Giải Thuật Lập Trình

Nơi tổng hợp và chia sẻ những kiến thức liên quan tới giải thuật nói chung và lý thuyết khoa học máy tính nói riêng.

longest common subsequence

You are currently browsing articles tagged **longest common subsequence**.

Một số kĩ thuật cải tiến quy hoạch động I -- Advanced Dynamic Programming

June 13, 2015 in Uncategorized | 2 comments

Trong loạt bài này mình sẽ giới thiệu những kĩ thuật để cải tiến thuật toán quy hoạch động. Kĩ thuật trong bài này mình viết ở đây chủ yếu dựa trên notes của Jeff Erickson [1 (#jeff)].

Tiết kiệm bộ nhớ bằng chia để trị

Trong bài viết trước (http://www.giaithuatlaptrinh.com/?p=99) , mình giới thiệu phương pháp quy hoạch động để tính khoảng cách Edit Distance (ED) giữa hai xâu $A[1,2,\ldots,n]$, $B[1,2,\ldots,m]$ với thời gian $O(mn)$. Gọi $ED[i,j]$ là khoảng cách giữa hai xâu con $A[1,2,\ldots,i]$ và $B[1,2,\ldots,j]$. Công thức đệ quy sau chính là ý tưởng chính của thuật toán quy hoạch động:

$$ED[i,j] = \min\{ \begin{array}{l} ED[i-1,j]+1, \\ ED[i,j-1]+1, \\ ED[i-1,j-1]+I[A[i] \neq B[j]]\} \end{array}$$

Trong bài viết trước, chúng ta đã thảo luận hai cách thực thi thuật toán quy hoạch động với bộ nhớ tương ứng là $O(mn)$ và $O(m)$. Nếu chúng ta chỉ quan tâm đến khoảng cách giữa hai xâu, thuật toán thứ hai hiển nhiên sẽ là lựa chọn tốt. Bây giờ nếu bài toán yêu cầu in ra một dãy với tổng số lượng các các thao tác *"chèn", "xóa", "thay thế"* nhỏ nhất để biến A thành B, chúng ta sẽ làm thế nào?

Với thuật toán $O(mn)$, bằng cách truy ngược bảng $ED[1,2,\ldots,n][1,2,\ldots,m]$, chúng ta có thể dễ dàng thu được dãy tối ưu các thao tác. Tuy nhiên với cách thực thi với bộ nhớ $O(m)$, gần như không thể truy ngược lại vì chúng ta không lưu toàn bộ bảng $ED[1,2,\ldots,n][1,2,\ldots,m]$. Tuy nhiên, vào năm 1975, Dan Hirshberg (http://www.ics.uci.edu/~dan/) đề xuất phương pháp chia để trị không

những tính được khoảng cách giữa hai xâu với thời gian $O(mn)$ và bộ nhớ $O(m)$ mà còn tìm được một dãy tối ưu các thao tác để chuyển A thành B với cùng thời gian và bộ nhớ. Giải thuật của Dan Hirshberg sẽ là chủ đề chính của phần này.

Problem 1: Tìm dãy các thao tác: chèn, xóa, thay thế với số lượng các thao tác nhỏ nhất để biến xâu kí tự $A[1,2,\ldots,n]$ thành $B[1,2,\ldots,m]$ trong thời gian $O(mn)$ và bộ nhớ $O(m)$.

Thuật toán của Hirshberg dựa trên quan sát rằng bài toán chuyển xâu $A[1,2,\ldots,n]$ thành xâu $B[1,2,\ldots,m]$ có thể được chia thành hai bài toán con: bài toán thứ nhất chuyển xâu $A[1,2,\ldots,n/2]$ thành $B[1,2,\ldots,h]$ và bài toán thứ hai chuyển xâu $A[n/2+1,\ldots,n]$ thành $B[h+1,2,\ldots,m]$. Vấn đề ở đây là làm sao để tìm điểm chia h. Hirshberg chứng minh bổ đề sau:

Lemma 1: Tồn tại một thuật toán tìm điểm chia h với thời gian $O(mn)$ và bộ nhớ $O(m)$.

Ta sẽ chứng minh bổ đề này sau. Bây giờ giả sử chúng ta đã có thuật toán Hirshberg$(A[1,2,\ldots,n],B[1,2,\ldots,m])$ tìm điểm chia h trong thời gian $O(mn)$ và bộ nhớ $O(m)$. Ta có thể gọi đệ quy trên các bài toán con để giải bài toán ban đầu. Chi tiết như trong giả mã sau:

```
ED(A[1,2,...,n], B[1,2,...,m]):
    if (n < 2 or (m < 2)
        print the edit sequence
    else
        h ← Hirshberg(A[1,2,...,n], B[1,2,...,m])
        ED(A[1,2,...,n/2], B[1,2,...,h]):
        ED(A[n/2+1,...,n], B[h+1,...,m]):
```

Có thể dễ thấy là bộ nhớ cần thiết của thuật toán là $O(m)$ vì bộ nhớ cần thiết của thủ tục Hirshberg$(A[1,2,\ldots,n],B[1,2,\ldots,m])$ là $O(m)$. Thời gian tính của thuật toán được cho bởi công thức đệ quy sau:

$$T(n,m)=\begin{cases} n, & \text{if } m\leq 1 \\ m, & \text{if } n\leq 1 \\ T(n/2,h)+T(n/2,m-h)+O(mn) & \text{otherwise} \end{cases}$$

Bằng phương pháp quy nạp (http://www.giaithuatlaptrinh.com/?p=22) , ta có thể chứng minh rằng $T(m,n)=O(mn)$.

Giải thuật Hirshberg tìm điểm chia

Giải thuật Hirshberg tìm điểm chia bằng phương pháp quy hoạch động. Gọi $H[i,j]$ là điểm chia khi biến xâu con $A[1,2,\ldots,i]$ thành $B[1,2,\ldots,j]$. Hay nói cách khác, bài toán chuyển xâu $A[1,2,\ldots,i]$ sang xâu con $B[1,2,\ldots,j]$ có thể được chia thành hai bài toán: bài toán chuyển xâu

con $A[1,2,\ldots,n/2]$ sang xâu $B[1,2,\ldots,H[i,j]]$ và bài toán chuyển xâu con $A[n/2+1,\ldots,i]$ sang xâu con $B[H[i,j]+2,\ldots,j]$. Dễ thấy:

1. Nếu $i < n/2$, $H[i,j]$ không tồn tại, ta sẽ khởi tạo là ∞.
2. Nếu $i = n/2$, j chính là điểm chia mà ta cần tìm. Như vậy $H[i,j] = j$.
3. Nếu $i > n/2$ ta có 3 trường hợp con:

1. Nếu $ED[i,j]$ thu được từ phép xóa $A[i]$, như vậy điểm chia của $A[1,2,\ldots,i]$ và $B[1,2,\ldots,j]$ chính là điểm chia của $A[1,2,\ldots,i-1]$ và $B[1,2,\ldots,j]$. Do đó, $H[i,j] = H[i-1,j]$
2. Nếu $ED[i,j]$ thu được từ phép chèn $B[j]$ vào sau $A[i]$, như vậy điểm chia của $A[1,2,\ldots,i]$ và $B[1,2,\ldots,j]$ chính là điểm chia của $A[1,2,\ldots,i]$ và $B[1,2,\ldots,j-1]$. Do đó, $H[i,j] = H[i,j-1]$
3. Nếu $ED[i,j]$ thu được từ phép thay thế $A[i]$ bằng $B[j]$, như vậy điểm chia của $A[1,2,\ldots,i]$ và $B[1,2,\ldots,j]$ chính là điểm chia của $A[1,2,\ldots,i-1]$ và $B[1,2,\ldots,j-1]$. Do đó, $H[i,j] = H[i-1,j-1]$

Tổng hợp lại, ta có công thức đệ quy sau của $H[i,j]$:

$$H[i,j] = \begin{cases} \infty, & \text{if } i \le n/2 \\ j, & \text{if } i = n/2 \\ H[i-1,j] & \text{if } i > n/2 \text{ and } ED[i,j] = ED[i-1,j]+1 \\ H[i,j-1] & \text{if } i > n/2 \text{ and } ED[i,j] = ED[i,j-1]+1 \\ H[i-1,j-1] & \text{otherwise} \end{cases}$$

Như vậy, điểm chia $\textsc{Hirshberg}(A[1,2,\ldots,n], B[1,2,\ldots,m]) = H[n,m]$. Dựa vào công thức đệ quy ở trê, ta có thể tính được $H[n,m]$ với thời gian $O(mn)$ và bộ nhớ $O(m)$. Chi tiết coi như bài tập cho bạn đọc (tham khảo thêm tại đây (http://www.giaithuatlaptrinh.com/?p=99)).

Ví dụ bảng $ED[1,2,\ldots,n][1,2,\ldots,m]$ và bảng $H[1,2,\ldots,n][1,2,\ldots,m]$ với hai xâu $A[1,2,\ldots,n] = ALTRUISTIC$, $B[1,2,\ldots,m] = ALGORITHM$. Hình ảnh được lấy từ [1 (#jeff)]

ED		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	0	1	2	3	4	5	6	7	8
L	2	1	0	1	2	3	4	5	6	7
T	3	2	1	1	2	3	4	4	5	6
R	4	3	2	2	2	2	3	4	5	6
U	5	4	3	3	3	3	3	4	5	6
I	6	5	4	4	4	4	3	4	5	6
S	7	6	5	5	5	5	4	4	5	6
T	8	7	6	6	6	6	5	4	5	6
I	9	8	7	7	7	7	6	5	5	6
C	10	9	8	8	8	8	7	6	6	6

H		A	L	G	O	R	I	T	H	M
	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
A	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
L	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
T	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
R	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
U	0	1	2	3	4	5	6	7	8	9
I	0	1	2	3	4	5	5	5	5	5
S	0	1	2	3	4	5	5	5	5	5
T	0	1	2	3	4	5	5	5	5	5
I	0	1	2	3	4	5	5	5	5	5
C	0	1	2	3	4	5	5	5	5	5

Chính vì kĩ thuật Hirshberg mà thông thường, với các bài toán quy hoạch động, nếu chúng ta có thể tìm được giá trị tối ưu trong thời gian T và bộ nhớ M, chúng ta có thể tìm được ít nhất một phương án tối ưu với cùng thời gian và bộ nhớ.

Lợi dụng tính thưa của bảng QHD để tiết kiệm thời gian

Các bài toán quy hoạch động thông thường liên quan đến việc xây dựng bảng quy hoạch hai hoặc ba chiều. Mỗi ô của bảng được cập nhật dựa trên các ô liền kề của bảng. Một điểm chúng ta có thể nhận thấy là không phải tất cả các ô đều được sử dụng trong xây dựng phương án tối ưu. Do đó một cách để tiết kiệm thời gian và bộ nhớ là chúng ta chỉ lưu lại giá trị của những ô mà có thể được sử dụng trong xây dựng phương án. Nếu số lượng các ô đó nhỏ hơn nhiều kích thước cuả toàn bảng, chúng ta sẽ tiết kiệm thêm được thời gian và bộ nhớ. Trong bài này mình lấy bài toán dãy con chung dài nhất (Longest Common Subsequence - LCS) làm ví dụ:

Problem 2: Cho hai xâu kí tự $A[1,2,\ldots,n], B[1,2,\ldots,m]$ có độ dài lần lượt là n và m. Tìm xâu con chung dài nhất của hai dãy này.

Ví dụ: $A[1,\ldots,11] = ALTRUISTIC?$ và $B[1,\ldots,11] = ALGORITHMS?$ có xâu con chung dài nhất là $C[1,\ldots,5] = ALRIT?$ có chiều dài là 6 (ta thêm kí tự ? vào cuối hai xâu để việc lập trình đơn giản hơn).

Tương tự như bài toán Edit Distance (http://www.giaithuatlaptrinh.com/?p=99), nếu ta gọi $LIS[i,j]$ là chiều dài dãy con chung dài nhất của hai xâu con $A[1,2,\ldots,i], B[1,2,\ldots,j]$. Ta có công thức đệ quy sau:

$$LCS[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ LCS[i-1,j-1]+1, & \text{if } A[i] = B[j] \\ \max\{LCS[i,j-1], LCS[i-1,j]\}, & \text{otherwise} \end{cases}$$

Dựa vào công thức trên, ta có thuật toán quy hoạch động giải bài toán trên với thời gian $O(mn)$. Thuật toán quy hoạc động cũng là thuật toán tối ưu để giải bài toán này theo nghĩa nếu $n = m$, không tồn tại thuật toán chạy với thời gian ít hơn $n^{2-\epsilon}$ với mọi $\epsilon > 0$ trừ khi Giả Thiết Thời Gian Lũy Thừa (Exponential Time Hypothesis (http://en.wikipedia.org/wiki/Exponential_time_hypothesis)) là sai [3].

```
LCS(A[1,2,...,n], B[1,2,...,m]):
    for i ← 0 to n
        LCS[i,0] = 0
    for j ← 1 to m
        LCS[0,j] = 0
    for i ← 1 to n
        for j ← 1 to m
            if A[i] = B[j]
                LCS[i,j] ←LCS[i-1,j-1] + 1
            else
                LCS[i,j] ← max{LCS[i-1,j], LCS[i,j-1]}
    return LCS[n,m]
```

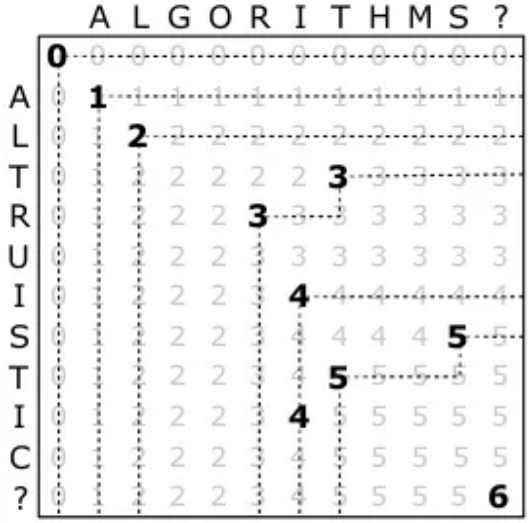
Code của thuật toán bằng C. Trong code này, đầu vào A, B là dãy các số nguyên từ 0-9 thay vì là kí tự.

```
+ expand source (#)
```

Tuy nhiên trong một số trường hợp đặc biệt, mà ở đây là trường hợp bảng QHD "thưa", ta có thể thiết kế thuật toán nhanh hơn $O(mn)$. Nếu để ý kĩ

ta sẽ thấy các ô của bảng $LCS[1,2,\ldots,n][1,2,\ldots,m]$ được sử dụng để xây dựng phương án tối ưu là các ô $LCS[i,j]$ với $A[i]=B[j]$. Ta gọi các ô $LCS[i,j]$ với $A[i]=B[j]$ là các ô *hữu ích*.

Với ví dụ $A[1,\ldots,10]=ALTRUISTIC$ và $B[1,\ldots,10]=ALGORITHMS$, ta có tất cả 9 ô hữu ích. Hình sau được lấy từ [1 (#jeff)].



Ta có thể thấy giá trị của các ô hữu ích phụ thuộc trực tiếp vào nhau theo công thức đệ quy sau:

$$LCS[i,j]=\begin{cases}0, & \text{if } i=j=0\\ \max\{LCS[i',j']\,|\,A[i']=B[j'],i'<i,j'<j\}+1, & \text{othwise}\end{cases}$$

Ở công thức trên, $LCS[i,j]$ là ô hữu ích ($A[i]=B[j]$). Gọi K là số lượng các ô hữu ích. Ta có thể tính trước danh sách các ô hữu ích và lưu trong mảng M trong thời gian $O(m\log m+n\log n+K)$ như sau:

```
FindUsefulPairs(A[1,2,...,n], B[1,2,...,m]):
    sort A and keep track of indices in array I
    sort B and keep track of indices in array J
    i ← 1; j ← 1
   while i ≤ m and j ≤ n
      if A[i] < B[j]
         i ← i + 1
      else if A[i] > B[j]
         j ← j + 1
      else                          [[found an useful pair]]
         ii ← i
         while A[ii] = A[i]
            jj ← j
            while B[jj] = B[i]
               add pair (I[ii], J[jj]) to M
               jj ← jj + 1
            ii ← ii + 1
         i ← ii; j ← jj
```

Trong code bằng C của giả mã dưới đây, thay vì lưu giữ mảng chỉ số I, J như trong giả mã, mình dùng cấu trúc dữ liệu *pair* để lưu chỉ số cùng với dữ liệu.

+ expand source (#)

Sau khi đã tìm được chỉ số của các ô hữu ích, ta có thể tìm chiều dài của dãy con chung dài nhất bằng quy hoạch động dựa vào công thức đệ quy ở trên (#rec). Giả mã như sau:

SparseLCS($A[1,2,\ldots,n], B[1,2,\ldots,m]$):
  $M[1,2,\ldots,K] \leftarrow$ FindUsefulPairs(A,B)
  $M[K+1] \leftarrow (n+1, m+1)$ *[[add dummy symbols to A, B]]*
  Sort M lexicographically
  for $k \leftarrow 1$ to $K+1$
    $(i,j) \leftarrow M[k]$
    LCS$[k] \leftarrow 1$
    for $\ell \leftarrow 1$ to $k-1$
      $(i',j') \leftarrow M[\ell]$
      if $(i' < i)$ and $(j' < j)$
        LCS$[k] \leftarrow \max\{$LCS$[k], 1+$LCS$[\ell]\}$
  return $LCS[K+1] - 1$

Code của thuật toán bằng C. Code đầy đủ được cho ở cuối bài.

+ expand source (#)

Phân tích thời gian Thuật toán tìm các ô hữu dụng có thời gian tính là $O(n \log n + m \log m + K)$. Sắp xếp mảng M có thể thực hiện trong thời gian $K \log K$. Phần quy hoạch động cập nhật mảng $LCS[1,2,\ldots,K+1]$ có thời gian tính $O(K^2)$. Do đó, tổng thời gian tính toán của thuật toán SparseLCS là $O(m \log m + n \log n + K^2)$. Như vậy, nếu $K = o(\sqrt{mn})$ (bảng quy hoạch động "thưa") thì thuật toán trên sẽ nhanh hơn thuật toán quy hoạch động ban đầu.

Bằng kĩ thuật tương tự với kĩ thuật trong bài Edit Distance (http://www.giaithuatlaptrinh.com/?p=99), chúng ta có thể giảm thời gian cập nhật mảng $LCS[1,2,\ldots,K+1]$ xuống còn $O(K \log K)$. Do đó thời gian tính có thể giảm xuống còn $O(m \log m + n \log n + K \log K)$. Chi tiết coi như bài tập cho bạn đọc.

Code: sparse-lcs (http://www.giaithuatlaptrinh.com/wp-content/uploads/2015/06/lcs.c.tar.gz).

Tham khảo

[1] Jeff Erickson, Advanced Dynamic Programming Lecture Notes (http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/06-sparsedynprog.pdf), UIUC.
[2] D. S. Hirschberg. 1975. *A linear space algorithm for computing maximal common subsequences*. Commun. ACM 18, 6 (June 1975), 341-343.
[3] Abboud, Amir, Arturs Backurs, and Virginia Vassilevska Williams. *Quadratic-Time Hardness of LCS and other Sequence Similarity Measures*. arXiv preprint arXiv:1501.07053 (2015).

Tags: advanced trick, dynamic programming, edit distance, longest common subsequence