CÁC KỸ NĂNG GIẢI BÀI CẦN RÈN LUYỆN

Đỗ Phan Thuận
Bộ môn Khoa Học Máy Tính – Đại học Bách Khoa Hà Nội
Email: thuan.dophan@hust.edu.vn
Hà Nội, 15/8/2016

Tổng thống Obama trong một lần thăm trụ sở của Google năm 2007, tiến sĩ Eric Schmidt, CEO của Google hỏi tổng thống một câu tuyển dụng quen thuộc cho kỹ sư lập trình vào công ty như sau:

"Cách nào hiệu quả nhất để sắp xếp một triệu số nguyên 32 bit?"

Theo bạn thì:

- Câu trả lời đúng là gì?
- Obama sẽ trả lời như thế nào?

Obama không phải là người làm tin học, tuy nhiên ông đã có một câu trả lời rất thông minh, thể hiện phần nào được trí tuệ của người đứng đầu đất nước số một thế giới:

"Theo tôi thì chắc chắn là không phải Bubble Sort!"

Câu trả lời đã gây bất ngờ cho tiến sĩ Schmidt, ông đã nói là không hề thấy kiến thức nền tảng khoa học máy tính của tổng thống mà sao trả lời được như vậy.

Cần bình luận thêm là từ Bubble Sort có thể là một từ thông dụng trong đời sống của người nói tiếng Anh khi nói đến cách sắp xếp các đồ vật hay công việc. Tư tưởng rất tự nhiên như sau: cứ nhìn vào hai đồ vật liên tiếp nếu thấy không đúng vị trí thì đảo chỗ chúng; cứ làm như vậy cho đến khi không còn cặp nào như vậy.

Tiến sĩ Schmidt cũng từng hỏi một câu tương tự như vậy với cựu phó tổng thống Mỹ John McCain nhưng có thêm điều kiện hạn chế bộ nhớ Ram là 2MB và McCain đã không có câu trả lời. Bạn hãy giúp McCain trả lời câu hỏi đó, điều kiện bài toán có thay đổi một chút cho phù hợp với việc giải bài trong thời gian hạn chế. Bài toán lấy tên là *Obama Sort* (Sort3MB), mở rộng cho việc sắp xếp n số thực 32-bit, hạn chế bộ nhớ tối đa **3MB** và hạn chế thời gian là **1 giây**.

Dữ liệu: Vào từ file văn bản SORT3MB.INP:

- Dòng đầu chứa một số nguyên dương $n \le 10^6$;
- Dòng thứ hai chứa n số thực giá trị tối đa có thể lưu trong biến 4 byte và có chính xác 2 chữ số sau dấu phẩy động.

Kết quả: Ghi ra file văn bản SORT3MB.OUT một dòng duy nhất chứa n số đã được sắp xếp tăng dần có chính xác 2 chữ số sau dấu phẩy động. Lưu ý là giá trị các số trong file input bao gồm cả phần lẻ cũng phải giống hệt như giá trị tương ứng trong dữ liệu đầu vào.

Ví dụ:

SORT3MB.INP	SORT3MB.OUT
6 2.22 5.25 6.26 1.21 4.24 3.23	1.21 2.22 3.23 4.24 5.25 6.26

Ràng buộc:

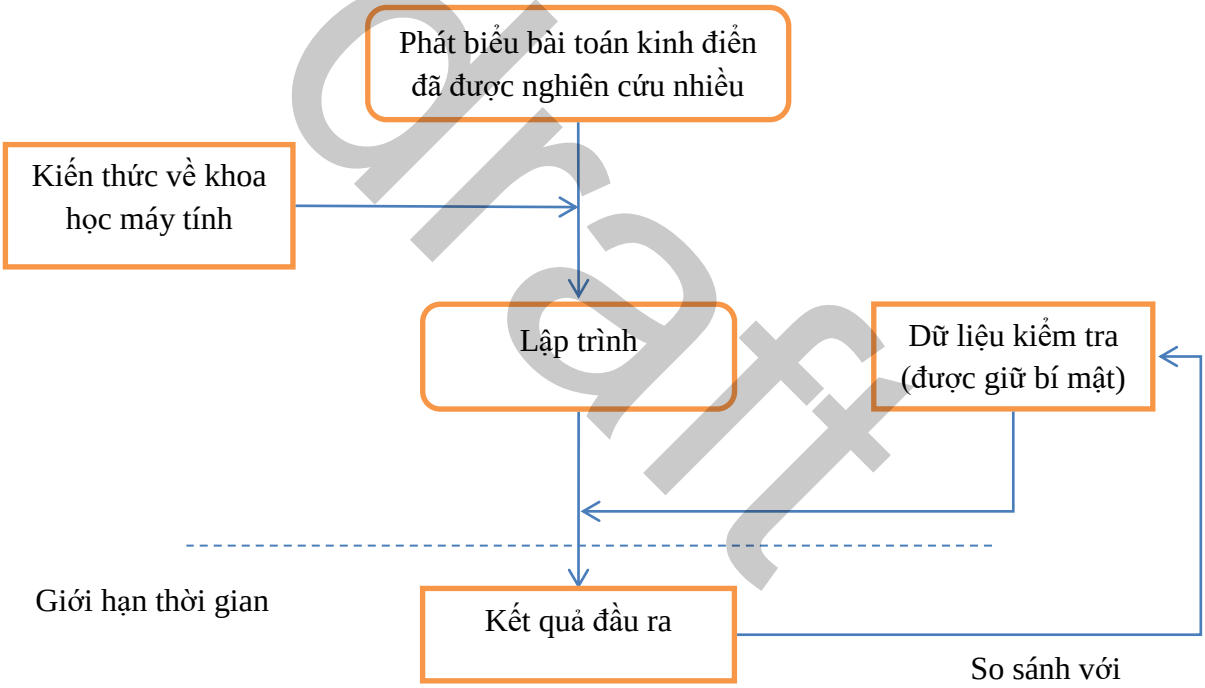
- Có 50% số test mà tất cả các giá trị là số nguyên (có dạng .00).
- Có 50% số test có $n \leq 700000$,

□

Bài toán trên cũng là một bài toán tuyển dụng ở một công ty mới khởi nghiệp ở Việt Nam nhưng rất có tiềm năng tương lai, www.bayo.vn. Bài toán ban đầu như sau: "*Hãy lập trình sắp xếp 1 triệu số nguyên mà chỉ được phép sử dụng 3MB bộ nhớ trong*". Bài toán tưởng chừng rất đơn giản nhưng đã phân loại rất tốt các ứng viên tham gia thi và kiểm tra được rất nhiều kỹ năng của lập trình viên, bao gồm các kỹ năng phân tích độ phức tạp thời gian và không gian thuật toán, các kỹ năng thiết kế thuật toán cũng như các kỹ thuật lập trình giải bài toán. Trong các phần tiếp theo ta sẽ thảo luận các kỹ năng giải bài quan trọng cần rèn luyện thông qua việc phân tích chi tiết bài toán *Obama Sort* và một số bài toán ví dụ khác.

1. Đôi nét về bài toán trong các kỳ thi lập trình

Thông thường các bài toán đưa ra là những vấn đề kinh điển đã được nghiên cứu nhiều của khoa học máy tính (KHMT), và yêu cầu giải chúng một cách nhanh nhất có thể.



Biểu đồ trên mô tả việc áp dụng kiến thức KHMT ở các mức độ khác nhau để lập trình giải quyết bài toán đặt ra. Đánh giá chương trình thông qua một bộ dữ liệu mẫu (test data) được giữ bí mật. Chương trình phải đưa ra bộ dữ liệu đầu ra (output) đúng tương ứng với các bộ dữ liệu đầu vào (input) trong một khoảng thời gian và bộ nhớ giới hạn cho biết trước. Do bài toán đã có sẵn lời giải nên yếu tố chính trong thi đấu lập trình là hãy *giải bài toán nhanh nhất có thể*.

2. Kỹ năng đọc đề

Đọc hiểu đề là kỹ năng đầu tiên cần sử dụng khi nhìn vào một bài toán. Hiểu sai đề dù chỉ một câu chữ nhỏ cũng có thể khiến công sức giải cả bài thành công cốc. Khi đọc đề cần đọc kỹ từng câu từng chữ trong các đoạn văn bản chính thức, các ví dụ hay hình vẽ minh hoạ chỉ để làm rõ thêm ý

trong văn bản. Đôi khi do sơ suất của người làm đề mà ví dụ minh hoạ có chỗ không khớp với văn bản, trong trường hợp đó, người giải bài cần phải làm theo những gì mô tả trong văn bản, cái dành cho mô tả mọi trường hợp bài toán, mà không nên làm theo ví dụ minh hoạ, cái mà chỉ đại diện cho một trường hợp ví dụ. Có thầy giáo trường chuyên đã từng chia sẻ rằng, trước kỳ thi quốc gia một tuần, không cho học sinh làm bài tập nữa, chỉ cho ngồi đọc đề bài các loại, đọc kỹ cho hiểu cặn kẽ ý tác giả muốn nói gì, rèn luyện kỹ năng đọc đề cho nhanh và chính xác. Quả nhiên lần đó đội tuyển không bạn nào bị bỏ sót hay hiểu nhầm đề và đạt thành tích tốt hơn hẳn so với các năm trước.

Cấu trúc đề bài thường bao gồm các phần theo trình tự sau:

- Kiến thức cơ sở của bài toán (Background);
- 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 là những trường hợp không quan trọng;
- Các hạn chế cho các test của bài toán;

Có khá nhiều tranh luận về việc có cần thiết hay không phần Background của bài toán. Đúng là nếu không có phần Background mà phát biểu trực tiếp bài toán như dạng các bài tập toán học thuần tuý thì cũng không ảnh hưởng gì đến việc tìm thuật toán và giải bài cả, thêm nữa người đọc dễ dàng hiểu bài toán hơn rất nhiều, có phần Background người đọc càng rối thêm. Vậy vì sao trong tất cả các kỳ thi lập trình, đại đa số các bài toán đều có phần Background với các lĩnh vực phong phú, rất nhiều bài dài dòng giải thích mô tả trên nhiều lĩnh vực rất khó hiểu, mà nếu như phát biểu thẳng tưng thì chỉ một vài dòng đơn giản?

Hãy cùng nhìn bài toán Obama Sort ở trên nếu như không có phần Background: "*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!

Tuy nhiên khi đọc đề bài như vậy liệu người đọc có hứng thú để giải bài, liệu môn tin học có còn đặc thù riêng hay chỉ như một nhánh của môn toán học, liệu học sinh giỏi có còn hứng thú ôn thi chọn đội tuyển môn Tin học? Nên nhớ rằng môn Tin học không có trong danh sách các môn thi tốt nghiệp phổ thông trung học. Học sinh ôn thi đội tuyển môn Tin học rất thiệt thòi vì sau khi tham dự kỳ thi Học Sinh Giỏi Quốc Gia xong lại phải quay lại ôn thi tốt nghiệp phổ thông trung học mà không bớt được ôn tập môn nào cả. Chỉ có khoảng hơn 30 học sinh trên tổng số hơn 400 học sinh dự thi được vào vòng 2 là được miễn thi tốt nghiệp sau đó. Vậy vì sao vẫn có số lượng đông đảo học sinh tham gia thi môn Tin học, con số không hề thua kém là bao so với các môn quan trọng khác có thi tốt nghiệp phổ thông trung học như Toán, Lý, Hoá, Văn, … ? Đó là vì sự đam mê và chấp nhận sự đánh cược của học sinh đối với môn Tin học. Vậy sự đam mê đó từ đâu mà có?

Tin học là lĩnh vực không thể thiếu ngày nay, có ứng dụng rộng khắp, hầu hết trong tất cả các ngành khác. Có thể nói đây là một lĩnh vực gây "nghiện" cho giới trẻ vì sự mới mẻ bất ngờ của nó. Rất nhiều thành tựu nhờ tin học đã được tạo ra và làm thay đổi điều kiện sống loài người. Thế hệ học sinh hiện tại từ bé đã được tiếp xúc với máy tính, mạng internet, những phần mềm thú vị, những sản phẩm công nghệ cao như máy tính bảng, điện thoại thông minh, game … Những thứ đó đã kích thích trí tò mò, sự sáng tạo của giới trẻ. Từ đó dẫn đến sự đam mê lúc nào không hay, khao khát khám phá chân trời bí ẩn của tin học.

Background của một bài toán tin học thường nói về một chủ đề ứng dụng của tin học. Những bài toán lập trình đó nếu phát biểu chân phương tưởng chừng như khô khan, nhưng thực ra đều có thể ứng dụng vào thực tế được. Chính phần Background là cầu nối vào ứng dụng thực tế của chúng. Tất nhiên có nhiều phần Background mà do trí tưởng tượng phong phú của người làm đề viết ra tuy có thể không ăn nhập với đề bài nhưng có tác dụng tạo hứng thú cho người đọc giải bài. Người làm đề phải có kiến thức đa ngành vừa sâu vừa rộng thì mới có thể viết được những phần Background hay và thuyết phục, thấy được sự ứng dụng hợp lý của bài toàn. Đằng sau phần Background đó là một sự gợi mở cả khối kiến thức đang chờ được khai phá tiếp. Dưới đây là một số ví dụ.

Ví dụ 1. Bài *Obama Sort* ở trên:

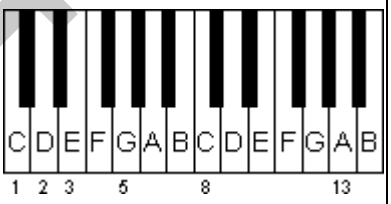
- Đây là một câu chuyện có thật rất hay và ý nghĩa, tham khảo thêm tại [2,3];
- Đó là cách tuyển dụng ở một số công ty có uy tín trong nước và quốc tế;
- Hiểu thêm một vấn đề thú vị về ngôn ngữ, thói quen sống;
- Sắp xếp một triệu phần tử mà chỉ sử dụng 1MB bộ nhớ trong là một bài toán kinh điển, được thảo luận rất nhiều về kỹ thuật và ứng dụng trên các diễn đàn lập trình như StackOverflow, Github, Quora,… Bài toán sử dụng các kỹ thuật rất hay về nén và các cấu trúc dữ liệu lưu trữ hiệu quả, tham khảo thêm tại [5,6].

Ví dụ 2. Bài *Bản vanxơ Fibonacci* VOI 2012:

Background: 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:

- số 1 tương ứng với nốt Đô (C),
- số 2 tương ứng với nốt Rê (D),
- số 3 tương ứng với nốt Mi (E),
- số 4 tương ứng với nốt Fa (F),
- số 5 tương ứng với nốt Sol (G),
- số 6 tương ứng với nốt La (A),
- số 7 tương ứng với nốt Si (B),
- số 8 tương ứng với nốt Đô (C),
- số 9 tương ứng với nốt Rê (D)



và cứ tiếp tục như vậy. Ví dụ, dãy gồm 6 số Fibonacci đầu tiên 1, 2, 3, 5, 8 và 13 tương ứng với dãy các nốt nhạc C, D, E, G, C và A.

Để xây dựng nhịp điệu vanxơ người ta đi tìm các đoạn nhạc có tính chu kỳ trong bản vanxơ Fibonacci. Đoạn nhạc được gọi là có tính chu kỳ nếu như có thể chia nó ra thành $k \geq 2$ đoạn giống hệt nhau. Ví dụ, đoạn nhạc GCAGCA là đoạn có tính chu kỳ, vì nó gồm hai đoạn giống nhau GCA.

Yêu cầu: Cho trước hai số nguyên dương u, v ($u < v$), hãy xác định độ dài đoạn nhạc dài nhất có tính chu kỳ của *bản nhạc* gồm dãy các nốt nhạc của bản vanxơ Fibonacci bắt đầu từ vị trí u kết thúc ở vị trí v.

Đây là một trong nhiều ứng dụng rất đẹp của dãy số tổ hợp Fibonacci, còn được gọi là dãy số của tự nhiên do rất nhiều hiện tượng tự nhiên tuân thủ theo tính chất của dãy số Fibonacci. Bạn đọc có thể thưởng thức bản nhạc vanxơ Fibonacci ở đây [4].

Ví dụ 3. Bài *QoS* VOI 2014:

> *Background*: Bài toán định tuyến kèm theo chất lượng dịch vụ bảo đảm trong các ứng dụng đa phương tiện như truyền hình ảnh và âm thanh theo yêu cầu là vấn đề thời sự trong những năm gần đây. Trong bài toán này, chúng ta quan tâm đến độ trễ của các đường truyền tin.
>
> Công ty cung cấp dịch vụ mạng ESI vừa thiết lập một mạng truyền thông giữa các điểm cung cấp dịch vụ và khách hàng, bao gồm n nút và m kênh nối trực tiếp một chiều giữa hai nút. Các nút được đánh số từ 1 đến n, trong đó nút 1 là nút nguồn. Các kênh nối được đánh số từ 1 đến m. Kênh nối thứ i cho phép truyền tin (một chiều) từ nút u_i tới nút v_i và có độ trễ là $c(u_i, v_i)$. Có không quá một kênh truyền tin từ một nút đến một nút khác. Một đường truyền tin từ nút nguồn đến nút t được biểu diễn dưới dạng một dãy liên tiếp các chỉ số của các nút, xuất phát từ 1 và kết thúc tại t. Độ trễ của đường truyền tin được định nghĩa là tổng độ trễ của các kênh nối trực tiếp trên đường truyền tin đó. Để khảo sát các đường truyền tin từ nút nguồn đến một nút t trong mạng, công ty ESI xác định C_{min} là độ trễ nhỏ nhất trong số tất cả các độ trễ của các kênh trong mạng và T_{min} là độ trễ của đường truyền tin từ nút nguồn đến nút t với độ trễ nhỏ nhất. Để đảm bảo dịch vụ đường truyền với chất lượng cao, đường truyền tin từ nút nguồn đến nút t phải thoả mãn điều kiện QoS (Quality of Service) sau đây: độ trễ của đường truyền tin phải nhỏ hơn hoặc bằng tổng số $T_{min}+C_{min}$. Sau đó, ESI sắp xếp tất cả các đường truyền tin từ nút nguồn đến nút t thoả mãn điều kiện QoS theo thứ tự từ điển. Theo định nghĩa của công ty ESI, đường truyền tin $(x_1, x_2, ..., x_p)$ được gọi là có thứ tự từ điển nhỏ hơn đường truyền tin $(y_1, y_2, ..., y_q)$, nếu:
>
> - hoặc là $x_1<y_1$;
> - hoặc là $(p < q)$ và $x_1 = y_1, ..., x_p = y_p$;
> - hoặc là tồn tại một chỉ số u $(1 < u \le p)$ sao cho $x_1 = y_1, ..., x_{u-1} = y_{u-1}, x_u < y_u$.
>
> *Yêu cầu*: Cho trước số nguyên dương k, hãy tìm đường truyền tin từ 1 đến t thoả mãn điều kiện QoS thứ k trong thứ tự từ điển.

Bản thân đề bài đã nêu bật lên ứng dụng thực tế bao gồm một số định nghĩa và kỹ thuật chuyên ngành mạng truyền thông sử dụng trong bài toán. Ở bài toán này, việc đọc kỹ từng câu từng chữ từng định nghĩa từng mô tả là rất quan trọng, bỏ sót hoặc nhầm ý nào có thể dẫn đến sai toàn bộ bài.

3. Kỹ năng phân loại bài toán

Sai khi đã hiểu đề bài, công việc tiếp theo là phải nhanh chóng phân loại được bài toán thuộc dạng nào. Thông thường mỗi bài toán thuộc vào một dạng chính nhất định. Tuy nhiên có nhiều bài là kết hợp của nhiều dạng khác nhau, ví dụ, bài toán tìm đường đi ngắn nhất giữa hai đỉnh trong đồ thị trọng số dương hay được giải bằng thuật toán Dijsktra, một thuật toán về đồ thị, nhưng còn phải áp dụng kỹ thuật tham lam, quy hoạch động và cấu trúc dữ liệu hàng đợi ưu tiên. Do đặc thù kỳ thi Quốc gia môn Tin học (VOI), các bài được tính điểm theo từng test nên có nhiều mức điểm khác nhau cho từng loại thuật toán áp dụng để giải bài. Dưới đây là bảng phân loại dạng bài chính trong các kỳ thi VOI những năm gần đây.

Loại bài	2016	2015	2014	2013	2012	2011
Duyệt toàn bộ		1	1	1	2	1
Chia để trị		1				
Tham lam	1		1		2	1
Quy hoạch động	3	1	1	2		1
Đồ thị	2	1	2	2	1	1
Toán học		1				1
Xử lý xâu				1		
Hình học tính toán		1	1			1
Ad-Hoc					1	

Bảng trên cũng là các dạng bài toán tối ưu hay gặp trong ngành KHMT. Tuy nhiên đó không phải là tất cả những kỹ thuật áp dụng trong việc giải bài. Một số kỹ thuật khác tuy không liệt kê ở bảng trên nhưng cũng rất quan trọng như: kỹ thuật sắp xếp, kỹ thuật quay lui nhánh cận, kỹ thuật sử dụng các cấu trúc dữ liệu đặc biệt, các kỹ thuật khó và hiếm gặp… Một người giải bài sau khi đọc đề sẽ gặp phải một trong ba tình huống sau:

- Loại A: Đã giải quyết loại bài toán này trước đây;
- Loại B: Đã gặp loại bài toán này nhưng chưa giải quyết được;
- Loại C: Chưa bao giờ gặp loại bài toán này.

Vậy để có thể xác định nhanh được loại bài toán, từ đó rút ra được thuật toán tối ưu nhất, hãy luyện tập để:

- Tăng số lần rơi vào trường hợp loại A, giảm số lần rơi vào trường hợp loại B;
- Luyện tập với các bài toán mang tính cổ điển để tăng khả năng tư duy thuật toán;
- Phát triển khả năng phân tích bài toán để phát hiện những điểm đặc biệt, mấu chốt của bài toán.

Với bài *Obama Sort*, người đọc có thể thấy ngay là sẽ phải áp dụng các thuật toán sắp xếp hợp lý để giải. Cái hay ở các thuật toán sắp xếp đó là sự đa dạng, mỗi thuật toán có một kỹ thuật xử lý riêng độc đáo: Chia để trị với Quick-sort và Merge-sort; cấu trúc dữ liệu hàng đợi ưu tiên với Heap-sort; xử lý bit với Raddix-sort… Các thuật toán sắp xếp có sự hiệu quả hơn kém nhau tùy vào tính chất của bộ dữ liệu. Ở bài toán này sẽ phải phân tích bài toán kỹ càng và chọn lựa giải thuật sắp xếp phù hợp để có thể giải được trọn vẹn.

4. Kỹ năng phân tích thuật toán

Phân tích độ phức tạp thuật toán là một lĩnh vực khó trong KHMT đòi hỏi rất nhiều tư duy cũng như kiến thức thuật toán và toán học. Tuy nhiên để giải bài hiệu quả trong kỳ thi, ta không nhất thiết phải học các kỹ thuật khó của lĩnh vực độ phức tạp thuật toán. Một số kinh nghiệm sau đây có thể giúp việc giải bài hiệu quả trong các kỳ thi lập trình.

Khi thiết kế thuật toán giải bài, điều quan trọng là phải đánh giá được chi phí của thuật toán về thời gian và bộ nhớ, đồng thời kiểm tra chi phí đó có thỏa mãn yêu cầu đề bài hay không. Nếu có nhiều

cách để giải bài toán thì hãy chọn thuật toán đối với mình là cài đặt đơn giản nhất mà vẫn đủ thỏa mãn yêu cầu, nghĩa là đúng và đủ nhanh trong giới hạn thời gian của đề bài. Tuy nhiên trong quá trình luyện tập, khi mà thời gian làm bài không bị giới hạn, thì hãy cố gắng tìm cách giải bài bằng thuật toán tốt nhất. Đó là cách chuẩn bị hiệu quả thực sự cho tương lai khi ta sẽ phải đối mặt các bài toán khó hơn.

Ví dụ 1.

> Bình thường lời giải với độ phức tạp $O(n^4)$ là một thuật toán tồi. Tuy nhiên nếu như hạn chế kích thước đầu vào $n \leq 50$ thì số lượng phép tính thực hiện tối đa là $c \times 50^4 = c \times 6.25$ triệu phép tính, với c là hệ số lập trình thông thường rất nhỏ. Thế thì thuật toán trên là chấp nhận được, thời gian chạy của nó sẽ dưới 1 giây vì đa phần máy tính hiện nay có thể thực hiện được ít nhất 100 triệu phép tính trong 1 giây.

Ví dụ 2.

> Bài toán có giới hạn kích thước đầu vào $n \leq 10^6$, hạn chế thời gian là 2 giây. Có hai thuật toán *A* và *B* giải đúng bài với độ phức tạp khác nhau.
>
> Thuật toán *A*: độ phức tạp $O(n\log n)$, số lượng phép tính ước lượng thực hiện tối đa là $c \times 10^6 \times \log(10^6) \approx c \times 1.7 \times 10^6$, thời gian chạy test lớn nhất sẽ dưới 1 giây.
>
> Thuật toán *B*: độ phức tạp $O(n^2)$, số lượng phép tính ước lượng thực hiện tối đa là $c \times 10^{12}$ và sẽ mất hàng trăm giây để thực hiện, không thể thõa mãn yêu cầu đề bài.

Tuy nhiên ngoài độ phức tạp thuật toán, cần phải lưu ý một số loại độ phức tạp khác không trực tiếp liên quan đến thời gian tính của thuật toán chính như: tính toán lượng lớn các phép toán dấu phảy động, có số lượng lớn các vòng lặp con, hoặc là có quá nhiều truy xuất vào ra (I/O). Trong các trường hợp đó thì chương trình sẽ mất nhiều thời gian hơn dự kiến. Trong kỳ thi, thời gian làm bài bị hạn chế, vì vậy hãy giảm bớt thời gian phân tích bài toán bằng cách luyện tập cách ước lượng xấp xỉ trong đầu. Hãy xử dụng một số mẹo ước lượng nhanh như: $2^{10} \approx 10^3$; cận trên của số nguyên 32-bit (*int, long* trong C hay *integer* trong pascal) là $2^{31}-1 \approx 2 \times 10^9$… Việc ước lượng nhanh và chính xác tương đối thời gian thực hiện thuật toán sẽ giúp ta sớm chọn được lời giải thích hợp.

Với bài *Obama Sort* thì cần phải phân tích rất nhiều khía cạnh của bài toán:

- Giới hạn kích thước đầu vào $n \leq 10^6$, như vậy chắc chắn phải sử dụng các thuật toán sắp xếp nhanh với độ phức tạp là $O(n\log n)$;
- Hạn chế thời gian là 1 giây. Việc đọc và ghi 1 triệu số thực với hai dấu phẩy động sẽ tốn rất nhiều thời gian. Nếu viết bằng C++ với các hàm *cin cout* sẽ mất vài giây cho việc đọc và ghi. Nếu viết bằng C hoặc Pascal với các hàm *scanf printf read write* sẽ mất khoảng 0.4-0.5 giây cho việc đọc ghi. Như vậy với các cách đọc ghi thông thường sẽ khiến chương trình chạy lâu hơn thời gian đề bài cho phép. Vì vậy cần phải dùng các kỹ thuật đọc ghi nhanh như: đọc từng ký tự kiểu *char* rồi ghép lại rồi đổi ra thành số thực và ngược lại khi ghi; hoặc đọc ghi cả cụm dữ liệu.
- Hạn chế bộ nhớ là 3MB. Nếu lưu toàn bộ 1 triệu phần tử 32 bit vào một mảng thì sẽ mất 4MB, vượt quá yêu cầu đề bài cho phép! Như vậy chỉ lưu được tối đa khoảng 700 nghìn phần tử. Nếu chỉ dừng lại ở đây thì có thể ăn điểm được 50% số test một cách dễ dàng. Làm thế nào để ăn điểm các test có một triệu phần tử?

Cách 1: Áp dụng tư tưởng chia để trị của Mergesort:

1. Đọc 400 nghìn phần tử vào mảng *A*;
2. Sắp xếp mảng *A* bằng một thuật toán sắp xếp nhanh;
3. Ghi mảng *A* ra một file tạm;
4. Đọc 600 nghìn phần tử vào mảng *A*;
5. Sắp xếp mảng *A* bằng một thuật toán sắp xếp nhanh;
6. Đọc các phần tử trong file tạm ở bước 3 và trộn với các phần tử mảng *A* rồi ghi ra file kết quả.

Cách 2: Bài toán có thể dễ dàng chuyển về làm việc với số nguyên bằng cách nhân mỗi số với 100. Mục tiêu của phương pháp này là lưu trữ cả 1 triệu số nguyên 32-bit trong 3MB bộ nhớ. Tư tưởng của phương pháp này là chỉ cần lưu số đầu tiên và tiếp theo là lưu trữ sự chênh lệch giữa các số liên tiếp nhau. Như vậy không phải giá trị nào cũng cần lưu 4 byte.

Cách 3: Áp dụng các thuật toán sắp xếp 1 triệu số nguyên 32-bit mà chỉ dùng 1MB bộ nhớ. Các phương pháp này đã được thảo luận nhiều trên các diễn đàn khoa học máy tính. Tư tưởng thuật toán chủ đạo sử dụng một số phương pháp nén số học hoặc sử dụng cấu trúc dữ liệu đặc biệt trên cây mã hóa Huffman để lưu trữ 1 triệu số nguyên trong 1MB, tham khảo thêm tại đây [5,6].

5. Một số kỹ năng khác

Kỹ năng làm chủ ngôn ngữ lập trình

Lợi ích sử dụng Pascal và C++ gần như tương đương nhau nếu như không sử dụng thư viện lập trình chuẩn. Làm chủ một ngôn ngữ lập trình bất kỳ mà không dùng thư viện thật đơn giản giống như người từ từ đi bộ rất dễ dàng vậy. Thư viện lập trình trang bị trong các ngôn ngữ lập trình có thể ví như người đi bộ được trang bị chiếc xe đạp vậy. Có thêm chiếc xe đạp có thể giúp người đi đến đích nhanh và đỡ mệt hơn. Tuy nhiên nếu điều khiển xe không thạo có thể đâm vào hàng rào. Sử dụng thư viện lập trình cũng có hai mặt của nó. Mặt tốt là giúp người lập trình hoàn thiện chương trình nhanh và ít lỗi hơn. Các thư viện chuẩn đã được cộng đồng lập trình dùng rất nhiều nên có thể nói việc sử dụng trực tiếp là tương đối dễ dàng, không có lỗi (bug), và kỹ thuật cài đặt tối ưu. Nhiều thuật toán cơ bản nhưng khó cài đặt và dễ gặp lỗi như Quicksort, tìm kiếm nhị phân, hàng đợi ưu tiên… đều có sẵn trong thư viện chuẩn, việc vận hành chỉ cần gọi một dòng lệnh đơn giản là xong.

Ví dụ sắp xếp một mảng *A* khai báo kiểu *vector<int>* trong C++:
```
sort(A.begin(),A.end();
```

Mặt hạn chế của việc dùng thư viện chuẩn đó là 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.

Ví dụ nếu gọi hàm sắp xếp của thư viện thì sau khi sắp xếp, thông tin về chỉ số vị trí ban đầu các phần tử trong mảng sẽ bị mất, muốn giữ lại thông tin này thì phải tùy biến hàm sắp xếp với kiểu dữ liệu cấu trúc struct sử dụng thêm một trường để lưu giữ lại thông tin về vị trí ban đầu và phải sửa lại toán tử so sánh cho trường chính của biến cấu trúc.

```
struct person {
   std::string name;
   int id;
};
bool sort_by_name( const person & lhs, const person & rhs )
{
   return lhs.name < rhs.name;
}
int main() {
   vector<person> people;
   std::sort( people.begin(), people.end(), sort_by_name );
}
```

Vì vậy nếu không thành thạo từng thư viện lập trình thì khi sử dụng rất dễ gặp phải lỗi và khó debug do không hiểu kỹ và khó can thiệp được vào phần code thuật toán chính của thư viện đó. Ngoài ra nếu thường xuyên dùng thư viện mà quên mất các thuật toán cơ bản và kỹ thuật cài đặt của nó thì sẽ rất khó khăn khi phải tự cài đặt lại thuật toán đó. Rất nhiều tình huống trong khi giải bài không thể dùng trực tiếp thư viện mà buộc phải tự cài đặt lại các thuật toán cơ bản.

Với bài *Obama Sort*, bước 2 và bước 5 ta chỉ cần gọi một hàm sắp xếp nhanh của thư viện chuẩn là đủ, ví dụ hàm `sort()`. Tuy nhiên vẫn phải tự cài đặt lại bước 6 là phần thuật toán trộn trong Mergesort.

Kỹ năng gõ nhanh

Hãy luyện tập để biến mình thành thợ đánh máy nhanh và chính xác, đừng để tốc độ gõ code là điểm yếu của mình. Đây là kỹ năng dễ dàng luyện tập thành thục nhất trong tất cả các kỹ năng. Luyện tập sao cho 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. Nếu như thiếu động lực để luyện tập kỹ năng này thì có thể thử dùng công cụ TypeRacer trên web tại địa chỉ http://play.typeracer.com/, công cụ có thể gây hứng thú cho người luyện tập với các tốc độ hiện nhanh chậm khác nhau phù hợp với trình độ tích lũy được của người chơi.

Kỹ năng test chương trình

Sau khi viết code chương trình xong, vấn đề tiếp theo rất quan trọng là kiểm nghiệm tính đúng đắn của chương trình. Một trong những kỹ thuật quan trọng để làm điều đó là viết chương trình kèm với tạo bằng tay bộ test kiểm tra chương trình. Dưới đây là một số kinh nghiệm test chương trình:

- Bộ test nên bao gồm cả ví dụ đầu vào đầu ra trong đề bài để đảm bảo chương trình đưa ra kết quả với khuôn dạng chính xác. Hãy sử dụng lệnh ‘fc’ trong windows hoặc ‘diff’ trong unix để so sánh output của chương trình với output mẫu;
- Với bộ dữ liệu nhiều test (multi test), hãy thêm trường hợp bộ dữ liệu vào có 2 bộ test giống hệt nhau và kiểm tra kết quả ra có giống nhau không nhằm loại bỏ trường hợp quên khởi tạo lại trong chương trình;
- Bộ test xây dựng nên bao gồm những test khó và những test biên như những trường hợp kích thước dữ liệu vào nhỏ nhất, ví dụ $n = 0$, $n = 1$ hoặc những trường hợp kích thước dữ liệu đạt giá trị lớn nhất để kiểm tra các lỗi tràn bộ nhớ, quá giới hạn hay xem chương trình chạy được trong giới hạn thời gian cho phép chưa;

- Quan sát kỹ đến định dạng kết quả ra của chương trình, vì có thể những trường hợp như dấu cách hay dấu xuống dòng cũng có thể làm chương trình ra kết quả sai;
- 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 bé mà thuật toán trực tiếp có thể ra kết quả nhanh được.

Kết luận

Với mỗi bài toán trong kỳ thi lập trình, đầu tiên ta phải đọc hiểu đề một cách chính xác, sau đó tìm cách thiết kế và phân tích thuật toán và cấu trúc dữ liệu sử dụng. Bước tiếp theo là chuyển lời giải thành chương trình càng nhanh càng tốt dưới áp lực thời gian và kết quả, đồng thời chương trình chạy cần không sinh lỗi và phải ra kết quả đúng trong thời gian hạn chế. Để làm tốt điều đó, kinh nghiệm của những lập trình viên đạt giải cao trong các kỳ thi lập trình là phải đam mê luyện tập tất cả các kỹ năng nêu trên trực tiếp trên càng nhiều bài toán càng tốt. 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. Cuối cùng giữ được tâm lý tốt và có được may mắn trong các kỳ thi cũng là một vấn đề quan trọng. Vấn đề này nên được thảo luận trong một chủ đề riêng.

Tài liệu tham khảo

1. Steven Halem. *Competitive Programming 3*. Lulu Publisher. 2013.
2. http://www.righto.com/2012/11/obama-on-sorting-1m-integers-bubble.html
3. https://www.youtube.com/watch?v=ZDDixe_N5sE
4. http://www.icompositions.com/music/song.php?sid=13228
5. http://stackoverflow.com/questions/12748246/sorting-1-million-8-digit-numbers-in-1-mb-of-ram
6. http://preshing.com/20121026/1mb-sorting-explained/