Cấu trúc dữ liệu và Thư viện

LẬP TRÌNH THI ĐẤU

Đỗ Phan Thuận

Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 10 tháng 6 năm 2016

Các vấn đề

- Các kiểu dữ liệu cơ bản
- Số nguyên lớn
- Tầm quan trọng của Cấu trúc dữ liệu
- Các cấu trúc dữ liệu thông dụng
- Sắp xếp và Tìm kiếm
- Sử dụng Bitmask để biểu diễn tập hợp
- Một số ứng dụng hay gặp
- Cây nhị phân tìm kiếm mở (Augmenting BST)
- 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`: 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	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	$\approx -3.4 \times 10^{-38}$	$\approx 3.4 \times 10^{-38}$	$\approx$ 7 chữ số
double	8	$\approx -1.7 \times 10^{-308}$	$\approx 1.7 \times 10^{-308}$	$\approx$ 14 chữ số

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

Bài toán ví dụ: Integer Inquiry

- http://uva.onlinejudge.org/external/4/424.html

Tầm quan trọng của cấu trúc dữ liệu

- Nhiều khi dữ liệu cần được biểu diễn theo cách thuận lợi cho
  - Truy vấn hiệu quả
  - Chèn hiệu quả
  - Xóa hiệu quả
  - Cập nhật hiệu quả
- Nhiều khi dữ liệu cần được biểu diễn theo cách tốt hơn nữa
  - Làm thế nào để biểu diễn số nguyên lớn?
  - Làm thế nào để biểu diễn đồ thị?
- Các cấu trúc dữ liệu giúp chúng ta thực hiện được những điều này

Các cấu trúc dữ liệu thông dụng

- Mảng tĩnh
- Mảng động
- Danh sách liên kết
- Ngăn xếp
- Hàng đợi
- Hàng đợi ưu tiên
- 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>
- Tập hợp - set<int>
- Ánh xạ - map<int, int>

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>
- Tập hợp - set<int>
- Ánh xạ - map<int, int>

- 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

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 lập trình sai tìm kiếm nhị phân

Biểu diễn tập hợp

- Cho một số lượng nhỏ ($n \leq 30$) phần tử
- Gán nhãn bởi các số nguyên $0, 1, \ldots, n-1$
- Biểu diễn tập hợp các phần tử này bởi một biến nguyên 32-bit
- Phần thử 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

- Tập rỗng:

  `0`

- Tập có một phần tử:

  `1<<i`

- Tập vũ trụ (nghĩa là tất cả các phần tử):

  `(1<<n)-1`

- Hợp hai tập:

  `x|y`

- Giao hai tập:

  `x&y`

- Phần bù một tập:

  `~x & ((1<<n)-1)`

Biểu diễn tập hợp

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

```
if (x & (1<<i)) {
    // yes
} else {
    // no
}
```


Biểu diễn tập hợp

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

Ứng dụng của Mảng và Danh sách liên kết

- Trường hợp có quá nhiều để liệt kê
- Phần lớn các bài toán cần lưu trữ dữ liệu, thường là lưu trong một mảng

Bài toán ví dụ: Broken Keyboard

- http://uva.onlinejudge.org/external/119/11988.html

Ứng dụng của Ngăn xếp

- Xử lý các sự kiện theo trình tự vào-sau-ra-trước
- Mô phỏng đệ quy
- Tìm kiếm theo chiều sâu trên đồ thị
- Đảo ngược chuỗi
- Ghép dấu ngoặc đúng
- ...

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
- ...

Ứng dụng của Hàng đợi ưu tiên

- Xử lý các sự kiện theo trình tự ưu tiên
- Tìm đường đi ngắn nhất trên đồ thị
- Một số thuật toán tham lam
- ...

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 vQ){(*this).fuel=fuel;}
    friend ostream& operator<<(ostream& os,const Plane& p){
        os<<p.fuel<<endl;return os;
    }
    bool operator>(const LabVer& p) const{
        return fuel>p.fuel;
    }
};
typedef priority_queue<Plane,vector<Plane>,greater<Plane> > PQPlane;
PQPlane PQ;

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;
}
```


Ứng dụng của tập hợp

- Giữ vết của các phần tử phân biệt
- Hỏi đã từng thấy một phần tử trước đây hay chưa?
- Nếu cài đặt như một cây nhị phân tìm kiếm:
  - Tìm cha của một phần tử (phần tử nhỏ nhất mà lớn hơn nó)
  - Tính xem có bao nhiêu phần tử nhỏ hơn một phần tử cho trước
  - Tính xem có bao nhiêu phần tử nằm giữa hai phần tử cho trước
  - Tìm phần tử lớn thứ kth
- . . .

Ứng dụng của kiểu Ánh xạ

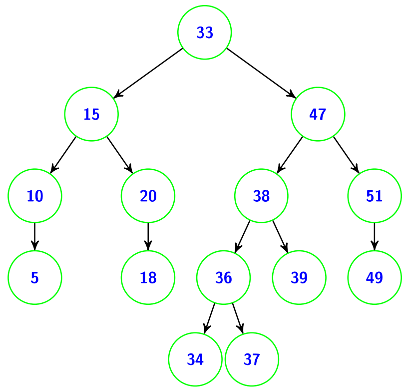
- Gắn một giá trị với một khóa
- Giống như Bảng tần xuất
- Giống như phần lưu trữ khi thực hiện thuật toán Quy hoạch động (chi tiết sau)
- . . .

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)

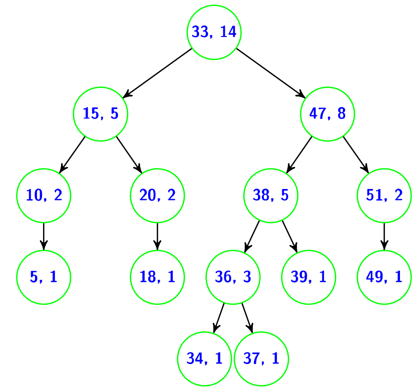
Cây nhị phân tìm kiếm mở

- 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 ngây thơ là duyệt qua tất cả các đỉnh: $O(n)$



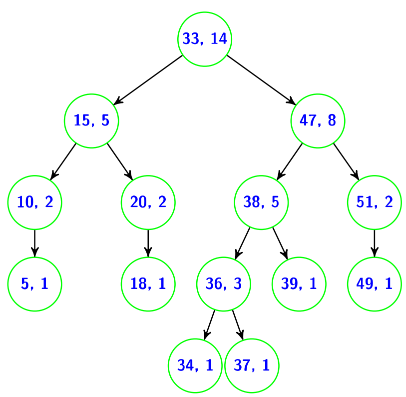
Cây nhị phân tìm kiếm mở

- 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 thuật toán chung



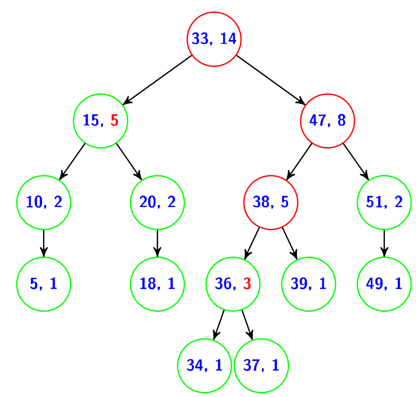
Cây nhị phân tìm kiếm mở

- 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



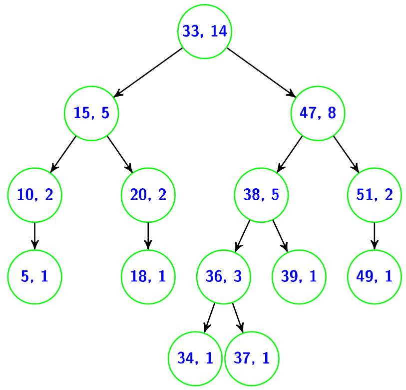
Cây nhị phân tìm kiếm mở

- 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 $O(\log n)$



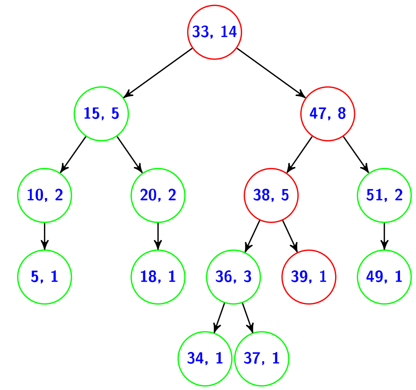
Cây nhị phân tìm kiếm mở

- 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: $k = 11$

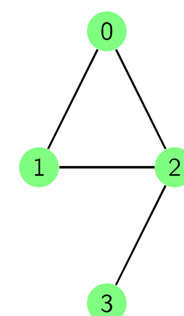


Biểu diễn đồ thị

- 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

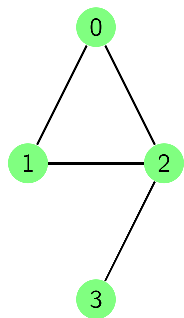
Danh sách kề

```
0: 1, 2
1: 0, 2
2: 0, 1, 3
3: 2

vector<int> adj[4];
adj[0].push_back(1);
adj[0].push_back(2);
adj[1].push_back(0);
adj[1].push_back(2);
adj[2].push_back(0);
adj[2].push_back(1);
adj[2].push_back(3);
adj[3].push_back(2);
```



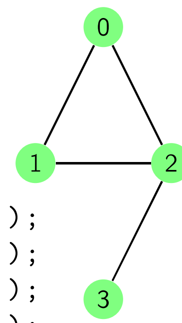
Ma trận kề

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

bool adj[4][4];
adj[0][1] = true;
adj[0][2] = true;
adj[1][0] = true;
adj[1][2] = true;
adj[2][0] = true;
adj[2][1] = true;
adj[2][3] = true;
adj[3][2] = true;
```



Danh sách cạnh

```
0, 1
0, 2
1, 2
2, 3

vector<pair<int, int> > edges;
edges.push_back(make_pair(0, 1));
edges.push_back(make_pair(0, 2));
edges.push_back(make_pair(1, 2));
edges.push_back(make_pair(2, 3));
```



Hiệu quả

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

- Các cách biểu diễn khác nhau hiệu quả tùy tình huống sử dụng
- Đôi khi cùng lúc sử dụng nhiều cách biểu diễn

Bài toán ví dụ:

- http://uva.onlinejudge.org/external/119/11991.html
- http://uva.onlinejudge.org/external/120/12049.html