Lời cảm ơn

Lời đầu tiên em xin chân thành cảm ơn thầy Đinh Điền, người đã trực tiếp hướng dẫn em hoàn thành luận văn này. Thầy là người đã truyền thụ cho em rất nhiều kiến thức về tin học và ngôn ngữ học, giúp em có được hiểu biết sâu hơn về một trong các ứng dụng có ý nghĩa vô cùng to lớn trong cuộc sống của tin học —- vấn đề dịch máy.

Em cũng xin chân thành cảm ơn các thầy cô trong khoa Công nghệ thông tin đã tận tình chỉ bảo và giúp đỡ cho em trong suốt thời gian em học đại học và hỗ trợ em trong quá trình thực hiện luận văn.

Con xin chân thành cảm ơn ba mẹ, các anh và những người thân trong gia đình đã nuôi dạy, tạo mọi điều kiện tốt nhất cho con học tập và động viên con trong thời gian thực hiện luận văn.

Và cuối cùng, xin gởi lời cảm ơn đến tất cả bạn bè và nhất là các bạn trong nhóm VCL (Vietnamese Computational Linguistics), những người đã giúp đỡ và hỗ trợ trong quá trình hoàn thiện luận văn này.

Tp. Hồ Chí Minh, tháng 7 năm 2004
Nguyễn Thái Ngọc Duy — 0012020

Mục lục

Danh sách hình vẽ

Danh sách bảng

Tóm tắt luận văn

Vấn đề nghiên cứu Xây dựng chương trình bắt lỗi chính tả tiếng Việt nhằm phát hiện và đề nghị từ thay thế cho các lỗi chính tả thường gặp. Đề tài này chỉ giới hạn bắt lỗi chính tả trong các văn bản hành chính.

Cách tiếp cận Sử dụng cách tiếp cận như sau: Phát sinh những câu có khả năng thay thế dựa trên các nguyên nhân gây lỗi chính tả, sau đó sử dụng mô hình ngôn ngữ dựa trên từ để xác định câu đúng nhất. Dựa trên sự khác biệt giữa câu gốc và câu được chọn, ta sẽ có thể biết được từ nào sai chính tả, và cách viết đúng chính tả là như thế nào. Mô hình sử dụng ngữ liệu thô chưa tách từ, tự huấn luyện để phù hợp với mục đích của mô hình.

Mô hình bắt lỗi chính tả theo hai giai đoạn. Giai đoạn thứ nhất tìm và yêu cầu người dùng sửa lỗi tiếng (những tiếng không tồn tại trong tiếng Việt). Giai đoạn này chủ yếu sửa những lỗi sai do nhập liệu từ bàn phím. Giai đoạn hai được dùng để bắt lỗi từ. Tất cả các cách tách từ có thể có của câu nhập vào được xây dựng dựa trên lưới từ. Sau đó lưới từ này được mở rộng để thêm vào những câu mới nhờ áp dụng các nguyên nhân gây lỗi chính tả, nhằm tạo ra câu đúng từ câu sai chính tả. Mô hình ngôn ngữ được áp dụng để đánh giá từng cách tách từ trong lưới từ và chọn ra cách tách từ tốt nhất. Dựa vào cách tách từ này và câu gốc, ta sẽ xác định từ sai chính tả và đưa ra từ đề nghị. Một số heuristic được áp dụng để hiệu chỉnh lưới từ nhằm tạo ra một kết quả

tốt hơn.

Mô hình ngôn ngữ được dùng là trigram dựa trên từ. Việc huấn luyện trigram dựa trên ngữ liệu đã tách từ sẵn có và tạo thêm ngữ liệu mới từ ngữ liệu thô chưa tách từ. Với ngữ liệu thô, mô hình ngôn ngữ được huấn luyện để thu thập tất cả cách tách từ có thể có của mỗi câu trong ngữ liệu huấn luyện thay vì sử dụng bộ tách từ rồi huấn luyện trên cách tách từ tốt nhất đó. Các trigram trong mỗi cách tách từ được thu thập dựa theo khả năng của mỗi cách tách từ. Trigram của cách tách từ tốt hơn sẽ có trọng số cao hơn các cách tách từ còn lại

Kết quả Chương trình hoạt động tốt và đạt được một số kết quả nhất định. Các lỗi sai âm tiết được phát hiện hoàn toàn. Lỗi sai từ có thể phát hiện đến trên 88%. Các loại lỗi khác đạt độ chính xác rất cao.

Chương trình có thể được cải tiến thêm bằng cách sử dụng các thông tin cao cấp hơn như thông tin từ loại, thông tin cú pháp, ngữ nghĩa . . . nhằm nâng cao độ chính xác hơn nữa.

Chương 1

Mở đầu

Mục lục

Ngôn ngữ là một phần quan trọng của đời sống, là phương tiện chuyển tải thông tin trong đời sống. Trong thời đại bùng nổ thông tin hiện nay thì ngôn ngữ đóng vai trò hết sức quan trọng, đặc biệt là ngôn ngữ viết.

Khi viết, đôi khi ta mắc phải những lỗi sai chính tả. Chữ quốc ngữ là thứ chữ ghi âm nên một số âm tiết rất dễ nhầm lẫn, khó phân biệt rõ ràng. Ngôn ngữ nói ở những vùng khác nhau lại có những điểm khác nhau. Những điểm khác nhau này rất dễ gây ra những lỗi chính tả khi viết nếu người viết không để ý khi sử dụng tiếng Việt.

Những thao tác chuyển thông tin ở dạng khác thành văn bản cũng có thể gây ra lỗi chính tả. Ví dụ, nếu nhập liệu không cẩn thận dẫn đến lỗi sai chính tả. Khi ghi lại lời nói của người khác mà người đó sử dụng giọng địa phương cũng có thể dẫn đến lỗi chính tả. Quét các văn bản giấy thành văn bản điện

tử, sử dụng chương trình nhận dạng chữ, cũng có thể dẫn đến lỗi chính tả do chương trình nhận dạng nhầm lẫn ...

Văn bản dễ bị sai chính tả do nhiều yếu tố khách quan. Để kiểm lỗi chính tả những văn bản này đòi hỏi nhiều công sức và thời gian, đặc biệt khi khối lượng văn bản bùng nổ như hiện nay. Do đó cần có một công cụ hỗ trợ kiểm lỗi chính tả, giúp nhanh chóng phát hiện lỗi chính tả và đề nghị cách khắc phục.

Trong thời đại tin học hoá, máy tính được tận dụng để giảm thiểu công sức của con người, đồng thời tăng tính hiệu quả. Tin học đã được áp dụng trong nhiều lĩnh vực khác nhau và chứng tỏ tính hiệu quả của nó. Tuy nhiên, việc ứng dụng tin học nhằm hỗ trợ bắt lỗi chính tả tiếng Việt chỉ mới được bắt đầu trong thời gian gần đây. Những ứng dụng bắt lỗi chính tả hiện có vẫn còn khá đơn giản, hoặc chưa hiệu quả, chưa đáp ứng được nhu cầu thực tế. Luận văn này đề ra một giải pháp khác để bắt lỗi chính tả, với hy vọng góp phần nâng cao chất lượng ứng dụng bắt lỗi chính tả tiếng Việt bằng máy tính.

1.1 Nội dung bài toán

Bài toán có thể được phát biểu như sau: Cho một văn bản tiếng Việt. Tìm tất cả các từ sai chính tả trong văn bản và đề nghị cách giải quyết lỗi nếu có.

Do ngôn ngữ là một lĩnh vực quá rộng. Việc bắt lỗi chính tả tiếng Việt tổng quát là cực kỳ khó khăn. Do vậy đề tài này chỉ giới hạn bắt lỗi chính tả trong các văn bản hành chính.

Chỉ sử dụng từ điển từ, từ điển tiếng và ngữ liệu thô làm đầu vào.

Khái niệm từ ở đây là "từ từ điển" — tức là các từ đơn, từ ghép, cụm từ được lưu trong từ điển.

Lỗi chính tả ở đây bao gồm chủ yếu hai loại lỗi sau:

- Lỗi nhập liệu sai: lỗi gõ thiếu chữ, gõ dư chữ, gõ nhầm vị trí hai chữ liên tiếp nhau, gõ nhầm một chữ bằng một chữ khác, sai sót do bộ gõ tiếng Việt ...

- Lỗi phát âm sai: chủ yếu là do đặc điểm phát âm của từng vùng, dẫn đến sai chính tả khi viết.

Không xử lý lỗi từ vựng, lỗi cú pháp.

Giả định rằng, nếu từ bị sai chính tả, thì chỉ sai bởi một trong những lý do nêu trên *một lần* (mỗi từ chỉ sai một lỗi chính tả, lỗi đó thuộc một trong những loại đã nêu). Nghĩa là không xét những trường hợp sai chính tả, vừa gõ nhầm chữ này bằng chữ khác, vừa gõ dư chữ.

Giả định người dùng chỉ sử dụng một trong hai cách gõ tiếng Việt là VNI hoặc TELEX.

Văn bản tiếng Việt được coi là thuần Việt. Không kiểm tra chính tả đối với những từ nước ngoài. Những từ nước ngoài và các ký hiệu khác đều bị coi là sai chính tả.

1.2 Đặc điểm

Bắt lỗi chính tả, xét từ quan điểm tin học, là một bài toán khó. Khó bởi vì ngôn ngữ là một phần rất quan trọng của đời sống xã hội, nó bao hàm rất nhiều khía cạnh của văn hoá, xã hội. Ngôn ngữ dùng để diễn đạt suy nghĩ, chuyển tải thông tin, nên nó chứa đựng một khối lượng tri thức đồ sộ. Để xử lý ngôn ngữ tự nhiên một cách đúng đắn đòi hỏi một trình độ nhất định. Bởi vậy, việc giải quyết bài toán bắt lỗi chính tả bằng máy tính là hết sức khó khăn.

Bắt lỗi chính tả đôi khi được mở rộng để phát hiện những lỗi khác trong văn bản như lỗi cú pháp, lỗi từ vựng ... Điều này cũng dễ hiểu vì người sử

dụng cần một chương trình giúp họ phát hiện và loại bỏ tất cả các lỗi trong văn bản, không quan trọng lỗi đó thuộc loại lỗi nào. Thông thường những lỗi từ vựng thường bị nhầm lẫn với lỗi chính tả, buộc chương trình bắt lỗi chính tả phải phát hiện cả lỗi từ vựng. Đây là một vấn đề khó vì để bắt lỗi từ vựng, đôi khi cần phải hiểu nội dung cả văn bản.

Nếu tìm hiểu sâu hơn về bài toán này, ta lại gặp một khó khăn khác do bản chất của tiếng Việt. Đối với tiếng Việt, cũng như một số ngôn ngữ châu Á khác, một từ chính tả có thể không tương ứng với một "từ" trên văn bản. Đối với các thứ tiếng châu Âu, ta có thể dễ dàng nhận ra một từ, do các từ được phân cách bằng khoảng trắng. Điều đó không đúng với tiếng Việt. Trong tiếng Việt, các tiếng được phân cách bởi khoảng trắng, không phải các từ. Điều này dẫn đến một bài toán mới: tách từ trong tiếng Việt. Do tiếng Việt là ngôn ngữ nói sao viết vậy, nên rất ít khi gặp lỗi sai về tiếng. Đa số các lỗi chính tả là lỗi sai từ, nên việc xác định đâu là từ cực kỳ quan trọng.

Vấn đề càng trở nên khó khăn hơn khi phải thực hiện cùng lúc hai bài toán là tách từ tiếng Việt và kiểm tra chính tả. Thật sự là tách từ tiếng Việt trước, sau đó bắt lỗi chính tả. Tuy nhiên, do khi tách từ thường ngầm định là dữ liệu đúng chính xác. Nên khi phải tách từ trước bước kiểm tra chính tả, ngầm định trên không còn đúng. Bài toán tách từ trở thành một bài toán khác, phức tạp hơn.

Đề tài này chỉ sử dụng các cách hình thành lỗi chính tả, từ điển từ tiếng Việt và ngữ liệu văn bản dạng thô. Việc không thể áp dụng được những thông tin cấp cao hơn như từ loại, cú pháp, ngữ nghĩa ... sẽ làm chương trình không thể phát huy tối đa khả năng.

1.3 Hướng giải quyết

Bài toán bắt lỗi chính tả đã được tìm hiểu từ rất lâu. Tuy nhiên đa số đều tập trung vào các ngôn ngữ phổ dụng ở châu Âu. Trong khi đó các ngôn ngữ châu Á, đặc biệt là tiếng Việt, có những đặc trưng riêng, đặt ra nhiều thách thức mới. Bài toán bắt lỗi chính tả trên các ngôn ngữ châu Á như tiếng Trung Quốc, tiếng Hàn Quốc, tiếng Nhật, tiếng Thái và tiếng Việt chỉ bắt đầu được nghiên cứu gần đây.

Đối với các ngôn ngữ châu Âu, cách giải quyết đơn giản là dựa vào từ điển. Nếu một từ trên văn bản không có trong từ điển nghĩa là từ đó sai chính tả.

Đối với các ngôn ngữ như tiếng Trung Quốc, tiếng Nhật . . . , nhiều giải pháp được đề ra để giải quyết bài toán. Tuy nhiên hầu hết các giải pháp đều dựa trên ý tưởng áp dụng tập nhầm lẫn để phát sinh các từ gần đúng, sau đó sử dụng mô hình ngôn ngữ để định lượng, xác định xem từ nào là đúng nhất.

Đề tài này áp dụng cách giải quyết truyền thống, so sánh từ dựa trên từ điển. Nếu từ không có trong từ điển nghĩa là sai chính tả, từ đó đưa ra những gợi ý thích hợp.

Bài toán đặt ra một bài toán con khác là tách từ tiếng Việt trong điều kiện văn bản bị sai chính tả. Cách giải quyết bài toán này là phát sinh mọi cách tách từ có thể, sử dụng tập nhầm lẫn, và sau đó áp dụng mô hình ngôn ngữ để tìm ra cách tách từ đúng nhất. Tập nhầm lẫn được phát sinh dựa vào nguồn gốc gây lỗi. Các lỗi về phát âm sẽ dựa trên các thói quen phát âm của từng vùng để tạo tập nhầm lẫn. Các lỗi về nhập liệu sẽ dựa trên các nghiên cứu về lỗi nhập liệu để đưa ra tập nhầm lẫn tương ứng.

1.4 Bố cục luận văn

Luận văn được chia thành các chương sau:

- Chương 1 giới thiệu chung về luận văn, các vấn đề cần giải quyết, đặc điểm, phạm vi của bài toán và hướng giải quyết.
- Chương 2 trình bày cơ sở lý thuyết ngôn ngữ học.
- Chương 3 trình bày cơ sở lý thuyết toán học/tin học. Các mô hình được áp dụng để giải quyết bài toán.
- Chương 4 trình bày mô hình đề nghị cho bắt lỗi chính tả tiếng Việt.
- Chương 5 trình bày các chi tiết khi cài đặt chương trình.
- Chương 6 tóm tắt luận văn, các kết quả đạt được, tìm hiểu các đặc điểm của mô hình cũng như chương trình cài đặt, các hạn chế và các hướng giải quyết trong tương lai.
- Phần phụ lục trình bày các thông tin liên quan.

Chương 2

Cơ sở lý thuyết ngôn ngữ

Mục lục

2.1 Âm tiết

Ngôn ngữ là một hệ thống tín hiệu. Khi nói, vỏ vật chất của tín hiệu là âm thanh, khi viết nó được thể hiện bằng chữ. Không phải chữ viết lúc nào cũng phản ánh chính xác các âm tố tương ứng. Vì vậy, các âm tố được biểu diễn bằng những ký hiệu đặc biệt, gọi là phiên âm. Các ký hiệu phiên âm thường đặt giữa / / hoặc [ ].

Âm thanh trong tự nhiên được tạo thành nhờ sự rung động của một vật thể đàn hồi. Âm thanh của tiếng nói được hình thành nhờ "bộ máy phát âm"

của con người — bao gồm môi, răng, lưỡi, khoang miệng, khoang mũi, yết hầu, thanh hầu, phổi .... Ngoài ra, tai người chỉ có thể tiếp nhận một khoảng âm thanh nhất định. Những chấn động không nghe được gọi là siêu âm và âm ngoại.

Âm học phân biệt các âm thanh theo những đặc trưng khác nhau, bao gồm: độ cao, độ mạnh, độ dài. Độ cao phụ thuộc vào tần số dao động. Tần số dao động càng lớn thì âm thanh càng cao. Tai người có khả năng nhận biết độ cao trong khoảng từ 16 đến 20.000 H_z. Độ mạnh (cường độ) phụ thuộc vào biên độ dao động. Biên độ càng lớn, âm thanh càng to. Cường độ âm thanh trong ngôn ngữ đảm bảo sự xác minh trong giao tế và là cơ sở để tạo thành các kiểu trọng âm khác nhau. Độ dài (trường độ) là khoảng thời gian kéo dài của âm thanh. Ngôn ngữ chỉ quan trọng thời gian tương đối của âm thanh. Ví dụ, các nguyên âm có trọng âm thường dài hơn nguyên âm không có trọng âm.

2.1.1 Nguyên âm và phụ âm

Các âm tố có thể chia thành nguyên âm và phụ âm, dựa vào các đặc điểm âm học, cấu âm và vai trò trong cấu tạo âm tiết.

Nguyên âm có đặc điểm cấu tạo:

- Luồng hơi ra tự do, không bị cản trở, không có vị trí cấu âm.
- Bộ máy phát âm căng thẳng toàn bộ.
- Luồng hơi ra yếu.

Phụ âm có đặc điểm cấu tạo hoàn toàn trái ngược với nguyên âm:

- Luồng hơi bị cản trở do sự xuất hiện chướng ngại trên lối ra của luồng không khí, chướng ngại thường xuất hiện ở các khoang trên thanh hầu

do các khí quan tiếp xúc nhau hay nhích gần nhau mà thành, điểm có chướng ngại được gọi là vị trí cấu âm của phụ âm.

- Bộ máy phát âm không căng thẳng toàn bộ mà sự căng thẳng cơ thịt tập trung ở vị trí cấu âm.
- Luồng hơi ra mạnh.

Nguyên âm và phụ âm có chức năng khác nhau trong cấu tạo âm tiết. Các nguyên âm thường làm hạt nhân hay đỉnh của âm tiết, còn phụ âm thường là yếu tố đi kèm, không tạo thành âm tiết (trừ các âm phụ vang).

Những âm tố có đặc tính giống nguyên âm nhưng thường chỉ đi kèm, bản thân không tạo thành âm tiết được gọi là *bán nguyên âm*. Ví dụ, các âm tố viết là u, i trong các âm "sau", "mai" trong tiếng Việt.

2.1.2 Âm vị

Âm vị là đơn vị nhỏ nhất của cơ cấu âm thanh ngôn ngữ, dùng để cấu tạo và phân biệt hình thức ngữ âm của những đơn vị có nghĩa của ngôn ngữ — từ và hình vị. Ví dụ, các từ "tôi" và "đôi", "ta" và "đa" được phân biệt bởi các âm vị /t/ và /d/.

Âm vị là đơn vị nhỏ nhất, vì về mặt tuyến tính nó không thể phân chia nhỏ hơn nữa. Nếu thay âm vị này bằng âm vị khác trong cùng một bối cảnh ngữ âm sẽ làm cho từ thay đổi nghĩa hoặc mất nghĩa. Ví dụ, thay âm /t/ trong từ "toàn" bằng âm /h/ sẽ được "hoàn" có nghĩa khác, hoặc nếu thay bằng âm /n/ sẽ được "noàn" hoàn toàn vô nghĩa.

Âm vị có thể được so sánh như những viên gạch trong việc xây dựng mỗi ngôn ngữ. Các viên gạch thường giống nhau, nhưng các âm vị về nguyên tắc phải khác nhau, ít nhất ở một đặc trưng nào đó. Sự khác biệt này tạo ra khác biệt về hình thức âm thanh của hình vị và từ, tạo ra tín hiệu khác biệt đối với

sự thụ cảm của con người. Vậy âm vị có hai chức năng cơ bản là *chức năng khu biệt* (vỏ âm thanh của hình vị và từ) và *chức năng cấu tạo* (chất liệu để cấu tạo nên những thành tố của những đơn vị có nghĩa).

2.1.3 Âm tiết

Chuỗi lời nói của con người được chia ra làm những khúc đoạn khác nhau, từ lớn đến nhỏ. Âm tiết là đơn vị phát âm nhỏ nhất, được phân định tự nhiên trong lời nói con người.

Về phương diện phát âm, dù lời nói chậm đến đâu cũng chỉ phân chia đến giới hạn của âm tiết mà thôi. Nhưng về phương diện thính giác thì âm tiết là một tổ hợp âm thanh, có thể gồm nhiều âm tố hoặc đôi khi chỉ có một âm tố. Mỗi âm tiết chỉ có một âm tố âm tiết tính (có khả năng tạo thành âm tiết), còn lại là những yếu tố đi kèm, không tự mình tạo thành âm tiết. Âm tố âm tiết tính thường được phân bố ở đỉnh hay ở trung tâm, làm hạt nhân âm tiết, thường là các nguyên âm. Các phụ âm thường là các yếu tố đi kèm, đứng ngoài biên, hay ở ranh giới của âm tiết. Đôi khi âm tiết chỉ gồm một nguyên âm.

Trong một số trường hợp, âm tiết có thể có hai hoặc ba nguyên âm. Tuy nhiên trong số đó chỉ có một nguyên âm tạo đỉnh, các âm tố khác không tạo thành âm tiết, gọi là bán nguyên âm.

Âm tiết có một số chức năng sau:

- Âm tiết có chức năng tổ chức chất liệu âm thanh của ngôn ngữ bằng cách hợp nhất các âm tố trong một đơn vị phát âm nhỏ nhất.
- Âm tiết là môi trường để hiện thực hoá các hiện tượng ngôn điệu như trọng âm, âm điệu.

- Âm tiết có chức năng cấu thành tiết điệu của lời nói ... Chức năng này thể hiện rõ trong ngôn ngữ thơ.

Trong các ngôn ngữ âm tiết tính như tiếng Trung Quốc, tiếng Miến Điện, tiếng Việt ... nói chung âm tiết trùng với hình vị — đơn vị cơ bản của ngữ pháp. Âm tiết có chức năng là vỏ ngữ âm của hình vị, tạo nên một đơn vị đặc biệt, gọi là *hình tiết*.

Tính chất âm tiết của tiếng Việt đưa đến nhiều hệ quả quan trọng về ngữ âm cũng như về ngữ pháp. Về mặt ngữ âm, do mỗi âm tiết là vỏ ngữ âm của một hình vị, và cũng thường là vỏ ngữ âm của từ đơn, nên số lượng các âm tiết là hữu hạn[1].

Là vỏ ngữ âm của một hình vị hay một từ đơn, mỗi âm tiết Tiếng Việt bao giờ cũng tương ứng với một ý nghĩa nhất định, nên việc phá vỡ cấu trúc âm tiết trong ngữ lưu, tức xê dịch vị trí các âm tố (âm vị) của cùng một hình vị từ âm tiết này sang âm tiết khác, là điều ít xảy ra. Kết quả là trong tiếng Việt, âm tiết có một cấu trúc chặt chẽ, mỗi âm tố (âm vị) có một vị trí nhất định trong âm tiết. Đứng đầu âm tiết bao giờ cũng là một phụ âm, cuối âm tiết là một phụ âm hoặc một bán nguyên âm. Phụ âm cuối luôn luôn ở cuối âm tiết, không thể trở thành âm đầu được. Do đó, phụ âm cuối và âm đầu làm thành hai đối hệ khác nhau, có vị trí và chức năng khác nhau trong cấu trúc âm tiết.

Một đặc điểm khác của âm tiết tiếng Việt là mỗi âm tiết đều mang một thanh điệu nhất định. Việc thể hiện thanh điệu đòi hỏi âm tiết phải có một trường độ cố định. Tính chất này làm cho các yếu tố bên trong âm tiết, trừ phụ âm đầu, không có một trường độ cố định, mà đắp đổi lẫn nhau, liên quan với nhau rất chặt chẽ.

[1]Theo Nguyễn Phan Cảnh "tiếng Việt đưa ra hơn 17.000 âm tiết — tín hiệu với tự cách là vỏ ngữ âm khả năng, và chỉ sử dụng hơn 6.900 với tư cách là các âm tiết tồn tại thực" (Nguyễn Phan Cảnh, "Bản chất cấu trúc âm tiết tính của ngôn ngữ: Dẫn luận vào một miêu tả không phân lập đối với âm vị học Việt Nam, tạp chí ngôn ngữ, H. 1978, số 2)

Cấu trúc âm tiết tiếng Việt

Trên bình diện ngữ âm học, các cứ liệu thực nghiệm cho thấy âm tiết Tiếng Việt được cấu tạo bởi ba thành tố độc lập là thanh điệu, phụ âm đầu và phần còn lại.

Thanh điệu là yếu tố luôn có mặt trong mọi âm tiết tiếng Việt. Tính chất độc lập về mặt ngữ âm của thanh điệu thể hiện ở chỗ nó có đường nét và trường độ tương đối ổn định tùy thuộc vào các loại hình âm tiết.

Phụ âm đầu là yếu tố mở đầu của âm tiết. Tính chất độc lập của phụ âm đầu thể hiện ở chỗ nó không tham gia vào việc đắp đổi về trường độ giữa các yếu tố bên trong âm tiết.

Phần còn lại của âm tiết có từ một đến ba yếu tố, gồm một bán nguyên âm chiếm vị trí trung gian giữa phụ âm đầu và phần còn lại, một nguyên âm âm tiết tính và một phụ âm hoặc bán nguyên âm cuối, có vai trò kết thúc âm tiết. Trừ bán nguyên âm trước nguyên âm tiết tính, các yếu tố của phần còn lại liên kết với nhau rất chặt chẽ, làm thành một khối. Để đảm bảo cho tính chất cố định về trường độ của âm tiết, các yếu tố của phần còn lại có sự đắp đổi nhau về trường độ: nếu nguyên âm dài thì phụ âm hay bán âm cuối ngắn, ngược lại nếu nguyên âm ngắn thì âm cuối dài. Các yếu tố của phần còn lại không có một trường độ cố định, và do đó mức độ độc lập về mặt ngữ âm của chúng thấp hơn so với phụ âm mở đầu âm tiết. Phần còn lại của âm tiết được gọi là phần vần, vì đây là bộ phận đoạn tính kết hợp với thanh điệu tạo nên vần thơ.

Tóm lại, các yếu tố của âm tiết tiếng Việt có mức độ độc lập khác nhau, chia làm hai bậc:

- Bậc một là những yếu tố độc lập về mặt ngữ âm và có thể được tách rời về mặt hình thái học. Đó là thanh điệu, âm đầu và vần.
- Bậc hai là các yếu tố của phần vần, gồm bán nguyên âm trước nguyên

âm âm tiết tính (được gọi là âm đệm), nguyên âm âm tiết tính (được gọi là âm chính), phụ âm hoặc bán nguyên âm cuối (được gọi là âm cuối). Các yếu tố này gắn liền với nhau về mặt ngữ âm do tính chất cố định về trường độ của âm tiết và chỉ được tách ra bằng những ranh giới thuần túy ngữ âm học.

Các thành tố của âm tiết tiếng Việt và quan hệ hai bậc giữa các thành tố được trình bày trong hình 2.1.

<table>
<tr><td colspan="4">Thanh điệu</td></tr>
<tr><td rowspan="2">Âm đầu</td><td colspan="3">Vần</td></tr>
<tr><td>Âm đệm</td><td>Âm chính</td><td>Âm cuối</td></tr>
</table>

Hình 2.1: Cấu trúc âm tiết

Khái niệm âm tiết liên quan mật thiết đến sự biến hoá ngữ âm. Vì các âm tố lời nói không phát âm đơn lập mà được phát âm trong dòng lời nói liên tục, cho nên các âm tố có thể ảnh hưởng lẫn nhau, đặc biệt là những âm tố lân cận được phát âm trong cùng một âm tiết, hoặc ở những âm tiết đi liền nhau. Một số hiện tượng biến hoá ngữ âm thường gặp trong tiếng Việt:

- Sự thích nghi. Xuất hiện giữa phụ âm và nguyên âm đứng cạnh nhau. Nếu âm tố sau biến đổi cho giống âm tố đi trước, đó là thích nghi xuôi. Nếu âm tố trước biến đổi cho hợp với âm tố sau là thích nghi ngược. Trong tiếng Việt, nguyên âm và phụ âm cuối kết hợp với nhau rất chặt chẽ, tạo thành vần của âm tiết. Hiện tượng thích nghi biểu hiện rõ rệt trong những vần có nguyên âm dòng trước và dòng sau tròn môi kết hợp với phụ âm cuối "ng" và "c".

- Sự đồng hoá (một yếu tố thay đổi để giống yếu tố kia). Ví dụ, "vỏn vẹn" và "vẻn vẹn".

- Sự dị hoá (hiện tượng rút gọn cho dễ phát âm). Ví dụ, "ba mươi mốt" và "băm mốt".

2.1.4 Phụ âm đầu

Phụ âm đầu luôn gắn liền với vị trí và chức năng mở đầu âm tiết. Đi sau âm đầu trong âm tiết là bán nguyên âm không thành âm tiết (hay còn gọi là âm đệm).

Hệ thống phụ âm đầu tiếng Việt với số lưỡng đối lập âm vị học tối đa được thể hiện trên chữ viết. Riêng những âm tiết như "ăn", "uống" ... tuy không ghi phụ âm đầu, nhưng thực tế vẫn tồn tại phụ âm đầu (âm tắt thanh hầu /ʔ/). Trong từng phương ngữ, một số đối lập có trên chữ viết có thể bị mất đi hoặc bị thay thế. Ví dụ, trong tiếng Hà Nội không còn đối lập các phụ âm đầu giữa ch–tr,x–s và gi,d với r. Trong tiếng miền Nam, /v/ và /z/ được thay bằng /j/.

Hiện nay, hệ thống phụ âm đầu được sử dụng thực tế trong nhà trường và trên các văn bản, chung cho các phương ngữ, là hệ thống phụ âm đầu hình thành trên cơ sở phát âm Hà Nội với sự phân biệt các phụ âm ch–tr, x–s, g,gi–r gồm 22 phụ âm sau: /b, m, f, v, t, tʰ, d, n, s, z, l, ʈ, ʂ, ʐ, c, ɲ, k, ŋ, x, ɣ, ʔ, h/[2]

Hệ thống phụ âm đầu của tiếng địa phương miền Bắc, mà cở sở là phát âm Hà Nội có 19 phụ âm (kể cả âm tắc thanh hầu /ʔ/). Trong phát âm Hà Nội không có loạt phụ âm uốn lưỡi /ʈ, ʂ, ʐ/. Các phụ âm này đều được chuyển thành các âm đầu lưỡi hoặc mặt lưỡi tương ứng /c, s, z/. Ví dụ,

- "cha" và "tra" đều phát âm thành "cha" /ca/
- "sa" và "xa" đều phát âm thành "xa" /sa/

[2]Phụ âm /p/ gặp trong từ vay mượn hoặc phiên âm tiếng nước ngoài, không được đưa vào hệ thống này

- "da", "gia" và "ra" đều được phát âm thành "da" /da/

Trong các thổ ngữ vùng Bắc Trung Bộ (Nghệ Tĩnh — Bình Trị Thiên) còn giữ loạt các phụ âm cong lưỡi /ʈ, ʂ, ʐ/. Ở một số nơi thuộc Nghệ Tĩnh, phụ âm "ph" được phát âm như âm mặt lưỡi sau bật hơi /kʰ/. Vì vậy hệ thống phụ âm đầu những nơi này có thêm dãy âm bật hơi /pʲ, ʈʰ, kʰ/. Trong khi đó các thổ ngữ miền Bắc và miền Nam chỉ còn lại một âm bật hơi /tʰ/ mà thôi. Vùng Bình Trị Thiên không có phụ âm "nh". Phụ âm này thường được phát âm thành /j/. Ví dụ, "nhà" được phát âm thành "dà". Nếu coi hệ thống phụ âm đầu vùng Vinh là đại diện cho phương ngự Bắc Trung Bộ thì hệ thống này có 22 phụ âm đầu.

Hệ thống phụ âm đầu miền Nam (từ đèo Hải Vân trở vào) không có các phụ âm xát hữu thanh /v, z/. Tương ứng với /v, z/ trong phát âm Hà Nội, phát âm miền Nam có phụ âm mặt lưỡi giữa /j/. Đôi khi âm /v/ được phát âm thành âm môi-môi, xát, vang ngạc hoá /βj/. Hiện nay các âm cong lưỡi đang trong quá trình biến đổi trong tiếng miền Nam. Phụ âm /ʂ/ là phụ âm ít bền vững nhất thường được phát âm thành /s/. Các phụ âm cong lưỡi khác như /ʈ/ và /ʐ/ vẫn còn giữ lại, phân biệt với /c/ và /j/ nhưng không đều đặn ở các thổ ngữ. Trong phát âm miền Nam có phụ âm đầu /w/[3] xát, môi-môi, tương ứng với các phụ âm tắc, lưỡi sau và thanh hầu tiếng Bắc khi kết hợp với âm đệm /-u-/. Ví dụ, "qua" /wa/, "ngoại" /wai/, hoa /wa/. Nếu lấy hệ thống phụ âm đầu của tiếng thành phố Hồ Chí Minh làm cơ sở cho phương ngữ miền Nam thì hệ thống này có 21 phụ âm đầu.

Quan hệ phân bố giữa phụ âm đầu và âm đệm

Âm đệm là thành tố đi sau phụ âm đầu trong âm tiết. Trong tiếng Việt chỉ có một âm đệm là /-u-/, thể hiện trên chữ viết bằng hai chữ "u" và "o". Ví dụ,

[3]Giá trị âm vị học của /w/ là vấn đề còn đang bàn cãi

"hoa", "quế". Trong phát âm, âm đệm chỉ được thể hiện ở tiếng địa phương miền Bắc và Bắc Trung Bộ, còn trong tiếng địa phương miền Nam thường không có âm đệm /-u-/.

Trong phát âm Hà Nội, hầu hết loạt phụ âm lưỡi và thanh hầu có thể phân bố trước âm đệm. Ví dụ, "toa", "đoán", "nhoà" ... Riêng loạt âm môi /b, m, v, f/ không phân bố trước âm đệm /-u-/ vì chúng có cấu âm môi giống nhau. Trong tiếng Việt, hễ những âm có cấu âm giống nhau hay tương tự nhau thì không phân bố cạnh nhau.

Ngoài các âm môi, một vài phụ âm lưỡi như /n, z, ɣ/ cũng rất ít xuất hiện trước âm đệm.

2.1.5 Vần

Âm đệm

Trong âm tiết, âm đệm /-u-/ đứng sau phụ âm đầu và đứng trước âm chính. Nó đóng vai trò một âm lướt trong kết cấu âm tiết. Về mặt cấu âm, âm đệm /-u-/ được phát âm giống như nguyên âm [u] nhưng không làm đỉnh âm tiết. Đó là một bán nguyên âm môi-ngạc mềm, được phiên âm là [-u-] hay [-w-]. Động tác cấu âm này diễn ra đồng thời với các giai đoạn phát âm của phụ âm đầu và phần vần đầu của nguyên âm làm âm chính. Về mặt âm học, âm đệm /-u-/ có tác dụng làm biến đổi âm sắc của âm tiết, làm trầm hoá âm sắc của âm tiết.

Âm đệm /-u-/, với tính chất là một bán nguyên âm môi-ngạc mềm, có độ mở rộng hay hẹp tương ứng với độ mở của nguyên âm đi sau nó. Trước nguyên âm hẹp **i**, âm đệm /-u-/ được thể hiện bằng một bán âm hẹp tương ứng là [u], ví dụ "tuy". Trước các nguyên âm có độ mở trung bình **ê, ơ, â**, âm đệm /-u-/ được thể hiện bằng một bán âm độ mở vừa [o], ví dụ "khuê", "huơ", "huân". Trước các nguyên âm có độ mở rộng **e, a, ă**, âm đệm /-u-/

được thể hiện bằng một bán âm có độ mở tương ứng là [ɔ], ví dụ "khỏe", "khoắn", "khoan".

Âm đệm /-u-/ xuất hiện phần lớn ở các từ gốc Hán như "thuyền", "loan", "uyên". Về mặt phân bố, như đã nói, âm đệm có thể xuất hiện sau hầu hết các phụ âm đầu, trừ các phụ âm môi /b, m, f, v/. Sau các phụ âm môi, âm đệm chỉ có mặt trong một ít từ phiên âm tiếng nước ngoài như "buýt", "phuy", "voan". Ngoài ra, sau các phụ âm /n, z, ɣ/, âm đệm /-u-/ cũng chỉ xuất hiện trong một vài từ như "noãn", "roa", "goá".

Âm đệm /-u-/ cũng không xuất hiện trước các nguyên âm tròn môi **u, uô, ô, o**. Sự phân bố của âm đệm sau phụ âm đầu và trước các nguyên âm thể hiện một quy luật của ngữ âm tiếng Việt: các âm có cấu âm giống nhau hoặc gần gũi nhau không được phân bố cạnh nhau.

Về mặt chữ viết, âm đệm /-u-/ được ghi bằng con chữ "o" trước ba nguyên âm rộng **e, a, ă** và được ghi bằng con chữ "u" trước các nguyên âm còn lại. Ví dụ, "thuý", "thuê", "loe", "loa". Riêng trường hợp sau phụ âm đầu /k-/, âm đệm /-u-/ luôn được ghi bằng con chữ "u" dù sau nó là nguyên âm rộng. Ví dụ: "quạ", "quý" (trong những trường hợp này âm /k-/ được ghi bằng con chữ "q")[4].

Âm đệm /-u-/, vốn là yếu tố có mặt trong phương ngữ Bắc và Bắc Trung Bộ, lại hoàn toàn vắng mặt trong phương ngữ Nam Bộ. Do đó, cấu trúc âm tiết của phương ngữ Nam Bộ chỉ có ba thành phần đoạn tính: âm đầu, âm chính, âm cuối.

Sự vắng mặt của âm đệm trong phương ngữ Nam Bộ có thể đưa đến một số biến đổi ở âm đầu và âm chính. Đáng chú ý là sự biến đổi của các phụ âm mặt lưỡi sau và thanh hầu, thành các phụ âm môi. Ví dụ, "hoa" thành "wa",

[4]Do đó về mặt chữ viết, sau con chữ "q", con chữ "u" luôn luôn có giá trị là một âm đệm. Điều này giúp ta phân biệt "ua" là một nguyên âm đôi trong từ "của" với "ua" trong tổ hợp âm đệm+nguyên âm trong "quả". Riêng trường hợp "quốc" thì "uô" là nguyên âm đôi nhưng /k-/ vẫn được ghi bằng "q". Sự phân biệt về mặt con chữ ở đây có giá trị phân biệt nghĩa hai từ đồng âm "cuốc" và "quốc" đều được phát âm là /kuok/.

khuya thành "phia".

Hiện nay dưới sự ảnh hưởng của ngôn ngữ văn học, đã thấy xuất hiện âm đệm sau các phụ âm đầu lưỡi, mặt lưỡi giữa và mặt lưỡi sau trong cách phát âm của tầng lớp trí thức, của giới trẻ, trừ trường hợp hai phụ âm thanh hầu /h-,ʔ-/ và phụ âm mặt lưỡi sau /k-/, vẫn được phát âm thành [w-] trong các từ "hoa", "oa", "qua" (đều phát âm là [wa]).

Âm chính

Âm chính trong âm tiết tiếng Việt có thể là một nguyên âm đơn hoặc một nguyên âm đôi.

Nguyên âm đơn Tiếng Việt có 11 nguyên âm đơn làm âm chính. Căn cứ vào vị trí lưỡi, hình dáng môi, các nguyên âm đơn được chia ra:

- Các nguyên âm giòng trước không tròn môi: /i, e, ɛ/.
- Các nguyên âm giòng sau không tròn môi: /ɯ, ɤ, ɤ̆, a, ă/.
- Các nguyên âm giòng sau tròn môi: /u, o, ɔ/.

Căn cứ vào độ mở miệng, có thể chia thành:

- Các nguyên âm có độ mở miệng hẹp: /i, ɯ, u/.
- Các nguyên âm có độ mở trung bình: /e, ɤ, ɤ̆, o/.
- Các nguyên âm có độ mở rộng: /ɛ, a, ă, ɔ/.

Căn cứ vào âm sắc, có thể chia ra:

- Các nguyên âm bổng: /i, e, ɛ/.

- Các nguyên âm trung bình: /ɯ, ɤ, ɤ̆, a, ă/.
- Các nguyên âm trầm: /u, o, ɔ/.

Căn cứ vào trường độ, có thể chia ra:

- Các nguyên âm dài: /i, e, ɛ, ɯ, ɤ, a, u, o, ɔ/.
- Các nguyên âm ngắn: /ɤ̆, ă/.

Nguyên âm đôi Ngoài 11 nguyên âm đơn, còn có 3 nguyên âm đôi âm vị tính là /ie, ɯɤ, uo/.

Âm cuối

Âm cuối là yếu tố kết thúc âm tiết. Các âm tiết trong tiếng Việt có thể kết thúc bằng cách biến đổi âm sắc của âm chính do động tác khép lại của bộ máy phát âm, làm cho nó bổng hơn hoặc trầm hơn. Âm cuối trong trường hợp này là hai bán nguyên âm /-u/ và /-i/. Âm tiết tiếng Việt còn có thể kết thúc bằng động tác khép của bộ máy phát âm với một phụ âm tắc (mũi hoặc miệng).

Hệ thống âm cuối trong tiếng Việt gồm có 2 bán nguyên âm và 6 phụ âm. Sau phụ âm bao gồm: /m, p, n, t, ŋ, k/.

Quy luật phân bố của các âm cuối sau âm chính

Về mặt phân bố, các bán nguyên âm cuối /-u/ và /-i/ chỉ xuất hiện sau các nguyên âm không cùng âm sắc với nó. Bán nguyên âm cuối /-i/ chỉ xuất hiện sau các bán nguyên âm không phải giòng trước. Bán nguyên âm cuối /-u/ chỉ xuất hiện sau các bán nguyên âm không tròn môi. Sự kết hợp giữa nguyên âm và bán nguyên âm cuối, giống như sự kết hợp giữa âm đệm và

nguyên âm làm âm chính, tuân theo quy luật dị hoá. Theo đó, các âm có cấu âm giống nhau hoặc gần nhau không bao giờ được phân bố cạnh nhau.

Có thể hình dung khả năng kết hợp giữa nguyên âm làm âm chính với hai bán nguyên âm cuối /-i/ và /-u/ như sau:

- Các nguyên âm có thể đứng trước bán nguyên âm /-i/ bao gồm các âm biểu hiện bởi các chữ: ư, ươ, ơ, â, a, ă, u, uô, ô, o.
- Các nguyên âm có thể đứng trước bán nguyên âm /-u/ bao gồm các âm biểu hiện bởi các chữ: i, iê, ê, e, ư, ươ, ơ, â, a, ă.

Các phụ âm cuối khác, nói chung được phân bố đều đặn sau các nguyên âm, trừ hai âm cuối mũi /-m, -p/ không xuất hiện sau /ɯ/.

Sự thể hiện của nguyên âm và phụ âm trong các tiếng địa phương

Trong phương ngữ Nam Bộ, các nguyên âm đôi /ie, ɯɤ, uo/ khi kết hợp với các âm cuối /-i, -u, -m, -p/ được thể hiện thành các nguyên âm đơn /i, ɯ, u/. Ví dụ, "chuối" — "chúi", "bưởi" — "bửi", "tiếp" — "típ".

Ở một vài địa phương thuộc phương ngữ Trung Bộ, các nguyên âm đôi được thể hiện bằng các nguyên âm cùng dòng, độ mở rộng. Ví dụ, "người" — "ngài", "ruột" — "rọt", "miếng" — "méng".

Hai phụ âm cuối /-n, -t/ được thể hiện thành /-ŋ, -k/ trong phương ngữ Nam Bộ, khi chúng đi sau các nguyên âm đơn và đôi, trừ /i, e/ là hai nguyên âm giòng trước, độ mở hẹp và trung bình. Ví dụ, "đen" – "đeng", "đét" — "đéc".

Sau ba nguyên âm giòng trước /i, e, ɛ/, hai phụ âm /-ŋ, -k/ được thể hiện trong các phương ngữ Nam Bộ thành /-n, -t/, đồng thời các nguyên âm này có cấu âm lui về phía sau nhiều hơn so với các nguyên âm trong phương

ngữ Bắc Bộ, trở thành các nguyên âm giòng giữa nghe gần giống như ư, ơ (hoặc â) và ă.

Điểm đáng lưu ý là trong phương ngữ Nam Bộ, sau /i, e/ hai âm cuối /-n, -t/ vẫn được phát âm không đổi. Sự khác biệt trong các vần này giữa phương ngữ Bắc Bộ và Nam Bộ xảy ra ở nguyên âm.

Trong phương ngữ Nam Bộ không có các âm cuối /-ɲ, -c/. Âm cuối này được phát âm thành /-n, -t/.

2.1.6 Thanh điệu

Thanh điệu là đặc trưng ngôn điệu của âm tiết. Người ta gọi thanh điệu là âm vị siêu đoạn tính. Số lượng thanh điệu trong tiếng Việt khác nhau giữa các tiếng địa phương. Số lượng nhiều nhất là 6 thanh trong phát âm Hà Nội, hay trong các tiếng Bắc nói chung, và được phản ánh trên chữ viết. Đó là các thanh: sắc, huyền, ngã, hỏi, nặng, và thanh không dấu.

Trong các tiếng địa phương từ Thanh Hoá trở vào Nam thường chỉ có năm thanh, thanh ngã trùng với thanh hỏi (trong một số vùng Thanh Hoá, tiếng Bình Trị Thiên, Nam Trung Bộ và Nam Bộ), hoặc thanh ngã trùng với thanh nặng (trong tiếng vùng Nghệ An, Hà Tĩnh). Ngoài ra trong một vài thổ ngữ lẻ tẻ ở Nghệ An và Quảng Bình chỉ có 4 thanh điệu.

Sự phân bố của thanh điệu

Như đã biết, thanh điệu là đặc tính siêu đoạn của âm tiết. Các đặc trưng của thanh điệu được thể hiện đồng thời với các thành phần cấu trúc khác của âm tiết. Vì vậy, trong chừng mực nào đó nó bị chế định bởi các thành phần này.

Về mặt âm vị học, âm tiết tiếng Việt trước hết được chia thành hai đơn vị là phụ âm đầu và vần. Phần vần, trong đó có nguyên âm, là phân luôn luôn mang thanh tính của âm tiết. Các đặc điểm về âm vực và âm điệu của thanh

điệu chỉ được biểu hiện trong phần mang thanh tính mà thôi. Vì vậy, trong sự đối lập và thống nhất các thanh điệu, phần vần đóng vai trò quan trọng. Phụ âm đầu hầu như không đóng vai trò gì trong sự đối lập các thanh. Về mặt ngữ âm, đặc tính của thanh điệu cũng hầu như không lan truyền lên phụ âm đầu, hoặc có chăng (trong trường hợp phụ âm đầu hữu thanh) thì trong đoạn đầu của âm tiết, các đặc trưng khu biệt của thanh điệu cũng chưa thể hiện rõ.

Phần vần có thể bao gồm âm đệm, một âm chính và có thể có bán nguyên âm hoặc phụ âm cuối. Sự khác nhau của thanh điệu biểu hiện tập trung ở giữa và cuối vần (tức phần nguyên âm và phụ âm cuối).

Trong các vần không có âm cuối, có âm cuối là bán nguyên âm hoặc phụ âm vang, các đặc trưng của thanh điệu được thể hiện dễ dàng. Với các vần kết thúc bằng các phụ âm cuối vô thanh, khép, các đặc trưng của thanh được biểu hiện rất hạn chế. Có thể nói rằng, trong mối quan hệ với các thành phần chiết đoạn của âm tiết, thanh điệu bị sự chế định rõ ràng nhất của âm cuối. Vì vậy sự phân bố của thanh điệu trong âm tiết phụ thuộc vào loại hình kết thúc âm tiết.

Số lượng các thanh điệu xuất hiện trong những âm tiết kết thúc bằng phụ âm cuối vô thanh rất hạn chế, thường chỉ có thể có thanh sắc hoặc thanh nặng.

Thanh sắc và thanh nặng trong những âm tiết có âm cuối vô thanh có những đặc điểm riêng về độ dài và đường nét âm điệu khác với thanh sắc và thanh nặng trong các âm tiết còn lại. Vì vậy trước đây đã từng có quan niệm cho rằng các thanh điệu trong các âm tiết có âm cuối vô thanh là những thanh điệu đặc biệt, tạo thành hệ thống 8 thanh điệu: tan, tàn, tãn, tản, tán, tạn, tát, tạt.

2.2 Từ

Khái niệm từ, mặc dù nghe qua rất thông dụng, dễ hiểu, nhưng định nghĩa chính xác thế nào là từ không đơn giản. Từ trước đến nay đã có nhiều định nghĩa về từ được đưa ra. Các định nghĩa đều đúng, tuy nhưng không hoàn chỉnh. Viện sĩ L. V. Sherba thừa nhận rằng: "Trong thực tế, từ là gì? Thiết nghĩ rằng trong các ngôn ngữ khác nhau, từ sẽ khác nhau. Do đó, tất sẽ không có khái niệm từ nói chung"[5]. Chính vì tính đa dạng và phức tạp của từ mà một số nhà ngôn ngữ học chối bỏ khái niệm từ, hoặc né tránh định nghĩa từ một cách chính thức. Nhà ngôn ngữ học Ferdinand de Saussure đã nhận xét: "... Ngôn ngữ có tính chất kỳ lạ và đáng kinh ngạc là không có những thực thể thoạt nhìn có thể thấy ngay được, thế nhưng người ta vẫn biết chắc là nó tồn tại, và chính sự giao lưu giữa những thực thể đó đã làm thành ngôn ngữ. Trong số những thực thể đó có cái mà ngôn ngữ học vẫn gọi là từ.". Theo ông thì "... Từ là một đơn vị luôn luôn ám ảnh toàn bộ tư tưởng chúng ta như một cái gì đó trọng tâm trong toàn bộ cơ cấu ngôn ngữ, mặc dù khái niệm này khó định nghĩa".

2.2.1 Định nghĩa từ

Thời Hy Lạp cổ đại, trường phái ngôn ngữ Alexandri đã định nghĩa: "*Từ là đơn vị nhỏ nhất trong chuỗi lời nói*". Ngoài ra A. Meillet trong *Ngôn ngữ học lịch sử và ngôn ngữ học đại cương* đã định nghĩa: "*Từ là kết quả của sự kết hợp một ý nghĩa nhất định với một tổ hợp các âm tố nhất định, có thể có một công dụng ngữ pháp nhất định*".

Theo E. Sapir thì "*Từ là một đoạn nhỏ nhất có ý nghĩa, hoàn toàn có khả năng độc lập và bản thân có thể làm thành câu tối giản*".

[5]Nguyễn Kim Thản, *Nghiên cứu ngữ pháp tiếng Việt*. NXB GD, 1997. Trang 28

Theo L. Bloomfield thì từ là "*một hình thái tự do nhất*".

Theo B. Golovin thì từ là "*đơn vị nhỏ nhất có ý nghĩa của ngôn ngữ, được vận dụng độc lập, tái hiện tự do trong lời nói để xây dựng nên câu*".

Theo Solncev thì "*Từ là đơn vị ngôn ngữ có tính hai mặt: âm và nghĩa. Từ có khả năng độc lập về cú pháp khi sử dụng trong lời*".

Theo B. Trơ-nơ-ka thì "*Từ là đơn vị nhỏ nhất có ý nghĩa, được cấu tạo bằng âm vị và có khả năng thay đổi vị trí và thay thế lẫn nhau trong câu*".

Theo Lục Chí Vỹ thì "*Từ là đơn vị nhỏ nhất có thể vận dụng tự do trong câu*". Theo một số tác giả khác của Trung Quốc thì "*Từ là đơn vị từ vựng, là đơn vị vật liệu kiến trúc của ngôn ngữ, và cũng là đơn vị nhỏ nhất có khả năng vận dụng tư do trong lời nói*".

Theo V. G. Admoni thì "*Từ là đơn vị ngữ pháp, do hình vị cấu tạo nên, dùng để biểu thị đối tượng, quá trình, tính chất và những mối quan hệ trong hiện thực, có tính đặc thù rõ rệt và có khả năng kiến lập nhiều mối quan hệ đa dạng với nhau*".

Theo R. A. Bunđagôp thì "*Từ là đơn vị nhỏ nhất và độc lập, có hình thức vật chất (vỏ âm thanh và hình thức) và có nghĩa, có tính chất biện chứng và lịch sử*".

Đối với tiếng Việt, cũng có một số định nghĩa từ được đưa ra. Theo M. B. Émeneau thì "*Từ bao giờ cũng tự do về mặt âm vị học, nghĩa là có thể miêu tả bằng những danh từ của sự phân phối các âm vị và bằng những thanh điệu*"[6]. Émeneau đã dựa trên mặt ngữ âm để định nghĩa từ, xem mỗi từ trước hết là những âm tiết. Với quan niệm như vậy chủ yếu dựa vào tính hoàn chỉnh về mặt âm thanh và trong thực tế thì người Việt luôn có khuynh hướng mong đợi mỗi tiếng như vậy sẽ mang một nghĩa nào đó và coi đó như "từ".

Theo Trương Văn Trình và Nguyễn Hiến Lê thì "*Từ là âm có nghĩa, dùng*

[6]Nguyễn Thiện Giáp. *Từ và nhận diện từ tiếng Việt*. NXB GD, Hà Nội 1996. Trang 17

trong ngôn ngữ để diễn tả một ý đơn giản nhất, nghĩa là ý không thể phân tích ra được". Định nghĩa này chủ yếu dựa vào tính nhất thể của nghĩa, nghĩa là mỗi từ có một nghĩa tối giản nào đó, và nghĩa của từ có tính võ đoán và tính thành ngữ.

Lê Văn Lý cho rằng từ tiếng Việt "*là một tín hiệu ngữ âm có thể cấu tạo bằng một âm vị hay sự kết hợp với âm vị, mà sự phát âm chỉ tiến hành trong một lần, hoặc là một âm tiết mà chữ viết biểu thị bằng một đơn vị tách rời và có một ý nghĩa hiểu được*"[7]. Định nghĩa này dựa vào cả ba mặt: ngữ âm, chữ viết và ý nghĩa. Tuy nhiên định nghĩa này mâu thuẫn với định nghĩa từ ghép của chính tác giả, vì tác giả định nghĩa từ ghép dựa trên chức năng ngữ pháp và gồm nhiều âm tiết.

Theo Phan Khôi thì "*Từ là một lời để tỏ ra một khái niệm trong khi nói*". Theo Nguyễn Lân thì "*Từ là những tiếng có nghĩa, tức là mỗi khi nghe thấy, trong óc chúng ta đều có một khái niệm*". Nếu xem từ tương đương với khái niệm thì những từ hình thái như à, ư, nhỉ, nhé ... hay những hư từ như cũng, với, bởi ... sẽ mang khái niệm gì? Trên thực tế, từ và khái niệm không tương ứng 1-1 với nhau. Có những khái niệm có thể biểu thị bằng nhiều từ.

Theo Nguyễn Kim Thản thì "*Từ là đơn vị cơ bản của ngôn ngữ, có thể tách khỏi các đơn vị khác của lời nói để vận dụng một cách độc lập và là một khối hoàn chỉnh về mặt ý nghĩa (từ vựng hay ngữ pháp) và cấu tạo*". Quan niệm của ông về "đơn vị cơ bản" là những đơn vị có số lượng hữu hạn để thông báo, trao đổi tư tưởng cho nhau. Đơn vị này phải có nghĩa, và khi sử dụng, người sử dụng phải có ý thức về nó. Chính vì vậy mà đơn vị cơ bản này không thể là câu (vì số lượng câu là vô hạn) và cũng không thể là âm tiết (vì nhiều âm tiết không có nghĩa và khi sử dụng, người sử dụng không ý thức về nó). Vậy đơn vị cơ bản là cái gì đó nhỏ hơn câu và lớn hơn âm tiết.

Theo Hồ Lê thì "*Từ là đơn vị ngữ ngôn có chức năng định danh phi liên*

[7]Nguyễn Kim Thản, *Nghiên cứu ngữ pháp tiếng Việt*. NXB GD, 1997. Trang 30

kết hiện thực, hoặc chức năng mô phỏng tiếng động, có khả năng kết hợp tự do, có tính vững chắc về cấu tạo và tính nhất thể về ý nghĩa". Theo ông, từ khác với âm tiết chủ yếu về mặt ý nghĩa. Từ có ý nghĩa ngữ ngôn, còn âm tiết thì chỉ có ý nghĩa tiền ngữ ngôn. Từ khác từ tố ở khả năng kết hợp. Từ có khả năng kết hợp tự do trong lời nói, còn từ tố thì chỉ có khả năng kết hợp hạn chế. Từ khác với cụm từ tự do bởi tính vững chắc về cấu tạo, tính nhất thể về ý nghĩa và bởi chức năng định danh phi liên kết hiện thực. Từ khác cụm từ cố định (thành ngữ, ngạn ngữ) chủ yếu bởi chức năng định danh phi liên kết hiện thực của nó.

Đái Xuân Ninh chủ trương không định nghĩa từ, vì "từ trước đến nay, trong ngôn ngữ học đại cương cũng như trong tiếng nói cụ thể như tiếng Việt, chưa có một định nghĩa nào thỏa đáng cả". Theo ông thì "đứng về mặt chức năng và cấu trúc của ngôn ngữ, chỉ cần xác định đơn vị từ và mối quan hệ của nó với các đơn vị khác trong tiếng nói". Ông cho rằng ta có thể nhận diện từ một cách khái quát như sau: "*Từ là đơn vị cơ bản của cấu trúc ngôn ngữ ở giữa hình vị và cụm từ. Nó được cấu tạo bằng một hay nhiều đơn vị ở hàng ngay sau nó tức là hình vị và lập thành một khối hoàn chỉnh*".

Nguyễn Tài Cẩn, tuy không định nghĩa trực tiếp từ tiếng Việt, nhưng ông đã chứng minh những tính chất đặc biệt của "tiếng", một đơn vị mà ông coi chính là hình vị và có tính năng rất gần với "từ", nó cũng chính là "từ đơn" và là thành tố trực tiếp để tạo nên "từ ghép". Theo ông, mọi đặc thù về từ pháp của tiếng Việt bắt nguồn từ tính đơn lập của tiếng Việt mà thể hiện rõ nét nhất là qua một đơn vị đặc biệt, đó chính là tiếng. Quan điểm này cũng được Cao Xuân Hạo đồng tình.

Kế thừa quan điểm coi tiếng gần trùng với từ. Nguyễn Thiện Giáp đã phát triển tư tưởng này lên đến mực cực đoan là coi tiếng trong tiếng Việt chính là từ trong các ngôn ngữ Ấn-Âu. Theo ông "*Nếu quan niệm từ không chỉ là đơn vị ngôn ngữ học mà còn là đơn vị tâm lý-ngôn ngữ học, nếu chú ý*

đến tính nhiều mặt của từ và đặc điểm của từ trong từng ngôn ngữ, nếu nhận diện từ căn cứ vào những quan hệ đối lập trong nội bộ từng ngôn ngữ thì cái đơn vị gọi là "tiếng" của Việt ngữ có đủ tư cách để được gọi là "từ"". Như vậy Nguyễn Thiện Giáp đã không sử dụng đến khái niệm hình vị trong tiếng Việt (đơn vị dùng để cấu tạo từ trong các ngôn ngữ Ấn-Âu). Trong quan niệm về từ của ông, ông chủ yếu dựa trên các tiêu chí nhận diện thuộc về hình thức mà không nhấn mạnh tiêu chí về ngữ nghĩa và khả năng độc lập về ngữ pháp.

2.2.2 Đặc điểm của từ

Từ các định nghĩa trên, có thể rút ra các đặc điểm chính của từ nói chung như sau:

- Về hình thức, từ phải là một khối về cấu tạo (chính tả, ngữ âm . . .).
- Về nội dung, từ phải có ý nghĩa hoàn chỉnh.
- Về khả năng, từ có khả năng hoạt động tự do và độc lập về cú pháp.

Đối với từ tiếng Việt, ta có thể rút ra những đặc điểm của từ tiếng Việt so với các ngôn ngữ thuộc loại hình khác. Tiếng Việt là một ngôn ngữ đơn lập với các đặc điểm chính như sau:

- Trong hoạt động ngôn ngữ, từ không biến đổi hình thái. Ý nghĩa ngữ pháp nằm ở ngoài từ.
- Phương thức ngữ pháp chủ yếu là trật tự từ và từ hư.
- Tồn tại một đơn vị đặc biệt là hình tiết mà vỏ ngữ âm của nó trùng khít với âm tiết. Đơn vị đó còn được gọi là tiếng.

- Không có hiện tượng cấu tạo từ bằng cách ghép thêm phụ tố vào gốc từ.

2.2.3 Các quan niệm về hình vị và từ trong tiếng Việt

Đối với từ trong tiếng Việt, đến nay có một số quan điểm như sau:

- Coi mọi tiếng đều là từ (Nguyễn Thiện Giáp). Điều này thuận tiện trong xử lý nhưng không đúng với tiêu chí ngôn ngữ học đại cương vì có nhiều tiếng không có nghĩa, như "phê" trong "cà phê", "bù" trong "bù nhìn" ...

- Coi tiếng chưa hẳn là từ (đa số các nhà Việt ngữ học). Trong số này chia thành ba nhóm sau:

  - Xem tiếng là hình vị. Quan niệm có thể chấp nhận được nếu coi hình vị là hình vị tiếng Việt (gồm tha hình vị và á hình vị)
  - Xem tiếng lớn hơn hình vị (Trần Ngọc Thêm, Lưu Văn Lang ...) cho là tiếng có những hình vị (khuôn vần).
  - Xem tiếng nhỏ hơn hoặc bằng hình vị. Đa số các tiếng đều là hình vị, ngoại trừ "hấu" trong "dưa hấu", "bù" trong "bù nhìn" ... vì những tiếng này không có nghĩa. Quan điểm này được nhiều người chấp nhận.

- Xem tiếng châu Âu (Anh, Pháp ...) cái nào là từ thì trong tiếng Việt cái đó là từ. Quan điểm này chưa xét đến sự khác biệt về sự từ vựng hoá giữa hai ngôn ngữ do khác biệt về văn hoá.

Theo quan điểm ngôn ngữ học đại cương, từ được cấu tạo bởi các hình vị, và hình vị chính là các đơn vị có nghĩa nhỏ nhất. Vì vậy, từ trong tiếng

Việt cũng phải được cấu tạo bởi các hình vị nêu trên, nhưng có điều khác là các hình vị thành phần ở đây không hoàn toàn giống khái niệm hình vị của ngôn ngữ học đại cương, mà là "hình vị tiếng Việt" hay còn gọi là "hình tiết" (hình vị + âm tiết) hay "tiếng" (vì chỉ tiếng Việt mới có đơn vị tiếng đặc biệt như vậy).

2.3 Từ láy

Từ láy là từ mà các thành tố kết hợp với nhau chủ yếu là theo quan hệ ngữ âm. Số lượng từ láy trong tiếng Việt rất lớn, khoảng 4000 từ. Quan hệ ngữ âm trong từ láy thể hiện ở hai mặt:

- Tương ứng về yếu tố siêu đoạn tính (thanh điệu)
- Tương ứng về yếu tố âm đoạn tính (phụ âm đầu, vần và các yếu tố trong vần)

Các thành tố của từ láy thường phải có thanh thuộc cùng một âm vực: hoặc thuộc âm vực cao (ngang, hỏi, sắc), hoặc thuộc âm vực thấp (huyền, ngã, nặng)[8]

Các từ láy có nhiều kiểu, bao gồm láy toàn bộ và láy bộ phận (láy vần, láy phụ âm đầu). Luật hài thanh của mỗi kiểu láy có đặc điểm riêng:

- Trong các từ láy toàn bộ, âm tiết đầu thường là một trong các thanh bằng (1, 2) còn âm tiết thứ hai thường là một trong các thanh trắc (3, 4, 5, 6) cùng âm vực với nó.
- Trong các từ điệp vận, thường có xu hướng thống nhất các thanh điệu ở cả hai âm tiết. Theo thống kê của Nguyễn Thiện Giáp, có 81% số

[8]Trong tiếng Việt hiện đại, thanh ngã thuộc âm vực cao, thanh hỏi thuộc âm vực thấp. Tuy nhiên về mặt lịch sử, thanh hỏi trước kia thuộc âm vực cao còn thanh ngã lại thuộc âm vực thấp (A.G. Haudricourt, 1954)

từ láy vần có thanh điệu hai âm tiết giống nhau hoàn toàn. Trong một số trường hợp, sự kết hợp của thanh điệu trong từ láy không theo đúng luật hài thanh (như *khe khẽ, se sẽ, xốp xộp* . . .) có thể giải thích bằng sự thay đổi lịch sử của thanh ngã từ âm vực thấp lên âm vực cao, kéo theo sự thay đổi của các thanh điệu khác kết hợp với nó, hoặc do quan hệ với cơ chế láy ba.

- Trong các từ láy phụ âm đầu, thanh điệu của hai âm tiết không bắt buộc phải giống nhau, chỉ cần hai thanh điệu ở hai âm tiết cùng âm vực là được.

Sự phân bố thanh điệu trong các từ láy tiếng Việt tuân theo luật phù-trầm. Luật hài hoà thanh điệu này bị chế định rõ rệt trong kiểu láy vần do mối quan hệ chặt chẽ giữa vần và thanh điệu.

2.4 Chính tả tiếng Việt

2.4.1 Tổng quan về chữ viết tiếng Việt

Chữ viết là một trong những phương tiện giao tiếp hiệu quả. Chữ viết cho phép vượt qua những giới hạn về không gian và thời gian của tiếng nói. Nhờ đặc điểm này, chữ viết được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau của đời sống.

Có nhiều hệ thống chữ viết khác nhau được sử dụng trên thế giới, nhưng nhìn chung có thể phân thành hai loại chữ viết sau:

Chữ viết ghi ý Đây là loại chữ viết biểu hiện từ bằng một ký hiệu duy nhất, không liên quan gì đến những âm thanh cấu tạo nên từ. Ký hiệu này liên quan với cả từ và do đó cũng gián tiếp có quan hệ với ý niệm mà từ đó biểu hiện. Loại này bao gồm chữ Trung Quốc, chữ Ai Cập . . .

Vì các ký hiệu chữ viết không phản ánh mặt âm thanh và hình thức ngữ pháp của từ mà phản ánh mặt ý nghĩa, nên trong tiếng Trung Quốc những từ đồng âm được biểu hiện bằng những chữ hoàn toàn khác nhau.

Chữ viết ghi âm Đây là loại chữ viết nhằm tái hiện chuỗi âm thanh nối tiếp nhau trong từ. Các hệ thống chữ viết ngữ âm học có thể ghi âm tiết hay âm tố.

Chữ ghi âm tiết Mỗi ký hiệu ghi một âm tiết. Dẫn chứng cho loại chữ viết này là hệ thống chữ Nhật Hiragana và Katakana.

Chữ ghi âm tố Mỗi ký hiệu ghi một âm tố (hay âm vị). Ví dụ như chữ Anh, chữ Pháp, chữ Nga . . .

Hệ thống chữ viết được sử dụng hiện nay của nước ta là chữ quốc ngữ. Nước ta trước đây vẫn dùng chữ Hán và chữ Nôm. Chữ quốc ngữ được hình thành từ thời Pháp đô hộ nước ta, được người Pháp sử dụng trong các văn tự chính thức và càng ngày càng được sử dụng rộng rãi.

Chữ quốc ngữ ra đời cách nay khoảng ba thế kỷ. Đó là công trình của một nhóm các cố đạo người châu Âu cộng tác cùng một số người Việt. Người để lại nhiều tác phẩm có giá trị trong giai đoạn đầu của chữ quốc ngữ là Alexandre de Rhodes.

Chữ quốc ngữ là một lối chữ ghi âm, dùng chữ cái Latin. Nó dùng những ký hiệu (tức là những con chữ, mượn từ chữ cái Latin, có thêm các dấu phụ) để ghi lại những âm vị, âm tố và các thanh điệu tiếng Việt. Chữ quốc ngữ về căn bản khác với chữ Hán và chữ Nôm. Chữ Hán là lối chữ ghi ý. Chữ Nôm của chúng ta ngày xưa về căn bản cũng là lối chữ ghi ý, tuy có nhiều thành phần ghi âm.

So với chữ Nôm, chữ quốc ngữ có tiến bộ rất lớn vì nó là chữ ghi âm

rất giản tiện, sử dụng vài chục ký hiệu giản tiện là có thể biểu diễn được hệ thống âm thanh tiếng Việt.

So với các hệ thống chữ ghi âm khác như chữ Anh, chữ Pháp thì chữ quốc ngữ là một hệ thống "chữ viết trẻ", mới được dùng phổ biến hơn một thế kỷ nay, nên giữa chữ và âm tương đối có sự phù hợp.

Nguyên tắc chính tả cơ bản của chữ quốc ngữ là nguyên tắc ngữ âm học, có nghĩa là "phát âm thế nào thì viết thế ấy", nên có sự tương ứng khá lớn giữa chữ viết và phát âm.

2.4.2 Chính tả tiếng Việt

Nói ngắn gọn, chính tả là toàn bộ những tiêu chuẩn và những qui luật thực hành chữ viết, bao gồm:

1. Những luật dùng các con chữ của bảng chữ cái để viết các từ.

2. Luật viết các từ độc lập với những chữ cái khi viết chúng.

   Ví dụ: Cách dùng các dấu câu, cách viết hoa, tên người, tên đất ...

Chuẩn mực của cách viết thường tuân theo những nguyên tắc khác nhau.

Đối với những luật chính tả liên quan đến việc sử dụng các con chữ của bảng chữ cái ghi âm, có thể kể đến các nguyên tắc cơ bản sau đây:

Nguyên tắc âm vị học Mỗi âm vị được thể hiện bằng một chữ cái, không phụ thuộc vào vị trí của nó trong các từ và tổ hợp từ.

Nguyên tắc ngữ âm học Chữ cái phản ánh phát âm của âm vị ở những vị trí hay bối cảnh khác nhau.

Nguyên tắc từ nguyên Nguyên tắc viết theo lịch sử, truyền thống. Phản ánh trên chữ viết không phải là trạng thái hiện tại mà là trạng thái quá khứ của hệ thống âm thanh.

Trong bất kỳ một hệ thống chữ viết nào cũng có thể thấy sự kết hợp các nguyên tắc khác nhau. Nhưng mỗi hệ thống chữ viết có những nguyên tắc chủ yếu. Chữ quốc ngữ được xây dựng chủ yếu trên nguyên tắc âm vị học và ngữ âm học. Ngược lại, chữ Pháp và chữ Anh chủ yếu dùng nguyên tắc từ nguyên, viết theo truyền thống lịch sử.

Âm tiết trong tiếng Việt có 5 thành phần, đó là thanh điệu, âm đầu, âm đệm, âm chính và âm cuối.

Âm đầu các âm vị phụ âm đảm nhiệm. Các âm tiết mà có chữ trên chữ viết không ghi phụ âm đầu có thể có âm đầu là âm tắt thanh hầu /ʔ/.

Âm đệm do các âm vị bán nguyên âm /-u-/ đảm nhiệm.

Âm chính do các âm vị nguyêm âm đảm nhiệm như trong bảng 2.1.

Âm vị	Chữ cái	Âm vị	Chữ cái
/i/	i,y	/o/	ô,ôô
/e/	ê	/ɔ/	o,oo
/ɛ/	e,a	/ɤ̆/	â
/ɯ/	ư	/ă/	a,ă
/ɤ/	ơ	/ie/	iê,ia,yê,ya
/a/	a	/uo/	uô,ua
/u/	u	/ɯɤ/	ươ,ưa

Bảng 2.1: Bảng nguyên âm

Âm cuối do các âm vị phụ âm bán nguyên âm đảm nhiệm như trong bảng 2.2 ở trang kế tiếp.

Trên chữ viết, các âm vị âm đầu được thể hiện như trong bảng 2.3 ở trang kế tiếp.

Một số âm như k và q, gh và g, ngh và ng là cùng âm vị. Tuy nhiên, do khi hình thành chữ quốc ngữ, ngữ âm tiếng Việt chưa được nghiên cứu đầy đủ, nên các giáo sĩ đã phải mượn nhiều con chữ ghép trong chữ Bồ Đào Nha, Hi Lạp, Pháp, Ý ... dẫn đến sự không đồng nhất khi biểu diễn âm vị.

Phụ âm cuối		Bán nguyên âm cuối	
Âm vị	Chữ cái	Âm vị	Chữ cái
/-p/	p	/-u/	u,o
/-t/	t	/-i/	i,y
/-k/	c,ch		
/-m/	m		
/-n/	n		
/-ŋ/	ng,nh		

Bảng 2.2: Bảng phụ âm và bán nguyên âm cuối

Âm vị	Chữ cái	Âm vị	Chữ cái
/b/	b	/m/	m
/f/	ph	/v/	v
/t^h/	th	/t/	t
/d/	đ	/n/	n
/s/	x	/z/	d,gi[a]
/l/	l	/ʈ/	tr
/ʂ/	s	/ʐ/	r
/c/	ch	/ɲ/	nh
/k/	q[b], k[c], c	/ŋ/	ngh[c], ng
/x/	kh	/ɣ/	gh[c], g
/h/	h	/ʔ/	khuyết

Bảng 2.3: Bảng phụ âm đầu

[a] Dựa vào nguyên tắc từ nguyên để phân biệt

[b] Dùng khi đứng trước bán nguyên âm /-u-/

[c] Dùng khi đứng trước các nguyên âm /i,e,ɛ,ie/

Hệ thống âm chính tiếng Việt dựa trên cách phát âm Hà Nội bao gồm 9 nguyên âm dài, 2 nguyên âm ngắn, 3 nguyên âm đôi. Tóm gọn các cách biểu diễn nguyên âm chính gồm: i, y, ê, e, a, ư, ơ, a, u, ô, o, â, ă, iê, ia, yê, y, uô, y, uô, ươ, ưa.

Các phụ âm cuối được ghi bằng "nh" khi đứng sau các nguyên âm i, y, ê, e, a. Ví dụ: *minh, mênh, manh*. Trong các trường hợp khác lại được ghi bằng "ng". Ví dụ: *mang, vâng, hồng, xuống*.

Các bán nguyên âm cuối /-u/ ghi bằng "o" khi đứng sau các nguyên âm đơn dài, ở bậc thanh lượng lớn như e, a. Các viết này biểu diễn sự biến dạng của các bán âm sau các mở rộng. Trong các trường hợp còn lại, bán nguyên âm này được ghi bằng "u".

Các bán nguyên âm cuối /-i/ được ghi bằng "y" khi đứng sau các nguyên âm ngắn ă, a, â. Trong các trường hợp khác nó được ghi bằng "i".

Tóm lại, các âm vị cuối được thể hiện bằng những chữ cái: p, t, c, ch, m, n, ng, nh, u, o, i, y.

Tiếng Việt có sáu thanh điệu: sắc, huyền, ngã, hỏi, nặng và thanh không dấu.

Về việc bỏ dấu, có ba nguyên tắc bỏ dấu sau:

Nguyên tắc bỏ dấu khoa học Dấu thanh được đặt ở âm chính của vần, tức là đặt trên hoặc dưới nguyên âm có vai trò quyết định âm sắc chủ yếu của âm tiết.

Nguyên tắc thẩm mỹ (nguyên tắc thứ yếu) Dấu thanh được đặt ở vị tri cân đối trong âm tiết. Nguyên tắc này trước đây hay dùng nhưng nay trong một số trường hợp nếu đặt dấu thanh sai sẽ làm cho phát âm không đúng và hiểu sai nghĩa từ.

Nguyên tắc thực dụng Dấu thanh thường được đặt vào một con chữ nguyên âm chứ không đặt ở giữa hai con chữ, để tiện việc in ấn.

- Nếu âm chính là một nguyên âm đơn thì dấu thanh luôn luôn được ghi ở trên hoặc ở dưới âm chính.
- Nếu âm chính là một nguyên âm đôi thì tùy trường hợp có thể bỏ dấu thanh ở yếu tố thứ nhất hoặc yếu tố thứ hai của âm chính.

2.4.3 Lỗi chính tả

Theo [Hoa02] thì:

> Chữ viết là hệ thống ký hiệu bằng đường nét đặt ra để ghi tiếng nói và có những qui tắc, qui định riêng. Muốn viết đúng chính tả tiếng Việt, ta phải tuân theo những qui định, qui tắc đã được xác lập.
>
> Chính tả là cách viết chữ được xem là chuẩn, tức là viết đúng âm đầu, đúng vần, đúng dấu (thanh), đúng quy định về viết hoa, viết tắt, viết thuật ngữ.

Các lỗi chính tả thường rơi vào loại lỗi do phát âm sai dẫn đến viết sai (lỗi hỏi-ngã, lỗi sai âm đầu, sai âm chính, sai âm cuối). Ngoài ra còn các loại lỗi khác như viết hoa không đúng qui cách, viết tên riêng, thuật ngữ, tên tiếng nước ngoài không đúng qui cách.

Chương 3

Cơ sở tin học

Mục lục

3.1 Bắt lỗi chính tả

Trình bắt lỗi chính tả có thể được đánh giá theo nhiều cách khác nhau. Nhưng chủ yếu vẫn được phân loại từ quan điểm người dùng: khả năng phát hiện lỗi sai, và khả năng đề nghị những từ thay thế cho lỗi sai đó.

3.1.1 Phân loại lỗi chính tả

Có nhiều cách phân loại lỗi khác nhau. Tuy nhiên, xét theo quan điểm của chương trình bắt lỗi chính tả thì lỗi chính tả có thể phân làm hai loại là lỗi non-word và lỗi real-word (được sử dụng trong [TPLT98]):

- Lỗi non-word là lỗi tạo ra từ sai, hoàn toàn không có trong từ điển. Đây là loại lỗi dễ phát hiện. (Ví dụ, “hoa2”, “nhưg” . . .)

- Lỗi real-word là lỗi chính tả mà từ/tiếng đó có trong từ điển. Nếu không dựa vào ngữ cảnh chung quanh thì không thể xác định đó có phải là lỗi chính tả hay không. (Ví dụ, "Anh ta là một người bàng quang" — từ "bàng quang" không đúng, nhưng vẫn có trong từ điển). Đây là loại lỗi rất khó nhận ra và xử lý.

Ngoài ra có thể phân loại lỗi theo nguồn gốc phát sinh lỗi. Theo cách phân loại này, có hai loại lỗi chiếm đa số là lỗi phát âm sai và lỗi nhập sai.

- Lỗi phát âm sai. Lỗi này do sự nhầm lẫn giữa cách đọc và cách viết giữa những từ đồng âm hoặc gần với nhau. Với tiếng Việt, do có nhiều khác biệt cách phát âm giữa các vùng trong khi hệ thống chữ viết dựa trên hệ thống phát âm tiếng Hà Nội, nên dễ dẫn đến các lỗi sai loại này.

- Lỗi nhập sai. Lỗi gây ra do gõ sai phím, gõ sót phím hoặc dư phím.

- Các lỗi khác. Ngoài hai loại lỗi trên, còn có nhiều nguyên nhân khác dẫn đến lỗi chính tả. Một trong những nguyên nhân đó là lỗi dùng từ sai (do hiểu sai, hoặc không hiểu rõ cách dùng từ). Đây thực chất thuộc về lỗi từ vựng, nhưng đôi khi người dùng lại đòi hỏi trình bắt lỗi chính tả phải tìm ra những lỗi này.

  Ngoài lỗi dùng từ sai, còn có những lỗi phát sinh do máy móc. Hai công cụ liên quan đến xử lý văn bản và dễ gây ra lỗi chính tả là nhận dạng tiếng nói và nhận dạng chữ viết. Đối với nhận dạng tiếng nói, lỗi thường gặp giống với dạng lỗi phát âm sai. Tuy nhiên, đối với một số ngôn ngữ như tiếng Anh — mỗi từ gồm nhiều âm tiết — thì có thể gây ra lỗi tách từ sai. Đối với nhận dạng văn bản, lỗi chủ yếu do sự giống nhau giữa các chữ cái khi viết. Thông thường, bản thân các công

cụ này cũng được cài đặt một trình bắt lỗi chính tả tự động (dạng đơn giản hoặc phức tạp) nhằm giảm thiểu các lỗi chính tả.

Theo [Cha98] thì lỗi bao gồm:

- Giống phiên âm
- Giống hình dạng chữ viết
- Giống nghĩa
- Giống cách gõ

3.1.2 Phát hiện lỗi chính tả

Giải pháp đơn giản để phát hiện lỗi chính tả là dùng một cấu trúc dữ liệu để lưu tất cả các từ đã biết (được lưu trong từ điển). Nếu từ không có trong từ điển nghĩa là từ đó bị sai. Giải pháp này cần thêm một số heuristic để tránh không xem các con số, ngày tháng ... là lỗi sai.

Đối với trình bắt lỗi chính tả truyền thống thì từ điển là một phần rất quan trọng. Từ điển có thể được lưu theo các dạng cấu trúc dữ liệu như bảng băm hoặc cấu trúc dữ liệu dạng cây có thể được sử dụng [McI82, Pet80a]

Với những lỗi sai dạng lỗi từ vựng, ta phải dùng một số phương pháp khác phức tạp hơn để phát hiện (chi tiết trong phần 3.4 ở trang 59).

3.1.3 Các sai lầm của trình bắt lỗi chính tả

Khi bắt lỗi chính tả, trình bắt lỗi không tránh khỏi các sai lầm. Có thể phân ra làm hai loại sai lầm: sai lầm tích cực[1] và sai lầm tiêu cực[2].

[1] false positive
[2] false negative

Sai lầm tích cực xảy ra khi trình bắt lỗi báo lỗi ở những từ hoàn toàn không sai chính tả. Sai lầm tiêu cực xảy ra khi trình bắt lỗi bỏ qua những từ bị sai chính tả. Nói cách khác, trình bắt lỗi cho rằng những từ sai chính tả này không sai. Sai lầm tích cực có thể tránh được nhờ tăng kích thước từ điển. Tuy nhiên đây không phải là giải pháp hoàn hảo. Việc tăng kích thước từ điển sẽ tốn kém (về bộ nhớ, CPU, cũng như công sức bỏ ra để xây dựng từ điển). Hơn nữa, càng có nhiều từ thì việc đề nghị các từ thay thế càng trở nên kém hiệu quả do bị phân tán bởi những từ rất ít gặp, không thể tập trung vào những lỗi phổ biến.

Sai lầm tiêu cực có thể xem là lỗi không phát hiện được. Phần nhiều những lỗi này thường đòi hỏi phải hiểu văn bản (ít nhất là một phần văn bản) để có thể phát hiện lỗi. Những dạng lỗi từ vựng, lỗi cú pháp thường rơi vào dạng này. Tuy nhiên vẫn có một số lỗi chính tả rơi vào loại này. Những loại lỗi này được phát hiện nhờ những chương trình bắt lỗi chính tả cảm ngữ cảnh (xem phần 3.4 ở trang 59).

Trong hai loại sai lầm thì sai lầm tích cực thường gây khó chịu cho người sử dụng, dễ gây tâm lý không tin tưởng vào trình bắt lỗi chính tả. Ngược lại, sai lầm tiêu cực phản ánh tính hiệu quả của trình bắt lỗi chính tả. Sai lầm tiêu cực càng nhiều thì trình bắt lỗi càng kém hiệu quả.

3.1.4 Vấn đề chữ hoa, chữ thường

Vấn đề chữ hoa/chữ thường gây nhiều khó khăn cho trình bắt lỗi chính tả. Trong từ điển, hầu hết các từ là chữ thường. Tuy nhiên cũng có chữ hoa (tên riêng, từ viết tắt ...). Các quy tắc chính tả về viết hoa cũng khá phức tạp. Ngoài ra, đôi khi các chữ được viết hoa hoàn toàn để nhấn mạnh, để làm tiêu đề ...

Thuật toán để xử lý trường hợp chữ hoa, chữ thường có thể được mô tả

như trong thuật toán 3.1.

1. Đặt w_t là chữ viết thường của w.
2. Đặt c là kết quả tìm kiếm w_t.
3. Nếu không tìm được c, từ bị sai chính tả.
4. Nếu c giống w, từ đúng.
5. Đặt c_c là chữ thường, viết hoa chữ cái đầu tiên của w. Nếu c giống c_c, từ đúng.
6. Đặt c_u là chữ hoa của w. Nếu c giống c_u, từ đúng.
7. Ngược lại, từ w sai.

Thuật toán 3.1: Xử lý chữ hoa, chữ thường

3.2 Lập danh sách từ đề nghị

Sau khi phát hiện ra từ bị sai chính tả, ta cần đưa ra một số từ "gần giống" có khả năng thay thế từ bị sai chính tả. Trong trường hợp lý tưởng, ta nên đưa ra *một từ duy nhất*, đó chính là từ đúng chính tả, lẽ ra cần phải được dùng thay cho từ bị sai chính tả.

Tuy nhiên, việc tìm ra từ đúng của từ bị sai chính tả là một công việc không dễ dàng, ngay cả với con người. Khi gặp một từ sai chính tả, ta thường phải suy nghĩ nhiều, chọn ra một số từ có khả năng thay thế, kiểm nghiệm xem từ nào là từ thích hợp nhất. Quá trình kiểm nghiệm xem từ nào là thích hợp thường đòi hỏi phải hiểu về nội dung của văn bản đang xem (đối với con người). Đối với máy tính, việc hiểu văn bản, đến nay vẫn là một vấn đề khó. Tuy nhiên, máy tính cũng có khả năng tìm ra kết quả đối với một số trường

hợp lỗi thông dụng (chi tiết trong phần 3.4 ở trang 59). Việc tìm ra chỉ một kết quả duy nhất đưa đến một thuận lợi đáng kể. Bởi vì chỉ có một kết quả, không cần phải lựa chọn, nên ta có thể tạo ra chương trình bắt lỗi chính tả (và sửa lỗi chính tả) tự động. Việc tạo ra một chương trình bắt lỗi chính tả tự động hoàn toàn mở ra một khả năng to lớn khi áp dụng vào thực tế, giúp giảm đáng kể công sức của con người.

Trong trường hợp không thể đưa ra một đề nghị duy nhất, ta có thể đưa ra một danh sách các từ "có khả năng" để người dùng chọn lựa. Yêu cầu đặt ra là từ đúng phải nằm trong danh sách từ lựa chọn. Và tốt hơn nữa là từ đúng nên được đặt trên cùng danh sách để gây sự chú ý của người dùng (chi tiết trong phần 3.3 ở trang 55). Để đảm bảo từ đúng nằm trong danh sách, ta cần tìm hiểu nguyên nhân dẫn đến lỗi, sau đó cố gắng phục hồi lỗi để tạo lại những từ có khả năng. Do có nhiều nguyên nhân khác nhau dẫn đến lỗi chính tả, nên cũng có nhiều cách khác nhau để phát sinh danh sách từ đề nghị.

3.2.1 Lỗi phát âm sai

Đối với các ngôn ngữ như tiếng Việt — vốn "nói sao viết vậy", giải pháp khá đơn giản. Ta có thể phân tích cấu trúc tiếng trong tiếng Việt, sau đó dựa vào các cách phát âm giống nhau để tạo ra danh sách các tiếng phát âm giống nhau.

Đối với các ngôn ngữ như tiếng Anh — cách viết không còn tương ứng với cách đọc nữa, thì giải pháp sẽ phức tạp hơn. Cơ bản là ta cần một cách nào đó để chuyển từ được viết thành một dạng phiên âm, sau đó áp dụng như bình thường. Một số heuristic được đưa ra để giải quyết vấn đề này. Thuật toán cơ bản là Soundex [Knu73]. Nhiều thuật toán khác được đưa ra để cải

tiến Soundex như Double Metaphone[3], Phonetex [AHD01]. Soundex cũng được cải tiến để áp dụng cho các ngôn ngữ khác, như tiếng Thái [KSM97]. Nói chung, các kỹ thuật này biến đổi về cơ bản thay thế các ký tự trong từ bằng như ký tự khác chung hơn, với mục đích làm cho sau khi biến đổi, các từ có cách đọc giống nhau sẽ trở nên giống nhau. Ví dụ như trong Soundex:

- Các ký tự "aeiouhwy" được thay bằng "0".
- "bpfv" được thay bằng "1".
- "cgjkqsxz" được thay bằng "2".
- "dt" được thay bằng "3".
- "l" được thay bằng "4".
- "mn" được thay bằng "5".
- "r" được thay bằng "6".

Cách thay thế khác nhau tùy vào từng thuật giải. Ngoài ra, các thuật giải có thể giữ lại một số ký tự mà không thay thế.

3.2.2 Lỗi nhập sai

Lỗi nhập liệu xảy ra khi gõ không đúng phím cần gõ trên bàn phím. Damerau [Dam64] xác định bốn thao tác có thể gây ra lỗi như sau:

- Tráo đổi một cặp ký tự.
- Xóa một ký tự đã có.

[3]http://aspell.sourceforge.net/metaphone/

- Chèn một ký tự lạ.
- Thay một ký tự bằng một ký tự khác.

Damerau cho rằng 80% các lỗi là do thực hiện thao tác trên một lần (một trong bốn thao tác trên).

Phân loại lỗi theo các thao tác trên dẫn đến một kỹ thuật sửa lỗi đơn giản được dùng bởi [Pet80b]. Nếu phát hiện một từ bị sai chính tả, ta lần lượt thực hiện lại những thao tác trên để phục hồi từ bị sai chính tả. Những từ được phát sinh, nếu có trong từ điển, sẽ được lưu vào danh sách những từ đề nghị. Kỹ thuật này thường được gọi là Đảo ngược lỗi[4].

3.2.3 Các lỗi khác

Ngoài hai loại lỗi trên, còn có nhiều nguyên nhân khác dẫn đến lỗi chính tả. Một trong những nguyên nhân đó là lỗi dùng từ sai (do hiểu sai, hoặc không hiểu rõ cách dùng từ). Đây thực chất thuộc về lỗi từ vựng, nhưng đôi khi người dùng lại đòi hỏi trình bắt lỗi chính tả phải tìm ra những lỗi này.

Ngoài lỗi dùng từ sai, còn có những lỗi phát sinh do máy móc. Hai công cụ liên quan đến xử lý văn bản và dễ gây ra lỗi chính tả là nhận dạng tiếng nói và nhận dạng chữ viết. Đối với nhận dạng tiếng nói, lỗi thường gặp giống với dạng lỗi phát âm sai. Tuy nhiên, đối với một số ngôn ngữ như tiếng Anh — mỗi từ gồm nhiều âm tiết — thì có thể gây ra lỗi tách từ sai. Đối với nhận dạng văn bản, lỗi chủ yếu do sự giống nhau giữa các chữ cái khi viết. Thông thường, bản thân các công cụ này cũng được cài đặt một trình bắt lỗi chính tả tự động (dạng đơn giản hoặc phức tạp) nhằm giảm thiểu các lỗi chính tả.

[4]Error reversal

3.3 Sắp xếp danh sách

Việc chọn từ tốt nhất trong danh sách từ đề nghị là một công việc không dễ dàng. [AGSV98] mô tả cách lựa chọn trong trường hợp này, có thể chia làm các nhóm như sau:

- Sử dụng phân tích cú pháp để loại bỏ những từ sai từ loại, hoặc sai các đặc trưng hình thái (số đếm, chữ hoa/chữ thường ...)
- Khử nhập nhằng ngữ nghĩa để chọn từ phù hợp với ngữ cảnh nhất.
- Dùng thống kê để chọn từ thường xuất hiện nhất.
- Những từ có cách viết hoa/thường khác với từ bị sai sẽ bị loại (ví dụ, nếu từ viết sai là chữ thường thì các từ đề nghị viết hoa sẽ bị loại)

Một số kỹ thuật để sắp xếp danh sách từ được chọn sẽ được mô tả ngắn gọn bên dưới.

3.3.1 Văn phạm ràng buộc

Văn phạm ràng buộc[5] (CG) được thiết kế độc lập ngôn ngữ và là một công cụ mạnh giúp khử nhập nhằng các văn bản không giới hạn [LVHA94].

CG có thể được xem như một tập hợp các luật mẫu-hành động[6], không quá một luật với mỗi tag có nhập nhằng. Mỗi luật bao gồm một hoặc nhiều mẫu (các "ràng buộc") xác định khi nào tag đó không hợp lệ. Nếu thỏa một mẫu trong số các mẫu của luật, tag đó sẽ bị xoá. Các mẫu ngữ cảnh có thể là mẫu cục bộ hoặc toàn cục, có thể tham khảo những phân tích nhập

[5] Constraint Grammar

[6] pattern-action rule

nhằng hoặc không nhập nhằng. Thuật toán sẽ được chạy vài lần để giảm nhập nhằng từ từ, nhờ đó giúp các ngữ cảnh giảm nhập nhằng, hoặc không còn nhập nhằng, tạo điều kiện khử nhập nhằng những từ khác.

Mô tả cú pháp và hình thái được mã hoá bằng tag thay vì cấu trúc đóng mở ngoặc. Mô tả cú pháp rất nông. Mỗi từ được gắn với một tag chức năng cú pháp[7], quy định mô tả phụ thuộc về mặt chức năng.

Các ràng buộc giúp tránh các dự đoán có nhiều rủi ro chứ không chọn ra giải pháp đúng. Do đó CG chỉ giúp giảm số lượng các nhập nhằng. Văn phạm ràng buộc tiếng Anh (EngCG) đã giúp cải thiện đáng kể chất lượng bộ đánh nhãn từ loại tiếng Anh. Văn phạm ràng buộc giúp loại bỏ hầu hết các nhập nhằng có thể được.

Việc áp dụng CG để khử nhập nhằng cho trình bắt lỗi chính tả là một công việc khó khăn vì hiện nay CG cho tiếng Việt vẫn chưa được xây dựng.

3.3.2 Mật độ quan niệm

Đây thực chất là áp dụng khử nhập nhằng ngữ nghĩa dùng WordNet và độ đo khoảng cách giữa các khái niệm trong WordNet. Cách này được áp dụng cho danh từ.

WordNet là một mạng ngữ nghĩa về từ vựng tiếng Anh, bao gồm các mối liên hệ khác nhau giữa các từ tiếng Anh. WordNet định nghĩa các quan hệ khác nhau cho mỗi từ loại. Đối với danh từ thì hai loại quan hệ quan trọng nhất là hypernym và hyponym.

A được xem là hyponym của B (và B là hypernym của A) nếu ta có thể nói "A là một loại đặc biệt của B". Ví dụ, cây là một loại thực vật. Vậy cây là hyponym của thực vật (và thực vật là hypernym của cây)

WordNet được tổ chức theo đơn vị là các synset. Synset (Synonym set)

[7] syntactic function tag

là một nhóm các từ đồng nghĩa có thể dùng thay thế cho nhau. Mỗi từ có thể thuộc nhiều synset khác nhau. Trong trường hợp đó, các syset được gọi là sense của từ đó. Phần danh từ trong WordNet có thể xem như một đồ thị của các synset và các liên kết hypernym/hyponym giữa các synset đó.

Độ đo khái niệm[8] cung cấp một nền tảng để đo độ giống nhau về mặt nghĩa của các từ. Độ đo khái niệm được định nghĩa bởi [RMBB89] là độ dài đường đi ngắn nhất liên kết các khái niệm trong mạng ngữ nghĩa phân cấp.

Cho một khái niệm c nằm trên đỉnh cây con và $nhyp$ là số hypernym mỗi nút. Mật độ quan niệm[9] (CD) để khử nhập nhằng cho c khi cây con của nó chứa m sense của từ đó như sau:

$$CD(c,m) = \frac{\sum_{i=0}^{m-1} nhyp^{i^{0.20}}}{descendants_c}$$

Trong công thức trên, tham số $0,20$ được dùng để làm trơn hệ số mũ i khi m chạy từ 1 đến số sense tổng cộng trong WordNet. Nhiều giá trị đã được thử cho tham số này và tham số gần $0,20$ là tốt nhất.

Thuật toán khử nhập nhằng dựa trên CD như sau: Cho cửa sổ với kích thước nhất định, chương trình di chuyển cửa sổ mỗi danh từ một lần, từ đầu câu cho đến hết, khử nhập nhằng cho danh từ ở chính giữa cửa sổ, xem các danh từ còn lại trong cửa sổ là ngữ cảnh. Đặt cửa sổ các danh từ là W và danh từ chính giữa cửa sổ là w, ta có thuật toán 3.2 ở trang kế tiếp.

Đầu tiên, thuật toán thể hiện một dàn các danh từ trong cửa sổ, các sense và hypernym của chúng (bước 1). Sau đó thuật toán tính CD cho mỗi khái niệm trong WordNet tương ứng với sense nó chứa trong cây con của nó (bước 2). Thuật toán chọn khái niệm c với CD cao nhất (bước 3) và chọn sense đúng

[8] conceptual distance

[9] conceptual density

1. tree := compute_tree(words_in_window).
   Loop
2. tree := compute_conceptual_distance(tree)
3. concept := select_concept_width_highest_weight(tree)
   if concept = null then exitloop
4. tree := mark_disambiguated_senses(tree,concept)
   endloop
5. output_disambiguation_result(tree)

Thuật toán 3.2: Khử nhập nhằng danh từ dùng CD

bên dưới cho những từ tương ứng (bước 4).

Thuật toán tiến hành tính CD cho những sense còn lại trong dàn, tiếp tục khử nhập nhằng những danh từ còn lại trong cửa sổ (quay lại bước 2, 3, 4). Khi không thể khử nhập nhằng được nữa, những sense còn lại của w được xử lý và xuất kết quả ra (bước 5).

Giải pháp CD có hạn chế là chỉ áp dụng đối với danh từ. Những loại từ khác, do có các mối quan hệ phức tạp hơn nhiều so với quan hệ hypernym của danh từ nên rất khó áp dụng. CD đôi khi không thể khử nhập nhằng tuyệt đối (chỉ chừa lại một kết quả) mà nhiều khi vẫn còn lại vài nhập nhằng. Tuy nhiên việc giảm nhập nhằng bằng CD cũng giúp ít rất nhiều cho trình bắt lỗi chính tả.

Hạn chế quan trọng của CD khi áp dụng cho tiếng Việt là thiếu WordNet hoàn chỉnh cho tiếng Việt. Việc xây dựng một mạng ngữ nghĩa tiếng Việt có tầm vóc như WordNet sẽ tốn rất nhiều công sức, chưa kể các điểm khác biệt giữa tiếng Anh và tiếng Việt đòi hỏi các nhà ngôn ngữ học phải xem xét lại

có thể áp dụng hoàn toàn các mối quan hệ đã được sử dụng trong WordNet hay không, hay cần phải loại bỏ và thêm vào một số quan hệ khác cho phù hợp với tiếng Việt. Nói tóm lại, đây là một giải pháp hay tuy nhiên không thể áp dụng trong điều kiện hiện tại. Gần đây có nhiều đề tài nghiên cứu xây dựng WordNet tiếng Việt [TND03]. Hy vọng có thể áp dụng CD và các giải pháp dựa trên WordNet khác cho tiếng Việt trong tương lai không xa.

3.4 Bắt lỗi tự động

Tự động phát hiện và sửa lỗi chính tả được đặt ra để cải tiến các chương trình bắt lỗi chính tả. Các chương trình bắt lỗi chính tả truyền thống thường dựa trên từ điển, nên không thể bắt lỗi những từ sai, nhưng lại có trong từ điển. Ví dụ, "give me a peace of cake" (lẽ ra phải là "give me a piece of cake") hoặc "anh ấy là một người bàng quang" (trong khi phải là "anh ấy là một người bàng quan"). Hướng giải quyết là dựa vào tập nhầm lẫn để tìm ra những từ có khả năng viết sai (ví dụ, "peace-piece" và "bàng quang-bàng quan") sau đó dựa vào ngữ cảnh để xác định xem đang xét có phù hợp với ngữ cảnh hay không. Bởi vậy bài toán này còn được gọi là bắt lỗi chính tả cảm ngữ cảnh[10].

3.4.1 Mô hình TBL

TBL[11] là mô hình học có giám sát, được Eric Brill đưa ra vào năm 1993. Đây là mô hình học luật dựa trên lỗi, tạo ra các luật mới để khắc phục các lỗi còn lại sau khi đã áp dụng các luật trước đó. TBL được áp dụng để tự động phát hiện và sửa lỗi chính tả. TBL chỉ nhắm vào một tập lỗi thông dụng cho trước, chủ yếu là loại lỗi dùng từ sai, loại lỗi rất khó bị phát hiện bởi các trình bắt

[10] context-sensitive spelling checking
[11] Transformation-based Learning

lỗi chính tả thông thường. Những lỗi không phải từ (lỗi nhập liệu . . .) không được xử lý bởi TBL. Phương pháp này được áp dụng bởi Lidia Mangu và Eric Brill [MB97] cho kết quả rất cao (93,15%).

TBL hoạt động như một bộ luật sửa lỗi. Dữ liệu ban đầu cần được một chương trình khác (baseline) xử lý để phát hiện hiện và sửa lỗi chính tả. Mục tiêu của chương trình này phát hiện và sửa đúng lỗi chính tả càng nhiều càng tốt. Các lỗi gây ra bởi chương trình ban đầu này sẽ được sửa bởi TBL. Các luật học được từ quá trình huấn luyện TBL sẽ được áp dụng lần lượt theo thứ tự, sửa chữa các lỗi của do chương trình baseline gây ra cũng như các lỗi do chính việc áp dụng luật TBL gây ra. Kết quả là số lỗi sai chính tả sẽ giảm đáng kể.

Các luật trong TBL là các luật dạng mẫu-hành động[12] sử dụng nhiều loại thông tin khác nhau để xác định ngữ cảnh. Hành động trong luật thường là thay thế từ đang xét bằng một từ khác. Các thông được sử dụng trong mẫu bao gồm vị trí tương đối của các từ so với từ đang xét, từ loại, từ . . . Ba loại mẫu được dùng trong [MB97] là:

- Từ W xuất hiện trong phạm vi $\pm k$ từ chung quanh từ w đang xét.
- Một mẫu xác định gồm l từ/từ loại liên tiếp nhau xuất hiện chung quanh w.
- Một mẫu xác định gồm các từ/từ loại không liên tiếp, xuất hiện quanh w.

Huấn luyện TBL giống như cách áp dụng luật TBL. Dữ liệu đầu vào là một ngữ liệu đã được đánh dấu (từ đúng/từ sai — nếu từ sai thì đi kèm với từ đúng). Thực hiện các bước sau:

1. Gỡ bỏ các đánh dấu trong ngữ liệu, đưa trở về dạng ngữ liệu thô.

[12] pattern-action rule

2. Đánh baseline cho ngữ liệu thô.
3. Dựa vào các mẫu luật, phát sinh các luật.
4. Lần lượt áp dụng các luật lên ngữ liệu.
5. Tính điểm cho ngữ liệu dựa trên ngữ liệu đã đánh dấu ban đầu, sau khi áp dụng từng luật lên ngữ liệu. Điểm tăng nghĩa là kết quả đúng nhiều hơn so với khi chưa áp dụng luật. Điểm âm nghĩa là kết quả sai nhiều hơn.
6. Nếu điểm âm, bỏ qua luật này.
7. Nếu điểm dương, đưa luật vào danh sách luật.
8. Nếu điểm tăng ít hơn một giới hạn cho trước, dừng thuật toán.
9. Quay lại bước 4.

Sau khi chấm dứt thuật toán, ta chọn khoảng n luật đầu tiên. Những luật còn lại bị loại bỏ. n luật này chính là những luật kết quả của quá trình huấn luyện theo mô hình TBL.

Việc áp dụng TBL đòi hỏi phải có ngữ liệu đã đánh dấu (ngữ liệu vàng), một hàm tính điểm (được dùng trong bước 5), trình đánh dấu baseline, và các mẫu luật. Ngoài ra còn có một số tham số (ngưỡng dừng thuật toán, các tham số n, k, l ... đã nêu trên). Việc chọn mẫu luật và các tham số thích hợp ảnh hưởng nhiều đến hiệu quả của TBL.

Ngữ liệu đánh dấu có thể được tạo ra từ tập nhầm lẫn[13]. Tập nhầm lẫn xác định những từ thường bị nhầm lẫn (Ví dụ, "their" và "there" hay "đã" và "đả" ...) Từ văn bản đúng chính tả, ta có thể áp dụng tập nhầm lẫn để tạo ra ngữ liệu sai chính tả. Để thực hiện đều này cần có tập nhầm lẫn.

[13]confusion set

Nếu tập nhầm lẫn không chỉ bao gồm các nhập nhằng về tiếng (hoặc từ đơn) mà cả nhập nhằng về từ (Ví dụ "bàn quan" và "bàng quang") thì cần phải có thêm một bộ tách từ.

Hiệu quả của TBL phụ thuộc vào tập nhầm lẫn. Tập nhầm lẫn càng lớn thì khả năng sửa lỗi chính tả bằng TBL càng cao. Tuy nhiên, tập nhầm lẫn càng lớn thì khả năng sai sót cũng càng lớn, và chương trình không thể tập trung vào các lỗi thường gặp. Để TBL hiệu quả hơn, cần sử dụng thông tin từ loại (hoặc phân lớp từ). Tuy nhiên, một khi chưa sửa lỗi chính tả/tách từ xong thì việc tìm từ loại bằng các phương pháp thông dụng trở nên không an toàn.

3.4.2 Mô hình Winnow

Bài toán bắt lỗi chính tả được xem như là bài toán khử nhập nhằng từ. Các từ nhập nhằng được tập hợp thành tập nhầm lẫn. Tập nhầm lẫn $C = \{W_1, \ldots, W_n\}$ nghĩa là mỗi từ W_i trong tập C có thể bị dùng lẫn lộn với các từ còn lại trong C. Bài toán bao gồm một câu, và một từ cần sửa chữa. Thuật toán thể hiện bài toán như là một danh sách các đặc trưng tích cực[14]. Mỗi đặc trưng tích cực thể hiện cho một ngữ cảnh cụ thể. Hai loại đặc trưng được dùng là từ ngữ cảnh[15] và collocation. Từ ngữ cảnh là một tập các từ nằm xung quanh từ đang xét (giới hạn trong khoảng $\pm k$ từ tính từ từ đang xét). Collocation là một mẫu l từ/từ loại liên tiếp nhau xung quanh từ đang xét.

Một bộ rút trích đặc trưng[16] được sử dụng để chuyển văn bản gốc thành danh sách các đặc trưng tích cực. Bộ rút trích đặc trưng phải được huấn luyện trước, để chỉ lọc ra những đặc trưng nhất định (đặc trưng tích cực), thay vì

[14] active feature
[15] context word
[16] feature extractor

tất cả các đặc trưng.

Để huấn luyện bộ rút trích đặc trưng, ta cho chạy bộ rút trích đặc trưng trên ngữ liệu huấn luyện, rút trích tất cả các đặc trưng có thể có, đồng thời thống kê số lượng của mỗi đặc trưng. Sau khi chạy xong, danh sách đặc trưng này sẽ bị cắt bớt theo một tiêu chí cho trước, chỉ chừa lại những đặc trưng được xem là tích cực. Có thể thu gọn danh sách đặc trưng theo nhiều tiêu chí khác nhau. Tuy nhiên cách đơn giản nhất là dựa vào tần số xuất hiện của các đặc trưng. Nếu các đặc trưng xuất hiện ít hơn một ngưỡng nào đó thì đặc trưng đó bị loại bỏ.

Công việc mỗi bộ phân lớp là xác định từ W_i trong tập nhầm lẫn có thuộc về câu đang xét hay không. Mỗi bộ phân lớp chạy thuật toán Winnow. Bộ phân lớp nhận tập các đặc trưng tích cực (đại diện cho câu đang xét), trả về giá trị nhị phân cho biết từ W_i có thuộc về câu đang xét hay không. Đặt F là tập các đặc trưng tích cực. Với mỗi $f \in F$, đặt w_f là trong số của cung nối f với bộ phân lớp. Thuật toán Winnow trả về giá trị 1 khi và chỉ khi

$$\sum_{f \in F} w_f > \theta$$

trong đó θ là tham số ngưỡng.

Khởi đầu, bộ phân lớp không kết nối với bất kỳ đặc trưng nào trong mạng. Trong quá trình huấn luyện, các kết nối và trọng số của kết nối sẽ được thành lập. Một mẫu huấn luyện bao gồm một câu (tập đặc trưng tích cực) cùng với từ W_c trong tập nhầm lẫn. W_c là từ đúng cho câu trong mẫu huấn luyện đối với các mẫu khẳng định[17] và là từ sai trong các mẫu phủ định[18].

Quá trình huấn luyện được tiến hành theo như sau: lần lượt mỗi mẫu được

[17] positive example

[18] negative example

đưa vào hệ thống, các bộ phân lớp được cập nhật, sau đó mẫu bị hủy.

Bước đầu tiên huấn luyện bộ phân lớp là thiết lập các liên kết giữa bộ phân lớp và các đặc trưng tích cực F trong mẫu. Nếu đặc trưng tích cực $f \in F$ chưa được kết nối vào bộ phân lớp, và câu là mẫu khẳng định đối với bộ phân lớp, ta tạo một kết nối giữa đặc trưng đó và bộ phân lớp với giá trị trọng số khởi đầu là $0,1$. Chú ý rằng không có gì xảy ra với các mẫu phủ định.

Bước kế tiếp là cập nhật trong số cho các liên kết. Bước này được thực hiện nhờ vào luật cập nhật Winnow, chỉ cập nhật trọng số khi xảy ra lỗi. Nếu bộ phân lớp dự đoán là 0 đối với một mẫu khẳng định (nghĩa là lẽ ra bộ phân lớp phải dự đoán là 1), trong số sẽ được tăng:

$$\forall f \in F, w_f \leftarrow \alpha \cdot w_f$$

trong đó $\alpha > 1$ là tham số cho trước. Nếu bộ phân lớp dự đoán 1 với các mẫu phủ định (mà lẽ ra bộ phân lớp phải dự đoán là 0), trọng số sẽ được giảm:

$$\forall f \in F, w_f \leftarrow \beta \cdot w_f$$

với $0 < \beta < 1$ là tham số cho trước. [GR99] đề nghị α là $1,5$ và β là $0,5$ đến $0,9$. Như vậy, trọng số của các đặc trưng không tích cực sẽ giữ nguyên, không thay đổi. Thời gian cập nhật của thuật toán phụ thuộc vào số đặc trưng tích cực trong mẫu.

Thay vì xử lý từ W_i dựa trên một bộ phân lớp, ta có thể áp dụng kết quả trả về của nhiều bộ phân lớp đồng thời. Mô hình Weighted Majority được dùng để kết hợp nhiều bộ phân lớp. Ta cho chạy nhiều bộ phân lớp đồng thời. Các bộ phân lớp trả về các giá trị khác nhau. Hiệu suất của mỗi bộ phân lớp được theo dõi. Trọng số được tính toán để phản ánh độ chính xác của bộ phân lớp. Giá trị sau cùng là tổng của các dự đoán của các bộ phân

lớp được xét, kèm với trọng số của mỗi bộ phân lớp.

Mô hình này được áp dụng trong [GR99].

3.4.3 Mô hình Danh sách quyết định

Mô hình Danh sách quyết định được Yarowsky đưa ra để giải quyết bài toán khử nhập nhằng ngữ nghĩa. Mô hình này dựa trên các đặc trưng quan trọng để nhận dạng. Ngoài ra kết xuất của mô hình rất đơn giản, dễ hiểu, tạo thuận lợi trong nghiên cứu, cải tiến mô hình. Phương pháp này được [TTCV02] áp dụng để bắt lỗi chính tả tiếng Việt.

Mô hình có thể sử dụng nhiều loại đặc trưng khác nhau. Hai đặc trưng thường được áp dụng là từ ngữ cảnh và collocation.

Thuật toán của mô hình như sau:

1. Xét mỗi từ trong câu, có tập nhầm lẫn tương ứng là S.
2. Với mỗi từ $w \in S$:
   (a) Xác định tập đặc trưng C_w không chứa các đặc trưng xung đột với các đặc trưng đã được chấp nhận trước đó.
   (b) Tím điểm của từ:
   $$Score(w) = \max_{f \in C_w} P(w|f)$$
   và xác định
   $$f_w = \underset{f \in C_w}{argmax} P(w|f)$$
3. Từ được chọn là
   $$a = \underset{w \in S}{argmax} Score(w)$$

Ghi nhớ thuộc tính f_w để kiểm tra xung đột ở các vị trí khác.

Quá trình huấn luyện mô hình như sau.

- Bộ rút trích đặc trưng (tương tự như trong mô hình Winnow) được sử dụng để rút ra các đặc trưng tích cực từ các câu trong ngữ liệu huấn luyện.
- Đếm tần số xuất hiện của mỗi đặc trưng.
- Loại bỏ các đặc trưng không đáng tin cậy (Ví dụ, tần số quá thấp).
- Sắp xếp các đặc trưng theo thứ tự giảm dần khả năng quyết định.

3.4.4 Mô hình Trigram và Bayes

Mô hình sửa lỗi bằng Trigram rất đơn giản. Đối với mỗi câu, các từ trong tập nhầm lẫn được thay thế cho từ tương ứng trong câu, sau đó tính xác suất trigram của toàn bộ câu. Từ tương ứng với câu có xác suất lớn nhất sẽ là từ được chọn. Cho câu $W = w_1 \dots w_k \dots w_n$, w'_k là từ được dùng để thay thế cho w_k, tạo ra câu mới W'. Nếu $P(W') > P(W)$ thì w'_k sẽ được chọn, với $P(W)$ và $P(W')$ lần lượt là xác suất trigram của câu W và W'.

Một cải tiến của phương pháp này là áp dụng trigram dựa trên từ loại thay vì trigram từ. Từ câu W, ta tạo ra các chuỗi từ loại. Xác suất cuối cùng là:

$$
\begin{aligned}
P(W) &= \sum_T P(W,T) \\
P(W,T) &= P(W|T)P(T) \\
&= \prod_i P(w_i|t_i) \prod_i P(t_i|t_{i-2}t_{i-1})
\end{aligned}
$$

với T là một chuỗi từ loại của W, $T = t_1 \dots t_n$ và $P(t_i|t_{i-2}t_{i-1})$ là xác suất trigram từ loại.

Một mô hình khác để tìm và sửa lỗi chính tả là áp dụng bộ phân lớp Bayes. Có thể xem đây là bài toán phân lớp từ dựa vào một tập các đặc trưng. Từ cần xét là từ nằm trong tập nhầm lẫn, ta sẽ xét từ này và các từ khác trong tập nhầm lẫn trong cùng ngữ cảnh. Tập đặc trưng chính là ngữ cảnh của từ cần xét. Tập đặc trưng được rút trích từ câu đang xét. Các đặc trưng và cách rút trích đặc trưng tương tự như trong mô hình Winnow.

Mô hình Trigram và Bayes, mỗi cái có điểm mạnh riêng. Mô hình trigram hoạt động tốt nếu những từ trong tập nhầm lẫn không cùng từ loại. Ngược lại, khi không thể phân biệt dựa trên từ loại, mô hình Bayes sẽ hoạt động tốt hơn do dựa vào các thông tin về cú pháp, ngữ cảnh xung quanh. Do đó, giải pháp tốt nhất là kết hợp hai mô hình này với nhau. Đầu tiên ta áp dụng mô hình Trigram. Trong quá trình xử lý, nếu thấy mọi từ trong tập nhầm lẫn đều cùng từ loại, ta áp dụng mô hình Bayes. Ngược lại, ta sẽ chấp nhận kết quả của mô hình Trigram. Giải pháp này được áp dụng trong [GS96], tạo ra mô hình Tribayes.

3.4.5 Mô hình Bayes và Danh sách quyết định

Mô hình Danh sách quyết định là một mô hình mạnh. Tuy nhiên, có một điểm cần lưu ý là mô hình này chỉ sử dụng *một* đặc trưng tốt nhất để phân loại. Có lẽ tốt hơn là nên sử dụng *tất cả* các đặc trưng để phân loại. Không có lý do gì để chỉ sử dụng một đặc trưng, trong khi ta vẫn có thể khai thác các đặc trưng khác. Đây là nơi có thể cải tiến bằng cách sử dụng bộ phân lớp Bayes.

Bộ phân lớp Bayes cũng sử dụng một tập các đặc trưng, sắp theo thứ tự giảm dần hiệu quả. Bộ phân lớp Bayes phân loại từ đang xét bằng cách sử

dụng các đặc trưng. Tuy nhiên, thay vì dừng lại sau khi áp dụng đặc trưng đầu tiên có thể sử dụng, Bayes duyệt qua tất cả các đặc trưng, kết hợp tất cả các đặc trưng, và giải quyết các xung đột giữa các đặc trưng nếu có, để đưa ra kết quả sau cùng. Ta giả sử chỉ sử dụng các đặc trưng loại từ ngữ cảnh và collocation. Nếu hai đặc trưng cần xét đều là từ ngữ cảnh, vậy sẽ không có xung đột. Nếu hai đặc trung đang xét đều là collocation và chồng lên nhau, nghĩa là có xung đột. Đặc trưng xung đột với đặc trưng đã xét trước đó sẽ bị bỏ qua, không được xem xét.

Phương pháp này được đề nghị bởi [Gol95].

3.5 Bắt lỗi tiếng châu Á

Điểm đặc trưng dễ thấy giữa các ngôn ngữ châu Âu và các ngôn ngữ châu Á, như tiếng Trung Quốc, tiếng Nhật, tiếng Hàn Quốc . . . , là ranh giới từ. Đối với các thứ tiếng châu Âu, các cụm ký tự cách nhau bởi khoảng trắng cũng là từ, do đó việc tách từ rất dễ dàng (chỉ phải xử lý các trường hợp đặc biệt như dấu '-' . . .). Đối với các tiếng châu Á, đặc biệt là tiếng Việt, công việc phức tạp hơn nhiều. Bởi vì một "từ" như quan niệm châu Âu chỉ là một phần từ (một âm tiết trong tiếng Việt) trong các ngôn ngữ châu Á. Điều này đặt ra một vấn đề mà các chương trình bắt lỗi chính tả các thứ tiếng châu Âu chưa từng gặp phải: phân ranh giới từ.

Do các phương pháp dựa trên tiếng Anh đều ngầm định có thể tìm được ranh giới từ ngay tức thì, nên khi áp dụng cho tiếng Trung Quốc, tiếng Việt . . . không dễ dàng. Điểm khó khăn ở đây là nếu ta thay một từ bằng một từ khác (trong tập nhầm lẫn) có độ dài không giống từ trước đó, thì toàn bộ ranh giới từ phía sau sẽ bị thay đổi. Nói cách khác, câu đã bị biến thành một câu khác (xét trên quan điểm câu là một chuỗi các từ) khi thay đổi một từ trong câu.

Nếu không giải trực tiếp bài toán mà thực hiện tách từ trước, sau đó mới bắt lỗi chính tả, thì khó khăn lại đè nặng lên phần tách từ. Bài toán tìm ranh giới từ vốn đã phức tạp (trong một số trường hợp, nếu không dựa vào thông tin ngữ nghĩa thì không thể nào tách từ), lại càng phức tạp hơn khi áp dụng trong bài toán bắt lỗi chính tả, vì khi đó ta phải tìm ranh giới từ khi *các từ/tiếng đầu vào có thể không đúng*. Nói cách khác, ta phải thực hiện tách từ trong điều kiện dữ liệu đầu vào không hoàn toàn chính xác: tách từ mờ. Các kỹ thuật tách từ, nếu bỏ qua yếu tố này sẽ dễ dẫn đến sai lầm, vì mọi phương pháp tách từ đều ngầm định là dữ liệu đầu vào là chính xác. Sau khi tách từ xong, bài toán bắt lỗi chính tả trở nên đơn giản. Ta có thể áp dụng các kỹ thuật đã được áp dụng trên bắt lỗi chính tả các ngôn ngữ châu Âu dễ dàng.

Với tiếng Việt, do độ dài mỗi tiếng ngắn (khoảng năm chữ cái, tối đa bảy chữ cái). Tuy nhiên nếu tính theo cấu trúc âm tiết thì mỗi tiếng chỉ gồm tối đa bốn thành phần (không kể thanh điệu) là âm đầu, âm đệm, âm chính và âm cuối. Mỗi thành phần đều được thể hiện bằng một cụm chữ cái riêng biệt, có thể coi như là một đơn vị tương đương chữ cái. Vậy nên, xét một mặt nào đó, có thể xem mỗi tiếng trong tiếng Việt chỉ gồm tối đa bốn "chữ cái". Do độ dài tiếng quá ngắn nên số lượng nhập nhằng của một tiếng lớn hơn rất nhiều so với các ngôn ngữ châu Âu dù chỉ xét từ đồng âm.

Trình bắt lỗi tiếng Việt đã được nghiên cứu trong những năm gần đây [TPLT98, TTCV02, cHN99]. Giải pháp được đề nghị trong [TPLT98, cHN99] sử dụng phân tích cú pháp để đánh giá các cách tách từ. Trong khi đó [TTCV02] sử dụng danh sách quyết định để khử nhập nhằng cho từng từ một.

3.6 Tách từ

Bài toán tách từ cho ngôn ngữ đơn lập đã được đặt ra từ lâu, chủ yếu để giải quyết cho tiếng Trung Quốc, tiếng Nhật. Các thuật toán tách từ có thể được

phân loại như sau:

Dựa theo luật Bao gồm các cách sau:

- Longest Matching, Greedy Matching Models (Yuen Poowarawan, 1986; Sampan Rarurom, 1991)
- Mô hình khớp tối đa[19]. Mô hình này được chia thành "khớp tối đa tiến"[20] và "khớp tối đa lùi"[21]. Đối với phương pháp này thì một từ điển hoàn chỉnh là không thể thiếu. Một từ điển không hoàn chỉnh sẽ giảm hiệu suất của thuật toán. Tuy nhiên, dễ thấy là khó có thể có một từ điển hoàn chỉnh (đặc biệt khi các ngôn ngữ vẫn còn được tiếp tục phát triển hằng ngày trong thời đại ngày nay). Mô hình này tùy thuộc nhiều vào từ điển.

Dùng thống kê Giải pháp này dựa vào ngữ cảnh từ xung quanh để đưa ra quyết định thích hợp. Có hai vấn đề cần được giải quyết đối với giải pháp này: độ rộng ngữ cảnh, và cách áp dụng thống kê. Ngữ cảnh càng rộng thì thuật toán càng phức tạp.

Cho dù độ rộng ngữ cảnh thế nào, luôn có thể áp dụng mô hình first-order HMM. Tuy nhiên giải pháp này phụ thuộc rất nhiều vào ngữ liệu huấn luyện. Kết quả huấn luyện trên ngữ liệu chính trị khó có thể áp dụng trên các tài liệu văn học và ngược lại. Thêm vào đó, có những từ có xác suất rất cao, nhưng chỉ có chứng năng về mặt ngữ pháp, làm giảm vai trò của xác suất.

Các cách khác Hầu hết các giải pháp khác là sự lai tạo giữa các mô hình trên và các mô hình ngôn ngữ học như WFST, TBL. Thời gian xử lý

[19] Maximal Matching Model

[20] forward maximum match

[21] backward maximum match

các giải pháp loại này trở nên đáng kể, nhưng độ chính xác đạt được khá cao.

Tri thức về ngôn ngữ, thường được áp dụng cho các mô hình dựa trên luật, hiếm khi được áp dụng cho những mô hình trên.

Một số phương pháp tách từ được mô tả ngắn gọn bên dưới.

3.6.1 Khớp tối đa

Thuật toán so khớp tối đa hoạt động như tên của chính nó. Thuật toán giải quyết bài toán tách từ bằng cách chọn cách tách từ nào có nhiều từ nhất (so khớp được nhiều nhất). Thuật toán được áp dụng để xây dựng chương trình tách từ tiếng Trung Quốc MMSEG[22]. Thuật toán này có nhiều biến thể khác nhau.

- Dạng đơn giản, được dùng để giải quyết nhập nhằng từ đơn. Giả sử có một chuỗi ký tự (tương đương với chuỗi tiếng trong tiếng Việt) $C_1, C_2, \ldots C_n$. Ta bắt đầu từ đầu chuỗi. Đầu tiên, kiểm tra xem C_1 có phải là từ hay không, sau đó kiểm tra xem C_1C_2 có phải là từ hay không. Tiếp tục tìm cho đến khi tìm được từ dài nhất. Từ có vẻ hợp lý nhất sẽ là từ dài nhất. Chọn từ đó, sau đó tìm tiếp như trên trên những từ còn lại cho đến khi xác định được toàn bộ chuỗi từ.

- Dạng phức tạp. Quy tắc của dạng này là phân đoạn có vẻ hợp lý nhất là đoạn ba từ với chiều dài tối đa. Thuật toán bắt đầu như dạng đơn giản. Nếu phát hiện ra những cách tách từ gây nhập nhằng (ví dụ, C_1 là từ và C_1C_2 cũng là từ), ta xem các chữ kế tiếp để tìm tất cả các đoạn ba từ có thể có bắt đầu với C_1 hoặc C_1C_2. Ví dụ ta được những đoạn sau:

[22]http://casper.beckman.uiuc.edu/ c-tsai4/chinese/wordseg/mmseg.zip

- $C_1 \quad C_2 \quad C_3C_4$
- $C_1C_2 \quad C_3C_4 \quad C_5$
- $C_1C_2 \quad C_3C_4 \quad C_5C_6$

Chuỗi dài nhất sẽ là chuỗi thứ ba. Vậy từ đầu tiên của chuỗi thứ ba (C_1C_2) sẽ được chọn. Thực hiện lại các bước cho đến khi được chuỗi từ hoàn chỉnh. Cách này đạt được độ chính xác 99.69% [CL92].

3.6.2 Mô hình HMM

Trong cách áp dụng này, các trạng thái ẩn là các lớp từ, giả định rằng mỗi từ có thể thuộc mọi lớp với một xác suất nhất định. Bài toán được xem như tìm kiếm chuỗi lớp từ $C = c_1, \ldots, c_n$ từ một chuỗi từ $W = w_1, \ldots, w_n$. Mục tiêu là tìm W và C từ câu S cho trước, sao cho tối đại xác suất

$$\underset{W,C}{argmax} P(W|C)P(C)$$

Giả định rằng xác suất $P(W|C)$ chỉ phụ thuộc vào lớp từ của nó, và xác suất lớp $P(C)$ chỉ phụ thuộc vào lớp của từ đứng trước. Những xác suất này có thể được ước lượng bằng thuật toán Baum-Welch dùng ngữ liệu huấn luyện. Tiến trình học dựa trên thuật toán Baum-Welch và giống với bài toán đánh nhãn từ loại bằng HMM, trừ việc số trạng thái được xác định trước và xác suất khởi đầu được gán ngẫu nhiên.

3.6.3 Mô hình WFST và mạng nơ-ron

WFST[23] đã được [SSGC96] áp dụng để tách từ tiếng Trung Quốc. Ý tưởng cơ bản là áp dụng WFST kết hợp với trọng số là xác suất xuất hiện của mỗi từ trong ngữ liệu. Dùng WFST để duyệt qua câu cần xét. Cách duyệt có trọng số lớn nhất sẽ là cách tách từ được chọn. Giải pháp này cũng đã được áp dụng trong [DKT01, TH01] kèm với mạng nơ-ron để khử nhập nhằng.

3.6.4 Mô hình Source-Channel cải tiến

Mô hình này được đề nghị trong [GLH03].

Đặt S là một câu tiếng Trung Quốc, hay là một chuỗi các ký tự (tương đương chuỗi tiếng trong tiếng Việt). Với mỗi cách tách từ W có thể có, chọn cách tách từ tốt nhất W^*, tương ứng với xác suất điều kiện $P(W|S)$:

$$W^* = \underset{w}{argmax} P(W|S)$$

Theo công thức Bayes, bỏ mẫu số là hằng số, ta được:

$$W^* = \underset{w}{argmax} P(W)P(S|W)$$

Ta định nghĩa lớp từ C như sau:

- Mỗi từ được định nghĩa như một lớp.
- Mỗi từ dẫn xuất hình thái được định nghĩa như một lớp.
- Mỗi loại ký hiệu khác nhau được định nghĩa như một lớp. Ví dụ, các biểu thức thời gian thuộc về lớp TIME.

[23] Weighted Finite State Transducer

- Mỗi loại tên riêng thuộc về một lớp. Ví dụ, tên người thuộc lớp PN.

Ta chuyển công thức trên qua các lớp từ:

$$C^* = \underset{c}{argmax} P(C)P(S|C)$$

Công thức trên là công thức cơ bản của mô hình source-channel cho tách từ tiếng Trung Quốc. Mô hình giả định câu S được phát sinh như sau: Đầu tiên, một người chọn một chuỗi khái niệm (ví dụ, lớp từ C) để xuất ra, theo xác suất $P(C)$. Sau đó người đó cố gắng thể hiện các khái niệm đó bằng chuỗi các ký tự, theo xác suất $P(S|C)$.

Mô hình source-channel có thể được hiểu theo một cách khác: $P(C)$ là mô hình thống kê dự đoán xác suất của chuỗi lớp từ. Nó chỉ ra khả năng một lớp từ xuất hiện, dựa trên một ngữ cảnh cho trước. Vậy $P(C)$ còn được hiểu như *mô hình ngữ cảnh*. $P(S|C)$ là mô hình phát sinh, dự đoán khả năng một chuỗi ký tự được phát sinh dựa trên lớp từ cho trước. Vậy $P(S|C)$ còn được hiểu như *mô hình lớp*.

Mặc dù mô hình ngữ cảnh và mô hình lớp có thể được kết hợp bằng một phép nhân đơn giản. Tuy nhiên nếu thêm trọng số thì kết quả tốt hơn. Lý do là có một số mô hình lớp dự đoán kết quả rất không chính xác. Hơn nữa, các mô hình lớp của các lớp từ khác nhau được xây dựng theo những cách khác nhau. Vì vậy xác suất mô hình lớp khác nhau nhiều giữa các mô hình lớp. Một cách để cân bằng những xác suất này là thêm vào một trọng số CW cho mỗi mô hình lớp để điều chỉnh xác suất $P(S|C)$ thành $P(S|C)^{CW}$

Với mô hình đã có, thao tác tách từ bao gồm hai bước:

1. Cho chuỗi S, phát sinh mọi cách tách từ có thể có. Mỗi cách tách từ được đánh nhãn lớp từ và xác suất lớp $P(S'|C)$ với S' là bất kỳ chuỗi con nào của S.

2. Thuật toán tìm kiếm Viterbi được áp dụng để chọn cách tách từ có khả năng nhất theo công thức nêu trên.

Huấn luyện

Nếu có một dữ liệu được tách từ sẵn, công việc trở nên rất dễ dàng. Tuy nhiên, việc xây dựng một ngữ liệu tách từ sẵn đủ lớn sẽ tốn rất nhiều công sức (đặc biệt là các mô hình thống kê thường đòi hỏi lượng ngữ liệu cực kỳ lớn, lớn hơn rất nhiều so với các mô hình dựa trên luật). Để đơn giản vấn đề, ngữ liệu này được xây dựng tự động như sau:

1. Khởi đầu, sử dụng một bộ tách từ sẵn có (có thể áp dụng các thuật giải đơn giản, không cần huấn luyện, như Longest matching, Maximum matching . . .)

2. Sử dụng mô hình đề nghị để tách từ ngữ liệu huấn luyện.

3. Tái huấn luyện mô hình dựa trên kết quả tách từ có được ở bước 2. Bước 2–3 có thể được lặp lại nhiều lần cho đến khi hiệu suất của mô hình ngừng tăng.

3.6.5 Mô hình TBL

Mô hình TBL (xem thêm phần 3.4.1 ở trang 59) cũng có thể được áp dụng để tách từ tiếng Trung Quốc [Pal97].

Mô hình TBL có thể được áp dụng cho nhiều bài toán khác nhau. Tùy vào các hành động cụ thể của mẫu luật mà cách áp dụng sẽ khác nhau. Mẫu luật áp dụng cho TBL sử dụng các hành động sau:

- Nối hai ký tự (tiếng).

- Tách hai ký tự.
- Trượt ranh giới từ sang ký tự kế bên.

Trình tách từ baseline, có thể áp dụng cách khớp tối đa.

3.7 Tách từ mờ

Như đã nói trên, bài toán tách từ không đơn thuần là tách từ đúng như nghĩa ban đầu, mà là tách từ trong điều kiện dữ liệu đầu vào có khả năng bị sai. Nói cách khác, đây là bài toán tách từ có khả năng chịu lỗi[24]. Các phương pháp tách từ được nêu, thông thường không thể áp dụng trực tiếp mà phải có một số cải tiến nhất định nếu có thể. Ngoài ra cũng có thể áp dụng một số giải pháp khác, tuy không trực tiếp áp dụng cho tách từ mờ, nhưng phần nào có thể gợi ý cho một giải thuật tách từ mờ hiệu quả.

Kemal Oflazer [Ofl96] khi xử lý hình thái trong tiếng Thổ Nhĩ Kỳ gặp trường hợp khá giống với trường hợp này. Tác giả phải tách hình thái từ trong điều kiện từ đó bị sai chính tả. Do đặc tính ngôn ngữ chắp dính[25], số tiếp vĩ ngữ nhiều, liên tiếp nhau, gây khó khăn cho việc nhận dạng tiếp vĩ ngữ, cũng như không thể phân biệt những tiếng nào hợp thành một từ trong một chuỗi tiếng trong tiếng Việt. Tác giả dùng một hàm độ đo, tạo ra các tiếp đầu ngữ có khả năng thay thế dựa trên độ đo này, sau đó sử dụng WFST để tìm chuỗi tiếp vĩ ngữ thích hợp nhất.

Bài toán nhận dạng tiếng nói trong tiếng Anh cũng gặp trường hợp tương tự [Rav96]. Sau công đoạn xử lý âm thanh, ta nhận được một chuỗi các âm tiết. Ta phải chuyển nhóm âm tiết này thành chuỗi từ. Do âm thanh thường bị nhiễu, nên các âm tiết có thể không chính xác hoàn toàn. Ngoài ra, do

[24] error-tolerant word segmentation

[25] agglunative language

đặc tính của tiếng Anh nên cùng một chuỗi âm tiết có thể suy ra nhiều chuỗi từ khác nhau. Tác giả sử dụng lưới từ để tạo ra các chuỗi từ có khả năng từ chuỗi âm tiết, sau đó ngram trên từ để lượng giá các chuỗi từ.

Một điểm đáng chú ý ở đây là sự tương đồng về một mặt nào đó giữa tiếng Anh và tiếng Việt. Với tiếng Anh, từ có thể tách dễ dàng, nhưng từ bao gồm nhiều âm tiết. Việc phân ranh giới âm tiết trong tiếng Anh là một điều khá khó khăn. Với tiếng Việt, các âm tiết được tách rất dễ dàng vì mỗi âm tiết là một "tiếng" tương đương với một "từ" trong câu — cách nhau bởi khoảng trắng. Từ của tiếng Việt lại bao gồm nhiều tiếng, và việc tách từ lại gặp nhiều khó khăn. Với nhận xét này, có thể thấy các bài toán nhận dạng tiếng nói tiếng Anh gặp cùng vấn đề với bài toán bắt lỗi chính tả tiếng Việt!

[Cha98] cũng dùng mô hình ngôn ngữ dạng ngram để đánh giá các cách tách từ sau khi đã qua tiền xử lý nhập nhằng chính tả, tuy nhiên lại sử dùng nhiều mô hình ngôn ngữ khác nhau như character bigram, word bigram, inter-word character bigram (IWCB), POS bigram, word class bigram.

Dựa trên những nghiên cứu này, có thể thấy giải pháp khả thi cho việc tách từ khi bị sai chính tả, là phát sinh một loạt các từ có khả năng thay thế, với giả định trong tập từ này sẽ có từ đúng chính tả, thay thế từ sai chính tả ban đầu. Sau đó sử dụng tách từ tìm một cách tách tốt nhất. Sau khi tìm được cách tách từ, ta có thể tra từ điển để tìm xem từ nào bị sai.

3.7.1 Huấn luyện

Nếu có dữ liệu đã được đánh dấu sẵn các ranh giới từ, công việc đơn giản chỉ là áp dụng các công thức thống kê để tính ra các giá trị cần thiết.

Đối với việc huấn luyện các thuật toán tách từ truyền thống, ta có thể sử dụng bộ tách từ tiếng Việt sẵn có để tạo ngữ liệu. Chất lượng của bộ tách từ sẽ ảnh hưởng đến chất lượng của thuật toán.

Với các thuật toán tách từ mờ, đôi khi không thể áp dụng bộ tách từ sẵn có. Với điều kiện hiện tại, khó có thể tìm được một khối lượng ngữ liệu lớn đã được tách từ sẵn, do đó cần phải tìm giải pháp tính được các tham số cần thiết từ ngữ liệu chưa được tách từ (ngữ liệu thô).

Thuật toán EM[26] thường được áp dụng để vượt qua khó khăn này [PSG99, SSGC96]. Trên lý thuyết, áp dụng thuật toán EM đảm bảo kết quả sẽ hội tụ, và kết quả ở mỗi vòng lặp sau sẽ tốt hơn hoặc bằng kết quả của vòng lặp trước.

Do thuật toán EM cũng có nhiều hạn chế (đặc biệt là hạn chế tối ưu cục bộ), nhiều giải pháp đã được đưa ra để khắc phục các hạn chế này [WGLL00, PS01].

Một giải pháp khác là áp dụng HMM để tìm ra ranh giới từ, sau đó áp dụng các công thức thống kê thông thường. Giải pháp này gặp hạn chế bởi chính HMM, vì HMM là mô hình thống kê thuần tuý, không phát huy được một số đặc trưng của tách từ, cũng không sử dụng các tri thức về ngôn ngữ học, do đó phần nào hạn chế kết quả cuối cùng.

[26]Estimation Maximization

Chương 4

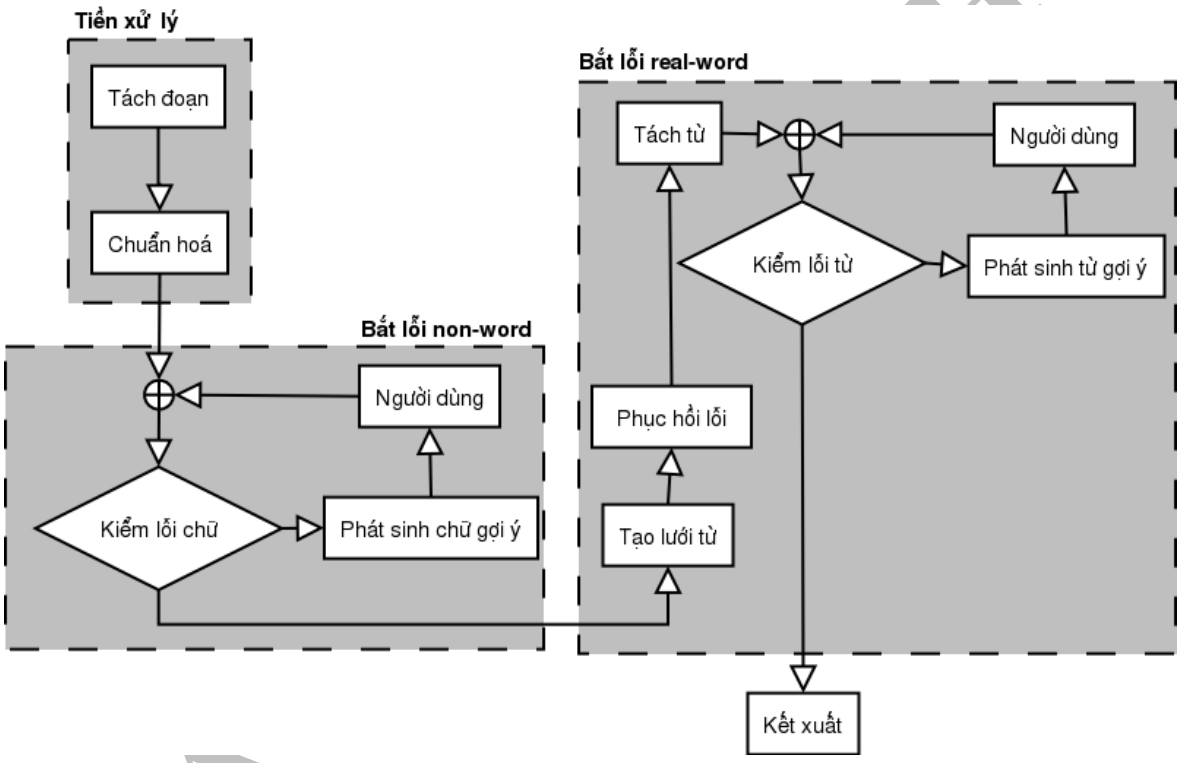
Mô hình

Mục lục

4.1 Mô hình chung

Việc bắt lỗi chính tả của một văn bản được xử lý lần lượt qua các bước (xem hình 4.1 ở trang kế tiếp), bao gồm ba khối chính là:

- Khối tiền xử lý. Tách văn bản thành những đoạn ngắn. Tách đoạn thành từng tiếng. Đánh dấu các ký hiệu, dấu ngắt dòng, các số, tên riêng ...
- Khối bắt lỗi non-word. Kiểm tra các tiếng với các tiếng đã có trong từ điển. Báo lỗi những tiếng không có trong từ điển. Sau đó đưa ra giải pháp thay thế.
- Khối bắt lỗi real-word. Tương tự như khối bắt lỗi real-word. Tuy nhiên cần phải tách từ trước khi thực hiện bắt lỗi chính tả.



Hình 4.1: Mô hình chung

4.1.1 Tiền xử lý

Phần tiền xử lý bao gồm nhiều công đoạn, bắt đầu bằng phần tách các token ("tiếng") (4.2.1 ở trang kế tiếp), và các phần tiền xử lý khác như đánh dấu các số, các ký hiệu ...

Do đây là tiền xử lý trên dữ liệu không chính xác, nên một số công đoạn tiền xử lý (như nhận dạng tên riêng) không được thực hiện ngay ở công đoạn này mà được thực hiện trong phần bắt lỗi real-word.

4.1.2 Bắt lỗi non-word

Phần này sử dụng từ điển để tìm ra những token sai, không có trong từ điển, lập danh sách từ đề nghị, sau đó yêu cầu người dùng chọn từ đúng.

Phần này được chạy lặp đi lặp lại cho đến khi không còn lỗi non-word. Sau đó sẽ đến phần thực hiện của bắt lỗi real-word.

4.1.3 Bắt lỗi real-word

Phần này cố gắng tìm ra tất cả các từ sai chính tả. Phần này gồm hai công đoạn là tách từ và bắt lỗi chính tả.

Đầu tiên, lưới từ của câu đang xét sẽ được xây dựng. Sau đó lưới từ này được bổ sung thêm từ cơ chế phục hồi lỗi và các bước tiền xử lý đã không thể thực hiện được trong phần tiền xử lý. Từ lưới từ này, ta sẽ chọn ra một cách tách từ tốt nhất dựa vào mô hình ngôn ngữ.

Sau khi đã tách từ xong, phần việc còn lại rất giống với các bước đã thực hiện trong phần tách từ non-word. Tuy nhiên, thay vì lần lượt tra cứu các từ với từ điển, ta sẽ so sánh giữa từ trong cách tách từ đúng nhất và từ trong câu có giống nhau hay không. Từ trong cách tách từ đúng nhất được xem là

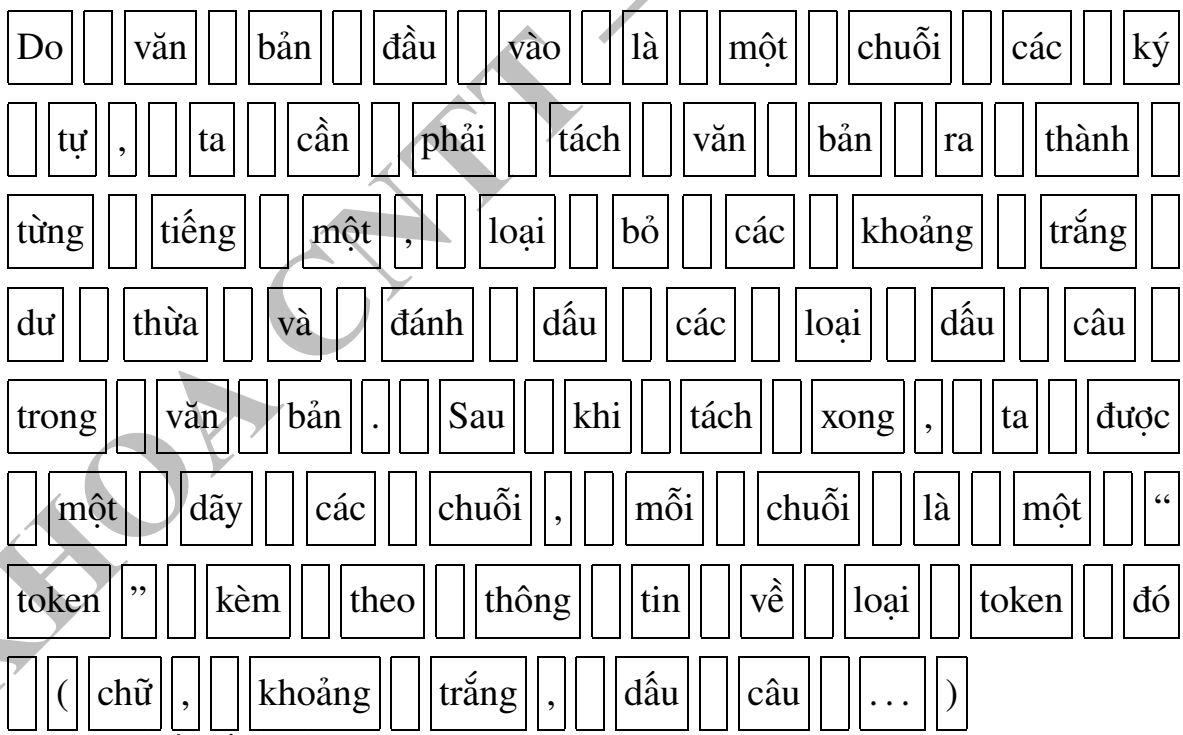
từ đúng. Nếu từ trong câu không giống, ta xem như đó là lỗi chính tả. Phần còn lại được thực hiện tương tự như trong phần bắt lỗi non-word.

4.2 Tiền xử lý

4.2.1 Tách token

Do văn bản đầu vào là một chuỗi các ký tự, ta cần phải tách văn bản ra thành từng tiếng một, loại bỏ các khoảng trắng dư thừa và đánh dấu các loại dấu câu trong văn bản. Sau khi tách xong, ta được một dãy các chuỗi, mỗi chuỗi là một "token" kèm theo thông tin về loại token đó (chữ, khoảng trắng, dấu câu . . .)

Ví dụ, đoạn văn bản trên được tách thành những phần như sau:



Quy luật tách như sau:

- Khoảng trắng là ký tự khoảng trắng (space) hoặc ký tự tab.

- Dấu kết thúc câu bao gồm ba dấu: [.] [?] [!]
- Dấu đóng ngoặc bao gồm: ["] [)] []] [}] [>] [’]
- Một token là:
  - Một hoặc nhiều dấu kết câu và các dấu đóng ngoặc nếu có. Ví dụ: [.)]
  - Một hoặc nhiều dấu nối chữ (hyphen) [-].
  - Một hoặc nhiều dấu mở nháy đơn [‘].
  - Một hoặc nhiều dấu đóng nháy đơn [’].
  - Một hoặc nhiều chữ, chữ số, một số dấu: [.] [,] [:] [’] [$] [%] [-] [\] [/]; kết thúc bằng chữ hoặc chữ số. Ví dụ: [tôi] [$12]
  - Một hoặc nhiều chữ, chữ số, kết thúc bằng dấu tỉnh lược [’] (quy tắc này chủ yếu áp dụng cho tiếng Anh, như [she’] [s] [ ] [a] [ ] [teacher])
  - Một chữ hoặc chữ số.
  - Một hoặc nhiều khoảng trắng, dấu xuống dòng và các ký tự lạ khác.

4.2.2 Tách câu

Mục đích của tách câu là tách ra thành từng câu để xử lý. Việc xác định đơn vị câu rất quan trọng trong các ứng dụng xử lý ngôn ngữ tự nhiên. Tuy nhiên, do chương trình này chưa sử dụng một số thông tin cần đến đơn vị câu, nên "câu" ở đây được hiểu như là một đoạn ngắn, một câu hoặc một phần của câu.

Việc tách câu dựa vào các dấu câu để ngắt câu ra thành từng đoạn để xử lý. Mỗi đoạn sẽ được xử lý độc lập với nhau. Các đoạn được phân cách bởi các token dấu câu [.] [,] [;] [(] [)] ... Phần tách token phải được thực hiện trước phần tách câu.

4.2.3 Chuẩn hoá

Chuẩn hoá dấu

Chuẩn hoá là đặt lại vị trí dấu trong các tiếng nhằm đảm bảo thống nhất một quy tắc đặt dấu chung. Do tiếng Việt có nhiều hơn một quy định về quy tắc bỏ dấu (bỏ dấu khoa học, bỏ dấu mỹ thuật ...), gây khó khăn cho các thao tác xử lý về sau, bởi vì cùng một chữ nhưng dùng hai quy tắc bỏ dấu khác nhau sẽ cho ra hai chuỗi ký tự hoàn toàn khác nhau (Ví dụ, "hoà" và "hòa").

Để tránh tình trạng này, dấu được tách riêng ra khỏi tiếng, được coi như là một ký tự nằm ở đầu tiếng. Như vậy một tiếng bất kỳ sẽ bao gồm ký tự đầu tiên đại diện cho thanh điệu, các ký tự theo sau đại diện cho âm đầu và vần của tiếng đó. Dấu được thể hiện theo quy ước gõ VNI:

- Thanh ngang được biểu diễn bằng ký tự '0'.
- Thanh sắc được biểu diễn bằng ký tự '1'.
- Thanh huyền được biểu diễn bằng ký tự '2'.

- Thanh hỏi được biểu diễn bằng ký tự '3'.
- Thanh ngã được biểu diễn bằng ký tự '4'.
- Thanh nặng được biểu diễn bằng ký tự '5'.

Theo quy tắc trên, "hoà" sẽ được biến đổi thành "2hoa"; "hòa" sẽ được biến đổi thành "2hoa". Như vậy ta có thể coi "hòa" và "hoà" là tương đương nhau khi so sánh dạng biến đổi "2hoa" của chúng. Một số ví dụ khác: "hồng" được biến đổi thành "2hông", "hoa" được biến đổi thành "0hoa".

Các tiếng nước ngoài, các ký hiệu ... không có dấu, sẽ được xem như có thanh ngang. Như vậy, "USA" sẽ được biến đổi thành "0USA".

Do việc biến đổi làm mất thông tin về cách bỏ dấu. Ta cần phải giữ lại chữ gốc bên cạnh chữ biến đổi ("chữ chuẩn hoá") để có thể dùng lại sau này. Ta cũng có thể phục hồi chữ từ chữ chuẩn hoá bằng cách phân tích cấu trúc âm tiết và bỏ dấu thích hợp theo quy tắc bỏ dấu cho trước. Việc này sẽ giúp chuẩn hoá cách bỏ dấu cho toàn văn bản.

Chuẩn hoá 'y' và 'i'

Ngoài việc chuẩn hoá cách bỏ dấu, một số chữ trong tiếng Việt kết thúc bằng 'y' có thể được đổi thành 'i'. Ví dụ, "quý" và "quí" đều hợp lệ. Tuy nhiên, không phải chữ nào kết thúc bằng 'y' cũng có thể chuyển thành 'i', ví dụ "thuý" và "thúi". Nguyên nhân là do khi chuyển thành 'i', chữ cái này kết hợp với 'u' tạo ra âm chính 'ui' thay vì âm chính 'i'. Một số chữ kết thúc bằng 'i' cũng không thể chuyển sang 'y', ví dụ "bí", "chí" ... Việc cho phép viết một từ ở hai cách sẽ làm giảm hiệu suất của chương trình do chương trình coi "quý" và "quí" là hai chữ hoàn toàn khác nhau. Giải pháp là lập danh sách những từ có âm chính là y/i và là âm tiết mở, sau đó chuyển tất cả những từ kết thúc bằng 'i' có trong danh sách trên sang 'y'. Trong quá trình

bắt lỗi chính tả, nếu người dùng yêu cầu chuẩn hoá thì ta có thể xem việc viết 'y' hoặc 'i' như là sai chính tả. Nếu không, ta sẽ bỏ qua khác biệt 'y' và 'i' ở bước báo lỗi chính tả. Gồm các chữ sau (không xét thanh điệu): "mi", "ti", "thi", "qui", "ki", "hi", "li" "si", "vi".

4.2.4 Chữ viết hoa

Chữ viết hoa dùng để biểu diễn tên riêng, từ viết tắt hoặc dùng cho chữ đứng đầu câu. Do đó cần phân biệt chữ đầu câu có phải là chữ bắt đầu tên riêng hay không. Ngoài ra, cần xác định tên riêng khi tìm được chữ viết hoa bắt đầu tên riêng. Các văn bản tiếng Việt chưa hoàn toàn thống nhất về quy tắc viết hoa. Ví dụ, có tài liệu dùng "Cộng hoà Xã hội Chủ nghĩa Việt Nam", nhưng có tài liệu lại dùng "Cộng Hoà Xã Hội Chủ Nghĩa Việt Nam".

Do văn bản đầu vào có khả năng bị sai chính tả, kèm theo sự không thống nhất trong quy cách viết tên riêng, nên khó có thể xác định tên riêng ngay ở bước tiền xử lý. Vì vậy phần này sẽ được thực hiện trong phần tách từ thay vì trong phần tiền xử lý.

4.2.5 Từ nước ngoài, từ viết tắt, các ký hiệu ...

Xử lý tiếng nước ngoài, các ký hiệu chuyên ngành, các từ viết tắt. Do trình bắt lỗi không có kiến thức về các lĩnh vực chuyên ngành, cũng như các thứ tiếng trên thế giới, nên việc áp dụng tri thức để phân loại là điều hết sức khó khăn. Giải pháp được dùng ở đây là xem các từ nước ngoài, từ viết tắt, các ký hiệu ... như là những chữ bình thường (và sẽ được xem như là lỗi chính tả trong phần bắt lỗi chính tả).

Phần này sẽ cố gắng phân loại một số loại thường gặp như số, ngày tháng ... nhằm giảm bớt các lỗi sai chính tả không đáng có. Các con số

được đánh dấu riêng bằng mã NUM. "Số" ở đây được coi là bất cứ chữ nào bắt đầu bằng số. Ví dụ, "0lit", "0.2", "0-4" ... đều được coi là số.

Ngày tháng được nhận dạng theo mẫu "ngày-tháng-năm" hoặc "ngày/tháng/năm". Nói cách khác, ngày tháng là các số liên tiếp, cách nhau bằng dấu '/' hoặc '–'. Ngày tháng cũng được đánh nhãn NUM.

4.3 Bắt lỗi non-word

4.3.1 Tìm lỗi chính tả

Việc tìm lỗi chính tả đơn giản là duyệt qua từng token, kiểm tra xem token đó có trong từ điển hay không. Nếu token không có trong từ điển, token đó bị sai chính tả.

Nếu thực hiện như trên, sẽ có rất nhiều chữ bị cho là sai chính tả, ví dụ như các con số, ngày tháng ... Ta cần phải nhận ra những token loại này và bỏ qua chúng khi tìm lỗi chính tả. Do phần tiền xử lý đã đánh dấu các token chứa số bằng mã NUM nên khi thực hiện tìm lỗi chính tả, ta sẽ không xét những token loại này.

4.3.2 Lập danh sách từ đề nghị

Sau khi đã xác định những chữ bị sai chính tả, ta cần đưa ra một số gợi ý để người dùng chọn, thay vì buộc người dùng tự tìm ra chữ đúng. Việc lập ra danh sách gợi ý chủ yếu dựa vào nguyên tắc phục hồi lỗi: Dựa vào nguyên nhân phát sinh ra lỗi, thực hiện thao tác ngược lại để tìm ra chữ đúng.

Lỗi được xử lý trong phần này là lỗi non-word. Lỗi này có thể do những nguyên nhân sau:

- Lỗi nhập liệu sai.

- Lỗi OCR.
- Lỗi nhận dạng tiếng nói.
- Lỗi phát âm sai.

Lỗi nhập liệu

Lỗi nhập liệu bao gồm những loại lỗi sau:

- Lỗi gõ sót phím.
- Lỗi gõ dư phím.
- Lỗi gõ sai phím (gõ nhầm phím này bằng phím khác).
- Lỗi gõ sai thứ tự (gõ đảo thứ tự hai phím liên tiếp nhau).

Ngoài ra, do phím spacebar phát sinh ký tự khoảng trắng dùng để phân cách các chữ với nhau, nên cần phải xử lý đặc biệt với phím này. Dựa vào các loại lỗi trên, ta có thêm các lỗi bổ sung:

- Lỗi gõ thiếu phím spacebar, gom hai chữ thành một chữ.
- Lỗi gõ dư phím spacebar, tách một chữ thành hai chữ.
- Lỗi gõ sai phím spacebar, có thể dẫn đến việc nhóm hai chữ hoặc tách chữ làm hai.
- Lỗi gõ sai thứ tự giữa một phím và phím spacebar dẫn đến việc một ký tự trong chữ này bị đẩy sang chữ khác.

Giải pháp khắc phục lỗi là thực hiện ngược lại quá trình tạo ra lỗi. Đối với lỗi gõ sót phím, dư phím, ta có thể thêm một phím hoặc bớt một phím để

tạo ra chữ mới. Với lỗi gõ sai thứ tự, ta duyệt qua chữ, lần lượt hoán vị hai ký tự liên tiếp để tạo ra chữ mới.

Đối với lỗi gõ nhầm phím, ta dựa vào bố trí bàn phím để phát sinh lỗi. Giả định sơ đồ bố trí của bàn phím EN-US được dùng. Do thông thường chỉ gặp một lỗi gõ nhầm với phím ngay bên cạnh này mỗi từ, nên chương trình chỉ lưu danh sách những phím lân cận với từng phím, dựa trên bàn phím EN-US. Ví dụ: $A \rightarrow \{S, Q, W, X, Z\}$.

Với những phím hai ký tự như phím '2' ('2' và '@') thì @ sẽ được thêm vào tập các phím lân cận với 2 và ngược lại.

Danh sách cụ thể được nêu trong bảng 4.1 ở trang kế tiếp.

Do với mỗi phím có khoảng 8 phím lân cận. Một chữ dài trung bình 5 ký tự sẽ phát sinh ra một tập 8^5 các chuỗi có khả năng. Trong số này chỉ có một số rất ít là chữ thật sự, đúng quy tắc chính tả. Tuy nhiên việc xử lý một khối lượng lớn như vậy là không thể. Vì vậy chương trình giả định chỉ nhập gõ sai tối đa hai phím với mỗi chữ, nhằm giảm thiểu bùng nổ tổ hợp.

Với lỗi spacebar, ta cũng xét tương tự. Tuy nhiên, với những lỗi có khả năng tách làm hai chữ, ta sẽ xét chữ hiện thời và chữ kế tiếp cùng lúc, xem như là một chữ. Lỗi gõ nhầm spacebar với một phím khác là điều khó có thể xảy ra vì phím spacebar tương đối lớn, dễ nhận diện khi gõ.

Với các kiểu gõ như VNI, TELEX. Chương trình cố gắng "phục hồi" từ những chữ gõ sai nếu phát hiện được. Ví dụ, "nguyê4n" sẽ tạo ra "nguyễn". Bộ gõ được VNI, TELEX cài đặt, sau đó nhận chuỗi các ký tự của chữ đang xét qua bộ gõ phục hồi các dấu. Bước này được thực hiện sau bước trên, để nếu gõ nhầm phím dấu kế bên thì vẫn có thể phục hồi lại.

Bộ gõ VNI và TELEX được sử dụng như trong bảng 4.2 ở trang 92. Nguyên tắc của các bộ gõ được cài đặt không quá cầu kỳ, do yêu cầu chỉ là để phục hồi lỗi. Các phím dấu của mỗi bộ gõ sẽ được duyệt qua chữ, từ vị trí của phím dấu về đầu chữ. Nếu phím dấu có thể kết hợp được với ký tự đang

Phím	Phím lân cận	Phím	Phím lân cận	Phím	Phím lân cận
`	~1!	~	`1!	1	!`~q2@
!	1`~q2@	2	@1!qw3#	@	21!qw3#
3	#2@we4$	#	32@we4$	4	$3#er5%
$	43#er5%	5	%4$rt6^	%	54$rt6^
6	^5%ty7&	^	65%ty7&	7	&6^yu8*
&	76^yu8*	8	*7&ui9(	*	87&ui9(
9	(8*io0)	(	98*io0)	0	)9(op-_
)	09(op-_	-	_0)p[=+	_	-0)p[=+
=	+-_[{]}\\|	+	=-_[{]}\\|	\	|=+]}
|	\=+]}	q	aw2@1!	w	qase3#2@
e	wsdr4$3#	r	edft5%4$	t	rfgy6^5%
y	tghu7&6^	u	yhji8*7&	i	ujko9(8*
o	iklp0)9(	p	ol;:[{-_0)	[	]}=+-_
{	]}=+-_	]	[{=+|	}	[{=+|
a	qwsz	s	awedxz	d	erfcxs
f	rtgvcd	g	tyhbvf	h	yujnbg
j	uikmnh	k	iol,<mj	l	op;:.>,<k
;	/?.>l	:	/?.>l	’	[{]}/?;:
"	’[{]}/?;:	z	asx	x	zsdc
c	xdfv	v	cfgb	b	vghn
n	bhjm	m	njk,<	,	<mkl.>
<	,mkl.>	.	>,<l;:/?	>	.,<l;:/?
/	?.>;:’"	?	/.>;:’"		

Bảng 4.1: Danh sách phím lân cận

được duyệt, ta xem như đã phục hồi lỗi. Lỗi VNI-TELEX được áp dụng lại, sau khi đã phục hồi lỗi theo phương pháp trên, vì khi gõ VNI-TELEX người dùng vẫn có khả năng gõ nhầm các phím dấu.

VNI		TELEX	
Phím	Dấu	Phím	Dấu
1	dấu sắc	s	dấu sắc
2	dấu huyền	f	dấu huyền
3	dấu hỏi	r	dấu hỏi
4	dấu ngã	x	dấu ngã
5	dấu nặng	j	dấu nặng
6	dấu mũ	aa,ee,oo	â,ê,ô
u7,o7	ư,ơ	uw,ow	ư,ơ
a8	ă	aw	ă
d9	đ	dd	đ

Bảng 4.2: Kiểu gõ VNI-TELEX

Các chữ được phát sinh sẽ được so sánh với từ điển tiếng. Nếu chữ nằm trong từ điển, ta cho chữ vào danh sách đề nghị. Ngược lại, chữ bị hủy bỏ.

Tóm lại, thuật toán 4.1 ở trang kế tiếp để phục hồi lỗi bàn phím.

Lỗi phát âm

Lỗi phát âm chủ yếu gây ra lỗi real-word. Tuy nhiên lỗi phát âm đôi khi vẫn có lỗi non-word. Đây là những từ phát âm giống nhau, chữ viết bao gồm các thành phần âm đầu, âm đệm, âm chính, âm cuối hợp lệ. Tuy nhiên kết hợp các thành phần lại không tạo thành một chữ nằm trong từ điển tiếng. Ví dụ, "nghành", "ngỉ", "ka". Do đặc điểm của chữ quốc ngữ, một số âm tố không thể kết hợp với nhau. Đây là nguyên nhân chủ yếu của lỗi loại này.

Lỗi phát âm sẽ được trình bày cụ thể trong phần 4.4.3 ở trang 100. Phần này chỉ nêu ra những điểm riêng của lỗi phát âm — non-word.

1. Phục hồi lỗi VNI-TELEX.
2. Phục hồi lỗi gõ sót phím. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
3. Phục hồi lỗi gõ dư phím. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
4. Phục hồi lỗi nhầm phím. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
5. Phục hồi lỗi sai thứ tự phím. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
6. Phục hồi lỗi gõ sót phím spacebar. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
7. Phục hồi lỗi gõ dư phím spacebar. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
8. Phục hồi lỗi sai thứ tự phím spacebar. Phục hồi lỗi VNI-TELEX dựa trên kết quả của lần phục hồi trước.
9. Lọc lại danh sách những từ đã có. Nếu từ không nằm trong từ điển tiếng thì loại bỏ.

Thuật toán 4.1: Phục hồi lỗi bàn phím

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Nếu ký tự đang xét là phím dấu, duyệt ngược từ vị trí ký tự đang xét về đầu chữ. Nếu có thể ghép dấu, thực hiện ghép dấu, lưu chữ vào danh sách.

Thuật toán 4.2: Phục hồi lỗi VNI-TELEX

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Ở từng vị trí, chèn thêm một ký tự vào bên phải ký tự đang xét. Lưu chữ vào danh sách.

Thuật toán 4.3: Phục hồi lỗi gõ sót phím

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Ở từng vị trí, xóa ký tự đang xét. Lưu chữ vào danh sách.

Thuật toán 4.4: Phục hồi lỗi gõ dư phím

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Ở từng vị trí, thay thế ký tự đang xét bằng ký tự lân cận (như trong bảng 4.1 ở trang 91). Lưu chữ vào danh sách.

Thuật toán 4.5: Phục hồi lỗi gõ nhầm phím

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Ở từng vị trí, hoán vị ký tự đang xét và ký tự bên phải ký tự đang xét. Lưu chữ vào danh sách.

Thuật toán 4.6: Phục hồi lỗi gõ sai thứ tự phím

1. Duyệt lần lượt từng ký tự trong chữ, từ trái sang phải.
2. Ở từng vị trí, tách chữ làm hai chữ tại vị trí ký tự đang xét. Lưu hai chữ vào danh sách.

Thuật toán 4.7: Phục hồi lỗi gõ sót phím spacebar

1. Nối chữ đang xét và chữ bên phải chữ đang xét lại làm một.
2. Lưu chữ vào danh sách.

Thuật toán 4.8: Phục hồi lỗi gõ dư phím spacebar

1. Chuyển một ký tự từ chữ bên phải chữ đang xét sang cuối chữ đang xét. Lưu vào danh sách.
2. Chuyển một ký tự từ chữ đang xét sang đầu chữ bên phải chữ đang xét. Lưu vào danh sách.

Thuật toán 4.9: Phục hồi lỗi gõ sai thứ tự phím spacebar

Với loại lỗi này, ta cần phân tích cấu trúc âm tiết tiếng Việt. Sau khi phân tích cấu trúc âm tiết tiếng Việt, ta sẽ thay thế từng thành phần của âm tiết bằng một thành phần khác có cách phát âm giống như thành phần được thay thế. Thuật toán lỗi phát âm như trong thuật toán 4.10.

1. Phân tích cấu trúc âm tiết.
2. Tạo danh sách âm tiết, bao gồm các âm tiết có phát âm tương tự với âm tiết ban đầu.
3. Lọc lại danh sách, những âm tiết không có trong từ điển tiếng bị loại bỏ.

Thuật toán 4.10: Phục hồi lỗi phát âm (non-word)

4.3.3 Sắp xếp danh sách từ đề nghị

Như đã nói trong phần 3.3 ở trang 55, có nhiều cách để sắp xếp danh sách từ đề nghị. Do không thể sử dụng các thông tin cú pháp, ngữ nghĩa, giải pháp được áp dụng là dùng mô hình ngôn ngữ (được dùng khi bắt lỗi real-word — phần 4.4.5 ở trang 103) sẽ được dùng để đánh giá các từ đề nghị, cùng với việc loại bỏ các chữ không cùng chữ hoa/thường với chữ gốc và áp dụng thống kê tần số sử dụng của mỗi từ.

Giải pháp được áp dụng như sau trong thuật toán 4.11.

1. Phát sinh lưới từ.
2. Thêm các chữ đề nghị vào, cập nhật lưới từ, thêm các từ mới từ những chữ được thêm.
3. Áp dụng mô hình ngôn ngữ, dựa vào điểm để sắp xếp danh sách từ.

Thuật toán 4.11: Sắp xếp danh sách từ đề nghị (non-word)

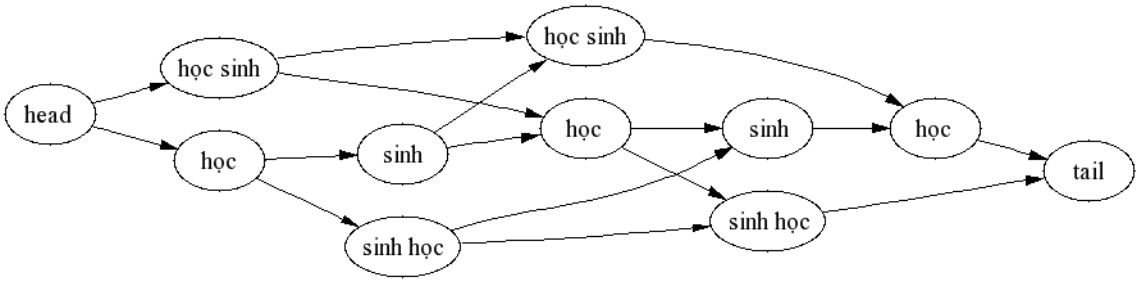
4.4 Bắt lỗi real-word

4.4.1 Lưới từ

Bắt lỗi real-word đòi hỏi tách từ và lượng giá các cách tách từ bằng mô hình ngôn ngữ. Để thuận tiện xử lý, chương trình sử dụng một cấu trúc dữ liệu dạng đồ thị để thể hiện các từ — gọi là lưới từ[1]. Đây là một đồ thị có hướng không chu trình, với các nút là các từ trong câu, cạnh là đường nối giữa hai từ kề nhau, hướng thể hiện thứ tự của các từ trong câu. Ngoài các nút là các từ trong câu, lưới từ có hai nút đặc biệt là nút “head” và nút “tail”. Nút

[1] word lattice

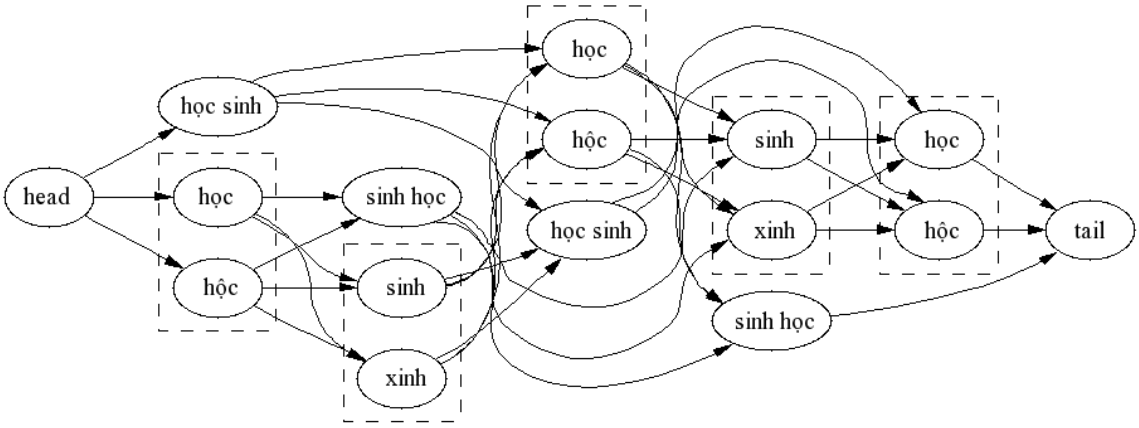
"head" nằm ở đầu lưới từ, nối với những nút tương ứng với những từ đầu tiên trong câu. Ngược lại, nút "tail" nằm ở cuối lưới từ, được nối với những từ cuối cùng trong câu. Lưới từ chứa tất cả các từ có khả năng xuất hiện trong câu. Các từ được liên kết với nhau theo trật tự như trong câu. Khi duyệt từ nút gốc đến nút đích, ta sẽ được một cách tách từ cho câu. Hình 4.2 thể hiện một lưới từ.



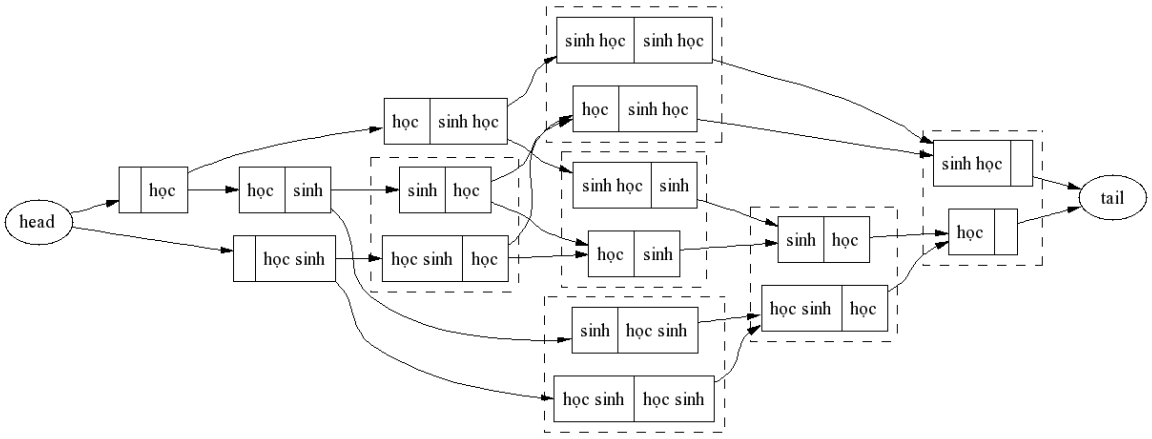
Hình 4.2: Lưới từ của câu "Học sinh học sinh học"

Ngoài lưới từ cơ bản như mô tả ở trên, ta có thể mở rộng lưới từ để chứa thêm những từ có khả năng, phát sinh từ công đoạn phục hồi lỗi, nhằm xác định xem từ nào là đúng nhất. Lưới từ này gọi là lưới từ mở rộng (hình 4.3 ở trang kế tiếp).

Một dạng khác của lưới từ, gọi là lưới 2-từ (xem hình 4.4 ở trang kế tiếp). Trong loại lưới từ này, mỗi nút không phải là một từ mà là hai từ đứng liền nhau. Hai nút nối liền nhau thì từ bên phải của nút bên trái và từ bên trái của nút bên phải là một. Nói cách khác, một cặp nút nối với nhau bằng một cạnh trong lưới từ chỉ có ba từ thay vì bốn từ. Lưới từ loại này dùng để thể hiện mô hình trigram (trong khi lưới từ bình thường được dùng để thể hiện bigram). Việc tạo ra lưới 3-từ, lưới 4-từ là có thể. Tuy nhiên những lưới từ này thường không hiệu quả.



Hình 4.3: Lưới từ mở rộng của câu "Học sinh học sinh học"



Hình 4.4: Lưới 2-từ của câu "Học sinh học sinh học"

4.4.2 Tạo lưới từ

Lưới từ cơ bản được tạo bằng thuật toán Viterbi. Mỗi tiếng trong câu được duyệt qua để tìm ra tất cả các từ có thể có trong đoạn. Sau đó tập hợp các từ này lại tạo nên lưới từ.

Cho câu S có n tiếng. State là đỉnh, còn "nút i" là cạnh. Ta duyệt lần lượt qua các cạnh để tìm ra các từ. Duyệt i từ 1 đến n:

- Tạo state gốc cho nút i.
- Xét các state, nếu tiến thêm được một bước thì lưu lại state mới (i+1).
- Nếu không tiến được thì xóa state.
- Nếu hoàn tất một từ thì lưu lại.

Thuật toán 4.12: Tạo lưới từ (cơ bản)

Thuật toán tạo lưới 2-từ được nêu trong [Rav96]. Thuật toán sẽ tạo ra lưới n-từ từ lưới $n-1$-từ. Lưới từ cơ bản nên trên được xem như là lưới 1-từ. Thuật toán được tóm tắt lại như trong thuật toán 4.13.

1. Nếu nút (w) có n từ đứng liền trước nó (w_i), $i = 1, 2, \ldots, n$ trong lưới từ gốc, nó sẽ được lặp lại n lần trong lưới từ mới, tên là $(w_i w)$, tương ứng với $i = 1, 2, \ldots, n$.
2. Nếu (w_i) nối với (w_j) trong lưới từ gốc, nối tất cả $(w_x w_i)$ với $(w_i w_j)$ x bất kỳ.
3. Giá trị của $(w_i w_j)$ là giá trị của cạnh (w_i) (w_j) trong lưới từ cũ.
4. Giá trị của cạnh $(w_i w_j)$ $(w_j w_k)$ là 3-gram của w_i, w_j và w_k.

Thuật toán 4.13: Tạo lưới n-từ từ lưới $(n-1)$-từ

4.4.3 Mở rộng lưới từ — Phục hồi lỗi

Sau khi có lưới từ cơ bản. Ta có thể áp dụng mô hình ngôn ngữ để tìm ra cách tách từ tốt nhất nếu bài toán là tách từ. Tuy nhiên, do câu không hoàn toàn chính xác, ta cần phải tiến hành phục hồi lỗi — thêm vào những từ có khả năng đúng, sau đó mới có thể tách từ. Phần phục hồi lỗi, thêm từ vào lưới từ để tạo lưới từ mở rộng là nội dung của công đoạn này.

Như đã xét ở phần 4.3.2 ở trang 88, có 4 loại lỗi chủ yếu là lỗi bàn phím, lỗi OCR, lỗi nhận dạng tiếng nói, lỗi phát âm (và các loại lỗi còn lại). Cả bốn loại lỗi này đều có thể gây ra lỗi real-word. Tuy nhiên, lỗi real-word chủ yếu là lỗi phát âm, lỗi bàn phím chiếm rất ít, phần còn lại là lỗi OCR và lỗi nhận dạng tiếng nói. Phần này chủ yếu tập trung vào lỗi phát âm. Lỗi bàn phím được trình bày trong phần 4.3.2 ở trang 89.

Lỗi phát âm

Lỗi phát âm phụ thuộc vào cách phát âm của từng vùng. [Hoa02] liệt kê các trường hợp lỗi thông dụng nhất. Những quy tắc này được áp dụng để tạo ra những từ gần giống phát âm. Theo [Hoa02], lỗi bao gồm các loại sau:

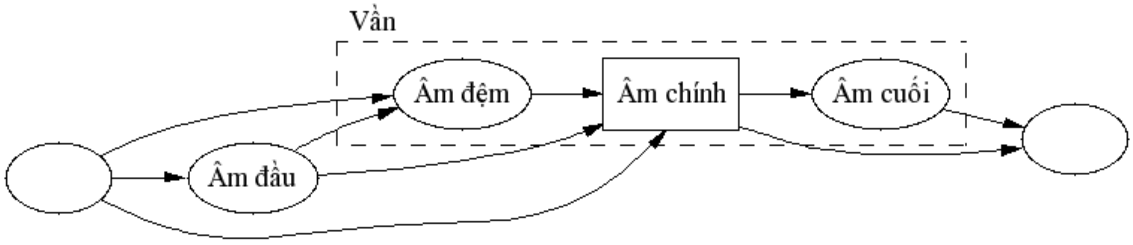
- Lỗi thanh điệu. Chủ yếu là lỗi nhầm lẫn hai thanh hỏi, ngã.
- Lỗi về âm đầu. Thường lẫn lộn các âm đầu sau: C/K, G/Gh, Ng/Ngh, Ch/Tr, S/X, V/D/Gi/R, W/Hw/Ngw/Qu.
- Lỗi về âm chính. Thường lẫn lộn các âm chính sau: ai/ay/ây, ao/au/âu, ăm/âm, ăp/âp, iu/iêu/êu, im/iêm/em, ip/iêp/êp/ep, oi/ôi/ơi, om/ôm/ơm, op/ôp/ơp, ong/ông, oc/ôc, ui/uôi, um/uôm, up/(uôp), ưi/ươi, ưu/ươu, ưm/ươm, (ưp)/ươp.

- Lỗi về âm cuối. Thường lẫn lộn chữ ghi các âm cuối trong các vần sau: an/ang, at/ac, ăn/ăng, ăt/ăc, ân/âng, ât/âc, en/eng, et/ec, ên/ênh, êt/êch, in/inh, it/ich, iên/iêng, iêt/iêc, ơn/(ơng), ơt/(ơc), un/ung, ut/uc, uôn/uông, uôt/uôc, ưn/ưng, ưt/ưc, ươn/ương, ươt/ươc.

- Sai quy cách viết hoa

Phân tích âm tiết

Để phát sinh từ dựa trên lỗi phát âm, cần phân tích cấu trúc của từng tiếng. Một tiếng bao gồm âm đầu, vần và thanh. Vần gồm âm đệm, âm chính và âm cuối. Trong các thành phần của tiếng, âm chính là bắt buộc phải có. Các thành phần còn lại có thể không có. Ta có thể biểu diễn cấu trúc âm tiếng theo sơ đồ trạng thái như hình 4.5.



Hình 4.5: Sơ đồ trạng thái phân tích cấu trúc tiếng

Ta có thể phân tích tiếng dùng FST. Tuy nhiên, qua hình 4.5 có thể thấy chỉ có tám cách để hình thành tiếng. Cài đặt theo tám cách này đơn giản và hiệu quả hơn dùng FST tổng quát.

Khi phân tích tiếng, có thể có một số nhập nhằng giữa các thành phần của tiếng. Ví dụ, "lúa" bao gồm âm đôi "ua" hay âm đệm "u" và âm chính "a"? Những âm có thể gây nhập nhằng xuất phát từ việc âm 'u' và 'o' có thể vừa là âm đệm, vừa là âm chính, bao gồm các âm "uô", "ua". Ngoài ra còn

Kiểm tra mẫu A, bao gồm việc phân tích âm tiết theo mẫu A. Nếu âm tiết có thể phân tích theo mẫu A và không vi phạm các quy luật phân bố thì chấp nhận âm tiết, dừng thuật toán.

1. Kiểm tra mẫu "Âm đầu, âm chính, âm cuối".
2. Kiểm tra mẫu "Âm chính, âm cuối".
3. Kiểm tra mẫu "Âm đầu, âm chính".
4. Kiểm tra mẫu "Âm chính".
5. Kiểm tra mẫu "Âm đầu, âm đệm, âm chính, âm cuối".
6. Kiểm tra mẫu "Âm đệm, âm chính, âm cuối".
7. Kiểm tra mẫu "Âm đầu, âm đệm, âm chính".
8. Kiểm tra mẫu "Âm đệm, âm chính".
9. Không thể phân tích âm tiết. Dừng.

Thuật toán 4.14: Phân tích cấu trúc âm tiết

có nhập nhằng giữa âm chính và âm cuối, bao gồm "ui", "uy", "oi". Cuối cùng là trường hợp "qu". Hầu hết các nhập nhằng này được loại bỏ bằng thứ tự áp dụng các mẫu âm tiết (ưu tiên xét các trường hợp không có âm đệm trước). Hai trường hợp cần được xử lý đặc biệt là "uy" và "qu".

Nếu chỉ sử dụng những luật này, sẽ có nhiều tiếng phân tích được, nhưng thực tế không tồn tại. Ví dụ, "cỳ", "kông" ... Tuy nhiên cách phân tích này vẫn chấp nhận được. Những tiếng sai như vậy sẽ được phát hiện và sửa chữa trong phần bắt lỗi non-word.

4.4.4 Hoàn chỉnh lưới từ

Sau khi tạo lưới từ mở rộng, cần bảo đảm có thể tìm được một cách tách từ từ từ đầu tiên cho đến từ cuối cùng và mọi cách tách từ đều kết thúc ở từ cuối cùng trong câu. Nói cách khác, phải đảm bảo luôn tìm được đường đi từ bất kỳ đỉnh nào trên đồ thị đến đỉnh của các từ kết thúc câu. Do từ điển bị giới hạn, có khả năng lưới từ thiếu một vài cạnh làm đồ thị không còn liên thông. Bước này thêm vào các đỉnh để bảo đảm đồ thị liên thông. Lưới từ sẽ được duyệt lại, tìm những nơi không liên thông, thêm vào các đỉnh (mã "UNK") để đảm bảo tính liên thông.

4.4.5 Áp dụng mô hình ngôn ngữ — Tách từ

Phần này sử dụng mô hình ngôn ngữ ngram để đánh giá các cách tách từ, từ đó đưa ra cách tách từ tốt nhất. Dựa theo cấu trúc lưới từ, một cách tách từ chính là đường đi từ nút head đến nút tail. Ta có thể tiến hành việc lượng giá bằng cách duyệt đồ thị theo chiều sâu, tìm mọi cách tách từ. Với mỗi cách tách từ tìm được, mô hình ngôn ngữ sẽ được áp dụng để tính giá trị của cách tách từ đó.

Cách trên đơn giản, nhưng không hiệu quả. Với bài toán tách từ bình thường (không tách từ mờ), số phân nhánh trong đồ thị không nhiều, mô hình này có thể áp dụng được. Tuy nhiên, với các câu dài phương pháp này trở nên không hiệu quả vì số lượng các cách tách từ tăng rất nhanh. Hãy xét một trường hợp cực đoan, một câu dài 86 chữ: "*Đó là trả lời của Bộ Ngoại giao nước ta tại cuộc họp báo thường kỳ ngày hôm qua trước câu hỏi của một số phóng viên nước ngoài về phản ứng của Việt Nam đối với việc Ủy ban về Tự do Tôn giáo Quốc tế của Hoa Kỳ tổ chức điều trần vấn đề tôn giáo Việt Nam và việc một số tổ chức tôn giáo hải ngoại kêu gọi trì hoãn việc phê chuẩn Hiệp định Thương mại song phương với Việt Nam*". Đây là một câu trích từ một tin tức thời sự. Khi đọc lướt qua câu này, ta không cảm nhận được độ dài của nó. Những câu loại này không phải hiếm gặp trong các văn bản hành chính. Câu này có $76101451776 \approx 76 \cdot 10^9$ cách tách từ khác nhau ($842851528992620544 \approx 842 \cdot 10^{15}$ nếu tách từ mờ), dẫn đến việc tính giá trị của từng cách tách từ một là điều không thực tế.

Thật ra, đối với bài toán tách từ thông thường, ta có thể phân tách câu trên thành từng đoạn ngắn hơn, do có một số từ trong câu hoàn toàn không nhập nhằng trong tách từ. Những từ như thế được biểu diễn trên lưới từ là những điểm thắt nút. Nhờ những điểm này, ta có thể tách câu ra thành từng đoạn ngắn hơn và xử lý từng đoạn độc lập. Cách này tùy thuộc vào loại mô hình ngôn ngữ được áp dụng. Với mô hình unigram, có thể áp dụng cách này. Với mô hình bigram hoặc cao hơn, ta cần tìm những nút có điều kiện khắt khe hơn. Cụ thể với bigram, ta cần tìm ra một cặp nút thắt thì vì chỉ cần một nút. Như vậy, việc sử dụng mô hình bigram hay trigram, để có được mô hình ngôn ngữ hiệu quả hơn, sẽ làm giảm đi số phân đoạn trong câu, làm tăng thời gian xử lý.

Bài toán tách từ mờ lại càng khó khăn hơn. Do kết quả của việc phục hồi lỗi, số nút trong lưới từ tăng rất nhiều, hậu quả là các nút có thể dùng để

phân đoạn giảm đáng kể, hầu như không còn. Do vậy cải tiến trên coi nhưng không thể phát huy tác dụng.

Cải tiến trên dùng nguyên tắc quy hoạch động để cải tiến. Điều đó dẫn đến suy nghĩ tại sao không tận dụng quy hoạch động triệt để hơn? Thật sự, ta có thể áp dụng thuật toán Viterbi để giải quyết bài toán này. Thuật toán Viterbi đã được áp dụng thành công trong những bài toán tìm kiếm tương tự trong lĩnh vực nhận dạng tiếng nói.

Thuật toán Viterbi có thể hoạt động, nhưng không tận dụng triệt để ưu điểm của lưới từ. Do mô hình ngôn ngữ được sử dụng là ngram, ta sẽ dùng mô hình bigram. Do mỗi nút đại diện cho một từ, cạnh nối giữa hai từ có thể đại diện cho xác suất bigram $logP(w_2|w_1)$, việc tính mô hình ngôn ngữ cho một cách tách từ:

$$P(w_1 \dots w_n) = P(w_1|head) \prod_{i=2}^{n} P(w_i|w_{i-1}) P(tail|w_n)$$

$$\begin{aligned} logP(w_1 \dots w_n) &= log(P(w_1|head) \prod_{i=2}^{n} P(w_i|w_{i-1}) P(tail|w_n) \\ &= P(w_1|head) + \sum_{i=2}^{n} P(w_i|w_{i-1}) + P(tail|w_n) \end{aligned}$$

Với cách tính này, việc tính $logP(w_1 \dots w_n)$ tương đương với giá trị đường đi từ head đến tail (cạnh head đến nút của từ đầu tiên là $P(w_1|head)$, còn cạnh từ nút của từ cuối cùng đến tail là $P(tail|w_n)$). Việc tính $maxP$ tương đương với $minlogP$, cũng là tìm đướng đi ngắn nhất từ head đến tail. Bài toán cuối cùng quy về bài toán tìm đường đi ngắn nhất trong đồ thị có hướng, có trọng số.

Thuật toán tìm đường đi ngắn nhất được dùng ở đây là thuật toán tìm

kiếm theo độ ưu tiên PFS[2] để tìm đường đi ngắn nhất trên đồ thị. Khoảng cách giữa hai điểm trong đồ thị là xác suất bigram của hai từ. Bài toán tìm đường đi ngắn nhất, có thể áp dụng PFS, Prime, Djisktra. PFS được chọn vì lưới từ, qua khảo sát, có thể được coi là một đồ thị thưa.

Cách trên được áp dụng cho bigram, không thể áp dụng cho trigram do giả định dùng trọng số của cạnh là xác suất bigram, bigram là xác suất giữa hai từ liên tiếp, trong khi trigram là xác suất giữa ba từ liên tiếp. Để áp dụng trigram thay vì bigram, ta sẽ sử dụng lưới 2-từ. Sau đó áp dụng thuật toán PFS cho đồ thị mới vì mỗi nút trong lưới 2-từ thực chất bao gồm 3 từ liên tiếp nhau.

Cách làm này không thể thực hiện với ngram ($n > 3$) vì khi đó số nút/cạnh trong đồ thị sẽ tăng đáng kể. Ngoài ra, PFS cũng không áp dụng được trong trường hợp sử dụng những độ đo như đo số lỗi sai đã xảy ra ... Trong trường hợp đó ta nên sử dụng thuật toán Viterbi.

4.4.6 Tìm lỗi chính tả

Sau khi tách từ xong, việc bắt lỗi chính tả cũng giống như phần bắt lỗi chính tả non-word (4.3.1 ở trang 88).

4.4.7 Lập danh sách từ đề nghị

Việc lập danh sách từ đề nghị tương tự như lập danh sách từ đề nghị khi bắt lỗi non-word (4.3.2 ở trang 88). Tuy nhiên, ta sẽ sử dụng từ điển thay vì dùng từ điển tiếng.

Việc phục hồi lỗi trong bước này là không cần thiết vì đã được thực hiện trong phần tách từ.

[2]Priority-First Search — PFS

4.4.8 Sắp xếp danh sách từ đề nghị

Trong phần tách từ, ta đã xác định được cách tách từ đúng nhất nhờ áp dụng mô hình ngôn ngữ. Áp dụng mô hình ngôn ngữ giúp ta chọn được câu hợp lý nhất trong số các câu có thể hình thành từ lưới từ, việc tìm ra ranh giới từ chỉ là hiệu ứng phụ. Do đã chọn được câu hợp lý nhất nên từ đúng nhất chính là từ nằm trong câu được chọn này. Do đó, từ nằm trên câu được chọn cần được ưu tiên sắp ở đầu danh sách từ đề nghị. Những từ còn lại có thể được sắp xếp như trong phần 4.3.3 ở trang 96.

4.4.9 Các heuristic để cải thiện độ chính xác

Phân biệt từ gốc và từ phát sinh

Với lưới từ hiện có, ta coi từ gốc và những từ phát sinh là như nhau. Tuy nhiên, trên thực tế ta thường ưu tiên chọn từ gốc hơn là các từ phát sinh, do số lỗi trong câu chỉ chiếm một tỉ lệ nhỏ, phần còn lại là các từ đúng. Việc đánh đồng hai loại từ này làm cho chương trình dễ có khuynh hướng sử dụng từ phát sinh không giới hạn chừng nào những từ này còn tạo ra một câu hợp lý. Tiếng Việt có số lượng từ hạn chế hơn so với tiếng Anh (do đặc điểm ngôn ngữ đơn lập, hình thái từ nằm ngoài từ, nên số lượng từ giảm xuống), những câu phát sinh nghe có vẻ hợp lý cũng nhiều hơn. Nếu không khống chế số lượng từ phát sinh được sử dụng thì khả năng chọn nhầm câu đúng sẽ cao hơn.

Vấn đề đầu tiên có thể được giải quyết bằng cách tăng giá trị các cạnh nối đến từ phát sinh một tỉ lệ nhất định để phản ánh sự khác biệt. Ví dụ, trong hình 4.3 ở trang 98, ta có thể biến đổi giá trị các cạnh nối đến từ "xinh" bằng cách tăng 0,2% giá trị của các cạnh nối đến từ "xinh" đó. Vậy, bất cứ cách tách từ nào chứa từ xinh cũng sẽ bị tăng một khoảng giá trị nhất định, so với

các cách tách từ chứa từ "sinh" vẫn giữa nguyên giá trị gốc. Nhờ vậy, từ gốc sẽ được ưu tiên hơn các từ phát sinh.

Cách này nói chung hạn chế việc sử dụng từ phát sinh, tăng khả năng sử dụng từ gốc. Việc điều chỉnh tỉ lệ tăng cho từ phát sinh cần phải được xem xét cẩn thận. Việc tăng quá nhiều sẽ làm cho chương trình bám lấy từ gốc, giảm khả năng phát hiện lỗi. Nhưng nếu không tăng thì chương trình có khả năng tìm nhầm câu đúng, dẫn đến sai lầm dây chuyền trong các bước tiếp theo.

Trên nguyên tắc, độ giảm của các cạnh nên phụ thuộc vào khả năng xuất hiện của các từ phát sinh. Những từ phát sinh có khả năng nhất sẽ ít bị giảm nhất, những từ hiếm gặp nhất sẽ bị giảm nhiều nhất. Việc này sẽ giúp hạn chế lỗi chọn sai từ của mô hình ngôn ngữ. Tuy nhiên các thông số về khả năng xuất hiện của các từ phát sinh đòi hỏi phải có ngữ liệu sai trong thực tế và ngữ liệu đúng tương ứng để rút trích thông tin này. Đây là một hướng phát triển trong tương lai.

Chuẩn hoá độ dài câu

Do mô hình ngôn ngữ được tính toán dựa trên tổng giá trị của các cạnh, nên các câu có ít từ hơn (ít cạnh hơn) thường có xu hướng được chọn vì tổng giá trị sẽ nhỏ hơn. Do đó cần có một cách để chuẩn hoá độ dài câu, loại bỏ sự ảnh hưởng về số lượng từ trong câu lên giá trị của câu.

Việc chuẩn hóa có thể được thực hiện bằng cách lấy giá trị của câu chia cho số lượng từ trong câu. Nếu ta sử dụng Viterbi hay WFST thì chỉ cần chuẩn hoá giá trị mỗi lần duyệt nút kế. Khi áp dụng đồ thị để giải quyết, ta không thể làm như vậy vì giá trị các cạnh được ngầm định là không thay đổi, và công thức tính độ dài đường đi là tổng các cạnh, cũng không thay đổi. Tuy nhiên ta có thể khắc phục bằng cách thay đổi giá trị của các cạnh *chưa*

được xét. Việc thay đổi giá trị các cạnh chưa được xét sẽ không ảnh hưởng đến tính đúng đắn của thuật toán, vì thuật toán tính toán đường đi ngắn nhất dựa vào các cạnh đã xét. Nếu ta không thay đổi giá trị các cạnh đã xét, thuật toán vẫn không đổi.

Giả sử có một đường đi qua n cạnh $e_1, e_2, \ldots, e_n$, giá trị đường đi qua các cạnh đó là $\sum_{i=1}^{n} e_i$. Nếu được chuẩn hóa, giá trị mới sẽ là $\frac{1}{n}\sum_{i=1}^{n} e_i$. Do

$$\frac{1}{n}\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} \frac{1}{n} e_i$$

ta thấy mỗi cạnh cần phải "co" lại còn một khoảng $\frac{1}{n}$ (hay nói cách khác, mất đi $\frac{n-1}{n}$) giá trị thực của chính nó. Các cạnh cứ co lại từ từ mỗi khi thêm vào một cạnh mới. Tuy nhiên, do các cạnh đã xét sẽ không thể thay đổi, nên ta sẽ gom tất cả các khoảng giá trị mất đi này trừ vào cạnh sẽ được xét.

Giả sử đã có n cạnh, ta cần thêm cạnh e_{n+1} vào. Độ dài n cạnh trước đó là N (đã chuẩn hoá). Ta cần tính N' là độ dài $n+1$ cạnh đã chuẩn hoá

$$\begin{aligned}
\Delta &= N' - N \\
&= \frac{Nn + e_{n+1}}{n+1} - N \\
&= \frac{e_{n+1} - N}{n+1} \\
N' &= N + \Delta
\end{aligned}$$

Ta có thể coi cạnh mới, cần được thêm vào là cạnh có giá trị là $\Delta = \frac{e_{n+1}-N}{n+1}$ thay vì e_{n+1} như ban đầu. Tuy nhiên, do Δ có thể âm, cần dùng những thuật toán có thể tìm đường đi trên đồ thị có trọng số âm như Bellman-Ford thay vì dùng PFS. Có thể hình dung đồ thị mới là một loại đồ thị động, các cạnh phía sau càng ngày càng bị thu ngắn lại.

Một vấn đề cần quan tâm là, liệu thuật toán Bellman-Ford có còn đúng với một đồ thị động như vậy hay không? Giả sử ta tính được giá trị cạnh khi dựa trên đường đi ngắn nhất phía trước nó là A1. Sau đó, vì một lí do nào đó, đường đi ngắn nhất tới điểm đó bị đổi sang A2, dẫn đến giá trị cạnh đã tính không còn hợp lệ. Với một đồ thị tổng quát, thuật toán có thể không còn chính xác. Tuy nhiên với đồ thị DAG của lưới từ, điều này vẫn còn đúng. Trong DAG, ta có thể xác định được tập các nút cha của một nút nào đó (những nút nối đến nút đó), tách rời hoàn toàn với phần còn lại của đồ thị. Do đó ta có thể tìm đường đi lớn nhất chỉ trong tập con đó mà thôi. Sau khi tìm được đường đi ngắn nhất trong tập con này, coi như đường đi đã ổn định và không thay đổi nữa, vì vậy việc tính giá trị cạnh kế tiếp sẽ không bị thay đổi sau mỗi lần tính toán. Vì lí do này nên thuật toán sẽ phải duyệt thứ tự các điểm theo thứ tự tôpô chứ không thể duyệt theo một thứ tự ngẫu nhiên.

Từ phát sinh phụ thuộc ngữ cảnh từ gốc

Mô hình hiện tại phát sinh từ cho mọi từ trong câu. Thực tế khi đọc một câu, ta thường coi như mọi từ đều đúng. Ta chỉ "phát sinh từ" ở những chỗ có vấn đề trong câu (thường là do từ không phù hợp với ý nghĩa của câu hoặc của đoạn văn). Ta có thể áp dụng ý tưởng này để biến đổi lưới từ nhằm hạn chế những lỗi tìm nhầm câu đúng.

Cách tiếp cận ở đây là tăng giá trị cạnh nối đến những từ phát sinh từ những từ gốc "có vấn đề". Từ gốc càng có vấn đề thì tỉ lệ tăng càng thấp. Từ gốc càng ít có vấn đề thì tỉ lệ tăng càng cao nhằm giảm khả năng sử dụng từ phát sinh của từ này. Độ đo vấn đề là ngram của liên quan đến các tiếng của từ gốc trong câu. Với bigram, độ đo này là trung bình xác suất của hai bigram nối với tiếng. Với trigram, độ đo này là trung bình xác suất của ba trigram nối với tiếng đó. Độ đo vấn đề của từ là tổng độ đo của các tiếng

thuộc từ đó.

Với cách làm này, các từ phát sinh sẽ không được đánh giá ngang nhau nữa. Những từ phát sinh từ những từ gốc có ít khả năng gặp lỗi sẽ bị tăng với một tỉ lệ nhất định, so với những từ phát sinh từ từ gốc có nhiều khả năng xảy ra lỗi. Việc này giúp giảm khả năng tìm nhầm câu đúng.

4.5 Huấn luyện

Do chương trình cần một số tham số như thống kê tần số xuất hiện của từ, mô hình ngôn ngữ bigram, trigram . . . nên ta cần phải thu thập các tham số này. Đối với tiếng Anh, việc này tương đối dễ dàng. Tuy nhiên, do không thể xác định ranh giới từ tiếng Việt, ta cần phải sử dụng một bộ tách từ trước khi huấn luyện. Chương trình sử dụng một tập ngữ liệu đã tách từ sẵn dựa trên [iiND03]. Tuy nhiên, mô hình ngôn ngữ đòi hỏi ngữ liệu huấn luyện càng lớn càng tốt. Do vậy chương trình tự tạo thêm ngữ liệu huấn luyện thêm ngữ liệu dựa thô chưa tách từ.

Đối với tiếng Việt, đã có một số nghiên cứu về tách từ tiếng Việt, tiêu biểu là [DKT01, TH01]. Chương trình này không sử dụng bộ tách từ hiện có vì các lý do sau:

- Bộ tách từ trong [DKT01, TH01] sử dụng unigram. Trên lý thuyết, bigram/trigram tốt hơn unigram do chứa đựng thông tin về ngữ cảnh.
- Do bộ tách từ thống kê thường phụ thuộc vào ngữ liệu, nếu sử dụng một bộ tách từ trên lĩnh vực khác với lĩnh vực được huấn luyện cho trình bắt lỗi chính tả, kết quả sẽ không như mong đợi.
- Cách tách từ được đề nghị tạo ra mô hình ngôn ngữ tốt hơn so với cách sử dụng bộ tách từ rồi thống kê trên đó (sẽ giải thích chi tiết bên dưới)

4.5.1 Huấn luyện mô hình ngôn ngữ

Như đã nói, tính toán mô hình ngôn ngữ cần phải tách từ. Trong phần bắt lỗi real-word, ta đã thực hiện được phần tách từ mờ. Do dữ liệu huấn luyện được xem là chính xác nên chỉ cần tách từ bình thường. Do đó ta sẽ bỏ qua phần Tạo lưới từ mở rộng, giữ nguyên lưới từ cơ bản, và phần bắt lỗi non-word. Tóm lại, các bước thực hiện gồm khối tiền xử lý, theo sau là các bước Tạo lưới từ và Tách từ. Trong phần tách từ, ta chỉ cần dùng PFS thay vì Bellman-Ford vì lưới từ không có trọng số âm.

Nếu dùng PFS để tách từ, ta chỉ có 1 cách tách từ tốt nhất. Việc đếm từ sẽ dựa trên cách tách từ này. Cách này cho kết quả tương tự như bộ tách từ rời, trừ việc áp dụng bigram/trigram thay cho unigram. Đối với đồ thị có trọng số âm, thuật toán Bellman-Ford sẽ được sử dụng thay cho PFS.

Thay vì chỉ sử dụng một cách tách từ, ta có thể thống kê trên nhiều cách tách từ của cùng một câu. Cách này phản ánh giá trị của các từ chính xác hơn. Ví dụ, ta có 3 cách tách từ với xác suất các cách tách từ tương ứng lần lượt là 0,5, 0,4 và 0,1. Cách chỉ dùng cách tách từ tốt nhất sẽ chỉ tính những từ trong cách tách từ đầu, với giá trị mỗi từ là 1,0. Trường hợp sau, các từ trong cách 2 và 3 vẫn được tính. Số đếm của mỗi từ không còn là 1, mà là xác suất của cách tách từ chứa từ đó. Trở lại ví dụ cũ, các từ trong cách một sẽ được cộng thêm 0,5 thay vì 1. Ngoài ra các từ trong cách 2 và 3 lần lượt được cộng 0,4 và 0,1. Dễ thấy, các từ trong cách tách từ thấp sẽ không tăng số đếm đáng kể, do đó không thể gây ảnh hưởng lớn đến quyết định tách từ. Cách cộng dồn số như vậy được gọi là “fractional count” (hay trong [PSG99] gọi là “soft-count”). Do bộ tách từ rời chỉ trả về cách tách từ tốt nhất, nên ta không thể áp dụng fractional count dựa trên bộ tách từ rời. Đó là một trong những lý do không sử dụng bộ tách từ rời.

Thay vì tính trên mọi cách tách từ (có thể rất lớn nếu câu dài), ta có thể

dùng WFST để tách từ. WFST phải dùng kèm với beam pruning để tránh bùng nổ số tổ hợp các cách tách từ. Sau khi dùng WFST, ta còn n cách tách từ tốt nhất, được dùng để đếm fractional count.

Tuy nhiên, tốt hơn hết vẫn là tính trên tất cả các cách tách từ, nếu có thể. [PSG99] đưa ra một giải pháp (tạm gọi là thuật toán "soft-count") áp dụng quy hoạch động để giải quyết vấn đề này. Soft-count thực tế không kết xuất ra một cách tách từ cụ thể nào (và do vậy nên cũng không thể áp dụng để tìm cách tách từ tốt nhất được!). Thuật toán đếm mọi từ thể có theo cách khác, không cần dựa trên cách tách từ hoàn chỉnh. Thuật toán được mô tả trong [PSG99] không dùng từ điển. Mọi chuỗi con trong câu đều được cho là từ. Điểm này, vừa là điểm mạnh vì không cần dùng từ điển, nhưng cũng là điểm yếu vì yếu tố này làm giảm độ chính xác một cách đáng kể (khoảng 20%, như kết quả trong [PSG99]).

Thuật toán Soft-count

Cho một cây dài n chữ. Có 2^{n-1} cách để tách câu. Giả sử chúng ta biết xác suất p_i của mỗi cách tách từ. Mỗi cách tách từ trong câu sẽ góp phần vào việc đếm từ, dựa trên xác suất của các cách tách từ đó. Ví dụ, với cách tách từ có xác suất p_i, ta tăng số đếm cho mỗi từ trong cách tách từ đó một khoảng $\frac{p_i}{\sum_{j=1}^{2^{n-1}} p_j}$. Cách đếm từ kiểu này được gọi là "soft-count" vì mọi cách tác từ đều được đếm.

Câu cho trước bao gồm các chữ $C_1C_2C_3 \dots C_n$. Mỗi từ $C_{j_1} \dots C_{j_2}$ trong câu sẽ được tăng một khoảng $\frac{S_{j_1}^{left} p(C_{j_1} \dots C_{j_2}) S_{j_2}^{right}}{\alpha}$, trong đó

- $S_{j_1}^{left}$ là tổng xác suất mọi cách tách từ có thể có của chuỗi con từ đầu câu đến C_{j_1}.
- $p(C_{j_1} \dots C_{j_2})$ là xác suất của cách tách từ $C_{j_1} \dots C_{j_2}$ hiện có.

- α là hằng số chuẩn hoá, bằng tổng xác suất mọi cách tách từ của câu — S_{n+1}^{left}.

$S_{j_1}^{left}$ và $S_{j_2}^{right}$ được tính bằng quy hoạch động.

$$S_i^{left} = \begin{cases} 1 & i = 1 \\ p(C_1) & i = 2 \\ \sum_{j=1}^{i-1} p(C_j \dots C_{i-1}) S_j^{left} & 2 < i \leq n+1 \end{cases}$$

$$S_i^{right} = \begin{cases} 1 & i = n \\ p(C_n) & i = n-1 \\ \sum_{j=i+1}^{n} p(C_{i+1} \dots C_j) S_j^{right} & 1 \leq i < n-1 \end{cases}$$

S_i^{left} được tính lần lượt với $i = 1, 2, \dots, n+1$ từ trái sang phải. Sau khi kết thúc, ta được $\alpha = S_{n+1}^{left}$. Sau đó tính S_i^{right} với $i = n, n-1, \dots, 3, 2, 1$ từ phải sang trái, vừa tính S_i^{right} vừa đếm từ.

Độ phức tạp của thuật toán là $O(kIN)$ với k là độ dài tối đa của từ, I là số lần duyệt (khoảng $5 - 10$ lần) và N là kích thước ngữ liệu.

Thuật toán Soft-count áp dụng trên bigram

Như đã nói, thuật toán Soft-count có hai hạn chế là không dùng từ điển, và tính toán dựa trên unigram. Thuật toán cải tiến được đưa ra để khắc phục hai hạn chế này.

Sử dụng từ điển và lưới từ Với hạn chế không dùng từ điển, khắc phục hạn chế này khá đơn giản. Với xác suất $P(C_i \dots C_j)$, nếu từ $C_i \dots C_j$ không tồn tại, ta cho xác suất này về 0. Tuy nhiên, thuật toán sẽ được thay đổi để tận dụng lưới từ khi tính toán, tránh việc tìm tất cả các chuỗi con rồi lại xét xem chuỗi con đó có phải là một từ hay không.

Cho lưới từ của câu S. Gọi $L(W)$ là tập những từ nối đến nút W. Tương tự, $R(W)$ là tập những từ được nối đến từ nút W. Với mỗi nút W trong S, tổng xác suất các cách tách từ có chứa nút W là:

$$P(W) = P^{left}(W)p(W)P^{right}(W)$$

Trong đó:

- $P^{left}(W)$ là tổng xác suất các cách tách từ tính từ đầu câu đến W.
- $P^{right}(W)$ là tổng xác suất các cách tách từ tính từ W đến hết câu.

$$P^{left}(W) = \begin{cases} p(W) & \text{nếu W là nút head} \\ \sum\limits_{W' \in L(W)} p(W')P^{left}(W') & \text{ngược lại} \end{cases}$$

Tương tự

$$P^{right}(W) = \begin{cases} p(W) & \text{nếu W là nút tail} \\ \sum\limits_{W' \in R(W)} p(W')P^{right}(W') & \text{ngược lại} \end{cases}$$

Thuật toán 4.15 ở trang kế tiếp được dùng để tính P^{left}. Thuật toán tương tự được áp dụng để tính P^{right}. Sau khi tính được P^{left} và P^{right}, ta có thể tính fractional count cho các từ trong câu bằng cách duyệt tất cả các nút trong lưới từ, cộng thêm vào $\frac{P(W)}{P^{left}(tail)}$ cho từ C. Thực tế, ta sẽ lồng bước này vào trong thuật toán tính P^{right}, vì thuật toán cũng phải duyệt qua tất cả các từ (thuật toán 4.16 ở trang kế tiếp).

1. Đặt $P^{left}(head) = p(head)$
2. Đặt $P^{left}(W) = 0$ với mọi W còn lại.
3. Duyệt lần lượt các nút W theo thứ tự từ trái sang phải, tính theo vị trí bắt đầu của W trong câu. Với $W' \in R(W)$ cộng thêm $p(W)P^{left}(W)$ vào $P^{left}(W')$

Thuật toán 4.15: Tính P^{left}

1. Đặt $P^{right}(tail) = p(tail)$
2. Đặt $P^{right}(W) = 0$ với mọi nút còn lại.
3. Duyệt tất cả các nút W từ phải sang trái, tính theo vị trí kết thúc của W trong câu.
   (a) Với $W' \in L(W)$, cộng thêm $p(W)P^{right}(W)$ vào $P^{right}(W')$
   (b) Tính fractional count cho W theo công thức $\frac{P^{left}(W)p(W)P^{right}(W)}{P^{left}(tail)}$

Thuật toán 4.16: Tính P^{right}

Áp dụng bigram Để áp dụng bigram thay vì unigram, giải pháp đầu tiên là sử dụng lưới 2-từ thay vì lưới từ. Với lưới 2-từ, mỗi nút tương đương hai từ liên tiếp nhau, do đó $P(C_i \dots C_j)$ tương ứng với $P(W_k|W_{k-1})$ (với W_k là từ thứ k). Tuy nhiên, để áp dụng với trigram, ta lại phải sử dụng lưới 3-từ. Điều này không hiệu quả vì lưới từ cấp càng cao càng to, xử lý kém hiệu quả.

Ta có thể áp dụng trực tiếp lưới từ cơ bản cho thuật toán Soft-count mà không cần lưới 2-từ. Thay vì tính toán các giá trị dựa trên nút, thuật toán sẽ được thay đổi để tính toán các giá trị dựa trên cạnh.

Thay vì vậy, thuật toán được hiệu chỉnh để áp dụng bigram với lưới từ thông thường. Thay vì dùng giá trị nút để tính, ta dùng giá trị cạnh để tính. $p(W)$ sẽ được thay bằng $p(W|W')$ với W và W' là hai từ kề nhau.

$$P(W) = P^{left}(W)P^{right}(W)$$

$$P^{left}(W) = \begin{cases} p(W) & \text{nếu W là nút head} \\ \sum_{W' \in L(W)} p(W/W')P^{left}(W') & \text{ngược lại} \end{cases}$$

Tương tự

$$P^{right}(W) = \begin{cases} p(W) & \text{nếu W là nút tail} \\ \sum_{W' \in R(W)} p(W'/W)P^{right}(W') & \text{ngược lại} \end{cases}$$

$P(W)$ đại diện cho xác suất tất cả các cách tách từ đi qua nút W. Do ta huấn luyện bigram, nên cần xác suất $P(W/W')$ chứ không cần $P(W)$. Nếu W và W_{+1} chỉ có một cạnh, $P(W)$ cũng chính là $P(W/W_{+1})$. Nếu có nhiều

hơn một cạnh, ta cần dùng cách khác để tính $P(W/W')$. Với thuật toán tính P^{right} hiện có, ta cộng dồn $p(W/W')P^{right}$ vào cho $P^{right}(W)$. Vậy trong quá trình cộng dồn ta có thể tính ngay $P(W/W')$ bằng cách nhân giá trị cộng dồn với $P^{left}(W)$. Thuật toán 4.17 và 4.18 là thuật toán cải tiến để tính P^{left} và P^{right}.

1. Đặt $P^{left}(head) = p(head)$
2. Đặt $P^{left}(W) = 0$ với mọi W còn lại.
3. Duyệt lần lượt các nút W theo thứ tự từ trái sang phải, tính theo vị trí bắt đầu của W trong câu. Với mỗi W tìm được, cộng thêm $p(W'/W)P^{left}(W)$ vào $P^{left}(W') \quad W' \in R(W)$

Thuật toán 4.17: Tính P^{left} cho bigram

1. Đặt $P^{right}(tail) = p(tail)$
2. Đặt $P^{right}(W) = 0$ với mọi nút còn lại.
3. Duyệt tất cả các nút W từ phải sang trái, tính theo vị trí kết thúc của W trong câu. Với $W' \in L(W)$:
   (a) Cộng thêm $p(W/W')P^{right}(W)$ vào $P^{right}(W')$
   (b) Tính fractional count cho W' theo công thức $\frac{P^{left}(W')p(W/W')P^{right}(W)}{P^{left}(tail)}$

Thuật toán 4.18: Tính P^{right} cho bigram

Thuật toán Soft-count mới để áp dụng trigram

Với cách tính áp dụng để tính Soft-count với bigram, ta chỉ cần dùng lưới 2-từ thay vì lưới từ để có thể áp dụng trigram (trong khi lẽ ra phải dùng lưới 3-từ để tính trigram, nếu dùng thuật toán gốc).

Chương 5

Cài đặt

Mục lục

Chương trình được cài đặt bằng C++ trên nền Linux. Phần giao diện được cài đặt bằng Gtk+. Phần giao diện này chỉ mang tính chất minh hoạ. Trong thực tế, chương trình nên được tích hợp vào các bộ soạn thảo văn bản như Open Office, AbiWord ... để có thể phát huy tác dụng. Ngoài ra chương trình sử dụng thư viện SRILM [Sto02] để tính toán mô hình ngôn ngữ.

Chương trình gồm hai phần: phần bắt lỗi chính tả và phần huấn luyện. Hai phần này sử dụng khá nhiều thành phần chung. Các thành phần sẽ được trình bày lần lượt bên dưới. Những điểm riêng của trình bắt lỗi chính tả và trình huấn luyện sẽ được trình bày riêng.

5.1 Cấu trúc dữ liệu

5.1.1 Lưu chuỗi

Để tiết kiệm chỗ và giảm chi phí khi so sánh các token và các từ, chương trình lưu toàn bộ các token có thể có vào một mảng chung. Mọi token đều được tham chiếu bằng một con số, tạm gọi là `strid` (chính là số thứ tự token trong mảng). Ngoài ra một bảng băm được dùng để ánh xạ chuỗi sang strid. Cả hai cấu trúc dữ liệu này được cài đặt trong lớp `StringArchive`. Chương trình chỉ sử dụng một đối tượng duy nhất thuộc lớp này, được truy cập thông qua hàm `get_sarch()`. Toán tử `[]`, được cài đặt để chuyển đổi từ strid sang chuỗi và ngược lại.

Từ và tiếng đều được lưu trong StringArchive. Với từ, các khoảng trắng được thay thế bằng dấu gạch chân '_' trước khi lưu. Ngoài ra từ cách nhau bởi nhiều khoảng trắng sẽ được gom lại thành một khoảng trắng. Ví dụ, từ "`học sinh`" sẽ được lưu trong StringArchive dạng "`học_sinh`". Việc chuẩn hoá sẽ được tiến hành trên từng tiếng. Sau khi chuẩn hoá, từ sẽ thành

"5hoc_0sinh".

5.1.2 Từ điển

Từ điển được lưu ở dạng văn bản, mỗi dòng tương ứng với một từ. Trong bộ nhớ, từ điển sử dụng cây đa nhánh (lớp `Node`). `Node` gồm hai lớp dẫn xuất là `BranchNode` (nhánh) và `LeafNode` (lá). Hàm `Node::is_leaf()` được dùng để xác định nút nhánh, nút lá. Trong lớp `BranchNode`, `nodes` sẽ chứa con trỏ đến các nút kế tiếp kèm theo chữ tương ứng được dùng để chuyển qua nút đó. Duyệt từ nút gốc (truy cập thông qua hàm `WordArchive::get_root()`) cho đến hết, ta sẽ được danh sách tiếng hình thành từ. Chữ được dùng để chuyển qua nút lá từ nút nhánh là `<mainleaf>` (với nút lá cơ bản), `<caseleaf>` ...

Để tiết kiệm bộ nhớ, tất cả các danh sách từ khác nhau trong chương trình được trộn chung vào một cây `BranchNode` duy nhất được quản lý bởi `WordArchive`. Các nút lá khác nhau sẽ phản ánh từ trong các danh sách khác nhau. Ví dụ, nút `<mainleaf>` được dùng để lưu từ trong từ điển từ. Trong khi nút `<caseleaf>` được dùng để lưu từ viết hoa của từ viết thường tương ứng (khi duyệt cây bằng từ viết thường, nhưng lá là từ viết hoa).

Lớp `WordArchive` chứa nút gốc của cây và một số hàm linh tinh để thao tác trên cây.

Từ điển tiếng không được lưu trực tiếp vào tập tin mà sẽ được phát sinh tự động khi nạp từ điển từ. Thực ra, hàm `WordArchive`::load()@ khi được gọi sẽ lưu các tiếng vào `StringArchive`, đồng thời đánh dấu tiếng đó thuộc từ điển tiếng. Sau đó, ta có thể xác định một chữ có phải thuộc về từ điển tiếng hay không nhờ hàm `StringArchive::in_dict()`.

5.1.3 Câu

Mỗi câu bao gồm danh sách strid của các token, kèm theo vị trí chuỗi con của token đó trong câu thật sự. Lớp `Sentence` bao gồm một mảng các đối tượng `Sentence::Syllable` (biến `syllables`) đại diện cho các tiếng. Mỗi đối tượng `Sentence::Syllable` bao gồm vị trí bắt đầu của tiếng trong câu `start`, mã strid của tiếng đó `id` và mã strid dùng để so sánh `cid`.

Chuỗi các câu sẽ được hàm `sentences_split()` tách thành những chuỗi con, đại diện cho các câu. Các chuỗi con này sẽ được chuyển cho `Sentence` để phân tích thành từng tiếng.

5.1.4 Lưới từ

Lưới từ bao gồm lưới từ dạng thô `Lattice` và lưới từ `DAG`.

Lưới từ thô `Lattice` không có nút head và tail, cũng không có các liên kết tường minh giữa các nút với nhau. Cấu trúc này lưu thành danh sách các từ theo vị trí bắt đầu của từ. Để biết từ này nối với từ nào, ta sử dụng thông tin về vị trí bắt đầu của từ và độ dài của từ để suy ra vị trí bắt đầu của từ kế tiếp, sau đó lấy danh sách của những từ bắt đầu tại vị trí của từ kế tiếp.

Tất cả các từ của lưới từ được lưu vào danh sách chung (biến `Lattice::we` kiểu `WordEntries`), chứa đựng toàn bộ thông tin liên quan đến một từ. Các từ được tham chiếu bằng số thứ tự của từ trong WordEntries — tạm gọi là WordID. Mỗi từ được lưu trong cấu trúc `WordEntry`. Cấu trúc này bao gồm:

- `len`: cho biết số tiếng của từ.
- `fuzid`: mỗi bit trong biến này cho biết trạng thái fuzzy của tiếng tương ứng trong từ. Nếu bit là 1 nghĩa là tiếng đó được biến đổi, không

phải là tiếng gốc.

- `id`: WordID.
- `node`: liên kết đến từ trong từ điển, dùng để lấy các thông tin chi tiết hơn về từ.

`Lattice` chứa một mảng các đối tượng `WordInfos`. Mỗi đối tượng `WordInfo` tương đương với một vị trí tiếng trong câu. `WordInfos` cho ta biết danh sách các từ bắt đầu từ vị trí tiếng đó, danh sách các từ fuzzy tại tiếng đó, chiều dài từ không bị fuzzy.

Lưới từ `DAG` được dùng để tổng quát hoá các thuật toán hoạt động trên cả lưới từ và lưới 2-từ. So với `Lattice`, lớp này giống một lưới từ hơn.

`DAG` là lớp cơ sở. Lớp dẫn xuất `WordDAG` hoạt động như lưới từ cơ bản, còn lớp `WordDAG2` hoặt động như lưới 2-từ.

5.1.5 Cách tách từ

Cách tách từ đơn giản là một mảng các WordID của những từ hình thành nên cách tách từ đó.

5.1.6 Mô hình ngôn ngữ

Mô hình ngôn ngữ được sử dụng là trigram. Mô hình ngôn ngữ được cài đặt trong thư viện SRILM. Mô hình ngôn ngữ được truy cập thông qua hàm `get_ngram()`.

5.2 Tiền xử lý

5.2.1 Tách token

Chương trình sử dụng flex để tách token. Các quy tắc flex được áp dụng được liệt kê trong hình 5.1 ở trang kế tiếp.

5.2.2 Tách câu

Trước hết, câu được tách thành token. Sau đó, các dấu "`?!()[]:;.,`" sẽ được dùng để tách thành những đoạn nhỏ hơn — những "câu". Hàm `sentences_split()` được dùng để thực hiện điều này.

5.3 Lưới từ

5.3.1 Tạo lưới từ

Tạo lưới từ Lattice

Lưới từ `Lattice` được tạo thông qua hàm `Lattice::construct()`. Thực tế, `construct()` chỉ là vỏ bọc gọi đến ba hàm `Lattice::pre_construct()`, `mark_proper_name()` và `post_construct()`.

`pre_construct()` thực hiện công đoạn đầu tiên để tạo lưới từ: phát sinh tất cả các từ có thể có dựa trên từ điển và so sánh mờ. Một thuật toán tương tự Viterbi được sử dụng để tạo lưới từ. Chương trình lần lượt quét qua các vị trí tiếng, từ một đến hết. Ở mỗi vị trí, chương trình cố gắng nối thêm tiếng mới vào danh sách những từ chưa hoàn chỉnh nếu được. Nếu từ hoàn chỉnh, từ sẽ được đưa vào danh sách (nhưng không bị xóa khỏi danh sách từ

```
SENTENCE_FINAL [.?!]
HYPHEN [\-]
OPEN_SINGLE_QUOTE [\‘]
CLOSE_SINGLE_QUOTE [\’]
RIGHT_PAREN [\"\)\]\}\>\’]

LETTERS_AND_NUMBERS [a-zA-Z0-9]
LETTERS_NUMBER_AND_THEN_SOME [a-zA-Z0-9]
APOSTROPHE \’

SINGLE_CHARACTER [a-zA-Z0-9]

WHITE_SPACE [ \t\n]
NEWLINE [\n]
INVISIBLE [^\040-\176]

{SENTENCE_FINAL}+{RIGHT_PAREN}* |
{HYPHEN}+ |
{OPEN_SINGLE_QUOTE}+ |
{CLOSE_SINGLE_QUOTE}+ |
{LETTERS_NUMBER_AND_THEN_SOME}+{LETTERS_AND_NUMBERS} |
{LETTERS_AND_NUMBERS}+{APOSTROPHE} |

{SINGLE_CHARACTER} { xuất từ }

({WHITE_SPACE}|{INVISIBLE}|{NEWLINE})+ { bỏ qua }
```

Hình 5.1: Quy tắc tách token dùng flex

chưa hoàn chỉnh, vì có thể có những từ dài hơn). Nếu không thể gắn thêm tiếng mới vào từ, từ đó sẽ bị loại bỏ khỏi danh sách từ chưa hoàn chỉnh. Ngoài ra, mỗi khi chuyển sang tiếng mới, tiếng đó cũng được đưa vào danh sách từ chưa hoàn chỉnh như là những tiếng bắt đầu từ mới. Công việc được thực hiện cho đến khi quét hết chiều dài câu (thuật toán 5.1 ở trang kế tiếp).

Cấu trúc `WordState` được dùng để lưu một từ chưa hoàn chỉnh trong danh sách các từ chưa hoàn chỉnh. Cấu trúc này bao gồm vị trí bắt đầu từ, mã `fuzid` của từ, và con trỏ đến nút của tiếng cuối cùng hiện thời của từ trong cây từ điển. Con trỏ này sẽ được `WordState::get_next()` dùng để tìm ra những tiếng kế tiếp giúp hình thành nên từ. Thành phần `fuzid` được cập nhật sau mỗi khi thêm tiếng mới vào từ, cho biết đó là tiếng chính xác trên câu, hay tiếng gần giống với tiếng trên câu (phát sinh nhờ so sánh mờ). Mỗi bit trong `fuzid` tượng trưng cho một tiếng trong từ.

Việc tìm từ dựa trên lớp `WordState` với hai hàm chính là `WordState::get_next()` và `WordState::collect_words()`. Hàm `get_next()` khi được gọi sẽ tạo ra các đối tượng `WordState` mới tương ứng với tiếng kế tiếp được nhận, sau đó tự hủy chính nó. `collect_words` sẽ thu thập tất cả các nút lá có thể có tại vị trí tiếng đang xét và đưa vào lưới từ. Các lớp dẫn xuất khác nhau từ `WordState` có các cách tìm kiếm từ khác nhau. Các lớp này sẽ được đề cập bên dưới.

Mỗi lớp dẫn xuất `WordState` có một lớp dẫn xuất `WordStateFactory` tương ứng, được dùng để tạo ra đối tượng dẫn xuất `WordState`. Hàm `WordStateFactory::create_new()` trong lớp dẫn xuất sẽ tạo ra đối tượng dẫn xuất `WordState` tương ứng với lớp đó. Danh sách các đối tượng `WordStateFactory` được chuyển cho `pre_construct()` nhờ đó `pre_construct()` biết cần phải sử dụng những lớp nào.

$states1$ và $states2$ chứa danh sách các `WordState`, được khởi động là rỗng.

1. Duyệt i từ tiếng đầu tiên đến hết câu.
   (a) Xóa $states2$.
   (b) Thêm state mới (bắt đầu tại vị trí i) vào $states2$ (`WordStateFactory::create_new()`).
   (c) Chuyển state cũ (trong $states1$) sang $states2$ mới (`WordState::get_next()`).
   (d) Tạo từ từ những state trong $states2$ (`WordState::collect_words()`), thêm vào `WordEntry` we. a
   (e) Hoán vị $states1$ và $states2$.
2. Xóa $states1$. Trả về we.

Thuật toán 5.1: Lattice::pre_construct()

Tìm từ chính xác được thực hiện bởi lớp `ExactWordState`. Lớp này sử dụng cid của các tiếng trong câu để tìm, không hề thực hiện các biến đổi nào trong quá trình tìm kiếm. Nút lá được dùng là nút `<mainleaf>`.

Tìm từ không phân biệt hoa thường được thực hiện bởi lớp `LowerWordState`. Lớp này so sánh dựa trên chữ viết thường của các tiếng trong câu. Nút lá được dùng là `<mainleaf>`.

Tìm từ viết hoa bị viết sai được thực hiện bởi lớp `UpperWordState`. Lớp này thực hiện tìm kiếm giống như `LowerWordState`. Tuy nhiên sử dụng nút lá `<caseleaf>` thay vì `<mainleaf>`. Trong quá trình tạo từ điển, các chữ viết hoa sẽ tạo ra các nút `<caseleaf>`, trong đó đường đi đến `<caseleaf>` là chữ thường chữ không phải chữ viết hoa.

Tìm từ phát âm tương tự được thực hiện bởi lớp `FuzzyWordState`. Lớp này thực hiện tìm kiếm dựa trên tất cả các tiếng có phát âm giống với tiếng tương ứng trong câu. Sau đó trả về các nút lá `<mainleaf>`. Việc tìm tiếng giống phát âm dựa trên tập nhẫm lẫn được lấy thông qua hàm `get_confusion_sets`. Xem thêm phần 5.3.4 ở trang 135 để biết cách hoạt động của `FuzzyWordState::get_next()`.

Hàm `post_construct()` thực hiện nốt những gì còn lại để tạo nên lưới từ, bao gồm việc duyệt qua lưới từ, thêm các nút (như UNK) để bảo đảm đồ thị liên thông, đồng thời đưa danh sách toàn bộ các từ đã thu được vào lưới từ. Việc này được thực hiện bằng cách duyệt qua lưới từ, đánh dấu tất cả các tiếng được bao gồm trong các từ đã được tạo. Những tiếng chưa được đánh dấu là những tiếng không nằm trong bất cứ từ nào, do đó sẽ phá hủy tính liên thông của đồ thị. Trong trường hợp xấu nhất, ta sẽ tạo một từ UNK chứa tiếng này.

Lưới từ lưu danh sách từ trong mảng để đảm bảo hiệu suất, tuy nhiên kết quả trả về từ bước `pre_construct()` lại là một tập hợp `std::set` chứ không phải một mảng. Mục đích của việc sử dụng tập hợp là để có thể truy xuất nhanh đến một phần từ trong lưới từ trong quá trình hiệu chỉnh lưới từ. `post_construct()` gọi hàm `construct()` (hàm override, không phải hàm `construct()` ban đầu) để điền các thông tin về `WordInfos` đảm bảo cho lưới từ có thể hoạt động.

Sau khi hoàn tất tạo lưới từ, lưới từ xem như đã ổn định và sẽ không bao giờ thay đổi. Không một hiệu chỉnh nào được phép xảy ra sau khi đã hoàn tất tạo lưới từ. Chính vì vậy, việc tạo lưới từ được tách làm hai phần để tạo cơ hội can thiệp vào lưới từ ngay trong giai đoạn hình thành để có thể thêm các nút khác (phát sinh trong bước phục hồi lỗi). Như đã nói, `construct()` gọi đến hàm `mark_propername()` hàm này tạo thêm những nút $PROP$ đại diện cho tên riêng. Các hàm khác có thể được gọi ở vị trí của hàm này nếu cần thay đổi lưới từ nhiều hơn.

Tạo lưới từ DAG

Lưới từ `DAG` được hình thành dựa trên lưới từ `Lattice`. Có thể coi đây là một vỏ bọc để dễ truy cập đến lưới từ, thêm vào những nút head, tail ... cho hoàn chỉnh. Các hàm trong `DAG` chỉ là vỏ bọc cho các hàm của `Lattice`. Hai nút giả head, tail được tạo ra và thêm vào lưới từ. Đó là trường hợp của lưới từ cơ bản `WordDAG`.

Với lưới 2-từ `WordDAG2`, công việc phức tạp hơn. Thuật toán mở rộng lưới từ thành lưới 2-từ được cài đặt trong constructor của `WordDAG2`. Do mỗi nút của `WordDAG2` bao gồm hai nút của lưới từ cơ bản, ta cần lưu trữ mã các nút này trong `WordDAG2::Node` (các biến `n1` và `n2`). Ngoài ra còn có biến `id` là mã nút của lưới 2-từ. Tất cả các nút của lưới 2-từ được lưu trong biến

thành phần `WordDAG2::nodes`. Hàm `WordDAG2::demange()` được dùng để chuyển mã từ mã nút trong lưới 2-từ thành mã nút trong lưới từ `WordDAG`.

5.3.2 Bổ sung lưới từ

Bổ sung lưới từ dựa theo lỗi phát âm được thực hiện ngay trong `pre_construct()` thông qua lớp `FuzzyWordState` (xem thêm phần 5.3.4 ở trang 135).

Lớp `PenaltyDAG` được dẫn xuất từ lớp `DAG`. Lớp này điều chỉnh lại hàm `DAG::edge_value()` để thay đổi độ dài cạnh nhằm tạo sự khác biệt giữa các cạnh phát sinh và cạnh gốc. Lớp nhận một đối tượng `DAG` khác, hoạt động như là một vỏ bọc cho đối tượng này.

Lớp sử dụng thông tin `WordEntry::fuzid` để biết có bao nhiêu tiếng trong từ là tiếng phát sinh, sau đó giảm giá trị của cạnh theo một tỉ lệ tương ứng. Nếu có n tiếng sai, giá trị cạnh sẽ bị mất đi một khoảng nwV với V là giá trị ban đầu, w là trọng số.

Giá trị w được sử dụng trong chương trình minh hoạ là 0.05.

5.3.3 Tìm cách tách từ tốt nhất

Thuật toán 5.2 ở trang kế tiếp trình bày cài đặt PFS. Thuật toán 5.3 ở trang 134 trình bày cài đặt Bellman-Ford kèm chuẩn hoá.

$last[head] \leftarrow head$
$val[head] \leftarrow 0$
$seen[head] \leftarrow true$
$candidates \leftarrow \{head\}$(candidates là một heap)

1. Nếu $candidates$ rỗng thì kết thúc thuật toán.
2. Lấy một đỉnh v ra khỏi $candidates$. $next\_nodes$ là tập điểm nối từ v đến.
3. Duyệt từng đỉnh vv trong $next\_nodes$.
   (a) Nếu $seen[vv]$ là $false$.
      i. Thêm vv vào $candidates$.
      ii. $seen[vv] \leftarrow true$
      iii. $val[vv] \leftarrow val[v] + edge\_value(v, vv)$
      iv. $last[vv] = v$
   (b) Nếu $seen[vv]$ là $true$ và $val[vv] > val[v] + edge\_value(v, vv)$
      i. $val[vv] \leftarrow val[v] + edge\_value(v, vv)$
      ii. $last[vv] \leftarrow v$
      iii. Sắp xếp lại $candidates$.
4. Về bước 1.

Lấy danh sách các nút.

1. $v \leftarrow tail$.
2. Lưu v.
3. $v \leftarrow last[v]$
4. Nếu $v \neq last[v]$, lặp lại bước 2.
5. Lưu v.
6. Đảo ngược danh sách các nút đã lưu.

Thuật toán 5.2: Thuật toán tìm đường PFS

$$last[head] \leftarrow head$$
$$length[head] \leftarrow 0$$
$$node\_count[head] \leftarrow 1$$
$$nexts \leftarrow \{head\}$$

Lập danh sách cạnh.

1. $i \leftarrow 0$.
2. $l \leftarrow size\_of(nexts)$
3. Nếu $i >= l$, dừng.
4. $v \leftarrow nexts[i]$. $i \leftarrow i + 1$
5. Nếu $done[v]$ là $true$ thì quay về bước 2. Nếu không thì $done[v] \leftarrow true$
6. Thêm các nút kề v vào phía sau $nexts$.
7. Lặp $ii = l \ldots size\_of(nexts)$, thêm các cạnh $(v, nexts[ii])$ vào $edges$
8. Quay về 2.

Tìm đường

1. $cont \leftarrow true$. Duyệt lần lượt từ 0 đến $n-1$ với điều kiện $cont$ vẫn còn là $true$.
2. $cont \leftarrow false$. Duyệt lần lượt các cạnh trong $edges$, đặt đỉnh đầu, đỉnh cuối lần lượt là i và v, bỏ qua các cạnh có $node\_count[i] = 0$.
3. Nếu $node\_count[v] \neq 0$ và $length[v] > length[i] + \frac{edge\_value(i,v) - length[i]}{node\_count[i]+1}$
   (a) $length[v] \leftarrow length[i] + \frac{edge\_value(i,v) - length[i]}{node\_count[i]+1}$
   (b) $last[v] \leftarrow i$
   (c) $node\_count[v] \leftarrow node\_count[i] + 1$
   (d) $cont \leftarrow true$

Lấy danh sách các nút tương tự như thuật toán 5.2 ở trang trước.

Thuật toán 5.3: Thuật toán tìm đường Bellman-Ford kèm chuẩn hoá

5.3.4 Lỗi phát âm

Phân tích cấu trúc âm tiết

Lớp `Syllable` được dùng để phân tích cú pháp âm tiết. Năm thành phần của âm tiết (âm đầu, âm đệm, âm chính, âm cuối, thanh điệu) được lưu trong mảng `components`. Hàm `Syllable::parse()` nhận một chuỗi của một âm tiết, sau đó tách âm tiết này ra và điền thông tin vào `Syllable::components`.

Tìm các tiếng phát âm tương tự

Việc tìm các tiếng tương tự cách phát âm được cài đặt trong lớp `Syllable` và hàm `FuzzyWordState::get_next()`. Lớp `Syllable` sẽ thực hiện các thao tác cơ bản liên quan đến âm tiết, bao gồm hai hàm `match` và `apply`.

`Syllable::match` so sánh âm tiết với một mẫu âm tiết. Một mẫu âm tiết bao gồm năm thành phần như âm tiết. Mỗi thành phần xác định những giá trị được phép đối với thành phần đó. Các thành phần có thể cho phép một giá trị, hoặc không quan tâm đến giá trị của nó, hoặc không được tồn tại thành phần đó, hoặc phải có thành phần đó. Các thành phần được liệt kê lần lượt theo thứ tự: âm đầu, âm đệm, âm chính, âm cuối, thanh điệu. Ví dụ, `[?,?,a,i,?]` sẽ khớp với bất cứ âm tiết nào có âm chính là 'a' và âm cuối là 'i'. `[?,?,?,?,Hook]` sẽ khớp với bất cứ âm nào dùng thanh hỏi.

Các mẫu âm tiết này được tập hợp lại để tạo nên tập nhầm nhằng. Một tập nhập nhằng bao gồm nhiều mẫu âm tiết. Ví dụ, ta có tập nhập nhằng `[?,?,ư,t,?]` và `[?,?,ư,c,?]` thể hiện sự nhập nhằng giữa những âm tiết "ưc" và "ưt". Tập nhập nhằng có kiểu là `confusion_set`. Tất cả các tập nhập nhằng được lưu vào một mảng, được truy cập thông qua hàm

`get_confusion_sets()`.

Hàm `Syllable::apply()` dùng để áp mẫu vào âm tiết đã có. Các thành phần tường minh trong mẫu sẽ ghi đè các thành phần tương ứng trong âm tiết. Nhưng thành phần quy định không tồn tại trong mẫu sẽ bị loại bỏ khỏi âm tiết. Những thành phần không được quy định trong mẫu sẽ được giữ nguyên. Các thành phần quy định tồn tại sẽ tạo ra một tập các âm tiết khác nhau, mỗi âm tiết tương ứng với một giá trị có thể có của thành phần đó.

Sử dụng hai hàm trên, hàm `FuzzyWordState::get_next()` làm phần việc còn lại là gắn kết `match()`, `apply()` và tập nhầm lẫn để tìm ra các âm tiết phát âm gần giống. Hàm lần lượt duyệt qua các mẫu âm tiết trong từng tập nhầm lẫn. Nếu âm tiết đang xét khớp với một mẫu âm tiết trong tập nhầm lẫn (`match()`), các âm tiết còn lại của tập nhầm lẫn sẽ được áp vào âm tiết đang xét (`apply()`) để tạo ra các âm tương tự. Âm tiết mới sẽ lại được duyệt qua tập nhầm lẫn như âm tiết đã xét để tạo ra thêm các âm tiết tương tự khác. Quá trình này sẽ dừng lại khi không có âm tiết mới nào được phát sinh. Sau đó, các âm tiết sẽ được so sánh với từ điển tiếng để lọc bỏ những âm tiết không đúng.

5.3.5 Danh từ riêng

Một danh sách 8000 tên riêng được sử dụng để tạo một tập các chữ thường hình thành tên riêng. Những chữ liên tiếp nhau nằm trong danh sách này được coi là một tên riêng. Hàm `mark_proper_name()` bổ sung những nút $PROP$ vào lưới từ cho những tên riêng được tìm ra.

Những từ viết hoa sai chính tả được bổ sung thêm vào lưới từ dựa vào lớp `UpperWordState`.

5.3.6 Lỗi bàn phím

Lỗi bàn phím được cài đặt thông qua các lớp `KeyRecover`, `CharInserter`, `CharEraser`, `CharTransposer` và hàm `im_recover()`.

Hàm `im_recover()`, gọi đến hai hàm `vni_recover()` và `telex_recover()` dùng để phục hồi phím từ hai kiểu gõ trên.

Các hàm này được sử dụng trong hàm `get_syllable_candidates()`. Các phần xử lý bàn phím khác cũng được xử lý trong hàm này.

5.4 Bắt lỗi chính tả

Trình bắt lỗi chính tả bao gồm hai lớp chính là lớp `Text`. Lớp `Text` đại diện cho một đoạn văn bản trong toàn bộ văn bản. Lớp này thực hiện toàn bộ công đoạn tìm lỗi chính tả. Các chuỗi được sử dụng để giao tiếp giữa lớp này và các lớp khác là UTF-8.

Các biến quan trọng trong lớp `Text`:

- `vspell` trỏ đến đối tượng `VSpell` quản lý đối tượng này.
- `offset`,`length`: vị trí và độ dài của đoạn văn bản do đối tượng này xử lý, xét trong toàn bộ văn bản của `vspell`.

Trong lớp `Text`, hàm `sentence_check()` tiến hành kiểm tra chính tả một câu. Hàm này thực hiện hầu như toàn bộ công đoạn bắt lỗi chính tả, mô tả trong thuật toán 5.4 ở trang kế tiếp. Hàm `syllable_check()` kiểm lỗi tiếng (thuật toán 5.5 ở trang 139). Hàm `word_check()` kiểm lỗi từ (thuật toán 5.6 ở trang 139).

1. Tách token.
2. Chuẩn hoá các tiếng.
3. Gọi `syllable_check()` để kiểm tra. Nếu có lỗi tiếng, gọi `ui_syllable_check()` để yêu cầu người dùng sửa lỗi, sau đó kết thúc (trả về `false` — vẫn còn lỗi).
4. Xây dựng lưới từ.
5. Bổ sung lưới từ:
   - Đánh dấu tên riêng.
   - Áp dụng separator.
6. Xây dựng `DAG` tương ứng (tùy theo các thông số về trigram, penalty . . .).
7. Tìm cách tách từ tốt nhất.
   - Nếu có chuẩn hoá, sử dụng Bellman-Ford.
   - Nếu không có chuẩn hoá, sử dụng PFS.
8. Gọi `word_check()` để kiểm tra. Nếu có lỗi, gọi `ui_word_check()` để yêu cầu người dùng sửa lỗi, sau đó kết thúc (trả về `false` — vẫn còn lỗi).
9. Kết thúc, trả về `true` — sạch lỗi.

Thuật toán 5.4: Text::sentence_check()

1. Duyệt lần lượt từng tiếng trong câu. Thực hiện:
   (a) Gọi `VSpell::in_dict()`, kiểm tra id xem tiếng đó có chưa. Nếu đã có, kết thúc (trả về `true` — không có lỗi).
   (b) Gọi `StringArchive::in_dict()`, kiểm tra xem cid của tiếng có trong từ điển tiếng không. Nếu có, phân tích tiếng, lấy dạng chữ thường, đúng quy cách bỏ dấu của tiếng. Kiểm tra xem tiếng mới có giống với tiếng cũ hay không (lỗi bỏ dấu sai vị trí). Nếu có, kết thúc (trả về `true` — không có lỗi).
   (c) Trả về `false` — lỗi.
2. Nếu trả về `false`, đưa tiếng vào danh sách tiếng gặp lỗi. Tiếp tục duyệt các tiếng còn lại.
3. Trả về `true` nếu danh sách các tiếng gặp lỗi là rỗng.

Thuật toán 5.5: Text::syllable_check()

1. Duyệt lần lượt các từ trong cách tách từ đã có.
2. Lấy danh sách id của các tiếng của từ — `sylls2`.
3. Lấy danh sách cid của các tiếng của từ — `sylls`.
4. Nếu `sylls` không trùng khớp với các tiếng trong câu (dùng cid) và `sylls2` không có trong từ điển riêng của `VSpell`, lưu từ vào danh sách từ gặp lỗi.
5. Trả về `true` nếu danh sách các từ gặp lỗi là rỗng.

Thuật toán 5.6: Text::word_check()

Ta thấy `Text` là một lớp abstract. Một số hàm như `ui_syllable_check()`, `ui_word_check()` cần phải được cài đặt trong lớp dẫn xuất. Đây là những hàm giao tiếp với người dùng. Tùy môi trường sử dụng mà các hàm này được cài đặt cho thích hợp. Dù cài đặt theo cách nào, nhiệm vụ của những hàm này vẫn như nhau: yêu cầu người dùng đưa ra quyết định. Sau khi lập danh sách từ đề nghị, `Text` sẽ gọi các hàm này. Các hàm này phải liệt kê danh sách từ đề nghị và chờ người dùng quyết định chọn từ nào.

Lớp `VSpell` là lớp chính, đại diện cho mô hình bắt lỗi chính tả, được dùng để giao tiếp với chương trình sử dụng bắt lỗi chính tả. Lớp này sử dụng `Text` để tiến hành bắt lỗi chính tả.

Lớp `VSpell` quản lý một từ điển riêng của chính nó, được bổ sung bởi người dùng. Do chương trình không thể đạt được một từ điển hoàn chỉnh, từ điển này được dùng để bổ sung cho từ điển đã có. Những ký hiệu, các từ nước ngoài ... không được nhận diện bởi chương trình cũng sẽ được thêm vào từ điển này sau khi người dùng đã kiểm tra những từ đó và xác định rằng những từ đó là chính xác. `VSpell` bao gồm hai từ điển: từ điển từ (`VSpell:words`) và từ điển tiếng (`VSpell::syllables`).

Ngoài ra `VSpell` còn duy trì một danh sách các separator (`VSpell::sepaparators`). Hàm `check()` tiến hành kiểm lỗi chính tả cho một đoạn văn bản (thuật toán 5.7 ở trang kế tiếp).

Lớp `VSpell` không sử dụng trực tiếp `Text` mà sử dụng lớp dẫn xuất từ `Text`. Để tạo ra một đối tượng `Text`, `VSpell` dựa vào đối tượng thuộc lớp `TextFactory`. Đối tượng này có nhiệm vụ tạo ra đối tượng `Text` cho `VSpell`.

1. Văn bản UTF-8 được lưu vào `utf8_text`.
2. Lặp
   (a) Chuyển văn bản từ UTF-8 sang VISCII, dùng `viet_to_viscii_force`.
   (b) Tách câu (`sentences_split()`).
   (c) Duyệt từng câu.
      i. Tạo đối tượng `Text` mới (`TextFactory::create()`).
      ii. Đặt `offset`,`length` cho đối tượng `Text`.
      iii. Gọi `Text::sentence_check()`.
      iv. Nếu trả về `false`, ngừng duyệt các câu còn lại.
   (d) Nếu `sentence_check()` trả về `true` với mọi câu, dừng lặp.
3. Trả về `true`.

Thuật toán 5.7: VSpell::check()

5.4.1 Separator

Phần trên có đề cập đến "separator". Separator tồn tại vì bộ tách từ của chương trình không thể nào hoàn chỉnh. Separator giúp hỗ trợ trình tách từ tách đúng hơn. Separator là những dấu hiệu, chỉ ra rằng một vị trí nào đó trong câu là ranh giới từ, không thể tồn tại một từ đi ngang qua vị trí đó. Ví dụ, chương trình tách từ "ông quan tài thật đấy". Ta sẽ chen một separator vào giữa "quan" và "tài". Nhờ vậy, chương trình sẽ chọn ra một cách tách từ khác "ông quan tài thật đấy".

Separator được sử dụng trong phần hiệu chỉnh lưới từ. Sau khi đã hoàn tất tạo lưới từ, thuật toán 5.8 sẽ duyệt qua toàn bộ lưới từ, loại bỏ bất cứ từ nào vi phạm separator.

1. Sắp xếp danh sách separator theo thứ tự xuất hiện.
2. Duyệt lần lượt từng separator, bắt đầu từ 0.
   (a) Duyệt lần lượt tất cả các từ trong lưới từ.
   (b) Loại bỏ những từ cắt ngang qua separator đang xét.

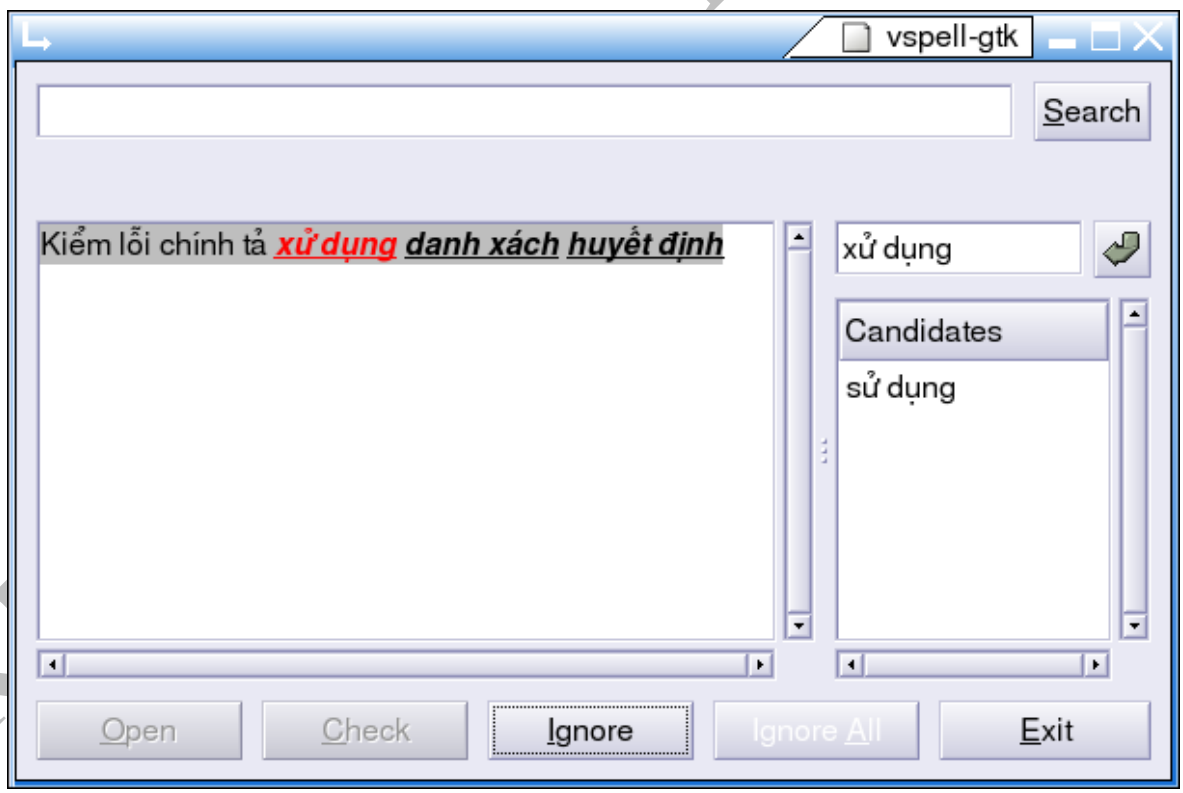
Thuật toán 5.8: Text::apply_separators()

5.4.2 vspell-gtk

Như đã nói, các lớp trong `VSpell`, `Text` chỉ tập trung vào phần xử lý văn bản. Phần giao tiếp được tách riêng (nằm trong hai hàm `Text::ui_word_check()` và `Text::ui_syllable_check()`). Chương trình vspell-gtk được tạo ra nhằm mục đích minh hoạ cách cài đặt hai hàm này. Chương trình là một

bộ bắt lỗi chính tả độc lập. Tuy nhiên, để chương trình thực sự hữu dụng, trình bắt lỗi chính tả nên được tích hợp với các chương trình liên quan đến xử lý văn bản như AbiWord, OpenOffice, GEdit, Evolution ...

Chương trình có giao diện như trong hình 5.2. Sau khi nhập liệu, ta nhấn nút "Check". Chương trình kiểm lỗi hai giai đoạn: kiểm lỗi tiếng và kiểm lỗi từ. Phần được kiểm tra sẽ được tô sẫm. Các từ nhiều tiếng được gạch dưới. Những từ/tiếng bị sai được tô đậm. Từ đang được sửa sẽ được tô đỏ. Nhấn "Ignore" sẽ bỏ qua lỗi hiện tại, chuyển sang lỗi kế, lần lượt xoay vòng lỗi. Nhấn "Ignore all" sẽ bỏ qua hoàn toàn một giai đoạn kiểm lỗi (tiếng hoặc từ), chuyển qua giai đoạn kế (hoặc kết thúc quá trình kiểm tra). Để sửa lỗi, ta chọn từ trong khung "Candidates" hoặc tự nhập từ mới, sau đó nhấn nút "Enter" bên cạnh.



Hình 5.2: Giao diện vspell-gtk

vspell-gtk tạo lớp `MyText` dẫn xuất từ `Text`. Các hàm `ui_syllable_check()` và `ui_word_check()` được cài đặt như trong thuật toán 5.9 và 5.10 ở trang kế tiếp. `MyText` có thêm vài hàm phụ để hiển thị như `show_wrong_syllables()`, `show_wrong_words()` và `show_words()`.

1. Duyệt lần lượt từng tiếng bị sai trong danh sách tiếng sai.
2. Hiện tiếng sai dùng hàm `show_wrong_syllables()`.
3. Lập danh sách tiếng đề nghị dùng `get_syllable_candidates()`, hiện thị danh sách tiếng lên màn hình.
4. Đặt `processed`, `ignore_all`, `ignore` là `false`.
5. Lặp liên tục (nhường quyền xử lý cho Gtk+) cho đến khi biến `ignore` hoặc `ignore_all` hoặc `processed` là `true`. (Khi các nút tương ứng được nhấn, giá trị các biến này sẽ bị thay đổi).
6. Nếu `ignore` là `true`, quay lại từ đầu (chuyển qua từ bị sai kế tiếp).
7. Nếu `ignore_all` là `true`, kết thúc (trả về `true` — "sạch lỗi" do người dùng không muốn kiểm tra lỗi tiếng nữa).
8. Nếu `processed` là `true`.
   (a) Lấy tiếng do người dùng chọn (hoặc nhập mới).
   (b) Thay thế tiếng bị sai trong câu bằng tiếng của người dùng.
   (c) Thêm tiếng này vào từ điển tiếng của `VSpell`.
   (d) Kết thúc, trả về `false` — còn lỗi.

Thuật toán 5.9: MyText::ui_syllable_check()

1. Duyệt lần lượt từng từ bị sai trong danh sách từ sai.
2. Hiện từ sai dùng hàm `show_wrong_words()`.
3. Lập danh sách từ đề nghị dùng từ được dùng trong cách tách từ đùng khi áp dụng mô hình ngôn ngữ.
4. Đặt `processed`, `ignore_all`, `ignore` là `false`.
5. Lặp liên tục (nhường quyền xử lý cho Gtk+) cho đến khi biến `ignore` hoặc `ignore_all` hoặc `processed` là `true`. (Khi các nút tương ứng được nhấn, giá trị các biến này sẽ bị thay đổi).
6. Nếu `ignore` là `true`, quay lại từ đầu (chuyển qua từ bị sai kế tiếp).
7. Nếu `ignore_all` là `true`, kết thúc (trả về `true` — "sạch lỗi" do người dùng không muốn kiểm tra lỗi từ nữa).
8. Nếu `processed` là `true`.
   (a) Lấy từ do người dùng chọn (hoặc nhập mới).
   (b) Tìm ký tự '`|`'. Các ký tự này đại diện cho separator. Nếu có, thêm những separator này vào `VSpell`, kết thúc — trả về `false`.
   (c) Thay thế từ bị sai trong câu bằng từ của người dùng.
   (d) Thêm từ này vào từ điển từ của `VSpell`.
   (e) Kết thúc, trả về `false` — còn lỗi.

Thuật toán 5.10: MyText::ui_word_check()

5.5 Huấn luyện

5.5.1 Dữ liệu huấn luyện

Dữ liệu gốc bao gồm: danh sách từ và các văn bản tiếng Việc thu thập từ mạng VnExpress[1].

Nhiệm vụ:

- Tạo danh sách tiếng, chuẩn hoá danh sách tiếng.
- Tiền xử lý ngữ liệu.
- Huấn luyện ngram, tạo ra dữ liệu ngram.

5.5.2 Dữ liệu nguồn

Dữ liệu nguồn bao gồm danh sách từ và ngữ liệu tiếng Việt. Dữ liệu được lưu dưới dạng mã VISCII.

Danh sách từ

Đây là một tập tin văn bản lưu tất cả các từ được dùng trong tiếng Việt, mỗi từ một dòng. Các chữ cách bởi một khoảng trắng.

Ngữ liệu huấn luyện

Bao gồm một loạt các tập tin văn bản tiếng Việt. Mỗi dòng là một đoạn văn bản.

[1] http://www.vnexpress.net

5.5.3 Tiền xử lý ngữ liệu huấn luyện

Dữ liệu thu được từ VnExpress bao gồm một loạt các file html, chia thành nhiều thư mục.

htmltidy được sử dụng để chuyển dữ liệu html về dạng xhtml. Các file html của VnExpress không hoàn toàn tuân theo chuẩn html, bao gồm hai trường hợp:

- Sử dụng tag <vne>
- Không đóng tag <frame>

Hai lỗi này được tự động phát hiện và sửa chữa trước khi chuyển dữ liệu cho htmltidy.

Sau khi chạy htmltidy, ta được dữ liệu xhtml hợp khuôn dạng. Phân tích các tài liệu này cho thấy phần nội dung chính thường nằm trong tag <vne>, hoặc trong tag <table> với thuộc tính id là "CContainer". XSLT được sử dụng để lọc ra phần nội dung nằm giữa hai tag này (`z.xslt`).

Trình `convert.sh` sẽ thực hiện bước htmltidy và xslt ở trên. Cách sử dụng:

```
convert.sh < input  > output
```

Sau bước này, ta cần lọc bỏ các tag, loại bỏ các entity ... để đưa dữ liệu về dạng văn bản thuần túy. Do số lượng tag được định nghĩa html rất nhiều nên chỉ xử lý những tag nào được sử dụng trong VnExpress (thông qua chương trình `gettags.pl`). Các tag này được loại bỏ bằng `html2text.pl`. Các mã utf-8 được sử dụng trong VnExpress (thông qua chương trình `z.c`) được chuyển về mã VISCII qua `z.cpp`. Các entity còn lại (lấy bằng `list-entity.pl`) được xử lý trong `html2text.pl`.

`html2text.pl` sử dụng các tag để tách dữ liệu thành các đoạn. Dữ liệu sau cùng là các tập tin văn bản, mỗi dòng là một đoạn.

5.5.4 Huấn luyện dữ liệu

Chương trình `wfst-train` được dùng để huấn luyện dữ liệu. Thông tin nhập là danh sách từ, danh sách tiếng, ngram (nếu có). Thông tin xuất là ngram mới.

Quy trình xử lý của trình huấn luyện được nêu trong thuật toán 5.11. Chương trình được lặp đi lặp lại nhiều lần.

1. Đọc từng dòng.
2. Tách câu.
3. Tácк token.
4. Xây dựng lưới từ (`Lattice::construct()`).
5. Tìm cách tách từ tốt nhất.
6. Thống kê ngram cho câu.
7. Xử lý câu kế tiếp.

Thuật toán 5.11: Quy trình huấn luyện

5.6 Linh tinh

5.6.1 Xử lý bảng mã

Chương trình giao tiếp bằng Unicode UTF-8. Tuy nhiên, để tiện việc xử lý, nội bộ chương trình sử dụng mã VISCII. Do VISCII không thể bao quát hết Unicode, những ký tự Unicode không có trong VISCII sẽ được thay bằng một ký tự đặc biệt (coi như là ký tự không biết). Sau khi xử lý VISCII xong,

chuỗi so sánh thật sự sẽ là chuỗi Unicode. Hầu hết các lớp sử dụng VISCII. Hai lớp sử dụng cả UTF-8 và VISCII là `VSpell` và `Text`.

5.6.2 So sánh chuỗi

Do mọi chuỗi đều được lưu thành strid. Việc so sánh chuỗi không phân biệt hoa thường là không thể thực hiện được. Do vậy, ngoài strid "id" tham chiếu đến chuỗi gốc, ta sử dụng thêm "cid" tham chiếu đến chuỗi được dùng để so sánh chuỗi. "cid" mặc định trong câu tham chiếu đến chuỗi đã chuẩn hoá. Nếu cần, "cid" sẽ tham chiếu đến chuỗi chuẩn hoá, viết thường.

Khi cần thiết phải so sánh chuỗi thật sự, chương trình sẽ dùng "id" để truy ngược đến chuỗi gốc, sau đó thực hiện phép so sánh chuỗi thông thường.

5.6.3 Xử lý tiếng Việt

Một số hàm được tạo ra để hỗ trợ việc xử lý tiếng Việt, bao gồm:

- `viet_toupper`, `viet_tolower`, `viet_isupper`, `viet_islower`, `viet_isalpha`, `viet_isdigit`, `viet_isxdigit`, `viet_isspace`, `viet_ispunct`: Các hàm tiếng Việt thay thế cho các hàm gốc (toupper, tolower ...)
- `viet_utf8_to_viscii()`: Chuyển từ UTF-8 sang VISCII. Trả về `false` nếu không thể chuyển đổi.
- `viet_utf8_to_viscii()`: Chuyển từ UTF-8 sang VISCII. Những ký tự không thể chuyển được sẽ được thay bằng 'z'.
- `viet_viscii_to_utf8()`: Chuyển từ VISCII sang UTF-8.

Chương 6

Đánh giá và kết luận

Mục lục

6.1 Tóm tắt

Sau đây là tóm tắt các đặc điểm mô hình được đề nghị:

- Áp dụng mô hình tách từ mờ dựa trên ngram (bigram và trigram).
- Cấu trúc dữ liệu dựa trên lưới từ — một dạng đồ thị có hướng không chu trình. Nhờ đó có thể áp dụng thuật toán tìm đường trên đồ thị có hướng để tìm ra cách tốt từ tốt nhất.
- Sử dụng lưới 2-từ cho trigram, các thuật toán làm việc trên bigram sẽ vẫn làm việc với trigram thông qua lưới 2-từ.
- Đồ thị được hiệu chỉnh để tạo ra sự khác biệt giữa từ gốc và các từ phát sinh bằng cách điều chỉnh giá trị các cạnh nối đến từ phát sinh.
- Loại bỏ khả năng cách tách từ chứa ít từ hơn có khả năng được chọn cao hơn cách tách từ có nhiều từ hơn bằng cách chuẩn hoá các cách tách từ thông qua lưới từ động.
- Áp dụng đảo ngược lỗi để phục hồi lỗi phát âm và lỗi bàn phím. Lỗi bàn phím bao gồm lỗi gõ thiếu phím, dư phím, nhầm phím, nhầm thứ tự hai lần gõ. Ngoài ra còn có lỗi do bộ gõ như VNI và TELEX.
- Sử dụng soft-count để huấn luyện ngram nhằm tạo ra mô hình ngôn ngữ tốt hơn.

6.2 Thử nghiệm

Chương trình được huấn luyện dựa trên ngữ liệu từ các bài báo của mạng VnExpress.net. Ngữ liệu sử dụng có kích thước 12283822 byte, bao gồm

2602105 chữ, khoảng 379000 câu[1]. Trình huấn luyện chạy trong vòng 18 phút, trung bình khoảng 350 câu một giây. Dữ liệu xuất bao gồm 76682 unigram, 561796 bigram, 1729882 trigram với kích thước tổng cộng 71084052 byte. Mô hình ngôn ngữ ngram-tiếng bao gồm 76224 unigram và 324326 bigram (8489570 byte). Dữ liệu tên riêng bao gồm 790 tiếng tạo thành tên người, rút trích từ 7567 tên. Từ điển từ bao gồm 73940 từ, bao gồm 12332 tiếng.

Chương trình được thử nghiệm dựa trên hai tập dữ liệu. Một tập được rút ra từ [Hoa02]. Tập dữ liệu thứ hai sử dụng đoạn đầu luận văn này, sau đó phát sinh lỗi dựa trên tập những tiếng nhập nhằng. Các lỗi về nhập liệu không được kiểm tra ở đây mà chủ yếu tập trung vào lỗi phát âm.

Chương trình được thử nghiệm với nhiều tham số khác nhau, bao gồm:

- Sử dụng trigram hay bigram (P).
- Chuẩn hoá độ dài câu hay không (C).
- Bắt lỗi chặt hay không (S).
- Áp dụng heuristic phát sinh từ dựa trên ngữ cảnh hay không (H).

Kết quả thử nghiệm tên tập dữ liệu 1 được thể hiện trong bảng 6.1 ở trang 155. Kết quả thử nghiệm tên tập dữ liệu 2 được thể hiện trong bảng 6.2 ở trang 156. Kết quả được phân thành ba nhóm: non-word, real-word và total. Nhóm non-word bao gồm những câu phạm cả lỗi non-word lẫn real-word. Nhóm real-word bao gồm những câu chỉ phạm lỗi real-word. Nhóm total tính trung bình cả hai trường hợp. Ba kết quả PE, NE, CE là:

- PE là tỉ lệ bắt đúng lỗi chính tả, và trong danh sách từ đề nghị có từ đúng.

[1] xem phần 4.2.2 ở trang 85 để biết các quy tắc tách câu

- NE là số lần bắt nhầm lỗi chính tả trong câu.
- CE là tỉ lệ bắt đúng lỗi chính tả, nhưng trong danh sách từ đề nghị không có từ đúng.

Với nhóm non-word, trong thực tế khi phạm lỗi non-word, chương trình sẽ yêu cầu người dùng sửa hết lỗi trước khi tìm lỗi real-word. Do chương trình kiểm tra chạy tự động nên bước yêu cầu người dùng sửa lỗi non-word được bỏ qua. Việc này làm giảm một phần độ chính xác so với thực tế.

Kết quả này cho thấy hiệu suất của chương trình phụ thuộc vào loại văn bản cần kiểm tra. Với tập dữ liệu 1, xác suất bắt lỗi đúng cao nhất chỉ có thể đạt khoảng 53% trong khi với tập dữ liệu 2 là 88%. Độ chênh lệch này có thể do tập dữ liệu 1 sử dụng văn phong khác so với ngữ liệu huấn luyện (văn phong báo chí). Ngoài ra do tập dữ liệu 1 được dùng để làm bài tập bắt lỗi chính tả nên đưa ra những trường hợp lỗi tương đối khó.

Việc thay đổi các tham số cũng ảnh hưởng đáng kể đến độ chính xác. Việc áp dụng chuẩn hóa độ dài câu, lẽ ra sẽ cải thiện kết quả, nhưng thực tế lại giảm độ chính xác đáng kể. Tham số S nhìn chung tăng PE, nhưng đồng thời cũng tăng NE. Việc tăng NE khi áp dụng S cho thấy mô hình ngôn ngữ chưa thật sự tốt, chưa tìm ra chính xác câu đúng mà có khuynh hướng tìm nhầm. Áp dụng H kèm với S giúp giảm bớt NE phần nào, nhưng đồng thời cũng giảm PE. Cuối cùng, việc sử dụng trigram nhìn chung giúp tăng độ chính xác. Tuy nhiên sử dụng trigram cũng làm giảm tính thực tế của chương trình vì dữ liệu trigram lớn hơn rất nhiều so với bigram (khoảng 6–7 lần).

Như vậy, qua khảo sát, ta thấy nên áp dụng trigram, bắt lỗi chặt kèm heuristic. Độ chính xác real-word trong trường hợp này là 42,91% và 86,62%. Ngoài ra tỉ lệ NE cũng không quá cao.

Tham số			Non-word			Real-word			Total		
T	C	S	H	PE(%)	NE(%)	CE(%)	PE(%)	NE(%)	CE(%)	PE(%)	NE(%)	CE(%)
				50.14	2.09	18.38	37.76	1.08	7.48	44.56	1.72	13.48
			✓	42.62	2.09	17.27	31.29	1.05	6.80	37.52	1.71	12.56
		✓		62.07	3.09	21.26	47.95	2.13	27.40	55.62	2.69	24.06
		✓	✓	53.05	2.81	16.77	40.65	1.91	19.42	47.36	2.37	17.99
	✓			37.88	2.12	15.60	25.51	1.10	6.12	32.31	1.75	11.33
	✓		✓	34.82	2.10	15.60	21.58	1.07	6.16	28.88	1.73	11.37
	✓	✓		55.11	3.52	20.45	43.10	2.52	23.79	49.69	3.14	21.96
	✓	✓	✓	47.46	3.14	15.52	34.29	2.15	16.79	41.46	2.70	16.10
✓				39.55	2.10	16.71	26.19	1.08	5.78	33.54	1.73	11.79
✓			✓	37.60	2.08	16.16	24.83	1.05	6.46	31.85	1.70	11.79
✓		✓		61.11	2.98	23.39	43.99	2.18	29.21	53.24	2.66	26.07
✓		✓	✓	59.26	2.64	16.05	42.91	1.85	17.45	51.75	2.25	16.69
✓	✓			31.48	2.13	14.48	19.73	1.09	6.12	26.19	1.75	10.72
✓	✓		✓	28.13	2.12	14.48	19.05	1.06	6.12	24.04	1.73	10.72
✓	✓	✓		58.70	3.09	18.58	43.36	2.12	23.08	51.68	2.70	20.64
✓	✓	✓	✓	55.73	2.74	15.17	41.01	1.82	16.91	48.92	2.30	15.97

Bảng 6.1: Kết quả thử nghiệm tập dữ liệu 1

Tham số			Non-word			Real-word			Total		
T	C	S	H	PE(%)	NE(%)	CE(%)	PE(%)	NE(%)	CE(%)	PE(%)	NE(%)	CE(%)
				64.74	2.71	12.41	67.87	0.95	0.93	66.66	0.65	5.39
			✓	64.74	2.71	12.41	67.76	0.95	0.93	66.59	0.65	5.39
		✓		74.31	6.17	19.31	78.21	4.27	9.60	76.69	1.92	13.38
		✓	✓	73.97	5.86	14.40	78.23	3.97	8.03	76.57	1.80	10.51
	✓			27.50	2.74	12.41	43.03	0.97	0.94	36.97	0.66	5.41
	✓		✓	24.40	2.72	12.41	46.28	0.98	0.77	37.75	0.66	5.31
	✓	✓		37.07	8.59	45.78	52.99	6.37	27.21	46.80	2.78	34.43
	✓	✓	✓	33.62	8.12	41.55	56.32	5.83	23.24	47.48	2.58	30.37
✓				50.95	2.77	3.79	50.77	0.95	0.82	50.84	0.66	1.98
✓			✓	50.78	2.74	4.83	50.77	0.95	0.71	50.77	0.65	2.31
✓		✓		91.46	4.81	5.61	86.46	2.67	11.02	88.40	1.35	8.92
✓		✓	✓	90.42	4.20	3.97	86.62	2.45	7.62	88.10	1.20	6.20
✓	✓			30.60	2.81	0.69	21.14	0.99	0.45	24.86	0.67	0.54
✓	✓		✓	30.78	2.81	0.69	22.64	0.99	0.45	25.84	0.67	0.54
✓	✓	✓		85.69	5.30	12.59	81.28	3.56	14.50	82.99	1.62	13.76
✓	✓	✓	✓	85.09	4.53	10.95	82.68	3.18	12.22	83.62	1.41	11.73

Bảng 6.2: Kết quả tập thử nghiệm dữ liệu 2

6.3 Đánh giá

So với các phương pháp khác đã được áp dụng để bắt lỗi chính tả tiếng Việt, phương pháp được tiến hành một cách có hệ thống, dựa trên ngữ liệu huấn luyện sẵn có (từ công trình Luận án Tiến sĩ Ngôn ngữ học của TS. Đinh Điền), bảo đảm tính chính xác khi hoạt động. Phương pháp được dùng trong [TPLT98] không khả thi do chưa thực hiện tách từ (dùng heuristic để đánh giá các cách tách từ). Phương pháp được dùng trong [TTCV02] sử dụng mô hình Markov ẩn để tạo ngữ liệu tách từ từ ngữ liệu thô, do đó không bảo đảm tính chính xác của ngữ liệu huấn luyện. Phương pháp được dùng trong chương trình bắt lỗi chính tả VietSpell của tác giả Lưu Hà Xuyên chỉ sử dụng heuristic, không tách từ.

Tuy nhiên, chương trình vẫn có một số hạn chế nhất định. Do chỉ sử dụng thông tin về xác suất xuất hiện của một chuỗi từ liên tục khi tách từ mở nên kết quả có phần hạn chế. Trong nhiều trường hợp ta có thể khử nhập nhằng bằng từ loại, bằng thông tin cú pháp, hoặc thông tin về ngữ nghĩa. Thông tin xác suất lẽ ra chỉ là giải pháp bổ trợ cho các thông tin trên. Khi những thông trên không hoàn chỉnh, không bao quát hết hoặc không thể khử nhập nhằng toàn bộ, khi đó ta mới dùng thống kê như một giải pháp dự phòng.

Chính vì chỉ sử dụng ngram dựa trên từ, đôi khi chương trình sẽ chọn lầm cách tách từ, dẫn đến việc thông báo từ sai chính tả trong khi từ đó hoàn toàn đúng.

Phần kiểm tra từ viết hoa vẫn còn là một vấn đề chưa thể giải quyết trọn vẹn do viết hoa đúng quy cách đôi khi đòi hỏi ta phải hiểu ý nghĩa của từ viết hoa. Điều này hiện tại không thể thực hiện được.

6.4 Hướng phát triển

Việc áp dụng mô hình ngôn ngữ để bắt lỗi chính tả không phải là lựa chọn duy nhất. Ta có thể áp dụng các mô hình khác để bắt lỗi chính tả. Tuy nhiên, phần này sẽ chỉ đề cập đến những hướng phát triển dự trên việc áp dụng mô hình ngôn ngữ.

Việc đầu tiên có thể nghĩ đến để cải thiện chất lượng bắt lỗi chính tả là làm giàu nguồn tri thức được sử dụng. Trình bắt lỗi chính tả hiện thời chỉ sử dụng dữ liệu thô. Đây là một trong yếu tố làm giảm chất lượng chương trình. Ta có thể sử dụng mô hình ngôn ngữ dựa trên từ loại, đồng thời áp dụng mạng ngữ nghĩa hoặc thông tin cú pháp (được dùng trong các phương pháp Mật độ ngữ nghĩa và Văn phạm ràng buộc). Đây là một hướng phát triển lâu dài, đòi hỏi rất nhiều công sức.

Ta có thể cải thiện tốc độ bằng cách cải tiến các heuristic được dùng để hiệu chỉnh lưới từ. Hai heuristic quan trọng là heuristic phân biệt từ gốc và từ phát sinh, và heuristic phân biệt các từ phát sinh với nhau (dựa vào ngữ cảnh từ gốc). Heuristic chỉ có thể được cải thiện bằng cách rút trích thông tin thực tế để biết được từ nào thường được phát sinh hơn từ nào (nói cách khác, những từ nào là lỗi sai thông dùng của từ nào). Heuristic phản ánh cách tìm lỗi chính tả của con người. Ở đây việc đánh giá những từ có khả năng sai chính tả dựa trên chuỗi ngram của các tiếng không phải là một giải pháp hoàn hảo. Ta cần tìm ra những phương pháp khác nhằm phát hiện từ có khả năng sai chính tả tốt hơn. Những giải pháp bắt lỗi chính tả cảm ngữ cảnh của tiếng Anh có thể là một khởi đầu để tìm cách giải quyết.

Ngoài ra bắt lỗi chính tả cảm ngữ cảnh cũng là một cách tiếp cận. Vấn đề của bắt lỗi chính tả cảm ngữ cảnh là áp dụng trên một cách tách từ biết trước, không thay đổi. Hướng phát triển có thể là hiệu chỉnh để áp dụng bắt lỗi chính tả cảm ngữ cảnh trên lưới từ, thay vì trên một cách tách từ cố định.

Một cách khác đơn giản hơn là áp dụng bắt lỗi chính tả cảm ngữ cảnh trên từng cách tách từ có thể có trên lưới từ rồi định lượng để phát hiện lỗi. Do số lượng cách tách từ trên lưới từ là rất lớn, có thể ta chỉ cần xét n cách tách từ tốt nhất mà thôi.

Tài liệu tham khảo

[AGSV98] Eneko Agirre, Koldo Gojenola, Kepa Sarasola, and Atro Voutilainen. Towards a single proposal in spelling correction. *COLING-ACL '98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 463–463, 1998.

[AHD01] Jim Austin, Victoria J. Hodge, and Yo Dd. An evaluation of phonetic spell checkers. November 21 2001.

[Cha98] Chao-Huang Chang. A new approach for automatic chinese spelling correction. March 01 1998.

[cHN99] Nguyễn Đức Hải và Nguyễn Phạm Hải Nhi. Phân tích cú pháp tiếng việt và bắt lỗi chính tả. *Luận văn Cử nhân Tin học. ĐH Khoa học Tự nhiên HCM*, 1999.

[CL92] K. J. Chen and S. H. Liu. Word identification for madarin chinese sentences. *Proceedings of the Fifthteenth International Conference for Computational Linguistics*, 1992.

[Dam64] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the Association for Computing Machinery*, 7(3):171–176, 1964.

[DKT01] Dinh Dien, Hoang Kiem, and Nguyen Van Toan. Vietnamese word segmentation. *NLPRS*, 11 2001.

[GLH03] Jianfeng Gao, Mu Li, and Chang-Ning Huang. Improved source-channel models for chinese word segmentation. 2003.

[Gol95] Andrew R. Golding. A bayesian hybrid method for context-sensitive spelling correction. *Proceedings of the Third Workshop on Very Large Corpora*, pages 39–53, 1995.

[GR99] Andrew R. Golding and Dan Roth. A winnow-based approach to context-sensitive correction. *Machine Learning, Special issue on Machine Learning and Natural Language Processing*, 34:107–130, 1999.

[GS96] Andrew R. Golding and Yves Schabes. Combining trigram-based and feature-based methods for context-sensitive spelling correction. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[Hoa02] TS. Lê Trung Hoa. *Lỗi chính tả và cách khắc phục*. NXB Khoa học Xã hội, 2002.

[iiND03] Đinh Điền, Nguyễn Thống Nhất, và Nguyễn Thái Ngọc Duy. Cách tiếp cận thống kê cho hệ dịch tự động việt-anh. *Tạp chí phát triển KHCN*, 6:27–33, 2003.

[Knu73] D. E. Knuth. *The Art of Computer Programming. Vol. 3: Sorting and Searching*. Addison Wesley, 1973.

[KSM97] Theppitak Karoonboonyanan, Virach Sornlertlamvanich, and Surapant Meknavin. A thai soundex system for spelling correc-

tion. *Proceedings of the National Language Processing Pacific Rim Symposium*, 1997.

[LVHA94] F. Larlsson, A. Voutilainen, J. Heikkila, and A. Anttila. *Constraint Grammar: a Language-Independant System for Parsing Unrestricted text*. Berlin and New York: Mouton de Gruyter, 1994.

[MB97] Lidia Mangu and Eric Brill. Automatic rule acquisition for spelling correction. 1997.

[McI82] M. Douglas McIlroy. Development of a spelling list. *IEEE Transactions on Communications*, COM-30(1.1):91–99, 1982.

[Ofl96] Kemal Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *CL*, 22(1):73–89, 1996.

[Pal97] David Palmer. A trainable rule-based algorithm for word segmentation. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 1997.

[Pet80a] James Lyle Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the Association of Computing Machinery*, 23(12):676–687, 1980.

[Pet80b] James Lyle Peterson. *Computer Programs For Spelling Correction*. Springer-Verlag, Inc., Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1980.

[PS01] Fuchun Peng and Dale Schuurmans. Self-supervised Chinese word segmentation. *Lecture Notes in Computer Science*, 2189:238–242, 2001.

[PSG99] A Pratt, Padhraic Smyth, and Xianping Ge. Discovering chinese words from unsegmented text. August 22 1999.

[Rav96] Mosur K. Ravishankar. *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon University, 1996.

[RMBB89] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development an application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.

[SSGC96] Richard W Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *CL*, 22(3):377–404, 1996.

[Sto02] Andreas Stolcke. Srilm - an extensible language modeling toolkit. *Proc. Intl. Conf. Spoken Language Processing, Denver, Colorado*, 9 2002.

[TH01] Nguyễn Văn Toàn và Nguyễn Thị Minh Hằng. Tách từ tiếng việt. *Luận văn Cử nhân Tin học. ĐH Khoa học Tự nhiên HCM*, 2001.

[TND03] Nguyễn Văn Toàn, Văn Chí Nam, và Nguyễn Thái Ngọc Duy. Xây dựng WordNet tiếng việt. *Hội thảo quốc gia lần 6. Một số vấn đề CNTT, truyền thông, chủ để xử lý ngôn ngữ*, 2003.

[TPLT98] PTS. Phan Thị Tươi, KS. Nguyễn Hứa Phùng, KS. Huỳnh Vụ Như Liên, và KS. Phạm Quyết Thắng. Bắt lỗi chính tả tự

động cho tiếng việt bằng máy tính. *Đề tài nghiên cứu khoa học Sở Khoa học Công nghệ và Môi trường*, 1998.

[TTCV02] Nguyễn Phương Thái, Nguyễn Quốc Toản, Lê Văn Cường, và Nguyễn Văn Vinh. Kiểm lỗi chính tả tiếng việt sử dụng danh sách quyết định. 2002.

[WGLL00] Hai-Feng Wang, Jianfeng Gao, Kai-Fu Lee, and Mingjing Li Li. A unified approach to statistical language modeling for chinese. June 2000.

Phụ lục A

Dữ liệu kiểm tra

Thị Kính không thể **giải** bày nỗi oan của mình.	(giải,giãi)
Mọi người phải **nỗ** lực lao động.	(nỗ,nổ)
Bà cụ bị **lảng** tai.	(lảng,lãng)
Trong phòng họp, mọi người nói chuyện **se sẽ**.	(se sẽ,se sẻ)
Chiếc tủ rất nặng, nhưng nhiều người khiêng, trở nên nhẹ **bỗng**.	(bỗng,bổng)
Cần chăm số kĩ mồ **mả** ông bà.	(mả,mã)
Con **cãi** cha mẹ, trăm đường con hư.	(cãi,cải)
Con anh Triều mới học lớp **vỡ** lòng.	(vỡ,vở)
Có vợ **lẽ** là điều không tốt.	(lẽ,lẻ)
Công việc còn đang **lỡ dỡ**.	(lỡ dỡ,lở dở)
Bị ép rước khách, Kiều tự **vẫn**.	(vẫn,vẩn)
Khi mới khỏi bệnh, nên **cữ** ăn.	(cữ,cử)
Bị bắt quả tang gian lận, thí sinh **bẽ** mặt.	(bẽ,bẻ)
Không được uống nước **lã**.	(lã,lả)
Chiếc thuyền đã chìm **nghỉm**.	(nghỉm,nghĩm)
Lũ lớn đã về.	(lũ,lủ)
Chiếc **chõng** tre đặt dưới gốc dừa.	(chõng,chỏng)
Ăn **cỗ** đi trước, lội nước đi sau.	(cỗ,cổ)
Mải học, nó không hay tôi bước vào.	(mải,mãi)
Vác nặng, người công nhân ướt **đẫm** mồ hôi.	(đẫm,đẩm)
Mảnh vải **lỡ cỡ**, may áo thì thừa, may quần thì thiếu.	(lỡ cỡ,lở cở)

Cháu bé đang uống **sữa**. (sữa,sửa)
Học sinh **ngả** mũ chào thầy giáo. (ngả,ngã)
Trong nhà chưa tỏ, ngoài **ngõ** đã hay. (ngõ,ngỏ)
Sinh viên **bãi** khoá, học sinh biểu tình. (bãi,bải)
Câu văn này còn **lủng củng**. (lủng củng,lũng cũng)
Nó bền **bỉ** theo đuổi ước mơ. (bỉ,bĩ)
Lan niềm **nở** tiếp bạn. (nở,nỡ)
Từ **giã** quê hương, tác **giả** luôn luôn nhớ đến con sông. (giã,giả)
Đến **ngã** bảy, nó không biết đi theo **ngả** nào. (ngã,ngả)
Nhà khoa học kẹp con **đỉa** bỏ lên cái **đĩa**. (đỉa,đĩa)
Hai cậu bé giành nhau thanh **gỗ** nên gây **gổ** nhau. (gỗ,gổ)
Cháu bé **mải** nhìn đồ chơi, mẹ gọi **mãi** không nghe. (mải,mãi)
Bà cụ **ngả** người trên lưng ghế, ghế gãy, cụ bị **ngã**. (ngả,ngã)
Ông **lão** hơi ngà ngà, bước đi **lảo** đảo. (lão,lảo)
Lực sĩ ấy mạnh đến **nỗi** vác **nổi** một bao cát nặng 120 kg. (nỗi,nổi)
Anh ấy thường **nghĩ ngợi** cả trong giờ **nghỉ ngơi**. (nghĩ ngợi,nghỉ ngơi)
Ông giám đốc **lẩn** tránh câu trả lời vì sợ nhầm **lẫn**. (lẩn,lẫn)
Người chủ bổ cái **cũi** hư để làm **củi**. (cũi,củi)
Bà nội trợ **mở** thùng **mỡ** ra xem. (mở,mỡ)
Vì **sĩ** diện, nhạc **sỉ** không thừa nhận sự sai lầm của mình. (sĩ,sỉ)
Nó **xả** nước trong ruộng ra con kinh của **xã**. (xả,xã)
Cô ấy giả vờ **bẻ** cành cây để đỡ **bẽ** mặt. (bẻ,bẽ)
Vào hang sâu, người mẹ khuyên **nhủ** con không nên rờ vào các thạch **nhũ**. (nhủ,nhũ)
Ông ấy cảm thấy **lẻ** loi vì **lẽ** vợ ông đã qua đời. (lẻ,lẽ)
Dù được khuyên giải, nó **vẩn** nghĩ vớ **vẩn**. (vẩn,vẫn)
Mây trôi **lửng lơ**, dòng nước **lững lờ**. (lửng lơ,lững lờ)
Ăn hết **nửa** con gà, người lực sĩ tiếp tục ăn **nữa**. (nửa,nữa)
Kẻ cẩn thận thường dặn dò cặn **kẽ**. (kẻ,kẽ)
Đồng **củ** lang này mới đào từ mảnh đất **cũ**. (củ,cũ)
Nên **lảng** tránh những người **lãng** mạng. (lảng,lãng)
Vỉ thuốc đặt cạnh cây **vĩ** cầm. (vỉ,vĩ)
Người **viễn** khách không thích nói chuyện **viển** vông. (viễn,viển)
Sau chuyến **lữ** hành, nó mệt **lử**. (lữ,lử)
Có tiếng la **bải** hải trên **bãi** biển. (bải,bãi)

Ông **sãi** đang bước **sải** trên sân chùa.	(sãi,sải)
Sửa xe xong, ba đi mua hộp **sữa**.	(sửa,sữa)
Gặp vận **bĩ**, chàng thanh niên ấy vẫn bền **bỉ** phấn đấu thực hiện lí tưởng.	(bĩ,bỉ)
Nó nói dối, **rủ** được bạn đi chơi, thích chí cười **rũ** rượi.	(rủ,rũ)
Ai **bảo** với anh năm nay không có **bão**?	(bảo,bão)
Bã rượu không thể làm **bả** chuột.	(bã,bả)
Đòn **bẩy** không phải là cái **bẫy**.	(bẩy,bẫy)
Kết quả chẳng **bõ** công so với công sức đã **bỏ** ra.	(bõ,bỏ)
Dũng mặc quần **ka** ki.	(ka,ca)
Ba của Tài mới mua một chiếc **ghe**.	(ghe,ge)
Ngọc làm việc trong **ngành** tin học.	(ngành,nghành)
Chàng trai **chiêm** ngưỡng bức tượng.	(chiêm,triêm)
Bài làm còn nhiều chỗ sơ **suất**.	(suất,xuất)
Dòng nước chảy xiết.	(dòng,giòng)
Bạn Dũng không thích mặc áo hoa **hòe**.	(hòe,ngòe)
Lỡ tay **trót** đã nhúng chàm.	(lỡ,lở) (trót,chót)
Người thợ săn bắt được con **cheo**.	(cheo,treo)
Cậu bé biết nhường cơm **sẻ** áo.	(sẻ,xẻ)
Người vợ ấy rất **son sắt**.	(son sắt,xon xắt)
Không nên **gièm** pha người khác.	(gièm,dèm)
Cô bé không bao giờ **giấu** mẹ điều gì.	(giấu,dấu)
Cháo bé khóc **oe oe**.	(oe oe,hoe hoe)
Anh Dũng **chèo** ghe ra biển.	(chèo,trèo)
Bài làm của Hùng y **chang** bài làm của Dũng.	(chang,trang)
Chiếc bàn đặt **sát** cửa sổ.	(sát,xát)
Mẹ **xẻ** dưa cho các con ăn.	(xẻ,sẻ)
Trăng tròn **vành vạnh**.	(vành vạnh,dành dạnh)
Hoàn cảnh gia đình ấy rất **gieo neo**.	(gieo neo,deo neo)
Duyên từ sân nắng bước vào nhà, **hoa** cả mắt.	(hoa,oa,qua)
Cuộc sống của nàng Kiều thật ê **chề**.	(chề,trề)
Hành động của Trịnh Hâm thật đáng chê **trách**.	(trách,chách)
Vân Tiên nấu sử **xôi** kinh trong nhiều năm dài.	(xôi,sôi)
Bùi Kiệm không biết thế nào là liêm **sỉ**.	(sỉ,xỉ,sĩ)
Nên nhường nhịn nhau để tránh **va** chạm.	(va,da,gia)

Ngọc Dũng đã **giành** giải nhất cuộc thi. (giành,dành)
Chớ coi trọng việc vinh thân thì **gia**. (gia,da)
Hội nghị tập **trung** thảo luận những vấn đề **chung**. (trung,chung)
Lan thích nghe kể **chuyện** hơn đọc **truyện**. (chuyện,truyện)
Anh hà tiện không chịu **chi** tiền để có nhiều **tri** thức. (chi,tri)
Học sinh **chong** đèn học bài **trong** đêm. (chong,trong)
Người thợ săn đang cột con **cheo**, **treo** lên cành cây. (cheo,treo)
Cậu bé **trằm** trồ khen ngợi người thợ **chằm** nón. (trằm,chằm)
Nó **trót** hẹn với bạn đi chuyến tàu **chót**. (trót,chót)
Tùng **xẻ** dưa để chia **sẻ** với bạn. (xẻ,sẻ)
Trời **sẩm** tối, bà **xẩm** dọn hàng vào. (sẩm,xẩm)
Rờ mặt bàn thấy nhám **sì**, người chị **xì** một tiếng. (sì,xì)
Y tá ghi số lượng thuốc **xổ** vào **sổ**. (xổ,sổ)
Sau khi **sinh** con, chị ấy trong hết **xinh**. (sinh,xinh)
Khi **xử** lí một vấn đề, ta phải xem xét lịch **sử** của nó. (xử,sử)
Ôm **xấp** báo quá nặng, cậu bé té **sấp** xuống đất. (xấp,sấp)
Nó **vừa** uống nước **dừa** xiêm vừa nói chuyện. (vừa,dừa)
Không nên **dành** nhiều thì giờ cho việc **giành** quyền lợi. (dành,giành)
Học sinh **giở** cuốn **vở** để đọc tiếp trang đọc **dở**. (giở,vở,dở)
Con **gián** căn nham nhở tờ giấy có **dán** hồ. (gián,dán)
Đôi **giày** này đế rất **dày**. (giày,dày)
Sáng nào cũng **vậy**, ông ta **dậy** sớm để tập thể **dục**. (vậy,dậy) (dục,giục)
Giao du với bạn xấu, tính tốt của nó bị **dao** động. (giao,dao)
Nó phóng nhanh **quá**, bị cảnh sát giữ lại, **hoá** ra lại chậm. (quá,hoá)
Đến **quãng** đường vắng và tối quá, chú bé **hoảng** sợ. (quãng,hoảng)
Người khác **oán** ông chủ **quán** tính giá thực phẩm quá cao. (oán,quán)
Để chạy tội, cô thủ **quĩ quỉ** quyệt đã **hủy** một số chứng từ. (quĩ,hủy,quỉ)
Sáng hôm **qua**, cây mai đã nở **hoa**. (qua,hoa)
Cậu bé này rất **hoang**, không còn **ngoan** như ngày trước. (hoang,ngoan)
Anh em **hoà** thuận, được cha mẹ cho **quà**. (hoà,quà)
Lịch sử đã **hun** đúc nên những anh hùng dân tộc. (hun,hung)
Các em bé rất thích **mút** kẹo. (mút,múc)
Rau **muống** có nhiều chất bổ. (muống,muốn)
Đôi bàn tay chị ấy trắng **muốt**. (muốt,muốc)

Hai **chưn** ông ta trong thật rắn chắc.	(chưn,chưng)
Mứt gừng là loại **mứt** rất cay.	(mứt,mức)
Cá để lâu sẽ bị **ươn**.	(ươn,ương)
Ăn bánh **ướt** với chả lụa rất ngon.	(ướt,ước)
Khay trầu của mẹ trông thật đẹp mắt.	(khay,khai)
Đôi môi em bé gái đỏ **au**.	(au,ao)
Trời rét **căm căm**, anh ấy vẫn tập thể dục.	(căm căm, câm câm)
Các học sinh tiểu học thích **cặp** kè nhau đi.	(cặp,cập)
Ở Huế, người ta thường thả **diều** nhân **dịp** lễ tết.	(diều,dìu) (dịp,diệp)
Không nên hoàn toàn tin những gì diễn ra trong giấc **chiêm** bao.	(chiêm,chim)
Đàn gà lạc mẹ, kêu **chiêm chiếp**.	(chiêm chiếp,chim chíp)
Tép riu là loại **tép** nhỏ.	(tép,tếp)
Phải bảo vệ bờ **cõi** do ông cha để lại.	(cõi,cỗi)
Về nông thông, ta thấy nhiều cây **rơm** khô.	(rơm,rôm)
Người bệnh chỉ còn **thoi thóp**.	(thoi thóp,thôi thốp)
Suốt ngày, cậu bé ấy chỉ chơi **rong**.	(rong,rông)
Ông nội tôi còn một mẫu ruộng **rộc**.	(rộc,rọc)
Bạn Thành rất thích ăn món mì **nui**.	(nui,nuôi)
Bé Diệu làm việc **luôm thuộm**.	(luôm thuộm,lụm thụm)
Suốt ngày, cô gái ngồi dệt **cửi**.	(cửi,cữi,cưỡi)
Tiếng hót con **khướu** rất du dương.	(khướu,khứu)
Đừng **nuối** tiếc những ngày thơ mộng đã qua.	(nuối,núi)
Trái đất có 24 **múi** giờ.	(múi,muối)
Cậu bé rất thích **cưỡi** ngựa gỗ.	(cưỡi,cữi)
An **táng** cha già xong, anh bộ đội trở về đơn vị.	(táng,tán)
Tiếng tranh luận **át** cả tiếng gọi giữ im lặng của người chủ tọa.	(át,ác)
Học sinh **căng** lều dựng trại để vui chơi.	(căng,căn)
Bây giờ chúng ta không còn dùng bạc **cắc**.	(cắc,cắt)
Toà nhà ấy có tất cả năm **tầng**.	(tầng,tần)
May áo xong, còn thừa ba **tấc** vải.	(tấc,tất)
Tiếng **phèng** la từ căn nhà dài vang lên.	(phèng,phèn)
Heo kêu **eng éc** vì nó bị chọc tiết.	(eng éc,en ét)
Cụ già đang ngồi **bện** thừng ngoài sân.	(bện,bệnh)
Da người bệnh trắng **bệch**.	(bệch,bệt)

Tinh thần ông cụ hết sức **tinh** anh. (tinh,tin)
Bánh **ít** lá gai ở Bình Định rất nổi **tiếng**. (ít,ích) (tiếng,tiến)
Nó cảm thấy **tiêng** tiếc chiếc xe đạp đã mất. (tiêng,tiên)
Đây là loại sữa đã **tiệt** trùng. (tiệt,tiệc)
Con **trùn** có lợi, còn côn **trùng** có hại. (trùn,trùng)
Theo bạn, giờ **phút** nào bạn cảm thấy hạnh **phúc** nhất? (phút,phúc)
Con chuồn **chuồn** bay ngang qua **chuồng** bồ câu. (chuồn,chuồng)
Con chão **chuộc** nhảy qua cái bẫy **chuột**. (chuộc,chuột)
Bánh chưng này **nhưn** nhiều **nhưng** ít thịt. (nhưn,nhưng)
Người thương binh thường bị **nhức** nhối, **nhứt** là những lúc trở trời. (nhức,nhứt)
Lãnh **lương** rồi, mẹ ghé chợ mua **lươn**. (lương,lươn)
Dáng cô giáo tầm **thước**, tà áo dài tha **thướt**. (thước,thướt)
Trời nhiều **mây**, gió heo **may** lại về. (mây,may)
Nó được **báo** tin vùng này có kho **báu**. (báo,báu)
Anh ta lằm **bằm** vì bị té **bầm** ở tay. (bằm,bầm)
Giải pháp trung **lập** hóa được **lặp** lại một lần nữa. (lập,lặp)
Những người biết **điều đều** được quí trọng. (điều,đều)
Đương **kim** Thủ tướng **kiêm** nhiệm chức Bộ trưởng Ngoại giao. (kim,kiêm)
Chúng tôi ngồi dưới gốc **bàng** để **bàn** công việc. (bàng,bàn)
Ăn nói rất hoạt **bát**, Tiến phản **bác** đối phương. (bát,bác)
Chú bé phục **lăn** những người xây dựng **lăng** vua Tự Đức. (lăn,lăng)
Đường bị **tắc** nên nó phải đi đường **tắt**. (tắc,tắt)
Chí Phèo **ngẩng** đầu, **ngẩn** ngơ nhìn mọi người. (ngẩng,ngẩn)
Ngọc nổi **bật** trong đám bạn bè vì nó tiến bộ vượt **bậc**. (bật,bậc)
Lửa đã **bén** vách nhà mà nó quên **béng**. (bén,béng)
Nó nghe bạn dọa **méc** ba má, mặt nó tái **mét**. (méc,mét)
Bắt được con kên **kên**, nó được bạn bè công **kênh**. (kên,kênh)
Trong bữa tiệc nó **thếch** đãi bạn hữu, món xúp cua nhạt **thết**. (thếch,thết)
Tính ông ấy rất **kín** đáo, ai cũng **kính** nể. (kín,kính)
Trời tối **mịch**, cảnh chuà thêm tịch **mịt**. (mịch,mịt)
Con sông ở **biên** giới, nước xanh **biêng** biếc. (biên,biêng)
Tiếc rằng thời **tiết** hôm nay không được thuận lợi cho trận đấu bóng đá. (tiếc,tiết)
Nhân dân hai nước **kíp** xóa bỏ mối thù truyền **kiếp**. (kíp,kiếp)
Muối rang trên **bếp** đang nổ lép **bép**. (bếp,bép)

Khối thị trường chung châu Âu có nền "công nghệ không **khói**" rất tiên tiến. (khối,khói)
Đứa bé **chồm** lên phía trước để vuốt **chòm** râu ông nội. (chồm,chòm)
Nằm trong chiếc **nóp**, không còn nơm **nớp** sợ muỗi cắn. (nóp,nớp)
Có con bế con **bồng**, chó đèo **bòng**. (bồng,bòng)
Gai **góc** nằm la liệt dưới **gốc** cây. (góc,gốc)
Nó ngủ **muồi**, không còn ngửi thấy **mùi** thơm ngát phòng. (muồi,mùi)
Về **hưu**, ông ấy không còn thích đi săn **hươu**. (hưu,hươu)
Bà ấy mới chi tiêu **khoản** phân nửa **khoảng** tiền mới lãnh. (khoản,khoảng)
Càng ngày tính nó **càng càn** rỡ. (càng,càn)
Vác cái **thang** nặng, cậu bé thở **than**. (thang,than)
Vấp hòn đá, nó ngã **chúi** vào bụi **chuối**. (chúi,chuối)
Chị ấy **đang** ngồi **đan** cái áo len cho con. (đang,đan)