weight;
3  }
4
5  vector<edge> mst(int n, vector<edge> edges) {
6      union_find uf(n);
7      sort(Edges.begin(), Edges.end(), Edge_cmp);
8      vector<edge> vRes;
9      for (int i = 0; i < Edges.size(); ++i) {
10         int u = Edges[i].u,
11             v = Edges[i].v;
12         if (uf.Find(u) != uf.Find(v)) {
13             uf.Unite(u, v);
14             res.push_back(Edges[i]);
15         }
16     }
17     return res;
18 }
```

1 Đồ thị có trọng số

2 Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3 Cây khung nhỏ nhất - MST

- Kruskal
- Prim

4 Đường đi ngắn nhất

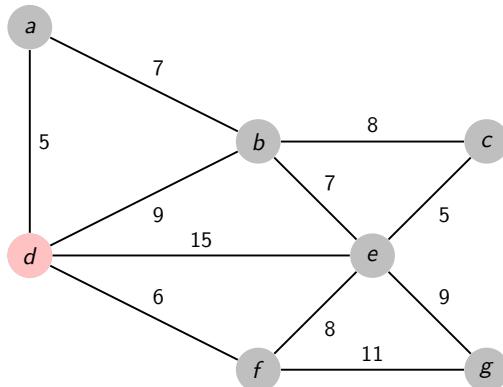
5 Một số bài toán kinh điển trên đồ thị

6 Một số đồ thị đặc biệt

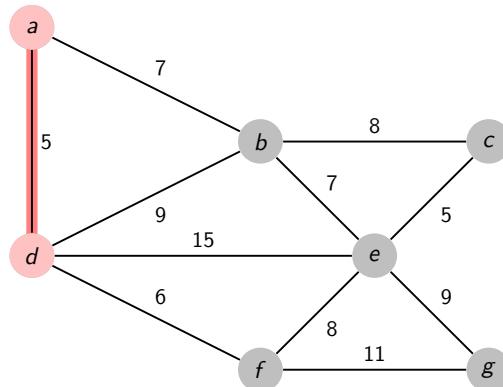
Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị dày (đồ thị với số cạnh $m \geq n(n-1)/2$). Trong trường hợp đó thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn gọi là phương pháp lân cận gần nhất.

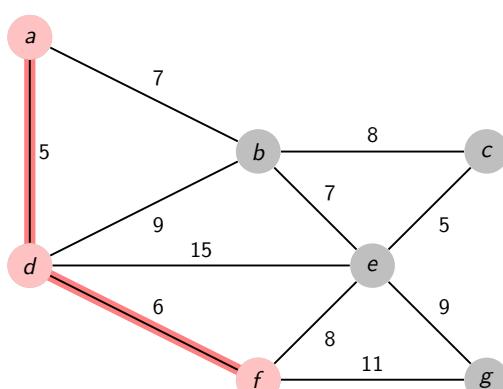
Demo thuật toán Prim



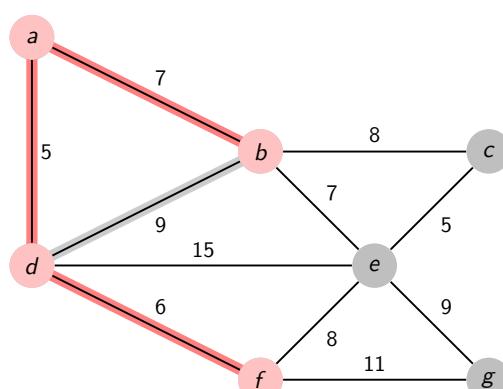
Demo thuật toán Prim



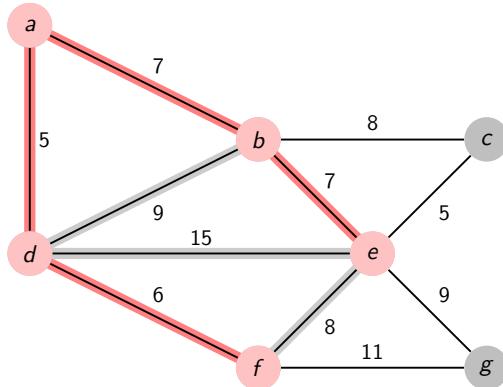
Demo thuật toán Prim



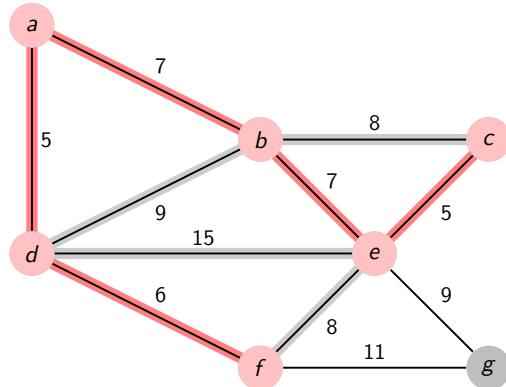
Demo thuật toán Prim



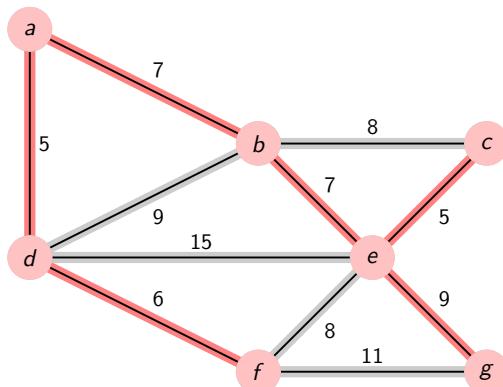
Demo thuật toán Prim



Demo thuật toán Prim



Demo thuật toán Prim



Thuật toán Prim: Mô tả

- B1: Chọn tùy ý một đỉnh s , khởi tạo $V_T = \{s\}$, $E_T = \emptyset$
- B2: Nếu $|E_T| = |V| - 1$ thì đưa ra cây $T = (V_T, E_T)$, kết thúc thuật toán
- B3: Chọn cạnh $e = (u, v, w)$ có $u \in V_T$, $v \notin V_T$ với w nhỏ nhất
- B4: Nạp đỉnh v và cạnh e vào cây: $V_T = V_T \cup \{v\}$, $E_T = E_T \cup \{e\}$. Quay lại B2
- Kết thúc thuật toán nếu chưa nạp được hết n đỉnh thì đồ thị đã cho không liên thông, đồ thị G không tồn tại cây khung
- Độ phức tạp $O(\min(|V|^2, (|V| + |E|) \log |V|))$

Thuật toán Prim: Kỹ thuật cài đặt

- Sử dụng mảng đánh dấu bIn_T . $bIn_T[v] = false$ nếu như đỉnh v chưa được kết nạp vào T .
 - Gọi $iBest_W[v]$ là khoảng cách từ v tới T . Ban đầu khởi tạo:
 - $iBest_W[1] = 0$
 - $iBest_W[2] = iBest_W[3] = \dots = iBest_W[n] = +\infty$
 - Gọi $iBest_A[v]$ là đỉnh gần v nhất của cây khung
 - Tại mỗi bước chọn đỉnh đưa vào T , ta sẽ chọn đỉnh u nào ngoài T và có $iBest_W[u]$ nhỏ nhất.
- Khi kết nạp u vào T rồi thì các nhãn $iBest_W[v]$ sẽ thay đổi nếu khoảng cách từ u đến v nhỏ hơn $iBest_W[v]$ hiện tại

Thuật toán Prim: Cài đặt $O(|V|^2)$

```

1  vector<edge> Prim(int sn, vector<vector<edge>> Adj) {
2    vector<bool> bIn_T(n+1, false); // Tap_Dinh_MST
3    vector<edge> res; // Tap_Canh_MST
4    vector<int> iBest_W(n+1, 1e9), iBest_A(n+1, -1);
5    iBest_W[1] = 0;
6    while (res.size() < n-1){
7      int u = -1, v = -1, w = 1e9;
8      for (int x = 1; x <= n; ++x)
9        if (bIn_T[x] == false && iBest_W[x] < w){
10          u = iBest_A[x], v = x, w = iBest_W[x];
11        }
12      if (v == -1) return res; // Do_Thi_Khong_Lien_Thong
13      bIn_T[v] = true;
14      for (edge e : Adj[v])
15        if (iBest_W[e.v] > e.weight){
16          iBest_W[e.v] = e.weight;
17          iBest_A[e.v] = e.u;
18        }
19      if (v != 1) res.push_back({u, v, w});
20    }
21    return res;
22  }

```

Thuật toán Prim: Cài đặt ($|V| + |E|$) log $|V|$)

```
1 vector<edge> mst(int n, vector<vector<edge>> Adj) {
2     priority_queue< pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > PQ;
3     vector<edge> res; // Tap_Canh_MST
4     vector<int> iBest_W(n+1, 1e9), iBest_A(n+1, -1);
5     iBest_W[1] = 0;
6     PQ.push({iBest_W[1], 1});
7     while (res.size() < n-1){
8         while (!PQ.empty()) &&
9             PQ.top().first != iBest_W[PQ.top().second]) PQ.pop();
10        if (PQ.empty()) return res; // G_Khong_Lien_Thong
11        int w=PQ.top().first, v=PQ.top().second, u=iBest_A[v];
12        for (edge e : Adj[v]){
13            if (iBest_W[e.v] > e.weight){
14                iBest_W[e.v] = e.weight;
15                iBest_A[e.v] = e.u;
16                PQ.push({iBest_W[e.v], e.v});
17            }
18            if (v != 1) res.push_back({u, v, w});
19        }
20    }
21    return res;
22 }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Dường đi ngắn nhất: Bài toán

- Cho đồ thị có trọng số $G = (V, E)$ (vô hướng hoặc có hướng)
- Cho hai đỉnh u, v , hãy tìm đường đi ngắn nhất từ u đến v ?
- Nếu tất cả trọng số trên các cạnh đều bằng nhau, bài toán có thể giải bằng tìm kiếm theo chiều rộng BFS
- Tất nhiên là đại đa số các trường hợp không như vậy...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật toán Dijkstra

```
1 vector<edge> Adj[100];
2 vector<int> iDist(100, INF);
3 void Dijkstra(int start) {
4     iDist[start] = 0;
5     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > PQ;
6     PQ.push(make_pair(iDist[start], start));
7     while (!PQ.empty()) {
8         int u = PQ.top().second;
9         PQ.pop();
10        for (int i = 0; i < Adj[u].size(); i++) {
11            int v = Adj[u][i].v;
12            int w = Adj[u][i].weight;
13            if (w + iDist[u] < iDist[v]) {
14                iDist[v] = w + iDist[u];
15                PQ.push(make_pair(iDist[v], v));
16            }
17        }
18    }
19 }
20 }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1 Đồ thị có trọng số

2 Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3 Cây khung nhỏ nhất – MST

4 Đường đi ngắn nhất

- Dijkstra
- Bellman-Ford
- Floyd-Warshall

5 Một số bài toán kinh điển trên đồ thị

6 Một số đồ thị đặc biệt

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Dường đi ngắn nhất: Bài toán

- Có rất nhiều thuật toán hiện biết giải bài toán tìm đường đi ngắn nhất
- Cũng giống như thuật toán tìm kiếm theo chiều rộng BFS, các thuật toán này tìm đường đi ngắn nhất từ đỉnh xuất phát đến tất cả các đỉnh còn lại
- Các thuật toán phổ biến và hiệu quả bao gồm: Dijkstra, Bellman-Ford, và Floyd-Warshall

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật toán Dijkstra

- Độ phức tạp thuật toán $O(|E| \log |V|)$
- Lưu ý là thuật toán chỉ đúng trong trường hợp trọng số không âm

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật toán Bellman-Ford

```
1 void Bellman_Ford(int n, int start) {  
2  
3     iDist[start] = 0;  
4     for (int k = 1; k < n - 1; ++k) {  
5         for (int u = 1; u <= n; ++u) {  
6             for (int j = 0; j < Adj[u].size(); ++j) {  
7                 int v = Adj[u][j].v;  
8                 int w = Adj[u][j].weight;  
9                 iDist[v] = min(iDist[v], w + iDist[u]);  
10            }  
11        }  
12    }  
13}
```

Thuật toán Bellman-Ford

- Độ phức tạp $O(|V| \times |E|)$
- Có thể sử dụng để tìm ra các chu trình trọng số âm

Thuật toán Floyd-Warshall

Sử dụng phương pháp Qui hoạch động:

- Gọi $DP(k, i, j)$ là trọng số đường đi ngắn nhất từ i đến j nếu như chỉ cho phép đi trong những đỉnh $0, \dots, k$
- Điều kiện biên 1: $DP(k, i, j) = 0$ nếu $i = j$
- Điều kiện biên 2: $DP(-1, i, j) = \text{Weight}[a][b]$ nếu $(i, j) \in E$
- Điều kiện biên 3: $DP(-1, i, j) = \infty$
- $DP(k, i, j) = \min \begin{cases} DP(k-1, i, k) + DP(k-1, k, j) \\ DP(k-1, i, j) \end{cases}$

Thuật toán Floyd-Warshall

```
1 int iDist[1001][1001];  
2 int Weight[1001][1001];  
3 void Floyd_Warshall(int n) {  
4     for (int i = 0; i < n; ++i) {  
5         for (int j = 0; j < n; ++j) {  
6             iDist[i][j] = i == j ? 0 : Weight[i][j];  
7         }  
8     }  
9     for (int k = 0; k < n; ++k) {  
10        for (int i = 0; i < n; ++i) {  
11            for (int j = 0; j < n; ++j) {  
12                dist[i][j] =  
13                    min(dist[i][j], dist[i][k] + dist[k][j]);  
14            }  
15        }  
16    }  
17}
```

Thuật toán Floyd-Warshall

- Tính toàn bộ các đường đi ngắn nhất giữa các cặp đỉnh
- Độ phức tạp $O(|V|^3)$
- Dễ cài đặt

① Đồ thị có trọng số

② Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

③ Cây khung nhỏ nhất – MST

④ Đường đi ngắn nhất

⑤ Một số bài toán kinh điển trên đồ thị

- Bài toán phủ đỉnh - MVC
- Bài toán tập độc lập - MIS
- Bài toán ghép cặp - Matching

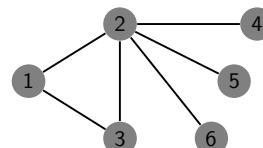
⑥ Một số đồ thị đặc biệt

Một số bài toán kinh điển trên đồ thị

- Đây là những bài toán quan trọng và kinh điển trên đồ thị
- Thường là những bài toán khó trên đồ thị tổng quát
- Ứng dụng rộng rãi trong các bài toán thực tế khi các đồ thị tương ứng có tính chất đặc biệt
- Cũng là những bài toán cơ bản rất hay được sử dụng làm đề bài trong các kỳ thi
- Thường xuyên được ẩn chứa kín trong phát biểu bài toán

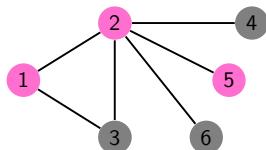
Bài toán phủ đỉnh - MVC

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S , sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



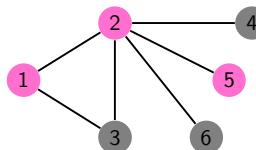
Bài toán phủ đỉnh - MVC

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S , sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



Bài toán phủ đỉnh - MVC

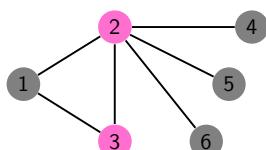
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S , sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất

Bài toán phủ đỉnh - MVC

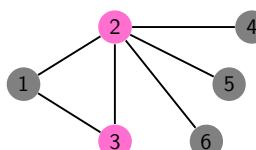
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S , sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất

Bài toán phủ đỉnh - MVC

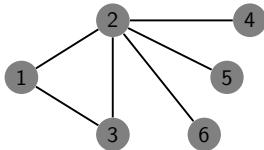
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S , sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất
- Đây là bài toán NP-khó trên đồ thị tổng quát

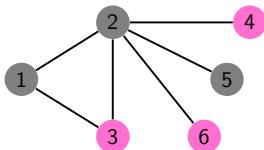
Bài toán tập độc lập - MIS

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S , sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



Bài toán tập độc lập - MIS

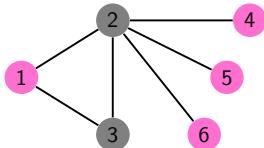
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S , sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất

Bài toán tập độc lập - MIS

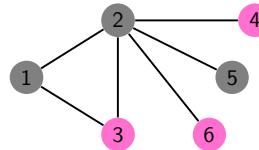
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S , sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất
- Đây là bài toán NP-khó trên đồ thị tổng quát

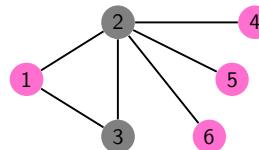
Bài toán tập độc lập - MIS

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S , sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



Bài toán tập độc lập - MIS

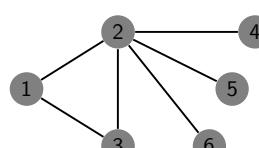
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S , sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất

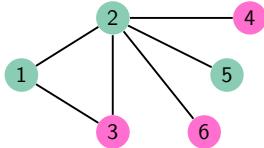
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



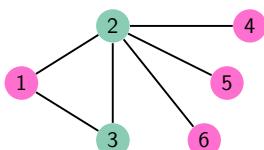
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



Mối quan hệ giữa MVC và MIS

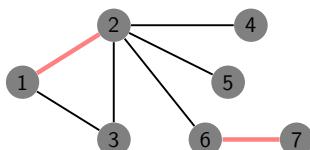
- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



- Lực lượng của một phủ tập có kích thước nhỏ nhất cộng với lực lượng của tập độc lập có kích thước lớn nhất bằng tổng số đỉnh của đồ thị

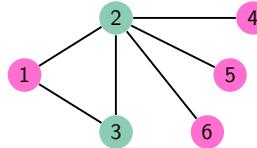
Bài toán ghép cặp

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một cặp ghép là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



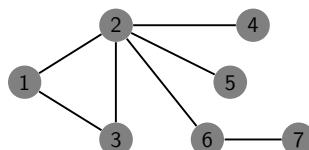
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



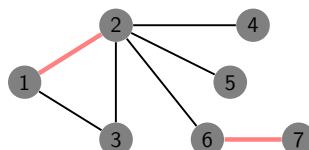
Bài toán ghép cặp

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một cặp ghép là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



Bài toán ghép cặp

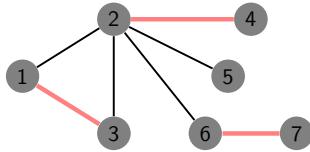
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một cặp ghép là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



- Hãy tìm một cặp ghép có lực lượng lớn nhất

Bài toán ghép cặp

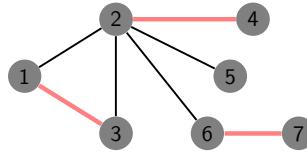
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một cặp ghép là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



- Hãy tìm một cặp ghép có lực lượng lớn nhất

Bài toán ghép cặp

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một cặp ghép là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F

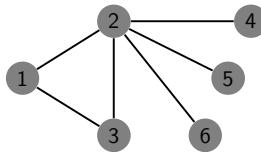


- Hãy tìm một cặp ghép có lực lượng lớn nhất

- Có thuật toán $O(|V|^4)$ cho đồ thị tổng quát, nhưng cài đặt khá phức tạp

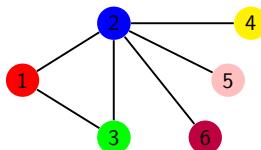
Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Bài toán tô màu đồ thị yêu cầu gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



Bài toán tô màu đồ thị

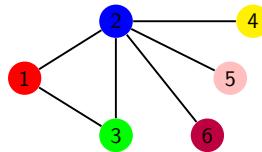
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Bài toán tô màu đồ thị yêu cầu gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất, hay còn gọi tìm *sắc số* của đồ thị

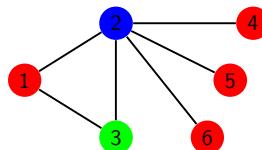
Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Bài toán tô màu đồ thị yêu cầu gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



Bài toán tô màu đồ thị

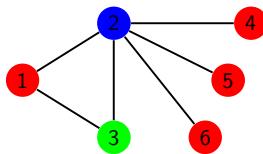
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Bài toán tô màu đồ thị yêu cầu gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất, hay còn gọi tìm *sắc số* của đồ thị

Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Bài toán tô màu đồ thi yêu cầu gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



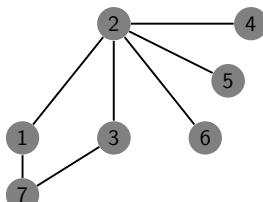
- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất, hay còn gọi là **sắc số** của đồ thị
- Đây là bài toán NP-khó trên đồ thị tổng quát

Một số đồ thị đặc biệt

- Hầu hết các bài toán trên đều là NP-khó trong trường hợp đồ thị tổng quát
- Còn trên một số loại đồ thị đặc biệt thì sao?
- Xét một số ví dụ

Đồ thị hai phía

- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

1. Đồ thị có trọng số

2. Cấu trúc dữ liệu các tập không giao nhau – UNION-FIND

3. Cây khung nhỏ nhất - MST

4. Đường đi ngắn nhất

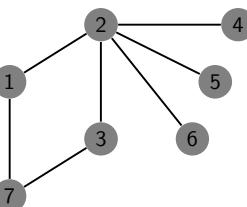
5. Một số bài toán kinh điển trên đồ thị

6. Một số đồ thị đặc biệt

- Đồ thị hai phía
- Cây
- Đồ thị có hướng không có chu trình - DAG

Đồ thị hai phía

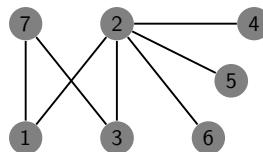
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

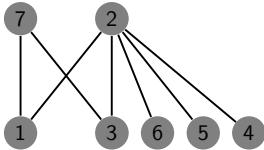
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phia

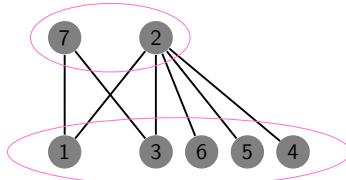
- Một đồ thị là hai phia nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phia?

Đồ thị hai phia

- Một đồ thị là hai phia nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phia?

Đồ thị hai phia

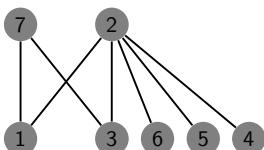
- Cần phải kiểm tra xem ta có thể chia tập đỉnh thành hai nhóm
- Lấy một đỉnh bất kỳ, giả sử nó thuộc nhóm 1
- Sau đó tất cả đỉnh kề với nó sẽ thuộc nhóm 2
- Tiếp theo tất cả đỉnh kề với các đỉnh nhóm 2 đó đều phải thuộc nhóm 1
- Và cứ kiểm tra như vậy...
- Ta có thể thực hiện với thuật toán tìm kiếm theo chiều sâu DFS
- Nếu thấy điều vô lý xảy ra (nghĩa là một đỉnh phải nằm trong cả hai nhóm 1 và 2), thì đồ thị đó không phải là đồ thị hai phia

Đồ thị hai phia

```
1 vector<int> Adj[1000];
2 vector<int> iSide(1001, -1);
3 bool is_bipartite = true;
4 void Check_Bipartite(int u) {
5     for (int i = 0; i < Adj[u].size(); ++i) {
6         int v = Adj[u][i];
7         if (iSide[v] == -1) {
8             iSide[v] = 1 - iSide[u];
9             Check_Bipartite(v);
10        } else if (iSide[u] == iSide[v]) {
11            is_bipartite = false;
12        }
13    }
14 }
15 int main() {
16     for (int u = 0; u < n; u++) {
17         if (iSide[u] == -1) {
18             iSide[u] = 0;
19             Check_Bipartite(u);
20         }
21     }
22     return 0;
23 }
```

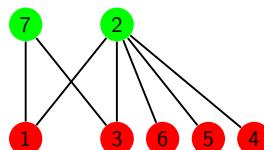
Bài toán tô màu trên đồ thị hai phia

- Làm thế nào để tô đỉnh bởi ít màu nhất trên đồ thị hai phia?



Bài toán tô màu trên đồ thị hai phia

- Làm thế nào để tô đỉnh bởi ít màu nhất trên đồ thị hai phia?



- Rất đơn giản, một phia tô bởi một màu, và phia kia tô bởi một màu khác

Bài toán ghép cặp trên đồ thị hai phía

- Bài toán ghép cặp trên đồ thị hai phía rất quen thuộc
- [xem ví dụ](#)
- Lưu ý là thuật toán hiệu quả tìm cặp ghép lớn nhất trên đồ thị tổng quát là phức tạp

Định lý König

- Định lý König chỉ ra rằng lực lượng của một phủ đỉnh nhỏ nhất trên đồ thị hai phía bằng với lực lượng của cặp ghép lớn nhất trên đồ thị đó
- Do đó, để tìm một phủ đỉnh nhỏ nhất trên đồ thị hai phía, ta chỉ cần tìm ghép cặp lớn nhất với thuật toán hiệu quả quen thuộc
- Và do lực lượng của tập độc lập lớn nhất chính là tổng số đỉnh của đồ thị trừ đi lực lượng của phủ đỉnh nhỏ nhất, nên ta có thể tính được tập độc lập lớn nhất một cách hiệu quả

Cây

- Cây là đồ thị vô hướng liên thông không có chu trình
- Dễ dàng kiểm tra một đồ thị là cây bằng cách kiểm tra có tồn tại cạnh ngược hay không trên cây DFS
- Một cây với n đỉnh có chính xác $n - 1$ cạnh
- Giữa mỗi cặp đỉnh u, v trên cây tồn tại duy nhất một đường đi đơn, có thể sử dụng DFS hoặc BFS để tìm đường đi này

Cây

- Các bài toán áp dụng trên cây thế nào?
- Tìm sắc số đỉnh trên cây?

Cây

- Các bài toán áp dụng trên cây thế nào?
- Tìm sắc số đỉnh trên cây?
- Thực chất cây cũng giống như đồ thị hai phía...
- Vì sao? Lấy một đỉnh bất kỳ và coi đỉnh đó là gốc của cây. Tiếp theo các đỉnh có chiều cao chẵn thì cho thuộc về một phía, còn các đỉnh chiều cao lẻ thì cho thuộc vào phía còn lại
- Vì vậy tất cả các thuật toán hiệu quả trên đồ thị hai phía đều đúng cho cây

Cây cũng rất thích hợp cho các thuật toán quy hoạch động, vì vậy rất nhiều bài toán trở nên dễ dàng hơn nhiều trên cây vì lý do đó

Đồ thị có hướng không có chu trình - DAG

- Một đồ thị có hướng là một DAG nếu như nó không chứa chu trình
- Dễ dàng kiểm tra một đồ thị là DAG bằng cách kiểm tra xem có cạnh ngược nào không trên cây DFS
- Rất nhiều bài toán trở nên dễ dàng ở trên DAG, do có thể áp dụng quy hoạch động nhờ tính chất không có chu trình trên DAG
 - ▶ tính số lượng đường đi đơn từ u đến v
 - ▶ đường đi dài nhất từ u đến v

Thank you for
your attentions!



25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thuật Toán Tham Lam

THUẬT TOÁN ỨNG DỤNG

- 1 Sơ đồ thuật toán tham lam
- 2 Bài toán đổi tiền
- 3 Bài toán cái túi
- 4 Tập đoạn thẳng không giao nhau

Các mô hình giải bài căn bản

Các phương pháp căn bản xây dựng lời giải bài toán:

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

- ① Sơ đồ thuật toán tham lam
 - Giới thiệu chung
 - Sơ đồ chung
 - Chứng minh tính tối ưu
- ② Bài toán đổi tiền
- ③ Bài toán cái túi
- ④ Tập đoạn thẳng không giao nhau

Thuật toán tham lam

- Các thuật toán tham lam là một phương pháp tiếp cận căn bản để xây dựng thuật toán giải hầu hết các dạng bài toán khác nhau
- Tại mỗi bước, quyết định được đưa ra dựa ngay vào thông tin đang có, và trong tương lai sẽ không xem xét lại tác động của các quyết định đã nhận trong quá khứ
- Do đó các thuật toán tham lam rất dễ đề xuất, và thông thường không đòi hỏi nhiều thời gian tính. Tuy nhiên, các thuật toán dạng này thường không cho kết quả tối ưu !
- Ứng dụng đưa ra lời giải chấp nhận được cho các bài toán tối ưu trong thực tế có độ khó cao

Sơ đồ chung: Mô tả bài toán

- Lời giải cần tìm có thể mô tả như là bộ gồm hữu hạn các thành phần (x_1, x_2, \dots, x_n) thoả mãn các điều kiện bài toán
- Thông thường giải quyết bài toán tối ưu: Tìm lời giải với giá trị hàm mục tiêu lớn nhất (hoặc nhỏ nhất)
- Xác định tập các ứng cử viên có thể lựa chọn làm các thành phần của lời giải

Sơ đồ chung: Thuật toán

GREEDY-ALGORITHM()

```
1 // C: tập các ứng viên
2 S = ∅; // S: lời giải xây dựng theo thuật toán
3 while (C ≠ ∅) && (Solution(S)==false) {
4     x ← Select(C);
5     C = C \ x;
6     if (Feasible(S ∪ x)==true)
7         S = S ∪ x;
8 }
9 if (Solution(S)==true)
10    return S;
```

Một số bài toán điển hình của thuật toán tham lam

- Bài toán đổi tiền
 - Đổi giá trị tiền x bởi các đồng mệnh trị 1, 5, 10, 25
 - Đổi với các mệnh giá khác thì sao? Ví dụ 1, 5, 7
- Bài toán cây khung nhỏ nhất
 - Thuật toán Prim:
 - Xuất phát từ cây là một đỉnh bất kỳ, mỗi bước bổ sung vào cây hiện tại đỉnh gần cây nhất
 - Thuật toán Kruskal:
 - Sắp xếp các cạnh theo trọng số giảm dần
 - Mỗi bước bổ sung vào tập đang xây dựng một cạnh theo thứ tự sắp xếp mà không tạo ra chu trình trong tập
- Bài toán đường đi ngắn nhất: Thuật toán Dijkstra:
 - Mỗi bước cố định đỉnh có nhãn nhỏ nhất
- Mã Huffman

Sơ đồ chung

- Xuất phát từ lời giải rỗng, thuật toán xây dựng lời giải của bài toán theo từng bước, ở mỗi bước sẽ chọn một phần tử từ tập ứng cử viên và bổ sung vào lời giải hiện có
- Hàm Solution(S) nhận biết tính chấp nhận được của lời giải S
- Hàm Select(C) chọn từ tập C ứng cử viên có triển vọng nhất để bổ sung vào lời giải hiện có
- Hàm Feasible($S ∪ x$) kiểm tra tính chấp nhận được của lời giải bộ phận $S ∪ x$

Chứng minh tính tối ưu

- Để chỉ ra thuật toán không cho lời giải tối ưu chỉ cần đưa ra một phản ví dụ: Một bộ dữ liệu mà đối với nó thuật toán không cho lời giải không tối ưu
- Chứng minh tính tối ưu của thuật toán khó hơn nhiều

Lập luận biến đổi (Exchange Argument)

Giả sử cần chứng minh thuật toán A cho lời giải tối ưu:

- Gọi $A(I)$ là lời giải tìm được bởi thuật toán A đối với bộ dữ liệu I , còn O là lời giải tối ưu của bài toán đối với bộ dữ liệu này
- Cần tìm cách xây dựng phép biến đổi ϕ cho phép biến đổi O thành lời giải O' sao cho
 - O' cũng tốt không kém gì O (nghĩa là O' vẫn là tối ưu);
 - O' **giống** với $A(I)$ nhiều hơn O

Lập luận biến đổi: Chứng minh trực tiếp

Giả sử có O là lời giải tối ưu, ta biến đổi O thành lời giải tối ưu O' giống với $A(I)$ hơn là O :

- Nếu $O' = A(I)$ thì $A(I)$ chính là lời giải tối ưu;
- Ngược lại, ta lại lặp lại phép biến đổi đối với O' để thu được lời giải tối ưu O'' giống với $A(I)$ hơn là O' ,
- ...
- Ta thu được dãy O', O'', O''', \dots ;
- Mỗi phần tử của dãy này đều là phương án tối ưu, và mỗi phần tử sau lại giống với $A(I)$ hơn phần tử trước nó
 \Rightarrow Dãy này phải kết thúc ở $A(I)$. Vậy $A(I)$ cũng là tối ưu !

Đổi tiền

Bài toán

Có các đồng tiền mệnh giá: 1, 5, 10, 25, 100 (xu), hãy tìm phương pháp đổi một lượng tiền x sử dụng một số lượng ít nhất các đồng tiền



25¢ 5¢ 1¢ 1¢ 1¢ 1¢

Hình: Đổi 34¢

Lập luận biến đổi: Chứng minh bằng phản chứng

Giả sử A không đúng đắn, khi đó phải tìm được bộ dữ liệu I sao cho $A(I)$ khác với lời giải tối ưu của bài toán:

- Gọi O là lời giải tối ưu giống với $A(I)$ nhất;
- Do $A(I)$ không là lời giải tối ưu nên $A(I) \neq O$;
- Khi đó sử dụng phép biến đổi ϕ ta có thể biến đổi O thành O' sao cho: O' vẫn là tối ưu đồng thời O' giống với $A(I)$ hơn là O
 \Rightarrow Mâu thuẫn với giả thiết O là lời giải tối ưu giống với $A(I)$ nhất !

1 Sơ đồ thuật toán tham lam

2 Bài toán đổi tiền

- Bài toán
- Thuật toán
- Chứng minh tính tối ưu

3 Bài toán cái túi

4 Tập đoạn thẳng không giao nhau

Đổi tiền: Thuật toán

Thuật toán Người Thu Ngân

Ở mỗi bước, trả đồng tiền mệnh giá lớn nhất không vượt quá lượng tiền cần đổi



1\$ 25¢ 10¢ 1¢

Hình: Đổi 2.89\$

Đổi tiền: Thuật toán

CASHIERS-ALGORITHM(x, C_1, C_2, \dots, C_n)

```

1 SORT( $n$  đồng tiền theo thứ tự:  $C_1 < C_2 < \dots < C_n$ )
2  $S \leftarrow \emptyset$  //  $S$  là tập các đồng tiền được chọn
3 while ( $x > 0$ ) {
4      $k \leftarrow$  đồng tiền mệnh giá lớn nhất  $c_k$  sao cho  $c_k \leq x$ 
5     if (không tìm được  $k$ )
6         return "no solution"
7     else
8          $x \leftarrow x - c_k$ 
9          $S \leftarrow S \cup \{k\}$ 
10    return  $S$ 
11 }
```

Thuật toán Người Thu Ngân có cho cách đổi tiền tối ưu không?



18 / 42

Đổi tiền: Các tính chất của lời giải tối ưu

Bổ đề 4: Số lượng đồng 5¢ + Số lượng đồng 10¢ ≤ 2

CM:

- Thay ba đồng 10¢ và không đồng 5¢ bởi 1 đồng 25¢ và một đồng 5¢;
- Thay hai đồng 10¢ và một đồng 5¢ bởi một đồng 25¢;
- Nhắc lại Bổ đề 2: có tối đa một đồng 5¢.

k	c_k	Tất cả lời giải tối ưu phải thoả mãn	Giá trị tối đa với các mệnh giá $1, 2, \dots, k-1$ trong bất kỳ OPT
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N+D \leq 2$	$4+5=9$
4	25	$Q \leq 3$	$20+4=24$
5	100	không giới hạn	$75+24=99$



20 / 42

Đổi tiền: Mở rộng

Câu hỏi: Thuật toán NTN có là tối ưu cho dãy mệnh giá khác?

Trả lời:

- KHÔNG. Thuật toán NTN không cho lời giải tối ưu với các mệnh giá tem của Bưu chính Hoa Kỳ: 1,10,21,34,70,100,350,1225,1500. **Phản ví dụ:** 140¢
 - Thuật toán NTN: $140=100+34+1+1+1+1+1+1$.
 - Lời giải tối ưu: $140=70+70$.
- KHÔNG. Thậm chí thuật toán NTN có thể không tìm được lời giải chấp nhận được nếu $c_1 > 1$: 7, 8, 9. **Phản ví dụ:** 15¢
 - Thuật toán NTN: $15=9+???$.
 - Lời giải tối ưu: $15=7+8$.



22 / 42

Đổi tiền: Các tính chất của lời giải tối ưu

Bổ đề 1: Số lượng đồng 1¢ ≤ 4.

CM: Thay năm đồng 1¢ bởi một đồng 5¢



Bổ đề 2: Số lượng đồng 5¢ ≤ 1

CM: Thay hai đồng 5¢ bởi một đồng 10¢

Bổ đề 3: Số lượng đồng 25¢ ≤ 3.

CM: Thay ba đồng 25¢ bởi một đồng 1\$



19 / 42

Đổi tiền: Chứng minh tính tối ưu

Dịnh lí

Thuật toán Người Thu Ngân (NTN) là tối ưu đối với dãy mệnh giá: 1,5,10,25,100
CM: (qui nạp theo x)

- Xét cách đổi tối ưu $c_k \leq x < c_{k+1}$: thuật toán NTN sẽ chọn đồng tiền k
- Ta khẳng định bất kì lời giải tối ưu nào cũng phải chọn đồng tiền k :
 - Nếu không, cần sử dụng các đồng tiền c_1, \dots, c_{k-1} để đổi x
 - Bảng trên đã cho thấy không tồn tại lời giải nào đổi được như vậy !
- Bài toán đưa về đổi $x - c_k$. Theo qui nạp, thuật toán NTN cho lời giải tối ưu !

k	c_k		
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N+D \leq 2$	$4+5=9$
4	25	$Q \leq 3$	$20+4=24$
5	100	không giới hạn	$75+24=99$



21 / 42

1 Sơ đồ thuật toán tham lam

2 Bài toán đổi tiền

3 Bài toán cái túi

- Bài toán
- Tham lam 1
- Tham lam 2
- Tham lam 3
- Tham lam 4

4 Tập đoạn thẳng không giao nhau



23 / 42

Bài toán cái túi

Phát biểu bài toán

- Có n đồ vật. Đồ vật i có trọng lượng W_i và giá trị C_i , $i = 1, \dots, n$.
- Yêu cầu:** Tìm cách chất các đồ vật này vào cái túi có dung lượng là b sao cho tổng trọng lượng của các đồ vật được chất vào túi là không quá b , đồng thời tổng giá trị của chúng là lớn nhất
- Ký hiệu $S = \{1, 2, \dots, n\}$ tập chỉ số các đồ vật. Bài toán đặt ra là: Tìm $I \subset S$ sao cho

$$\sum_{i \in I} W_i \leq b, \sum_{i \in I} C_i \rightarrow \max$$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 1

Phản ví dụ với bộ dữ liệu sau

- Số lượng đồ vật $n = 3$
- Trọng lượng cái túi $b = 19$

Đồ vật	1	2	3
Giá trị C_i	20	16	8
Trọng lượng W_i	14	6	10

- Greedy1 cho lời giải: $I_1 = \{1\}$ với giá trị 20
- Trong khi đó, ta có lời giải tốt hơn là $I^* = \{2, 3\}$ với giá trị 24



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 2

Phản ví dụ với bộ dữ liệu sau

- Số lượng đồ vật $n = 3$
- Trọng lượng cái túi $b = 11$

Đồ vật	1	2	3
Giá trị C_i	10	16	28
Trọng lượng W_i	5	6	10

- Greedy2 cho lời giải: $I_2 = \{1, 2\}$ với giá trị 26
- Trong khi đó, ta có lời giải tốt hơn là $I^* = \{3\}$ với giá trị 28



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 1

Greedy1: Sắp xếp các đồ vật theo thứ tự không tăng của giá trị

- Lần lượt xét các đồ vật theo thứ tự đã sắp, và xếp đồ vật đang xét vào túi nếu như dung lượng còn lại của cái túi đủ chứa nó (tức là tổng trọng lượng của các đồ vật đã xếp vào túi và trọng lượng của đồ vật đang xét là không vượt quá b)

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 2

Greedy2: Sắp xếp các đồ vật theo thứ tự không giảm của trọng lượng

- Lần lượt xét các đồ vật theo thứ tự đã sắp, và chất đồ vật đang xét vào túi nếu như dung lượng còn lại của cái túi đủ chứa nó (tức là tổng trọng lượng của các đồ vật đã xếp vào túi và trọng lượng của đồ vật đang xét là không vượt quá b)

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 3

Greedy3: Sắp xếp các đồ vật theo thứ tự không tăng của giá trị một đơn vị trọng lượng (C_i/W_i)

Nghĩa là

$$\frac{C_{i_1}}{W_{i_1}} \leq \frac{C_{i_2}}{W_{i_2}} \leq \dots \leq \frac{C_{i_n}}{W_{i_n}}$$

- Lần lượt xét các đồ vật theo thứ tự đã sắp, và chất đồ vật đang xét vào túi nếu như dung lượng còn lại của cái túi đủ chứa nó (tức là tổng trọng lượng của các đồ vật đã xếp vào túi và trọng lượng của đồ vật đang xét là không vượt quá b)

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 3

Phản ví dụ với bộ dữ liệu sau

- Số lượng đồ vật $n = 3$
- Trọng lượng cái túi $b \geq 2$

Đồ vật	1	2
Giá trị C_i	10	$10b - 1$
Trọng lượng W_i	1	b

- Rõ ràng

$$\frac{C_1}{W_1} = \frac{10}{1} \geq \frac{10b - 1}{b} = \frac{C_2}{W_2}$$

- Greedy3 cho lời giải: $I_3 = \{1\}$ với giá trị 10
- Trong khi đó, ta có lời giải tốt hơn là $I^* = \{2\}$ với giá trị $10b - 1$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 4

Phản ví dụ với bộ dữ liệu sau

- Số lượng đồ vật $n = 4$
- Trọng lượng cái túi $b = 11$

Đồ vật	1	2	3	4
Giá trị C_i	9	10	18	27
Trọng lượng W_i	4	5	6	10
C_i/W_i	2.25	2	3	2.7
Greedy <i>i</i>	27	19	27	27

- Greedy4 cho lời giải với giá trị 27
- Trong khi đó, ta có lời giải tốt hơn là $I^* = \{2, 3\}$ với giá trị 28



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1 Sơ đồ thuật toán tham lam

2 Bài toán đổi tiền

3 Bài toán cái túi

4 Tập đoạn thẳng không giao nhau

- Bài toán
- Các ý tưởng tham lam
- Chứng minh tính tối ưu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán cái túi: Tham lam 4

Greedy4:

Gọi I_j là lời giải thu được theo thuật toán Greedy $j = 1, 2, 3$. Gọi I_4 là lời giải đạt

$$\max\{\sum_{i \in I_1} C_i, \sum_{i \in I_2} C_i, \sum_{i \in I_3} C_i\}$$

- Định lí: Lời giải I_4 thoả mãn bất đẳng thức

$$\sum_{i \in I_4} C_i \geq \frac{1}{2} OPT,$$

với OPT là đáp số tối ưu của bài toán



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán thực hành

Money Changing

ATM Withdrawal

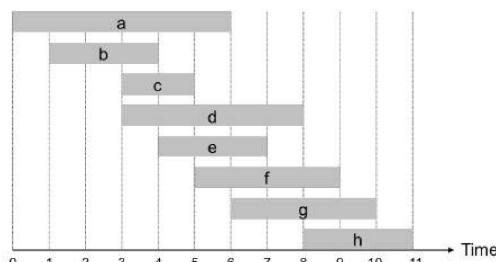


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Tập đoạn thẳng không giao nhau

Phát biểu bài toán

- Có n công việc, công việc j bắt đầu tại S_j và kết thúc tại F_j
- Hai công việc là **phù hợp** với nhau nếu chúng không chồng lên nhau
- Yêu cầu:** Tìm tập con nhiều nhất các công việc đôi một phù hợp với nhau



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Tập đoạn thẳng không giao nhau: Các ý tưởng tham lam

Xét các công việc theo một thứ tự ưu tiên nào đó. Tại mỗi bước chọn lần lượt công việc theo thứ tự ưu tiên mà phù hợp với tất cả các công việc đã chọn

- ① [Bắt đầu sớm xét trước] Xét các công việc theo thứ tự ưu tiên tăng dần của thời gian bắt đầu s_j
- ② [Kết thúc sớm xét trước] Xét các công việc theo thứ tự ưu tiên tăng dần của thời gian kết thúc f_j
- ③ [Ngắn hơn xét trước] Xét các công việc theo thứ tự ưu tiên tăng dần của tổng thời gian thực hiện công việc $f_j - s_j$
- ④ [Ít mâu thuẫn hơn xét trước] Với mỗi công việc j , tính c_j là số lượng công việc không tương thích với j . Xét các công việc theo thứ tự ưu tiên tăng dần của c_j



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Tập đoạn thẳng không giao nhau: Các ý tưởng tham lam

Các ý tưởng tham lam

Xét các công việc theo một thứ tự ưu tiên nào đó. Tại mỗi bước chọn lần lượt công việc theo thứ tự ưu tiên mà phù hợp với tất cả các công việc đã chọn

[Bắt đầu sớm xét trước]



[Ngắn hơn xét trước]



[Ít mâu thuẫn hơn xét trước]



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Tập đoạn thẳng không giao nhau: Kết-Thúc-Sớm-Xét-Trước

Demo

EARLIEST-FINISH-TIME-FIRST($n, S_1, \dots, S_n, F_1, \dots, F_n$)

```
1 SORT (các công việc theo thời gian kết thúc:  $F_1 \leq \dots \leq F_n$ )
2 A ← ∅ // tập các công việc đã được chọn
3 for j=1 to n
4     if (công việc j phù hợp với tập A)
5         A ← A ∪ {j}
6 return A
```

Dộ phức tạp: $\mathcal{O}(n \log n)$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

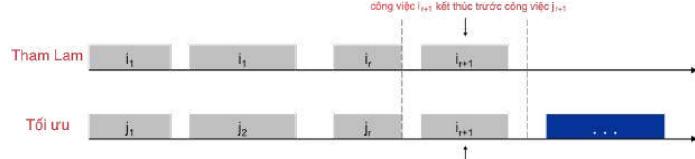
Tập đoạn thẳng không giao nhau: Chứng minh tính tối ưu thuật toán Kết-Thúc-Sớm-Xét-Trước

Dịnh lí

Thuật toán Kết-Thúc-Sớm-Xét-Trước cho kết quả tối ưu!

CM: [bằng phản chứng] Giả sử thuật toán tham lam không cho kết quả tối ưu:

- Gọi $\{i_1, i_2, \dots, i_k\}$ là tập các công việc được chọn bởi thuật toán tham lam;
- Gọi $\{j_1, j_2, \dots, j_m\}$ là tập các công việc được chọn của lời giải tối ưu với $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ với giá trị r lớn nhất có thể



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

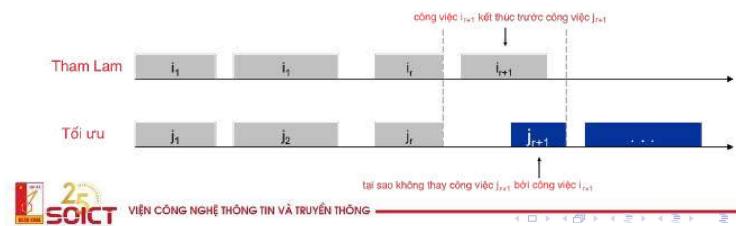
Tập đoạn thẳng không giao nhau: Chứng minh tính tối ưu thuật toán Kết-Thúc-Sớm-Xét-Trước

Dịnh lí

Thuật toán Kết-Thúc-Sớm-Xét-Trước cho kết quả tối ưu!

CM: [bằng phản chứng] Giả sử thuật toán tham lam không cho kết quả tối ưu:

- Gọi $\{i_1, i_2, \dots, i_k\}$ là tập các công việc được chọn bởi thuật toán tham lam;
- Gọi $\{j_1, j_2, \dots, j_m\}$ là tập các công việc được chọn của lời giải tối ưu với $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ với giá trị r lớn nhất có thể



Bài toán thực hành

Planting Trees

Postman

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thank you for
your attentions!



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



25 YEARS ANNIVERSARY
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Lý Thuuyết NP-Đầy-Đủ THUẬT TOÁN ỨNG DỤNG

- ① Giới thiệu
- ② Các lớp bài toán P, NP, NPC
- ③ Bài toán quyết định và Bài toán tối ưu
- ④ Phép qui đổi
- ⑤ Chứng minh NP-đầy-đủ
- ⑥ Các hướng tiếp cận giải bài toán NP-khổ

Độ khó của bài toán

- Đánh giá độ khó của bài toán là ước lượng thời gian tính của thuật toán tốt nhất trong số tất cả các thuật toán giải bài toán kể cả các thuật toán đã biết lẫn các thuật toán còn chưa biết
- Hai cách tiếp cận:
 - ▶ Cách 1: tìm cách đưa ra đánh giá cận dưới độ phức tạp của bài toán (Dành cho khóa học Phân Tích Thuật Toán Nâng Cao)
 - ▶ Cách 2: tập trung vào việc chỉ ra mức độ khó của nó có thể so sánh với bất kỳ bài toán khó hiện biết (Lý thuyết NP-đầy-đủ)
- Việc đánh giá được độ phức tạp tính toán của bài toán giữ vai trò định hướng trong việc thiết kế thuật toán để giải bài toán đặt ra

① Giới thiệu

- ② Các lớp bài toán P, NP, NPC
- ③ Bài toán quyết định và Bài toán tối ưu
- ④ Phép qui đổi
- ⑤ Chứng minh NP-đầy-đủ
- ⑥ Các hướng tiếp cận giải bài toán NP-khổ

Giới thiệu

- Từ những năm 1960, Steve Cook và Dick Karp đã quyết định rằng yêu cầu tối thiểu đối với một thuật toán hiệu quả là thời gian tính của nó phải là đa thức: $\mathcal{O}(n^c)$ với c là hằng số
- Người ta cũng nhận thấy rằng đối với nhiều lớp bài toán, việc tìm ra những thuật toán như vậy là rất khó khăn, hơn nữa chúng ta còn không biết là một thuật toán như vậy có tồn tại hay không
- Chính vì thế, Cook và Karp và một số người khác đã đưa ra định nghĩa lớp bài toán NP-đầy-dủ, mà cho đến hiện nay người ta vẫn tin rằng là không thể có thuật toán hiệu quả để giải chúng



Dick Karp (1972)

(Cook et. al., 1998)



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

6/51

1 Giới thiệu

2 Các lớp bài toán P, NP, NPC

3 Bài toán quyết định và Bài toán tối ưu

4 Phép qui đổi

5 Chứng minh NP-đầy-dủ

6 Các hướng tiếp cận giải bài toán NP-khó



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

7/51

Lớp bài toán P, NP

P là lớp các bài toán có thể giải được trong thời gian đa thức

- Bài toán về tính liên thông của đồ thị có thể giải được nhờ thuật toán với thời gian tính là $\mathcal{O}(n^2)$, vì vậy, nó là bài toán thuộc lớp P
- Bài toán nhân dãy ma trận giải được nhờ qui hoạch động với thời gian $\mathcal{O}(n^3)$, cũng thuộc vào lớp P



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

8/51

Lớp bài toán P, NP

P là lớp các bài toán có thể giải được trong thời gian đa thức

- Bài toán về tính liên thông của đồ thị có thể giải được nhờ thuật toán với thời gian tính là $\mathcal{O}(n^2)$, vì vậy, nó là bài toán thuộc lớp P
- Bài toán nhân dãy ma trận giải được nhờ qui hoạch động với thời gian $\mathcal{O}(n^3)$, cũng thuộc vào lớp P

NP là lớp các bài toán kiểm chứng được trong thời gian đa thức

nghĩa là đưa ra được thuật toán trong thời gian đa thức kiểm chứng tính chính xác của một bộ kết quả đầu ra với bộ dữ liệu đầu vào tương ứng

- Bài toán kiểm tra tính hợp số: "Có phải số n là hợp số?"
- Bài toán tìm chu trình Hamilton, việc kiểm tra dãy đỉnh $v_1, v_2, \dots, v_n, v_1$ có là chu trình Hamilton của đồ thị đã cho hay không có thể thực hiện sau thời gian đa thức



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

8/51

Lớp bài toán P, NP, NPC

NP-đầy-dủ là lớp các bài toán trong lớp NP và khó khôn kém bất cứ bài toán nào trong NP

Đây là lớp bài toán khó nhất trong lớp NP



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

9/51

Lớp bài toán P, NP, NPC

NP-đầy-dủ là lớp các bài toán trong lớp NP và khó khôn kém bất cứ bài toán nào trong NP

Đây là lớp bài toán khó nhất trong lớp NP

Vì sao không nói "Giải được trong thời gian hàm mũ" hay "Giải được không trong thời gian đa thức"?

$\mathcal{O}(n^{100})$ và $\mathcal{O}(2^n)$

- Thuật toán $\mathcal{O}(n^{100})$ vẫn là thuật toán đa thức, tuy nhiên thời gian tính là không có tính ứng dụng thực tế
- Đa phần các thuật toán đa thức có thời gian tính nhỏ hơn rất nhiều
- Khi một thuật toán đa thức được tìm ra, nhiều thuật toán hiệu quả mới từ đó sẽ được khám phá



VIEN CONG NGHIEP THONG TIN VÀ TRUYEN THONG

9/51

Mối quan hệ giữa P, NP, NPC

- $P \subseteq NP$ (chắc chắn)
- $NPC \subseteq NP$ (chắc chắn)
- $P = NP$ (hoặc $P \subset NP$, hoặc $P \neq NP$) ???
- $NPC = NP$ (hoặc $NPC \subset NP$, hoặc $NPC \neq NP$) ???



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Mối quan hệ giữa P, NP, NPC

- $P \subseteq NP$ (chắc chắn)
- $NPC \subseteq NP$ (chắc chắn)
- $P = NP$ (hoặc $P \subset NP$, hoặc $P \neq NP$) ???
- $NPC = NP$ (hoặc $NPC \subset NP$, hoặc $NPC \neq NP$) ???

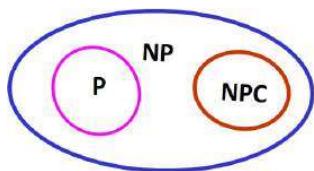
P = NP ?

- Một trong những vấn đề hóc búa nhất và là trung tâm của lý thuyết tính toán đó là chứng minh hay bác bỏ đẳng thức này!
- Cho đến hiện nay vấn đề này vẫn còn là vấn đề mở



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Mối quan hệ giữa P, NP, NPC



- Quan điểm của hầu hết các nhà nghiên cứu khoa học máy tính là:

$$P \subset NP, NPC \subset NP, P \cap NP = \emptyset$$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Vì sao cần quan tâm đến lý thuyết NP-dầy-đủ?

- Nếu một bài toán được chứng minh là thuộc lớp NP-dày-đủ thì ta có được bằng chứng về độ khó của bài toán
- Không lãng phí thời gian để cố gắng tìm ra thuật toán hiệu quả cho bài toán NP-dày-đủ
- Thay vào đó, hãy tập trung vào thiết kế thuật toán gần đúng hoặc heuristic/metaheuristic hoặc tìm lời giải hiệu quả cho một số trường hợp đặc biệt của bài toán
- Một số bài toán nhìn bề ngoài thì rất dễ, nhưng thực tế là khó (thuộc lớp NP-dày-đủ)

1 Giới thiệu

2 Các lớp bài toán P, NP, NPC

3 Bài toán quyết định và Bài toán tối ưu

4 Phép qui đổi

5 Chứng minh NP-dày-đủ

6 Các hướng tiếp cận giải bài toán NP-khó

Bài toán quyết định và Bài toán tối ưu

Bài toán tối ưu là bài toán yêu cầu tìm ra lời giải tối ưu max hoặc min

Ví dụ bài toán tối ưu: SHORTEST-PATH

- Cho G, u, v , hãy tìm một đường đi từ u đến v qua ít cạnh nhất



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài toán quyết định và Bài toán tối ưu

Bài toán tối ưu là bài toán yêu cầu tìm ra lời giải tối ưu max hoặc min

Ví dụ bài toán tối ưu: SHORTEST-PATH

- Cho G, u, v , hãy tìm một đường đi từ u đến v qua ít cạnh nhất

Bài toán quyết định là bài toán mà đầu ra chỉ có thể là 'YES' hoặc 'NO' (dúng/sai, 1/0, chấp nhận/từ chối)

- Đối với một bài toán quyết định, có những bộ dữ liệu vào của nó có câu trả lời (đầu ra) là 'YES' và cũng có những bộ dữ liệu vào có câu trả lời là 'NO'
- Những bộ dữ liệu vào với câu trả lời 'YES' ('NO') sẽ được gọi là bộ dữ liệu vào 'YES' ('NO')

Ví dụ bài toán quyết định: PATH

- Cho G, u, v, k , hỏi có tồn tại hay không một đường đi từ u đến v qua tối đa k cạnh?



Dạng quyết định của bài toán tối ưu

- Xét bài toán tối ưu hoá:

(PO): $\max\{f(x) : x \in D\}$

- Bài toán dạng quyết định (PD) tương ứng với bài toán tối ưu (PO) là:

(PD): "Cho giá trị k , hỏi có tìm được $u \in D$ sao cho $f(u) \geq k$?"

- Lời giải của bài toán tối ưu dẫn ra trực tiếp lời giải cho bài toán quyết định tương ứng
- Nếu bài toán quyết định tương ứng với một bài toán tối ưu có thể giải được hiệu quả (chẳng hạn, bằng thuật toán đa thức) thì bài toán tối ưu đó cũng giải được hiệu quả (bằng thuật toán đa thức)



(PO) và (PD): Bài toán tìm tập độc lập cực đại

MIS:

- Cho đồ thị vô hướng $G = (V, E)$. Một tập con các đỉnh của đồ thị mà hai đỉnh bất kỳ trong nó là không kề nhau trên đồ thị được gọi là **tập độc lập** của đồ thị
- Yêu cầu:** Tìm tập độc lập với lực lượng lớn nhất (**tập độc lập cực đại**)
- Bài toán dạng quyết định tương ứng có dạng: "Cho số nguyên dương k , hỏi đồ thị có chứa tập độc lập với lực lượng ít ra là k hay không?"
- Do $1 \leq k_{\max} \leq n$, (k_{\max} là lực lượng của tập độc lập cực đại), nên nếu bài toán dạng quyết định giải được bằng thuật toán đa thức thì bài toán tối ưu cũng giải được bằng sơ đồ trình bày trên sau không quá $\lceil \log n \rceil$ lần áp dụng thuật toán giải bài toán dạng quyết định



Bài toán quyết định và Bài toán tối ưu

Lớp NP-dầy-dủ chỉ gồm các bài toán quyết định!

- Đối với bài toán tối ưu, nếu kiểm chứng được tính tối ưu và chính xác của bộ kết quả đầu ra trong thời gian đa thức thì cũng có thể tìm được lời giải tối ưu trong thời gian đa thức
- Việc qui đổi giữa các bài toán quyết định dễ dàng hơn so với giữa các bài toán tối ưu



Mối liên hệ giữa (PO) và (PD)

- Giả thiết rằng hàm f nhận giá trị nguyên trên D và ta biết α_0 và β_0 là cận dưới và cận trên của giá trị hàm f trên D
- Giả sử A là thuật toán giải bài toán (PD) với độ phức tạp $\mathcal{O}(n^c)$

Bước lặp $k = 0, 1, 2, \dots$

- Tính $\gamma_k = (\alpha_k + \beta_k)/2$;
- Nếu tìm được $x_k \in D$ thoả mãn $f(x_k) > \gamma_k$ thì $\alpha_{k+1} = f(x_k)$, $\beta_{k+1} = \gamma_k$;
- Trái lại, đặt $\alpha_{k+1} = \alpha_k$, $\beta_{k+1} = \beta_k$;
- Chuyển sang bước $k + 1$;

Rõ ràng sơ đồ trên cho thời gian tính toán $\mathcal{O}(\log(\beta_0 - \alpha_0)n^c)$ để giải bài toán (PO)



1 Giới thiệu

2 Các lớp bài toán P, NP, NPC

3 Bài toán quyết định và Bài toán tối ưu

4 Phép qui đổi

5 Chứng minh NP-dầy-dủ

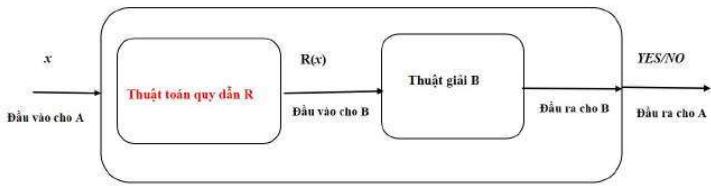
6 Các hướng tiếp cận giải bài toán NP-khó



Phép qui dẫn trong thời gian đa thức

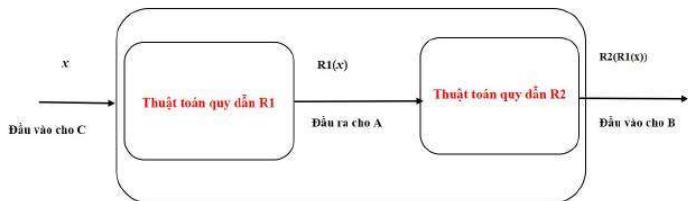
- Giả sử A và B là hai bài toán quyết định. Ta nói bài toán A có thể **qui dẫn** trong thời gian đa thức về bài toán B nếu tồn tại thuật toán thời gian đa thức R cho phép biến đổi bộ dữ liệu vào x của A thành bộ dữ liệu vào $R(x)$ của B sao cho x là bộ dữ liệu 'YES' (nghĩa là bộ dữ liệu mà câu trả lời cho nó là 'YES') của A khi và chỉ $R(x)$ là bộ dữ liệu 'YES' của B
- Trong phần tiếp theo ta chỉ xét phép qui dẫn sau thời gian đa thức, vì thế để ngắn gọn, ta sẽ gọi là phép qui dẫn thay cho phép qui dẫn sau thời gian đa thức

Phép qui dẫn trong thời gian đa thức



- Kí hiệu A qui dẫn về B: $A \prec B$
- Nếu tồn tại thuật toán đa thức giải B thì A cũng sẽ được giải trong thời gian đa thức
- A không khó hơn B (hay B không dễ hơn A)
- Nếu bài toán A là khó (ví dụ thuộc lớp NP-dài-dài), thì bài toán B cũng khó
- Phép qui dẫn này được sử dụng để chỉ ra một bài toán là khó.

Phép qui dẫn



- Nếu $C \prec A$ và $A \prec B$, thì $C \prec B$

NP-dài-dài và NP-khó

Định nghĩa

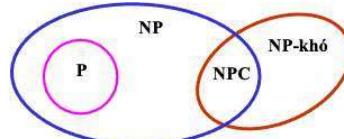
Bài toán quyết định A được gọi là NP-dài-dài nếu như

- A là bài toán trong NP
- Mọi bài toán trong NP đều có thể qui dẫn về A

Nếu bỏ đi điều kiện (1) mà chỉ thỏa mãn điều kiện (2) thì A được gọi là NP-khó

- Như vậy, có thể nói khái niệm về "bài toán khó nhất" trong lớp NP được xây dựng trên cơ sở phép qui dẫn
- Nếu tất cả các bài toán trong NP có thể qui dẫn về một bài toán A thì A khó không kém bất cứ bài toán nào trong số chúng
- Chỉ với điều kiện (2), nếu tồn tại thuật toán đa thức để giải A thì sẽ kéo theo sự tồn tại thuật toán đa thức để giải mọi bài toán trong NP

NP-dài-dài và NP-khó



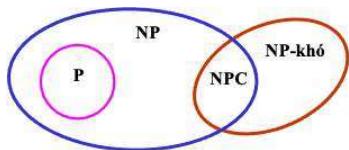
Hình: Bức tranh tạm thời chi tiết hơn

Khó khăn nhất là việc tìm ra được một bài toán thuộc NP để qui dẫn về A. Bởi vì hễ chúng ta đã có một bài toán NP-dài-dài thì có thể chứng minh nhiều bài toán khác là NP-dài-dài nhờ sử dụng kết quả sau đây:

Bổ đề

Nếu $A \in \text{NP-dài-dài}$, $B \in \text{NP}$, và $A \prec B$, khi đó $B \in \text{NP-dài-dài}$

NP-dầy-dủ và NP-khó



Hình: Bức tranh tạm thời chi tiết hơn

Khó khăn nhất là việc tìm ra được một bài toán thuộc NP để qui dẫn về A. Bởi vì hễ chúng ta đã có một bài toán NP-dày-dủ thì có thể chứng minh nhiều bài toán khác là NP-dày-dủ nhờ sử dụng kết quả sau đây:

Bổ đề

Nếu $A \in \text{NP-dày-dủ}$, $B \in \text{NP}$, và $A \prec B$, khi đó $B \in \text{NP-dày-dủ}$

NP-dày-dủ và NP-khó

Vậy làm thế nào để chứng minh bài toán NP-dày-dủ đầu tiên?

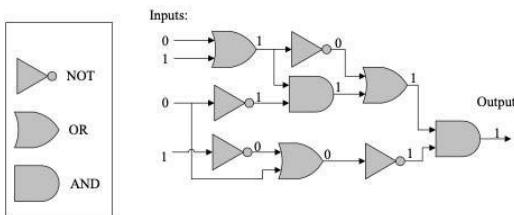
NP-dày-dủ và NP-khó

Vậy làm thế nào để chứng minh bài toán NP-dày-dủ đầu tiên?

- Bài toán về tính thực tiễn của mạch logic CIRCUIT-SAT (Circuit-satisfiability) là bài toán đầu tiên được chứng minh là bài toán NP-dày-dủ bằng cách đưa ra một thuật toán qui dẫn đa thức từ bất kì một bài toán L nào thuộc lớp NP
- Tuy nhiên về mặt lịch sử CIRCUIT-SAT không phải là bài toán NP-dày-dủ đầu tiên, mà đó là bài toán SAT

Định lí: CIRCUIT-SAT là NP-dày-dủ (Cook, 1971)

- ① CIRCUIT-SAT $\in \text{NP}$: Cho 1 đầu vào, dễ dàng tính được đầu ra tương ứng sau thời gian đa thức nhờ sử dụng thuật toán duyệt đồ thị



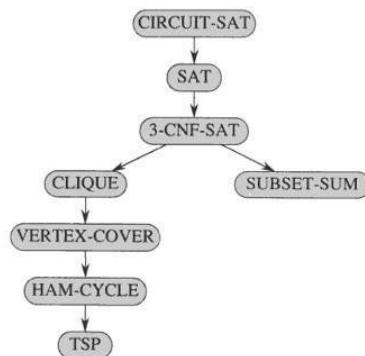
CIRCUIT-SAT là NP-dày-dủ

- ② Việc chứng minh mọi bài toán trong NP đều qui dẫn được về CIRCUIT-SAT là khá phức tạp. Ý tưởng chứng minh được phác thảo như sau:
- ▶ Mọi bài toán trong NP đều có thể tính được nhờ mạch logic
 - ▶ Mạch logic này có số thành phần giới hạn bởi đa thức và vì thế cũng tính được sau thời gian đa thức

Sơ đồ chứng minh L là NP-đầy-đủ

- ① Chứng minh $L \in NP$: thường là dễ
- ② Chọn bài toán L' là NP-đầy-đủ (hoặc NP-khó)
- ③ Xây dựng thuật toán thời gian tính đa thức xác định hàm f thực hiện việc biến đổi bộ dữ liệu của L' thành bộ dữ liệu của L
- ④ Chứng minh: x là bộ dữ liệu 'YES' của L' khi và chỉ khi $f(x)$ là bộ dữ liệu 'YES' của L

Cấu trúc cây qui dẫn các bài toán NP-đầy-đủ cơ bản



Một số bài toán trên cây qui dẫn NP-đầy-đủ

CIRCUIT-SAT

- Cho một mạch lôgic với đầu vào gồm n giá trị
- Hỏi có tồn tại một đầu vào của mạch để đầu ra của mạch là TRUE, hay mạch luôn đưa ra FALSE?
- $L \prec CIRCUIT-SAT, \forall L \in NP$

SAT

- Cho một biểu thức lôgic tạo thành bởi n biến Bun: x_1, x_2, \dots, x_n , các mệnh đề, các toán tử lôgic AND \wedge , OR \vee , NOT \neg , ... và các dấu ngoặc '(', ')'.
- Hỏi có tồn tại một bộ giá trị của các biến Bun để cho biểu thức nhận giá trị TRUE?
- $CIRCUIT-SAT \prec SAT$

Một số bài toán trên cây qui dẫn NP-đầy-đủ

CLIQUE

- Một **bè** của đồ thị vô hướng $G = (V, E)$ là một tập con các đỉnh $S \subseteq V$ sao cho mỗi cặp đỉnh trong đó được nối bởi một cạnh $\in E$
- Nói một cách khác, một bè là một đồ thị con đầy đủ của G
- Lực lượng của một bè là số lượng đỉnh của bè đó
- **Bài toán tối ưu:** Tìm bè cực đại
- **Bài toán quyết định:** Hỏi có tồn tại bè kích thước k cho trước trong G hay không?
- $3-CNF-SAT \prec CLIQUE$

Một số bài toán trên cây qui dẫn NP-đầy-đủ

3-CNF-SAT

- Cho một biểu thức lôgic dưới dạng 3-CNF (Conjunctive Normal Form), nghĩa là biểu thức Bun cấu thành từ hội của các mệnh đề mà mỗi mệnh đề là tuyển của đúng 3 toán hạng, mỗi toán hạng là 1 biến Bun (x) hoặc phủ định của nó ($\neg x$)
- Hỏi có tồn tại một bộ giá trị của các biến số để cho biểu thức nhận giá trị TRUE?
- $SAT \prec 3-CNF-SAT$

SUBSET-SUM

- Cho tập S gồm n số nguyên dương a_1, \dots, a_n và số nguyên dương t
- Hỏi có thể tìm được tập con S' của S với tổng các số trong S' là bằng t ?
- $3-CNF-SAT \prec SUBSET-SUM$

Một số bài toán trên cây qui dẫn NP-đầy-đủ

VERTEX-COVER

- Một **phủ định** của đồ thị vô hướng $G = (V, E)$ là một tập con các đỉnh của đồ thị $S \subseteq V$ sao cho mỗi cạnh của đồ thị có ít nhất một đầu mút trong S
- Lực lượng của một phủ định là số lượng đỉnh của bè đó
- **Bài toán tối ưu:** Tìm phủ định cực tiểu
- **Bài toán quyết định:** Hỏi có tồn tại phủ định kích thước k cho trước trong G hay không?
- $CLIQUE \prec VERTEX-COVER$

Một số bài toán trên cây qui dẫn NP-dầy-dủ

HAM-CYCLE

- Hỏi đồ thị vô hướng $G = (V, E)$ có chứa chu trình Hamilton không?
- VERTEX-COVER \prec HAM-CYCLE

TSP

- Cho ma trận chi phí $C = [C_{ij}]$ giữa các thành phố và một số nguyên k :
- Bài toán tối ưu: Tìm hành trình người du lịch với tổng chi phí nhỏ nhất
- Bài toán quyết định: Hỏi có tồn tại một hành trình của người du lịch với tổng chi phí không vượt quá số k hay không?
- HAM-CYCLE \prec TSP

Qui dẫn HAM-CYCLE \prec TSP

- Với mỗi trường hợp đồ thị $G = (V, E)$ của HAM-CYCLE, ta xây dựng một trường hợp tương ứng của TSP $< G', C, 0 >$ trong thời gian đa thức như sau:
 - $G' = (V, E')$, với $E' = \{< i, j > : i, j \in V, i \neq j\}$ và
 - hàm chi phí C được định nghĩa như sau:
 - * $C(i, j) = 0$ nếu $(i, j) \in E$,
 - * $C(i, j) = 1$, nếu trái lại
- Nếu G có một chu trình Hamilton H , thì H cũng là một hành trình hợp lệ trên G' với chi phí tối đa là 0
- Nếu G' có một hành trình H' với chi phí tối đa là 0, thì mỗi cạnh trên H' đều có chi phí bằng 0, vì vậy các cạnh đó đều $\in E$, và H' cũng là một chu trình Hamilton trên G

Các hướng tiếp cận giải bài toán NP-khó

Còn nhớ 4 mô hình giải bài cẩn bản ?

Định lí: TSP là NP-dầy-dủ

Chứng minh:

- ① TSP \in NP: việc kiểm tra xem dãy các thành phố đã cho có phải là hành trình hợp lệ với chi phí không vượt quá k có thể dễ dàng thực hiện xong trong thời gian đa thức
- ② Chứng minh TSP là NP-khó bằng phép qui dẫn HAM-CYCLE \prec TSP

1 Giới thiệu

- 2 Các lớp bài toán P, NP, NPC
- 3 Bài toán quyết định và Bài toán tối ưu
- 4 Phép qui dẫn
- 5 Chứng minh NP-dầy-dủ
- 6 Các hướng tiếp cận giải bài toán NP-khó

Các hướng tiếp cận giải bài toán NP-khó

Còn nhớ 4 mô hình giải bài cẩn bản ?

Duyệt nhánh cận (Lecture 3)

- Thời gian chạy lâu
- Thường chỉ đưa ra được đáp án tối ưu trong thời gian chấp nhận được với kích thước đầu vào đủ nhỏ hoặc với một số trường hợp đặc thù
- Khó ước lượng chính xác độ phức tạp tính toán

Các hướng tiếp cận giải bài toán NP-khó

Chia nhỏ / Qui hoạch động (Lecture 4/5)

- Có thể đưa ra được đáp án tối ưu trong một số bài toán (ví dụ: TSP, KNAPSAC)
- Độ phức tạp tính toán vẫn là hàm mũ
- Có thể sử dụng như là một phần của thuật toán heuristic để đưa ra được một lời giải chấp nhận được

Các hướng tiếp cận giải bài toán NP-khó

Thuật toán xấp xỉ (Approximation Algorithms)

- Thường là các thuật toán tham lam hay heuristic đơn giản
- Luôn chứng minh được chắc chắn độ tốt của lời giải của thuật toán đề xuất so với lời giải tối ưu luôn nằm trong giới hạn cận tỉ lệ ρ xác định
- Ít có tính ứng dụng trong các bài toán thực tế
- Là một lĩnh vực lý thuyết khó, thường xuất hiện trong các khóa học thạc sĩ chuyên ngành Khoa học máy tính

Các hướng tiếp cận giải bài toán NP-khó

Thuật toán metaheuristic

- Là các mô hình phát triển thuật toán cấp cao dùng để đưa ra các chiến lược điều khiển và điều chỉnh các thuật toán heuristic bằng cách thay đổi linh hoạt các tham số trong mô hình
- Lời giải bài toán thường sẽ được cải tiến so với các lời giải thuật toán heuristic đơn giản khi chọn được các bộ tham số của mô hình đủ tốt
- Thường phải chạy thí nghiệm nhiều lần để tìm ra được bộ tham số thích hợp
- Thời gian chạy và bộ nhớ sử dụng thường rất lớn và ít khả năng ứng dụng vào các bài toán thực tế đòi hỏi xử lý thời gian thực hoặc dữ liệu lớn
- Một số mô hình điển hình: Thuật toán di truyền/tiến hóa (genetic/evolutionary algorithms), tìm kiếm tabu (tabu search), mô phỏng tối luyên thép (simulated annealing), tìm kiếm vùng lân cận linh hoạt (variable neighborhood search), tìm kiếm vùng lân cận không gian lớn (large neighborhood search), thuật toán đàm kiêng (ant colony optimization), ...

Các hướng tiếp cận giải bài toán NP-khó

Thuật toán tham lam (Lecture 9)

- Nhanh chóng đưa ra được một lời giải chấp nhận được (feasible solution)
- Hay được dùng để tạo một lời giải khởi đầu cho các thuật toán heuristic/metaheuristic phức tạp
- Tính ứng dụng vào các bài toán thực tế cao, nhất là các bài toán đòi hỏi thời gian thực (real-time) với độ tốt của lời giải là chấp nhận được
- Về khoa học lý thuyết, thường được sử dụng trong lĩnh vực thuật toán xấp xỉ

Các hướng tiếp cận giải bài toán NP-khó

Thuật toán heuristic

- Là các kỹ thuật giải bài được phát triển *dựa trên kinh nghiệm* phân tích các đặc điểm của bài toán để đưa ra một hướng tiếp cận cho lời giải chấp nhận được đủ tốt (không đảm bảo là tối ưu)
- Hay sử dụng các kỹ thuật tìm kiếm địa phương (local search) để cải tiến lời giải hiện biết
- Có thể cùng một lúc áp dụng nhiều thuật toán heuristic khác nhau, gọi là multiheuristic
- Là nền tảng cho việc phát triển các thuật toán metaheuristic

