ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA: CÔNG NGHỆ PHẦN MỀM



**BÁO CÁO ĐỒ ÁN**

**Môn học: Ngôn ngữ lập trình Java**

**XÂY DỰNG ỨNG DỤNG**

**HỖ TRỢ HỌC HÓA HỌC**

*Gv: Huỳnh Tuấn Anh*

Sinh viên thực hiện:

Nguyễn Quốc Tài MSSV: 16521053

Huỳnh Gia Phát MSSV: 16520910

Vũ Gia Khiêm MSSV: 16520591

Mục lục

[**1.** **Giới thiệu** 4](#_Toc516669933)

[**1.1.** **Giới thiệu đề tài** 4](#_Toc516669934)

[**1.2.** **Lý do chọn đề tài** 4](#_Toc516669935)

[**1.3.** **Giới thiệu về ứng dụng:** 4](#_Toc516669936)

[**1.3.1.** **Tính phân tử khối của một chất:** 4](#_Toc516669937)

[**1.3.2.** **Cân bằng phương trình hóa học:** 5](#_Toc516669938)

[**1.3.3.** **Điều chế:** 5](#_Toc516669939)

[**1.3.4.** **Chuỗi phản ứng:** 5](#_Toc516669940)

[**1.3.5.** **Mô tả hiện tượng:** 6](#_Toc516669941)

[**1.3.6.** **Phân biệt các chất:** 6](#_Toc516669942)

[**1.3.7.** **Tìm chất:** 6](#_Toc516669943)

[**2.** **Cơ sở lý thuyết** 6](#_Toc516669944)

[**2.1.** **Biểu diễn tri thức** 7](#_Toc516669945)

[**2.1.1.** **Các phương pháp biểu diễn tri thức** 7](#_Toc516669946)

[**2.1.2.** **Biểu diễn tri thức bằng Logic vị từ** 7](#_Toc516669947)

[**2.1.3.** **Ngôn ngữ lập trình Prolog** 8](#_Toc516669948)

[**2.2.** **Lý thuyết đồ thị** 9](#_Toc516669949)

[**2.2.1.** **Tìm kiếm theo chiều sâu (Depth First Search - DFS)** 9](#_Toc516669950)

[**2.2.2.** **Tìm kiếm theo chiều rộng (Breadth First Search – BFS)** 10](#_Toc516669951)

[**2.3.** **Lập trình đa luồng** 11](#_Toc516669952)

[**3.** **Phân tích các chức năng** 11](#_Toc516669953)

[**3.1.** **Tính phân tử khối của một chất:** 11](#_Toc516669954)

[**3.2.** **Cân bằng phương trình hóa học:** 12](#_Toc516669955)

[**3.3.** **Điều chế:** 17](#_Toc516669956)

[**3.4.** **Chuỗi phản ứng:** 19](#_Toc516669957)

[**3.5.** **Mô tả hiện tượng:** 21](#_Toc516669958)

[**3.6.** **Phân biệt các chất:** 22](#_Toc516669959)

[**3.7.** **Tìm chất:** 24](#_Toc516669960)

[**4.** **Cài đặt** 25](#_Toc516669961)

[**4.1.** **Biểu diễn tri thức** 30](#_Toc516669962)

[**4.2.** **Tính phân tử khối của một chất:** 33](#_Toc516669963)

[**4.3.** **Cân bằng phương trình hóa học:** 33](#_Toc516669964)

[**4.4.** **Điều chế:** 40](#_Toc516669965)

[**4.5.** **Chuỗi phản ứng:** 42](#_Toc516669966)

[**4.6.** **Mô tả hiện tượng:** 48](#_Toc516669967)

[**4.7.** **Phân biệt các chất** 49](#_Toc516669968)

[**4.8.** **Tìm chất:** 54](#_Toc516669969)

[**5.** **Phân công công việc** 55](#_Toc516669970)

[**6.** **Demo** 56](#_Toc516669971)

1. **Giới thiệu**
   1. **Giới thiệu đề tài**

Môn Hóa học trong trường trung học phổ thông là một nhánh của khoa học tự nhiên, là ngành nghiên cứu về thành phần, cấu trúc, tính chất, và sự thay đổi của vật chất. Hóa học nói về các nguyên tố, hợp chất, nguyên tử, phân tử, và các phản ứng hóa học xảy ra giữa những thành phần đó. Hóa học đôi khi được gọi là "khoa học trung tâm" vì nó là cầu nối các ngành khoa học tự nhiên khác như vật lý học, địa chất học và sinh học.

Đối với y học thì hóa học không thể thiếu được trong cuộc tìm kiếm những thuốc trị bệnh mới và trong việc sản xuất các dược phẩm. Các kỹ sư thường tìm kiếm vật liệu chuyên dùng tùy theo ứng dụng (vật liệu nhẹ trong chế tạo máy bay, vật liệu xây dựng chịu lực và bền vững, các chất bán dẫn đặc biệt tinh khiết, ...).

Công nghiệp hóa học là một ngành kinh tế rất quan trọng. Công nghiệp hóa học sản xuất các hóa chất cơ bản như axít sunfuric hay amoniac, thường là nhiều triệu tấn hằng năm, cho sản xuất phân bón và chất dẻo và các mặt khác của đời sống và sản xuất công nghiệp. Mặt khác, ngành công nghiệp hóa học cũng sản xuất rất nhiều hợp chất phức tạp, đặc biệt là dược phẩm. Nếu không có các hóa chất được sản xuất trong công nghiệp thì cũng không thể nào sản xuất máy tính hay nhiên liệu và chất bôi trơn cho công nghiệp ô tô.

* 1. **Lý do chọn đề tài**

Từ những lí do trên, ta thấy Hóa học có ứng dụng cao trong cuộc sống vậy nên Hóa học cũng là một môn học hết sức quan trọng. Để đóng góp một phần nào đó hỗ trợ các bạn học sinh, sinh viên học tốt môn Hóa hơn. Nhóm em xin chọn đề tài viết phần mềm Hóa học để làm đồ án môn học.

* 1. **Giới thiệu về ứng dụng:**

Dưới đây, nhóm xin trình bày phần mô tả các chức năng của ứng dụng.

Ứng dụng gồm có 7 chức năng được xây dựng nhằm giải quyết các bài toán cơ bản của học sinh trình độ cấp 2 và một chút kiến thức cấp 3 trong môn hóa học.

* + 1. **Tính phân tử khối của một chất:**

*Bài toán:* Cho một công thức hóa học, hãy tính phân tử khối của công thức hóa học này.

*Mô tả chương trình:* Người dùng nhập vào một công thức hóa học của một chất bất kì. Hệ thống sẽ gợi ý người dùng nhập nhanh, chương trình thực thi tính toán. Trả về cho người dùng phân tử khối chất đó và cách tính ra số đó.

* + 1. **Cân bằng phương trình hóa học:**

*Bài toán:* Cho một chương trình hóa học, hãy cân bằng các hệ số trước các chất của phương trình hóa học này.

*Mô tả chương trình:* Người dùng nhập vào phương trình hóa học và chọn phương pháp cân bằng, sau đó nhấn nút cân bằng. Chương trình sẽ cân bằng phương trình vừa nhập theo cách phương pháp mà người dùng chọn. Chương trình có thể thực thi hai phương pháp cân bằng đó là cân bằng đại số và cân bằng electron.

* + 1. **Điều chế:**

*Bài toán:* Cho một tập các chất X và tập các chất Y. Hãy viết các phương trình phản ứng để điều chế ra tập Y từ các chất trong tập X đã cho. Sao cho số phương trình là ít nhất có thể.

*Mô tả chương trình*: Người dùng sẽ nhập các chất đã cho X và các chất cần điều chế Y. Chương trình sẽ liệt kê ra chuỗi phản ứng điều chế ngắn nhất.

* + 1. **Chuỗi phản ứng:**

*Bài toán:* Cho một chuỗi các chất X (có cùng cation kim loại). Hãy liệt kê phương trình hóa học để hoàn thành chuỗi các chất đó.

Ví dụ: Fe → FeSO4 → FeCl2 → FeCl3 → Fe(OH)3 → Fe2O3

Các phương trình hóa học là:

Fe + H2SO4 = FeSO4 + H2

FeSO4 + BaCl2 = FeCl2 + BaSO4

2FeCl2 + Cl2 = 2FeCl3

FeCl3 + 3NaOH = Fe(OH)3 + 3NaCl

2Fe(OH)3 = Fe2O3 + 3H2O

*Mô tả chương trình*: Người dùng sẽ nhập vào các chất có trong chuỗi điều chế. Chương trình sẽ đi tìm các phương trình hóa học phù hợp với chuỗi điều chế đó.

* + 1. **Mô tả hiện tượng:**

*Bài toán:* Lần lượt cho các dung dịch vào trong một ống nghiệm. Hãy mô tả hiện tượng quan sát được.

*Mô tả chương trình*: Chương trình vẽ ra một ống nghiệm. Ban đầu trong đó chỉ có nước. Người dùng có thể thả vào đó những những dung dịch mình muốn (lượng dư). Chương trình sẽ tính toán và mô tả hiện tượng cho người dùng như: Nếu có phản ứng tạo ra chất khí, thì trên ống nghiệm sẽ có khí thoát ra. Tương tự, nếu có phản ứng tạo ra chất kết tủa, thì dưới đáy ống nghiệm cũng sẽ có màu của chất kết tủa đó. Dung dịch cũng được mô tả màu một cách rất khách quan.

* + 1. **Phân biệt các chất:**

*Bài toán:* Cho một hỗn hợp các chất ở dạng dung dịch. Hãy trình bày cách phân biệt các chất đó.

*Mô tả chương trình*: Chương trình yêu cầu người dùng nhập các chất cần phân biệt. Chương trình tiến hành hiển thị trực quan các bước để phân biệt các chất.

* + 1. **Tìm chất:**

*Bài toán:* Đặt trường hợp, bạn là người bị lạc vào rừng. Bạn gặp một chất lỏng rất lạ. Và đang muốn biết chất đó là chất gì. Cho các hiện tượng phản ứng khi dung dịch đó tác dụng với các dung dịch khác. Hãy đoán xem chất đó là chất gì.

*Mô tả chương trình*: Người dùng sẽ chat với một chatbot và trả lời những câu hỏi của nó. Chatbot sẽ giúp bạn giới hạn tối đa các chất có thể là dung dịch. Giúp bạn một phần nào xác định được chất lỏng đó.

1. **Cơ sở lý thuyết**
   1. **Biểu diễn tri thức**
      1. **Các phương pháp biểu diễn tri thức**

Có rất nhiều cách biểu diễn tri thức trong máy tính như:

a) Biểu diễn tri thức bằng logic mệnh đề

b) Biểu diễn tri thức bằng logic vị từ

c) Biểu diễn tri thức bằng Frame, Classes

d) Biểu diễn tri thức bằng mạng ngữ nghĩa

e) Biểu diễn tri thức bằng hệ luật dẫn

Trong phạm vi báo cáo, nhóm chỉ đề cập và sử dụng biểu diễn tri thức bằng logic vị từ.

* + 1. **Biểu diễn tri thức bằng Logic vị từ**

Biểu diễn tri thức bằng mệnh đề gặp phải một trở ngại cơ bản là ta không thể can thiệp vào cấu trúc của một mệnh đề. Hay nói một cách khác là mệnh đề không có cấu trúc. Điều này làm hạn chế rất nhiều thao tác suy luận. Do đó, người ta đã đưa vào khái niệm vị từ và lượng từ ( - với mọi, - tồn tại) để tăng cường tính cấu trúc của một mệnh đề.

Trong logic vị từ, một mệnh đề được cấu tạo bởi hai thành phần là các đối tượng tri thức và mối liên hệ giữa chúng (gọi là vị từ). Các mệnh đề sẽ được biểu diễn dưới dạng:

Vị từ (<đối tượng 1>, <đối tượng 2>, …, <đối tượng n>)

Như vậy để biểu diễn vị của các trái cây, các mệnh đề sẽ được viết lại thành:

Cam có vị Ngọt 🡺 Vị (Cam, Ngọt)

Cam có màu Xanh 🡺 Màu (Cam, Xanh)

Công cụ vị từ đã được nghiên cứu và phát triển thành một ngôn ngữ lập trình đặc trưng cho trí tuệ nhân tạo. Đó là ngôn ngữ PROLOG.

* + 1. **Ngôn ngữ lập trình Prolog**

Prolog là một ngôn ngữ lập trình. Tên gọi Prolog được xuất phát từ cụm từ tiếng Pháp *Programmation en logique*, nghĩa là "lập trình theo lô gích". Xuất hiện từ năm 1972 (do Alain Colmerauer và Robert Kowalski thiết kế), mục tiêu của Prolog là giúp người dùng mô tả lại bài toán trên ngôn ngữ của logic, dựa trên đó, máy tính sẽ tiến hành suy diễn tự động dựa vào những cơ chế suy diễn có sẵn (hợp nhất, quay lui và tìm kiếm theo chiều sâu) để tìm câu trả lời cho người dùng.

Prolog được sử dụng nhiều trong các ứng dụng của trí tuệ nhân tạo và ngôn ngữ học trong khoa học máy tính (đặc biệt là trong ngành xử lý ngôn ngữ tự nhiên vì đây là mục tiêu thiết kế ban đầu của nó). Cú pháp và ngữ nghĩa của Prolog đơn giản và sáng sủa, nó được người Nhật coi là một trong những nền tảng để xây dựng máy tính thế hệ thứ năm mà ở đó, thay vì phải mô tả cách giải quyết một bài toán trên máy tính, con người chỉ cần mô tả bài toán và máy tính sẽ hỗ trợ họ nốt phần còn lại.

Một chương trình Prolog bao gồm các luật được biểu diễn dưới dạng mệnh đề Horn. Một mệnh đề Horn có dạng

Head :- Body.

Head là một vị từ logic, còn Body có thể là rỗng hoặc là một tập các vị từ logic. Ví dụ như sau:

chẵn (X) :- X chia\_dư 2 = 0.

* 1. **Lý thuyết đồ thị**
     1. **Tìm kiếm theo chiều sâu (Depth First Search - DFS)**

Tìm kiếm ưu tiên chiều sâu hay tìm kiếm theo chiều sâu (*Depth-first search* - DFS) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Thông thường, DFS là một dạng tìm kiếm thông tin không đầy đủ mà quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước. Trong dạng không đệ quy, tất cả các đỉnh chờ được phát triển được bổ sung vào một ngăn xếp LIFO.

Thuật toán:

**def DFS (u):**

visited[u] = true

foreach (u, v) in E:

if (not visited[v]):

do something;

DFS(v)

**end def**

* + 1. **Tìm kiếm theo chiều rộng (Breadth First Search – BFS)**

Trong lý thuyết đồ thị, tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại.

Thuật toán:

**def BFS(start):**

Push (start, Queue)

While not empty(Queue):

u = Pop(Queue)

Visited(u) = true

Foreach (u, v) in E:

Do something

If not Visited[v]:

Push (v, Queue)

**end def**

* 1. **Lập trình đa luồng**

Đa luồng (Multithreading) là một form chuyên dụng của đa nhiệm (multitasking) và một đa nhiệm là tính năng cho phép máy tính chạy hai hoặc nhiều chương trình đồng thời. Nói chung, có hai kiểu đa nhiệm là: process-based và thread-based tương ứng dựa trên tiến trình và dựa trên luồng.

Đa nhiệm dựa trên tiến trình xử lý việc thực thi đồng thời của các chương trình. Đa nhiệm dựa trên luồng xử lý việc thực thi đồng thời các phần của cùng một chương trình.

Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời. Mỗi phần của chương trình đó được gọi là một thread, và mỗi thread định nghĩa một path riêng biệt của sự thực thi.

Trong Chemistry App, phần đa luồng được sử dụng ở việc làm Chatbot với chức năng Tìm chất.

1. **Phân tích các chức năng**
   1. **Tính phân tử khối của một chất:**

*Bài toán:* Cho một công thức hóa học, tính phân tử khối của công thức hóa học này.

*Input:* Công thức hóa học được biểu diễn dưới dạng chuỗi (String).

*Output:* Phân tử khối của công thức đó

*Ý tưởng:* Xây dựng 1 file data.txt lưu trữ tên và phân tử khối của nguyên tố đó. Tạo danh sách str [] để lưu trữ tên của nguyên tố, cnt [] lưu trữ số lượng của nguyên tố đó trong hợp chất. Các tên nguyên tố trong danh sách str [] ta có thể tìm được phân tử khối nhờ vào file data.txt đã lưu sau đó nhân với số lượng nguyên tố đó xuất hiện trong hợp chất, từ 2 danh sách này ta đưa vào map <String, integet> để trả về 1 map. Sau đó duyệt công thức hóa học từ trái sang phải. Mặc định các nguyên tố trong hợp chất là 1. Nếu gặp dấu ‘)’ ta cập nhật lại danh sách cnt []. Sau tính được phân tử khối của công thức hóa học bằng cách lấy

B1: Khởi tạo các giá trị cần thiết.

B2: Duyệt từ trái sang phải.

B2.1: Nếu giá trị thứ i trong chuỗi là số ta cập nhật giá trị list str [k], cnt [k]// với k là biến chạy trong list str [], cnt [].

B2.2: Nếu giá trị thứ i số chuỗi là 1 nguyên tố ta đưa tên nguyên tố đó vào list st r[]

B2.3: Nếu giá trị i là dấu “(“cnt[k] =-1.

B2.4: Nếu giá trị i là dấu “)” cập nhật lại cnt cho tới khi gặp -1 thì dừng.

B5: Đưa list str [], cnt [] vào map

B6: return map

VD: Ta có hợp chất X= K2SO4

X [0] = K là một nguyên tố ta đưa str [0] = X [0], cnt [0] = 1;

X [1] = 2 là một số số ta cập nhật list cnt, cnt [0] = 2;

X [2] = S là nguyên tố ta đưa str [1] = S;

Tưởng tự với các bước tiếp theo ta có:

str [K, S, O]

cnt [2, 1, 4]

Sau đó từng tên của nguyên tố ta truy cập trong file data để lấy phân tử khối nhân với cnt [i] tương ứng.

* 1. **Cân bằng phương trình hóa học:**

*Input:* một chuỗi là phương trình phản ứng hóa học chưa được cân bằng.

*Output:* một chuỗi là phương trình hóa học đã được cân bằng hệ số, một đoạn text lưu hướng dẫn cho mỗi phương pháp.

Thực hiện bằng 2 cách:

*Cách 1: Cân bằng đại số:*

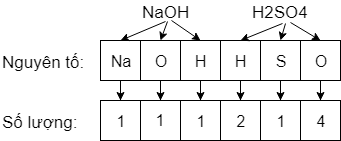
*Ý tưởng:*

* Xác định thành phần nguyên tố và số lượng của các nguyên tố bên các chất tham gia và sản phẩm tạo thành.
* Thành lập hệ phương trình theo nguyên tố, nghĩa là mỗi nguyên tố ta có thể thành lập được một phương trình. Như vậy, hệ phương trình ta thành lập được là hệ có n phương trình m ẩn (với n là số nguyên tố có trong phương trình phản ứng, m là tổng số chất tham gia và số chất tạo thành).
* Thực hiện giải hệ phương trình bằng cách thành lập thành ma trận giải ma trận đó bằng phương pháp Gauss, tìm được ẩn số và đưa lên phương trình.

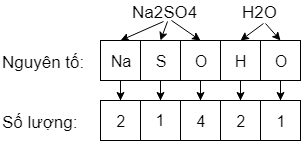
*Ví dụ minh họa:*

NaOH + H2SO4 → Na2SO4 + H2O.

Ta cần phân biệt chất tham gia phản ứng, sản phảm tạo thành. Sau đó ta tách thành các nguyên tố và số lượng của nó. Các chất tham gia phản ứng: NaOH, H2SO4.



Các chất sản phẩm tạo thành: Na2SO4, H2O.

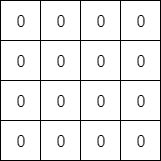


Sau khi biết được số lượng, ta cần thành lập hệ phương trình tìm nghiệm các ẩn số, với các ẩn số là các hệ số cân bằng ta cần tìm như sau:

+ Dùng một mảng phụ để lưu các nguyên tố không lặp lại có trong phương trình:



+ Tạo một mảng matrix lưu các hệ số của hệ n phương trình m ẩn, với n là số nguyên số trong phương trình phản ứng, m là số chất tham gia + số chất sản phẩm.



+ Đi qua từng chất của các chất tham gia và các chất sản phẩm, kết hợp với mảng phụ ta thành lập được hệ phương trình tìm được hệ số cân bằng của phương trình:

* Với mỗi chất tham gia phản ứng ta lấy ra từng nguyên tố
* Nếu nguyên tố trong chất thứ i trùng với nguyên tố thứ j trong mảng VT thì tại vị trí thứ i của mảng matrix ta lưu số lượng của nguyên tố thứ i.
* Tương tự với mỗi chất của sản phẩm.

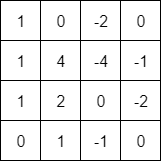
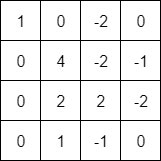
Na: 1a – 2c = 0

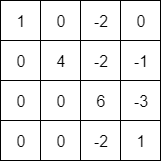
O: 1a + 4b – 4c – 1d = 0

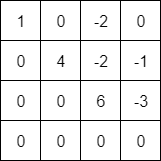
H: 1a + 2b – 2d = 0

S: 1b – 1c = 0

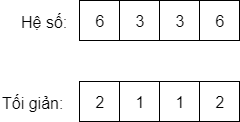
+ Sau khi có hệ phương trình, ta sẽ giải hệ phương trình bằng phương pháp tuyến tính (đã học ở môn đại số tuyến tính).





* Nghiệm ta thu được:



* Đưa lên phương trình: 2NaOH + H2SO4 Na2SO4 + 2H2O

*Cách 2: Cân bằng electron:*

*Ý tưởng:*

* Tương tự ở cách một ta cần xác định các chất tham gia và sản phẩm tạo thành, xác định các nguyên tố và số lượng của nguyên tố đó trong chất. Ở hàm này ta cần bổ sung thêm số oxi – hóa của từng nguyên tố trong chất.
* Thực hiện các bước như bên Hóa học, ta cần xác định quá trình khử và quá trình oxi – hóa, sau đó thực hiện thăng bằng electron sao cho tổng nhường bằng tổng nhận.
* Đưa các hệ số tìm được lên phương trình và thực hiện cân bằng lại với các các nguyên tố không thay đổi electron.

*Ví dụ minh họa:*

Fe + HNO3 Fe(NO3)3 + NO + H2O

Fe(NO3)3

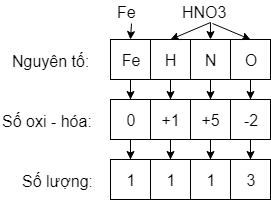
Số oxi - hóa của Fe là: +3

Số oxi – hóa của O là: -2

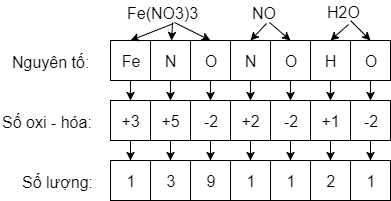
* Số oxi – hóa của N là: (-3 + 2\* 9)/3 = 15/3 = +5

Bước 1: Ta cần phân biệt chất tham gia phản ứng, sản phảm tạo thành. Sau đó ta tách thành các nguyên tố, số oxi hóa và số lượng của nó.

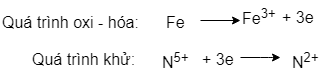
Các chất tham gia phản ứng: Fe, HNO3.



Các chất sản phẩm tạo thành: Fe(NO3)3, NO, H2O.



Bước 2: Xác định quá trình oxi hóa, quá trình khử và nhân hệ số sao cho cân bằng về số lượng của chất tham gia và chất sản phẩm. Sau đó thực hiện quá trình trao đổi ion sao cho ‘Tổng số electron cho = Tổng số electron nhận (tổng số oxi hóa giảm = tổng số oxi hóa tăng).



Bước 3: Sau khi thực hiện bước trên, ta đưa hệ số của các chất có trong quá trình oxi hóa, quá trình khử. Tiếp theo, thực hiện cân bằng lại các nguyên tố không thay đổ số oxi hóa. Kiểm soát lại số lượng nguyên tử oxi ở bên tham gia đảm bảo phải bằng bên sản phẩm.

* Fe + 4HNO3 Fe(NO3)3 + NO + 2H2O
  1. **Điều chế:**

*Nhắc lại bài toán:* Cho một tập các chất X và tập các chất Y. Hãy viết các phương trình phản ứng để điều chế ra tập Y từ các chất trong tập X đã cho. Sao cho số phương trình là ít nhất có thể.

*Mô hình hóa bài toán:* Ta coi việc điều chế từ tập X tới tập Y là một chuỗi các trạng thái {x1🡪x2 🡪 x3 🡪…🡪 xn}. Mà trong đó, mỗi trạng thái là một tập các chất. Vậy thì, ta có x1 = X và Y xn. Bài toán cần giải quyết lúc này là tìm chuỗi các trạng thái như vậy sao cho độ dài của chuỗi này là nhỏ nhất (tức n nhỏ nhất).

Xét hai trạng thái u và v, nếu v có thể đến được từ u khi và chỉ khi:

* Hai trong số các chất trong u phản ứng với nhau tạo ra chất mới
* Một chất nào đó trong u phản ứng phân hủy tạo ra chất mới

(định nghĩa chất mới ở đây nghĩa là chất mà ban đầu không có trong u)

Nếu một trong hai trường hợp trên xảy ra thì ta có thể nói có đường đi nối giữa trạng thái u và v. Tới đây, ta cũng có thể xác định cách để trang thái u đến được với trạng thái v là qua một phương trình phản ứng.

Từ đó, ta có ý tưởng sau:

Ý tưởng: Xem mỗi trạng thái (được mô tả ở trên) là một đỉnh của đồ thị. Các đỉnh là các trạng thái có thể sinh ra. Các đỉnh được nối với nhau bằng một phương trình hóa học. Ta cần tìm đường đi ngắn nhất từ đỉnh S đến đỉnh T (với S là trạng thái đầu tiên X, và T là trạng thái cuối cùng là trạng thái mà tập các chất trong nó có chứa Y).

Ta sẽ dùng thuật toán duyệt rộng đồ thị BFS để tìm đường đi ngắn nhất này. Sau đó, truy vết đường đi ta được các phương trình phản ứng để điều chế từ trạng thái S có thể đến được trạng thái T. Hay nói cách khác, cách phương trình đó xác định một cách điều chế nhanh nhất các chất trong X thành các chất trong Y. Đến đây, bài toán Điều chế đã được giải quyết.

Thuật toán:

S = makeState (X)

Push (S, Queue)

Trace (S) = Null

While not empty(Queue):

u = Pop(Queue)

// Xét chất đó tự phản ứng phân hủy

foreach s in u:

if canDestroy(s):

temp = add(destroy(s), u)

v = makeState(temp)

Trace(v) = u

If isGoal(v):

return v

foreach difference (s1, s2) in (u, u):

If canAccess (s1, s2):

temp = add (access (s1, s2), u)

v = makeState(temp)

if isGoal(v):

return v

Ở đây, hàm canDestroy(X) nghĩa là chất có công thức hóa học là X có thể thực hiện phản ứng phân hủy được hay không?

Hàm destroy(X) nghĩa là chất có công thức hóa học là X nếu phản ứng phân hủy sẽ tạo thành những chất nào?

Hàm canAccess (s1, s2) nghĩa là hai chất có công thức hóa học là s1 và s2 có phản ứng được với nhau hay khôn?

Hàm access (s1, s2) là hàm mà tạo ra chất sản phẩm nếu cho s1 phản ứng với s2.

Hàm isGoal(X) là hàm kiểm tra xem trạng thái X có chứa tập hợp tất cả các chất có trong Y hay chưa?

* 1. **Chuỗi phản ứng:**

*Nhắc lại bài toán:* Bài toán: Cho một chuỗi các chất X (có cùng cation kim loại). Hãy liệt kê phương trình hóa học để hoàn thành chuỗi các chất đó.

Ví dụ: Fe → FeSO4 → FeCl2 → FeCl3 → Fe(OH)3 → Fe2O3

Các phương trình hóa học là:

Fe + H2SO4 = FeSO4 + H2 ↑

FeSO4 + BaCl2 = FeCl2 + BaSO4

2FeCl2 + Cl2 = 2FeCl3

FeCl3 + 3NaOH = Fe(OH)3 + 3NaCl

2Fe(OH)3 = Fe2O3 + 3H2O

*Phân tích:* Ta thấy các chất có cùng cation kim loại, nên chúng chỉ biến đổi từ loại này đến loại kia của chất.

Như, thì trường hợp ví dụ ở trên chuỗi biến đổi các loại của chúng là: kim loại 🡪 muối 🡪 muối 🡪 muối 🡪 bazo 🡪 oxit bazo. Dựa vào chuỗi này, ta có thể phân tích toàn bộ trường hợp có thể xảy ra và chất điều chế đi kèm như sau:

1. Kim loại → Bazơ: chất điều chế có thể là: H2O.

2. Kim loại → Oxit bazơ thì chất điều chế có thể là: O2.

3. Oxit bazơ→ Oxit bazơ (số oxi hóa cao hơn) thì dùng chất điều chế có thể là: O2.

4. Oxit bazơ → Bazơ thì chất điều chế có thể là: H2O.

5. Oxit bazơ → Muối chất điều chế có thể là: Axit hoặc Oxit axit.

6. Oxit bazơ → Kim loại thì chất điều chế có thể là: H2, C, CO.

7. Bazơ → Oxit bazơ chất điều chế có thể là: không có (nhưng có cách là nhiệt phân).

8. Bazơ → Muối thì chất điều chế có thể là: Axit hoặc Muối hoặc Oxit axit.

9. Muối → Bazơ thì chất điều chế có thể là: Bazơ.

10. Muối → Oxit bazơ thì chất điều chế có thể là: không có (nhưng có cách là nhiệt phân).

11. Muối → Muối thì chất điều chế có thể là: Axit hoặc Bazơ.

Nhờ phân tích như vậy, ta có thể xác định được chất điều chế là gì một cách nhanh chóng.

Sau khi có được toàn bộ các chất điều chế, ta chỉ việc lấy chất thứ nhất trong chuỗi phản ứng tác dụng với chất điều chế thứ nhất tạo ra được một phương trình hóa học, rồi lại lấy chất thứ hai đem phản ứng với chất điều chế thứ hai…

Vậy tóm lại, khi gặp bài toán chuỗi, ta thực hiện thuật toán sau:

Thuật toán:

Def hoàn thành chuỗi phản ứng(chuỗi):

Result = []

For each (u, v) in chuỗi:

\_type = type (u, v)

Dc = Find (\_type, u, v)

Result += phanUng (u, Dc)

Return Result

Đến đây, ta đã giải quyết được bài toán hoàn thành chuỗi phản ứng.

* 1. **Mô tả hiện tượng:**

*Bài toán:* Lần lượt cho các dung dịch vào trong một ống nghiệm. Hãy mô tả hiện tượng quan sát được.

*Phân tích:* Giả sử ta đang có một hỗn hợp các dung dịch ở trong một ống nghiệm. Bây giờ, việc ta cần làm là mô tả hiện tượng khi cho một chất khác vào trong ống nghiệm đó. Ta nhận thấy, khi cho một chất vào trong hỗn hợp đó, thì nó sẽ đi tác dụng với các chất trong hỗn hợp đó (gặp chất nào trước thì phản ứng trước). Và khi cho một lượng dư, thì nó sẽ phản ứng chừng nào các chất trong ống nghiệm không còn đủ lượng để phản ứng nữa thì thôi.

Ý tưởng: Ta coi hỗn hợp các dung dịch trong ống nghiệm là một danh sách các chất. Sau đó, mỗi lần cho một chất gì đó vào, ta thấy có 2 trường hợp xảy ra:

* Có xảy ra phản ứng: Ta lấy chất đó đi phản ứng với toàn bộ các chất trong danh sách ấy (chừng nào hết sạch thì thôi). Sau phản ứng xong, ta đem chất kết quả của các phản ứng vào trong dung dịch.
* Không có phản ứng nào xảy ra: Ta đem chất đó vào hỗn hợp các dung dịch.

Thuật toán:

Def mô tả hiện tượng:

hh = [H2O]

foreach a in addition:

temp = []

foreach b in hh:

If phanUng (a, b):

sanpham = phanUng (a, b)

temp += [sanpham]

if size(temp) == 0:

hh += temp

else

hh += [a]

end def

* 1. **Phân biệt các chất:**

*Bài toán:* Giả sử có nhiều bình chứ các dung dịch mất nhãn khác nhau. Ta cần phân biệt các dung dịch đó, mô tả hiện tượng.

*Ý tưởng:* Ta quy bài toán phân biệt về mô hình đồ thị, sau đó duyệt Depth First Search kết hợp một chút Heuristic để tìm ra chuỗi phân biệt nhanh nhất, phổ biến nhất.

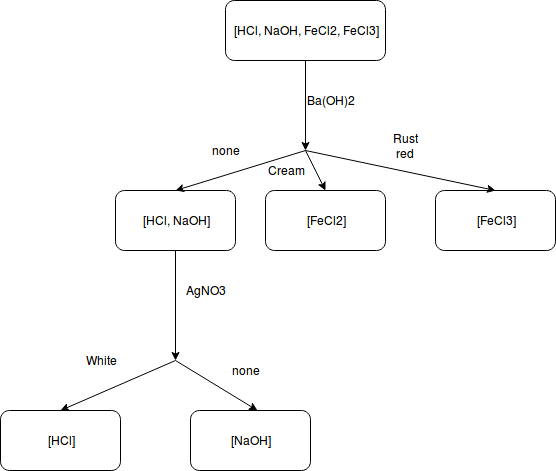
Quy về mô hình đồ thị như sau:

Ban đầu, cho tập hợp các chất X cần phân biệt - gọi trạng thái ban đầu này là S. Ta coi đây là đồ thị dạng cây với S là nút gốc.

Đồ thị G = (V, E) với:

Tập đỉnh V là tập hợp các chất cần phân biệt.

Tập cạnh E nối 2 đỉnh cha, con bằng cách phân biệt của tập hợp cha (ví dụ dưới đây sẽ rõ hơn).

Ví dụ dưới đây là *cây phân biệt* với tập hợp các chất là: HCl, NaOH, FeCl2, FeCl3.

Ta có nút gốc là nút có chứa [HCl, NaOH, FeCl2, FeCl3]. Sau đó, ta phân hoạch các chất trong nút gốc này ra thành các nút con, bằng dung dịch Ba(OH)2. Chất nào trong [HCl, NaOH, FeCl2, FeCl3] mà không phản ứng với Ba(OH)2 thì là chất [HCl, NaOH], nếu phản ứng với Ba(OH)2 sinh ra kết tủa màu Cream thì đó là FeCl2, nếu phản ứng ra màu Rust Red thì chất đó là FeCl3.

Nên nút có chứa chất [FeCl2] và [FeCl3] là nút lá. Ta còn cần phải phân biệt hai chất [NaOH, HCl] nữa. Tiếp theo, xét nút [NaOH, HCl] thì ta cần dùng chất AgNO3 để phân hoạch chúng thành 2 nút con. Chất nào phản ứng với AgNO3, mà ra kết tủa trắng thì đó là HCl, chất nào phản ứng không thấy hiện tượng thì đó là NaOH. Ta thấy, nút HCl và NaOH là nút lá. Vậy nên ta dừng việc phân biệt.

Tại đây, không phải ngẫu nhiên mà ta chọn Ba(OH)2 sau đó chọn AgNO3. Mà ta chọn sao cho số lượng nút trong phân hoạch càng lớn thì số lần phân biệt càng ít. Chương trình ưu tiên tìm cách phân biệt nhanh nhất có thể.

Số chất hóa học được chọn để phân hoạch: AgNO3, BaCl2, Ba(OH)2, HCl, H2SO4, Pb(NO3)2, CaCl2, Ca(OH)2.

Thuật toán:

Def DFS(u):

If size(u) <= 1: return;

If có\_thể\_phân\_biệt\_bằng\_màu:

v = phanHoach (mau, u)

foreach mỗi\_phân\_hoạch e in v: DFS(e)

return

if cho\_phép\_dùng\_quỳ\_tím:

v = phanHoach (quy, u)

foreach mỗi\_phân\_hoạch e in v: DFS(e)

return

// template = [AgNO3, BaCl2, Ba(OH)2, HCl, H2SO4, Pb(NO3)2, CaCl2, Ca(OH)2]

Choose = None

foreach temp in template:

v = phanHoach (temp, u)

if size(v) > size(Choose):

Choose = v

Foreach e in v: DFS(v)

End def

* 1. **Tìm chất:**

*Input:* Những câu truy vấn Yes, No (chatbot sẽ tự động yêu cầu).

*Output:* Sau khi hỏi đủ số lượng câu hỏi, chatbot sẽ xác định chất cho người dùng.

*Ý tưởng:* Ở phần này ta thực hiện tổ chức đa luồng, sẽ có hai luồng chạy song song nhau đó là luồng của con Bot xử lý thông tin người dùng, và một luồng nhận thông tin vào sự kiện của người dùng.

Cài đặt cho Bot thực hiện những câu hỏi để nhận biết một chất chưa các định cụ thể như sau:

* + - * Câu hỏi đầu tiên, sẽ liên quan đến quỳ tím. Mục đích khoanh vùng chất ở một phạm vi nhỏ hơn như là axit, bazo, muối.
      * Sau khi trả lời câu hỏi trên, Bot sẽ hỏi lần lượt các câu hỏi liên quan đến các thuốc thử thường dùng trong nhận biết hóa học, cụ thể là các chất “AgNO3, BaCl2, Ba(OH)2, HCl, H2SO4, Pb(NO3)2, CaCl2, Ca(OH)2”.
      * Để tránh những câu hỏi dư thừa ví dụ như đã xác định vùng chất là axit thì không cần hỏi lại các thuốc thử mang tính chất hóa học axit…
    - Người dùng cần trả lời chính xác những câu hỏi của Bot hỏi để Bot dễ dàng khoanh vùng một chất.

*Ví dụ minh họa:*

Bot: Bạn có quỳ tím không?

Person: Co

Bot: Ban thấy quỳ tím thay đổi như thế nào?

Person: Xanh

/\*Bot đã khoanh vùng được chất đó là một bazơ

Bot: Bạn có AgNO3 không?

Person: Không

Bot: Bạn có H2SO4 không?

Person: Co

Bot: Bạn quan sát thấy có hiện tượng gì?

Person: kết tủa

Bot: Bạn thấy kết tủa màu gì?

Person: trắng

/\* Bot khoanh vùng lại còn 2 chất đó là Ba(OH)2 và Ca(OH)2. \*/

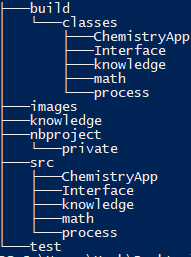
Đến đây hết câu hỏi nhưng còn 2 chất Bot sẽ xuất ra cho ta chất cần biết có thể là chất Ba(OH)2 và Ca(OH)2.

1. **Cài đặt**

Ta sẽ cài đặt biểu diễn tri thức và suy luận bằng ngôn ngữ lập trình Prolog.

Tuy nhiên, biểu diễn tri thức và suy luận chỉ là những nền tảng phía sau. Để xây dựng một ứng dụng hoàn chỉnh ta cần nhiều hơn thế. Ở phần cài đặt giao diện và những thao tác giải khác để quyết các bài toán ta dùng bằng ngôn ngữ lập trình Java.

Dưới đây là cấu trúc thư mục của ứng dụng:

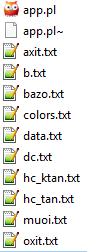


Ở đây, các thư mục như: **build, nbproject, test** là do Java tự tạo (chúng ta không cần quan tâm những thư mục này).

Các thư mục khác như: **images, knowledge, src** là những thư mục chính của ta.

Giải thích từng thư mục:

* **images**: Lưu những file ảnh hỗ trợ trong giao diện
* **src:** Lưu mã nguồn Java để chạy giao diện và các thao tác giải quyết các bài toán (cụ thể hơn ở dưới).
* **knowledge**: Có cấu trúc như sau:



Trong thư mục này, file **app.pl** chính là file biểu diễn tri thức và suy luận của ta được viết bằng Prolog. Các file khác hỗ trợ quá trình thông tin thêm cho các file mã nguồn.

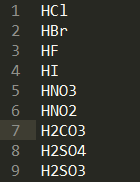
File axit.txt lưu danh sách những công thức hóa học của các axit

File bazo.txt lưu danh sách những công thức hóa học của các bazo

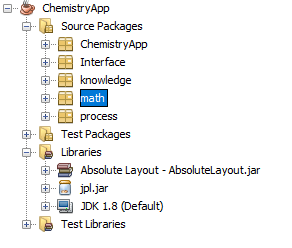
Tương tự, ta có muoi.txt lưu muối, oxit.txt lưu oxit, dc.txt lưu đơn chất, hc\_tan.txt ta lưu những danh sách hợp chất tan, hc\_ktan.txt lưu những hợp chất không tan.

Đặc biệt có 2 file khác khá đặc biệt là: data.txt và colors.txt (ta sẽ nói về 2 file này sau). Tiếp theo là file **app.pl~** là file được sinh ra trong quá trình chạy file app.pl (ta cũng không cần quan tâm đến file này).

Trong các file lưu các chất như axit.txt, ta lưu mỗi công thức hóa học ở dạng một chuỗi. Dưới đây là ví dụ:



(Mỗi dòng là một chuỗi đại diện cho công thức hóa học của một axit.)

Dưới đây là cấu trúc mã nguồn (cũng chính là những thứ có trong thư mục **src**)

Ở đây, ta chỉ xét **Source Packages** thôi. Vì những file khác là do Java tự động tạo ra, nên không cần quan tâm. Ta cũng cần quan tâm tới một file khác nữa đó là: **jpl.jar** đây là file thư viện giúp ta truy xuất Prolog từ Java.

Ta có, package ChemistryApp gồm có 3 class: functions.java, multiThread.java và main.java. Trong đó, class functions.java và multiThread.java được dùng để hỗ trợ đa luồng (nếu dùng chatbot). Class main.java thì chỉ chứa các lệnh gọi các file tạo giao diện.

Package Interface là nơi lưu các lớp hỗ trợ giao diện.

Package Knowledge chứa hai class là: knowledge.java và phanUng.java

* knowledge.java: gồm những thông tin sau:

Các biến và hằng:

public static List <String> allOfHC\_tan;

public static List <String> allOfHC\_ktan;

public static List <String> allOfAx;

public static List <String> allOfBz;

public static List <String> allOfMu;

public static List <String> allOfDC;

public static List <String> allOfOx;

public static List <String> all;

public static Map<String, Color> colors;

public static Map <String, Integer> oxh;

public static final String [] nt = {"H", "O", "Cl", "Br", "F", "I"};

public static final int [] it = {1, -2, -1, -1, -1, -1};

Như tên gọi, ta có biến allOfHC\_tan là nơi sẽ lưu tất cả các công thức hóa học của các hợp chất tan. Tương tự, allOfHC\_ktan: hợp chất không tan, allOfAx: hợp chất axit, allOfBz: hợp chất bazo, allOfMu lưu muối tan, allOfDC lưu đơn chất. allOfOx lưu những chất oxit. All lưu toàn bộ các chất có trong hệ thống.

colors là một map giúp ánh xạ từ tên màu (tiếng Anh) qua Kiểu dữ liệu màu.

Đến đây, ta cũng đề cập luôn đến file colors.txt ở trên: file này lưu như sau:

Cream

255 253 208

White

255 255 255

Nghĩa là: dòng đầu là tên màu ấy bằng tiếng Anh, dòng tiếp sẽ mô tả 3 thông số là R, G, B tương ứng với màu đó.

Tiếp theo là biến oxh nó giúp ánh xạ từ một tên nguyên tố sang số oxi hóa của chúng.

Có các hàm như sau:

// Hàm dùng để chuẩn bị trước dữ liệu khi vào thực thi các class khác

*public static void prepare ();*

//Hàm trả về một số thực là nguyên tử khối của chất đó

*public static float nguyenTo (String X);*

// Hàm lấy số oxi hóa của cation trong hợp chất T.

*public static int getOxh (String T);*

// Hàm trả về một ánh xạ từ nguyên tố sang số oxi hóa của chúng trong hợp chất X.

*public static Map<String, Integer> getAllOxh (String X);*

//Hàm trả về hai danh sách là tên nguyên tố và số oxi hóa của chúng trong toàn bộ danh sách hợp chất String [] L

*public static Pair<List<String>, List<Integer>> getAllOxhList (String [] L);*

// Hàm trả về chất cho và chất nhận electron

*public static Pair<String, String> chatChoNhan (List<String> name1, List<Integer> o1, List<String> name2, List<Integer> o2);*

// Hàm trả về nguyên tố tạo nên đơn chất có công thức hóa học là X

*public static String donChat (String X);*

// Hàm trả về True hay False nếu chất truyền vào là axit

*public static boolean axit (String X);*

// Tương tự

*public static boolean oxit (String X);*

*public static boolean bazo (String X);*

*public static boolean muoi (String X);*

// Hàm trả về màu quỳ tím của một chất

*public static String mauQuy (String X);*

//Hàm trả về danh sách các màu quỳ khi cho danh sách các chất X

*public static List<String> getAllMauQ(List<String> L);*

// Hàm kiểm tra một chất có kết tủa hay không?

*public static boolean kettua (String X);*

* phanUng.java: Là một lớp tĩnh có chứa các hàm sau

// Hàm trả về danh sách các chất sản phẩm khi phương trình hóa học truyền vào là v

*public static List <String> phanUng (String v);*

Package math: có chứa các lớp sau:

* Class LCS: là lớp để hỗ trợ tìm kiếm chuỗi con chung dài nhất giữa hai chuỗi
* Class Node: Sẽ được giới thiệu rõ hơn ở phần cài đặt bài toán Phân Biệt
* Class PwL (Process with List): là lớp chứa các thao tác với List trong Java, gồm một số các hàm như sau:

// Hàm copy một danh sách

*public static List<String> copy (List<String> L);*

// Hàm để chuyển đổi một mảng String sang một List String

*public static List<String> convertToList (String [] v);*

// Hàm kiểm tra một string X có ở trong danh sách L hay không?

*public static boolean checkIn (String X, List<String> L);*

// Hàm để kiểm tra một danh sách X có nằm trong một danh sách các List L không

*public static boolean checkIn (List <String> X, List <List<String>> L);*

// Hàm để kiểm tra một danh sách các phẩn tử phân biệt nhau đôi một hay không

*public static boolean unique (List <String> v);*

// Hàm phân hoạch L2 theo L1

*public static Map<String, List<String>> phanHoach (List <String> L1, List <String> L2);*

// Hàm kiểm tra hai List có giống nhau hay không

*public static boolean equals (List <String> L1, List<String> L2);*

Dưới đây, ta cùng đi sâu vào phần cài đặt.

* 1. **Biểu diễn tri thức**

Ở trên, ta đã nói sơ qua về các file axit.txt, bazo.txt và muoi.txt… nhưng chưa nói về cách tổ chức của file **data.txt** và file colors.txt. Ở mục này, ta đề cập đến file data.txt. File colors.txt sẽ sử dụng sau.

**data.txt** là file chứa những cấu trúc trong Prolog để giúp ta lấy đó làm dữ liệu giúp cho file **app.pl** suy luận. Cấu trúc trong file data.txt là do Prolog quy định, như sau:

Để bắt đầu định nghĩa về một loại vị từ, ta chèn vào file trên một dòng như sau:

:- dynamic tên\_vị\_từ/tham\_số\_của\_vị\_từ.

Sau đó, ta lần lượt thêm các vị từ như trong Prolog. Ví dụ, để định nghĩa một vị từ nguyên tố trong hóa học ta chèn vào file như sau:

:- dynamic nguyento/2.

nguyento ('H', 1).

nguyento ('He', 4).

nguyento ('Li', 7).

nguyento ('Be', 9).

…

Ta thấy, nguyento là tên vị từ, còn số 2 chính là muốn nói khai báo với Prolog vị từ này có 2 tham số.

Ở đây, ta lưu:

* Tham số thứ nhất là: kí hiệu nguyên tố
* Tham số thứ hai là: số hiệu nguyên tử của nguyên tố đó (nguyên tử khối)

Tương tự, ta cũng có các vị từ khác như sau:

:- dynamic kl/1.

kl ('Li').

kl ('Be').

kl ('Na').

kl ('Mg').

kl ('Al').

kl ('K').

…

:- dynamic klk/1.

klk ('Na').

klk ('K').

klk ('Li').

:- dynamic klkt/1.

klkt ('Be').

klkt ('Mg').

klkt ('Ca').

klkt ('Ba').

:- dynamic pk/1.

pk ('B').

pk ('C').

pk ('N').

:- dynamic anion/4.

anion ('Cl', 'Cl', 1, 35.5).

anion ('NO3', 'NO3', 1, 62).

anion ('NO2', 'NO2', 1, 46).

anion ('CO3', 'CO3', 2, 60).

anion ('SO4', 'SO4', 2, 96).

anion ('S\_2', 'S', 2, 32).

anion ('MnO4\_1', 'MnO4', 1, 119).

:- dynamic cation/4.

cation ('NH4', 'NH4', 1, 18).

cation ('Ag', 'Ag', 1, 108).

cation ('S\_4', 'S', 4, 32).

cation ('N\_5', 'N', 5, 14).

:- dynamic convert/2.

convert ('SO2', 'SO3').

convert ('SO3', 'SO4').

convert ('P2O5', 'PO4').

convert ('CO2', 'CO3').

:- dynamic dienhoa/3.

dienhoa ('Li', 'Li', 1).

dienhoa ('K', 'K', 2).

dienhoa ('Ba', 'Ba', 3).

dienhoa ('Ca', 'Ca', 3).

:- dynamic halogen/1.

halogen ('F').

halogen ('Cl').

halogen ('Br').

halogen ('I').

:- dynamic next/3.

next ('O', 'Fe', 'Fe\_2').

next ('O', 'Fe\_2', 'Fe\_3').

next ('O', 'S\_4', 'S\_6').

:- dynamic dangphantu/1.

dangphantu ('H').

dangphantu ('Br').

:- dynamic khongtan/2.

khongtan ('AgBr', 'cream').

khongtan ('AgCl', 'white').

khongtan ('AgI', 'yellow').

:- dynamic khongtontai/3.

khongtontai ('AgOH', ['Ag2O', 'H2O']).

khongtontai ('H2SO3', ['SO2', 'H2O']).

khongtontai ('H2CO3', ['CO2', 'H2O']).

khongtontai ('NH4OH', ['NH3', 'H2O']).

:- dynamic khi/1.

khi ('H2').

khi ('O2').

khi ('Cl2').

:- dynamic mau/2.

mau ('AgBr', 'Cream').

mau ('AgCl', 'White').

mau ('AgI', 'Yellow').

mau ('Ag3PO4', 'White').

Bên cạnh những vị từ biểu diễn tri thức này, ở trong file app.pl ta cũng có rất nhiều luật được xây dựng để suy ra chất kết quả từ nhiều chất tham gia và để điều chế các chất.

Một số dưới đây là các luật phản ứng cơ bản:

% oxit axit + H2O = dd axit

pu(X, 'H2O', [Z]):-

oxitAxit(X),

convert(X, A),

name(Z, 'H', A), !.

% oxit cua kim loai kiem, kiem tho + H2O = dd Bazo

pu(X, 'H2O', [Z]):-

% kiem tra oxit bazo

name(X, C, 'O'),

klk\_t(C),

name(Z, C, 'OH'), !.

%pu voi oxi

%don chat + oxi

pu('H2', 'O2', ['H2O']):- !.

pu(X, 'O2', [Z]):-

name(X, D, V),

isnumber(V),

\+ dangphantu(D),

name(Z, D, 'O'), !.

%oxit axit + bazo = muoi + h2o

pu(X, Y, [Z, 'H2O']):-

oxitAxit(X),

bazo(Y),

convert(X, A),

name(Y, C, \_),

name(Z, C, A), !.

%axit + bazo = muoi + h2o

pu(X, Y, [Z, 'H2O']):-

axit(X),

bazo(Y),

name(X, \_, A),

name(Y, C, \_),

name(Z, C, A), !.

* 1. **Tính phân tử khối của một chất:**

Ở trong file ptk.java có các biến sau:

String replace;//chuỗi hướng dẫn cho người dùng.

Map <String, Integer> // lưu trữ tên và số lượng nguyên tố

List <String> str\_ = new Vector<String> ();// lưu trữ tên nguyên tố

List <Integer> cnt\_ = new Vector<Integer> ();// lưu trữ số lượng nguyên tố trong công thức hóa học.

Các hàm sau:

// Hàm kiểm tra một kí tự là con số

*public static boolean isNum (char c);*

*public static String next (String X, int index, int n);*

// Hàm tách một chuỗi ra thành các nguyên tố và số lượng của chúng

*public static Map<String, Integer> split (String X);*

// Hàm lấy phần mô tả

*public static String getDescription (String X);*

* 1. **Cân bằng phương trình hóa học:**

Cách 1: Cân bằng đại số:

Cấu trúc dữ liệu:

* + List < String > tt; //Danh sách các chất tham gia
  + List < String > sp; // Danh sách các chất sản phẩm
  + List < String > TT1; //Danh sách các nguyên tố trong các chất tham gia
  + List < Integer > TT2; //Danh sách số lượng ứng với các nguyên tố ở list TT1
  + List < String > SP1; //Danh sách các nguyên tố trong các chất sản phẩm
  + List < Integer > SP2; //Danh sách số lượng ứng với các nguyên tố ở list SP1
  + List < String > VT; //Danh sách các nguyên tố không trùng lặp
  + List < String> e; //Danh sách lưu hệ số tìm được

Các hàm hỗ trợ thực hiện:

private Pair < List, List > tachchat (String s){

/\* Hàm này hỗ trợ thực hiện tách các chất tham gia và

các chất sản phẩm ra khỏi phương trình phản ứng.

Hàm input là phương trình phản ứng

Hàm trả về một Pair < List, List > với

key list là các chất tham gia,

value list là các sản phẩm \*/

* Bước 1: Chia chuỗi s thành nhiều chuỗi con dựa bằng hàm split () trong thư viện của java.
* Bước 2: Tìm vị trí của dấu ‘=’ trong list chuỗi con của s.
* Bước 3: Chạy qua list chuỗi con của s nếu phần tử string không phải là dấu ‘+’ trước dấu ‘=’ thì thêm vào danh sách chất tham gia, ngược lại được thêm vào danh sách các chất sản phẩm.

}

private List < String > getSet (Map < String, Integer > st) {

/\* Hàm này thực hiện lấy key ở Map < String, Integer >

Input: Map < String, Integer >

Ouput: List < String > \*/

List < String > res = new Vector < String > ();

for each st.keySet ():

Add vào res;

return res;

}

private List < String > getValue (Map < String, Integer > st) {

/\* Hàm này thực hiện lấy value ở Map < String, Integer >

Input: Map < String, Integer >

Ouput: List < String > \*/

List < Integer > res = new Vector < String > ();

for each st.values ():

Add vào res;

return res;

}

private List< Integer > giaimatran (int [][]a, int n, int m){

/\* Hàm này thực hiện giải ma trận có n hàng m cột

Input: ma trận, số hàng, số cột

Output: List < Integer > là các hệ số tìm được \*/

* Bước 1: Đưa ma trận về tam giác trên:

int t = 0;

while (t < n - 1){

for (int j = 0; j < m; j++){

if (a[t][j] == 0)

Đổi vị trí của hàng t và hàng t + 1;

else{

int v = a[t][j]; //Lưu giá trị a[t][j]

for (int k = t + 1; k < n; k++){

if (a[k][j] != 0)

int u = a[k][j];

for (int z =j; z < m; z++)

a[k][z] = a[k][z] \* v–a[t][z] \* u;

}

}

t++;

if (t >= n – 1) break;

}

}

* Bước 2: Tìm nghiệm

int [] x = new int [100]; //Lưu hệ số

Giả sử nghiệm thứ m – 1 có giá trị là phần tử a[m – 2][m-2]

x[m - 1] = a[m - 2][m - 2]; (Rút ra từ việc giải nhiều ma trận)

for (int i = m – 2; i >= 0; i--){

x[i] = 0;

for each phần tử của x

if (phần tử thứ j của x != i )

x[i] -= a[i][j] \* x[j];

if (a[i][i] != 1 && i < n - 1) //Đề phòng ra nghiệm lẻ

for each phần tử của x

mỗi phần tử nhân với a[i][i];

x[i] /= a[i][i];

}

* Bước 3: Tối giản hệ số tìm được

}

Thuật toán:

public String canbang (String s){

* Bước 1: Tìm các chất tham gia và sản phẩm trong chuỗi dùng hàm tachchat(String).
* Bước 2: Ở mỗi chất, ta thực hiện tìm các nguyên tố và số lượng của nguyên đó. Để thực hiện ta sử dụng hàm split (String) ở class ptk.java. Nếu chất ở bên các chất tham gia thì các nguyên tố được thêm vào TT1 và số lượng được thêm vào TT2, tương tự các chất bên sản phẩm thì thêm vào SP1, SP2.
* Bước 3: Thành lập ma trận, ở bước này ta dùng mảng VT để lưu các nguyên tố không trùng nhau, vì ta thành lập hệ phương trình dựa trên nguyên tố với mỗi nguyên tố ta thành lập được một phương trình.

for (int i = 0; i < TT1.size(); i++){

VT.add (TT1.get(i));

if (i > 0)

for each phần tử của VT:

Nếu phần tử vừa được thêm đã có trong VT:

remove(VT.size() – 1);

}

int [][] matrix;

for each chất trong tt{

Xác định nguyên tố và số lượng nguyên tố của mỗi chất trong tt;

for each mỗi nguyên tố{

for each phần tử của VT:

if (nguyên tố == phần tử thứ t của VT)

matrix[t][i] = số lượng nguyên tố đó;

}

}

for each chất trong sp{

Xác định nguyên tố và số lượng nguyên tố của mỗi chất trong sp;

for each mỗi nguyên tố{

for each phần tử của VT:

if (nguyên tố == phần tử thứ t của VT)

matrix[t][i+tt.size()] = -số lượng nguyên tố đó;

}

}

* Bước 4: Giải ma trận vừa tìm được bằng hàm giaimatran (int [][], int, int).
* Bước 5: Đưa hệ số tìm được ở bước 4 lên phương trình.

}

Cách 2: Cân bằng electron:

Cấu trúc dữ liệu:

* + List < String > tt; //Danh sách các chất tham gia
  + List < String > sp; // Danh sách các chất sản phẩm
  + List < String > TT1; //Danh sách các nguyên tố trong các chất tham gia
  + List < Integer > TT2; //Danh sách số oxh ứng với các nguyên tố ở list TT1
  + List < Integer > TT3; //Danh sách số lượng ứng với các nguyên tố ở list TT1
  + List < String > SP1; //Danh sách các nguyên tố trong các chất sản phẩm
  + List < Integer > SP2; //Danh sách số oxh ứng với các nguyên tố ở list SP1
  + List < Integer > SP3; //Danh sách số lượng ứng với các nguyên tố ở list SP1
  + List < String > VT; //Danh sách các nguyên tố không trùng lặp
  + List < String> e; //Danh sách lưu hệ số tìm được

Thuật toán:

public String canBangElectron (String s){

* Bước 1: Tìm các chất tham gia và sản phẩm trong chuỗi dùng hàm tachchat(String).
* Bước 2: Ở mỗi chất, ta thực hiện tìm các nguyên tố, số oxi - hóa và số lượng của nguyên đó. Để thực hiện ta sử dụng hàm split (String) ở class ptk.java, việc lấy số oxi – hóa của chất dùng hàm getAllOxh(String) ở class knowledge.java. Nếu chất ở bên các chất tham gia thì các nguyên tố được thêm vào TT1 và số oxi - hóa được thêm vào TT2, số lượng được thêm vào TT3, tương tự các chất bên sản phẩm thì thêm vào SP1, SP2, SP3.
* Bước 3: Tìm các chất có sự thay đổi electron sau quá trình phản ứng.

for each phần tử của TT1:

for each phần tử SP1:

if (phần tử TT1 == phần tử SP1 && số oxi–hóa khác nhau)

Thêm vào danh sách temp;

* Bước 4: Xác định quá trình khử – quá trình oxh và thực hiện thăng bằng electron.

int []a; // lưu số electron nhường và electron nhân

int k = 0;

for(int i = 0; i < temp.size() ; i += 2){

a[k] = số oxh của temp.get(i + 1) - số oxh của temp.get(i);

if(số lượng của temp.get(i+1)!=số lượng của temp.get(i))

a[k] \*= số lượng temp.get(i + 1)\*số lượng temp.get(i);

Cập nhật lại số lượng của temp.get(i+1) và temp.get(i);

}

Tìm tổng electron nhường là các phân tử của a là số âm, tổng eclectron nhận là các phần tử của a là số dương.

Tìm ước chung của tổng electron nhường và electron nhận.

Tổng electron nhường và tổng electron nhận chia cho ước.

Cập nhật lại số lượng của phần tử của các nguyên tốt trong temp

for(int i = 0; i < temp.size() ; i += 2){

if (a[k] > 0)

Số lượng phần tử của temp tại i, i + 1 nhân thêm -tổng electron nhường;

else

Số lượng phần tử của temp tại i, i + 1 nhân thêm tổng electron nhận;

}

* Bước 5: Đưa hệ số lên phương trình phản ứng và thực hiện cân bằng các nguyên tố không thay đổi số oxi – hóa.

int []hso;

int k = 0;

for each chất trong tt{

Xác định nguyên tố và số oxh nguyên tố của mỗi chất trong tt;

for each mỗi nguyên tố{

for each phần tử của temp:

if (nguyên tố==phần tử thứ t của temp && có số oxh bằng nhau)

hso[k] = số lượng phần tử thứ t của temp;

}

k++;

}

for each chất trong sp{

Xác định nguyên tố và số oxh nguyên tố của mỗi chất trong sp;

for each mỗi nguyên tố{

for each phần tử của temp:

if (nguyên tố==phần tử thứ t của temp && có số oxh bằng nhau)

hso[k] = số lượng phần tử thứ t của temp;

}

k++;

}

Dùng mảng VT lưu lại các nguyên tố không lặp lại như ở cách 1.

Tính lại số lượng các nguyên tố có trong VT ở bên các chất sản phẩm. int []ks;

for each chất trong sp{

Xác định nguyên tố và số lượng nguyên tố của mỗi chất trong sp;

for each mỗi nguyên tố{

for each phần tử của VT:

if (nguyên tố==phần tử thứ t của temp)

ks[t]=số lượng phần tử thứ t của temp\*hs[k];

}

k++;

}

Cân bằng lại các nguyên tố có trong VT ở bên các chất tham gia.

for each chất trong tt{

Xác định nguyên tố và số lượng nguyên tố của mỗi chất trong tt;

for each mỗi nguyên tố{

for each phần tử của VT:

if (nguyên tố==phần tử thứ t của temp&&số lượng nguyên tố\*hso[k] < ks[t])

hs[k] = ks[t];

}

k++;

}

Cân bằng lại các nguyên tố H, O có trong phương trình.

Thực hiện lại đếm số lượng nguyên tố của mỗi bên, và cân bằng lại.

}

* 1. **Điều chế:**

Theo cấu trúc file, file dieuChe.java này sẽ giải quyết bài toán điều chế

Class này có chứa:

* public static boolean found;
* private List <String> chats;
* private List <String> kq;
* private edge[] trace;

Biến found được dùng để trả về trạng thái có điều chế được hay không (sử dụng khi một chương trình, class khác gọi vào)

Biến chats lưu danh sách những chất trong tập X. Các chất được lưu dưới dạng chuỗi.

Biến kq lưu danh sách những chất trong tập Y. Và biến mảng trace, biến này dùng để lưu vết trong quá trình tìm đường đi ngắn nhất.

edge là lớp một cấu trúc (được đặt ở package math) xây dựng riêng cho việc này, chứa:

* private int id;
* private String pthh;

Khi xét cạnh (u, v) thì:

* id chính là vị trí của trạng thái u trong Queue và pthh là phương trình hóa học nối giữa hai đỉnh u và v.
* mảng trace sẽ lưu lại vị trí của trạng thái trước của trạng thái hiện hành.

Ví dụ: trace[u] sẽ là một edge, mà trong đó, trace[u].id chính là vị trí trước của u trong Queue. Nhờ vậy mà ta có thể truy vết một cách nhanh chóng.

Các hàm trong lớp dieuChe.Java như sau:

// Hàm trả về trạng thái có phải là kết thúc hay không? Tức là có chứa Y hay không.

private boolean isGoal (List <String> v){

for (String st : kq){

if (!PwL.checkIn(st, v)) return false;

return true;

}

// Hàm trả về id là vị trí của trạng thái cuối cùng (hoặc không tìm thấy) trong Queue.

public int find(): thực hiện như thuật toán đã nêu trên.

// Hàm trả về toàn bộ các phương trình hóa học để điều chế từ X tới Y.

public List<String> showPath(int id){

List <String> res = new Vector<String>();

int k = id;

while (k > 0){

res.add(trace[k].getPt());

k = trace[k].getId();

}

int n = res.size();

for (int i=0; i<n / 2; i++){

String st = res.get(i);

res.set(i, res.get(n - i - 1));

res.set(n - i - 1, st);

}

return res;

}

* 1. **Chuỗi phản ứng:**

Ta cài đặt các vị từ trong Prolog, sau đó dùng Java để lấy ra những chất điều chế. Tiếp đến, ta cho những chất điều chế đem phản ứng với chất đầu tiên trong chuỗi phản ứng, … để tạo thành cách phương trình hóa học:

Các vị từ trong Prolog là:

*Luật 1: Kim loại → Muối: chất trung gian là axit*

chuoi(X, Y, R):-

kl(X),

muoi(Y),

name(Y, C, A),

cation(C, X, \_, \_),

name(R, 'H', A),

pu(X, R, L),

member(Y, L), !.

*Luật 2: Kim loại → Muối: chất trung gian là đơn chất phi kim*

chuoi(X, Y, R):-

kl(X),

muoi(Y),

name(Y, \_, P),

anion(P, PP, \_, \_),

pk(PP),

xaydung(P, R),

pu(X, R, L),

member(Y, L), !.

*Luật 3: Kim loại → Bazơ: chất trung gian là: H2O*

chuoi(X, Y, 'H2O'):-

klk\_t(X),

bazo(Y),

name(Y, C, \_),

\+ dif(C, X), !.

*Luật 4: Kim loại → Oxit bazơ: Chất trung gian là: O2*

chuoi(X, Y, 'O2'):-

oxitBazo(X),

oxitBazo(Y),

name(X, C1, \_),

name(Y, C2, \_),

dif(C1, C2),

next('O', C1, C2), !.

*Luật 5: Oxit bazơ → Bazơ*

chuoi(X, Y, 'H2O'):-

oxitBazo(X),

bazo(Y),

name(X, C1, \_),

name(Y, C2, \_),

\+ dif(C1, C2),

klk\_t(C1), !.

*Luật 6: Oxit bazơ → Muối : chất trung gian là: axit*

chuoi(X, Y, R):-

oxitBazo(X),

muoi(Y),

name(X, C1, \_),

name(Y, C2, A),

\+ dif(C1, C2),

name(R, 'H', A).

*Luật 7: Oxit bazơ → Muối: chất trung gian là: Oxit axit*

chuoi(X, Y, R):-

oxitBazo(X),

muoi(Y),

name(X, C1, \_),

name(Y, C2, A),

\+ dif(C1, C2),

convert(R, A), !.

*Luật 8: Oxit bazơ → Kim loại : chất trung gian là: H2, C, CO*

chuoi(X, Y, 'H2'):-

oxitBazo(X),

name(X, C, \_),

cation(C, C1, \_, \_),

\+ dif(C1, Y),

pu(X, 'H2', L),

member(Y, L), !.

chuoi(X, Y, 'CO'):-

oxitBazo(X),

name(X, C, \_),

cation(C, C1, \_, \_),

\+ dif(C1, Y),

pu(X, 'CO', L), member(Y, L), !.

*Luật 9: Bazơ → Oxit bazơ: chất trung gian là: Không có, chỉ là phản ứng nhiệt phân*

chuoi(X, Y, 'none'):-

bazo(X),

oxitBazo(Y),

khongtan(X, \_),

name(X, C1, \_),

name(Y, C2, \_),

max('O', C1, C2),

pu(X, L), member(Y, L), !.

*Luật 10: Bazơ → Muối: Chất trung gian là: axit*

chuoi(X, Y, R):-

bazo(X),

muoi(Y),

name(X, C1, \_),

name(Y, C2, A),

\+ dif(C1, C2),

name(R, 'H', A),

pu(R, X, L),

member(Y, L), !.

*Luật 11: Bazơ → Muối : Chất trung gian là: muối*

chuoi(X, Y, R):-

bazo(X),

muoi(Y),

name(X, C1, \_),

name(Y, C2, A),

\+ dif(C1, C2),

name(R, 'Na', A),

pu(X, R, L),

member(Y, L), !.

*Luật 12: Bazơ → Muối : Chất trung gian là: oxit axit*

chuoi(X, Y, R):-

bazo(X),

muoi(Y),

name(Y, \_, A),

convert(R, A),

pu(X, R, L),

member(Y, L), !.

*Luật 13: Muối → Bazơ: Chất trung gian là: NaOH*

chuoi(X, Y, 'NaOH'):-

muoi(X),

bazo(Y),

pu('NaOH', X, L),

member(Y, L), !.

*Luật 14: Muối → Oxit bazơ : Chất trung gian không có, chỉ có phản ứng đốt cháy*

chuoi(X, Y, 'none'):-

muoi(X),

oxitBazo(Y),

pu(X, L),

member(Y, L), !.

*Luật 15: Muối → Muối: chất trung gian là: Axit*

chuoi(X, Y, R):-

muoi(X),

muoi(Y),

name(Y, \_, A),

cation(C, \_, \_, \_),

name(R, C, A),

\+ khongtan(R, \_),

pu(X, R, L),

member(Y, L), !.

chuoi('FeCl2', 'FeCl3', 'Cl2').

chuoi('FeCl3', 'FeCl2', 'Fe').

*Dưới đây là luật để rút ra chuỗi các chất điều chế khi truyền vào một chuỗi điều chế:*

dieuche([\_], []).

dieuche([H1, H2 | T1], [H | T]):-

chuoi(H1, H2, H), dieuche([H2 | T1], T), !.

Trong file chuoiPhanUng.java có các hàm:

// Hàm được dùng để tạo query trong Prolog

*public static String makeQuery (String X);*

*// Hàm dùng để gọi Prolog tìm kiếm giúp ta các chất điều chế*

*public static List<String> find (String X);*

// Hàm để tạo ra các phương trình phản ứng và trả về một danh sách các phương trình phản ứng

*public static List <String> showAll (String X);*

* 1. **Mô tả hiện tượng:**

Class hienTuong.java có chứa các biến như sau:

// Dùng để lưu danh sách các chất đang hiện có

private List <String> ds;

Các hàm như sau:

// Hàm để thêm một chất vào hỗn hợp các chất

public void add(String X){

boolean added = false;

List <String> tmp = new Vector<String>();

for (String Y : this.ds){

List <String> result = phanUng.phanUng(X + " " + Y);

if (result == null){

if (!added) push(tmp, X);

push(tmp, Y);

}

else{

added = true;

for (String one : result)

push(tmp, one);

}

}

this.ds = tmp;

}

* 1. **Phân biệt các chất**

Ở file phanBiet.java có các biến sau:

private boolean quyAllowed;

private Tree tree;

public static Vector<String> hetcach;

Biến quyAllowed nghĩa là cho phép dùng quỳ tím để phân biệt hay không.

Biến tree để lưu cây phân biệt (cấu trúc dữ liệu này ta sẽ nói sau, ở phía dưới)

Biến hetcach để lưu tập hợp các chất mà chương trình không có khả năng phân biệt.

Các hàm sau:

public void buildTree(){

DFS(tree.getRoot());

}

public void travel(){

tree.travel();

}

// Hàm này dùng để xây dựng cây phân biệt, cách làm giống như thuật toán đã mô tả ở phần phân tích ý tưởng thuật toán

public void DFS(Node u){

List <String> chats = u.getChats();

// Nếu có 1 con thì có thể xem như đã phân biệt được.

if (chats.size() == 1) return;

boolean found = false;

// Có thể phân biệt màu nè

List <String> mau = knowledge.getColor(chats);

// Nếu biết hết màu

if (PwL.available(mau)){

Map<String, List<String>> ph = PwL.phanHoach(mau, chats);

// Có nhiều hơn 1 màu

if (ph.keySet().size() > 1){

// set edge with mau

u.setEdge("mau");

for (String key : ph.keySet()){

u.addChild(key, new Node(ph.get(key)));

}

found = true;

// visit all childs

for (Node child : u.getChilds().values()){

DFS(child);

}

}

}

if (found) return;

// Giờ chúng ta sẽ xem có phân biệt được bằng quỳ tím không nà.

if (quyAllowed){

System.out.println(chats);

//System.out.println("vao quy");

List <String> mauQ = knowledge.getAllMauQ(chats);

Map<String, List<String>> ph = PwL.phanHoach(mauQ, chats);

// Có nhiều hơn 1 màu

if (ph.keySet().size() > 1){

for (String key : ph.keySet()){

u.addChild(key, new Node(ph.get(key)));

}

found = true;

// set edge with quy

u.setEdge("quy");

found = true;

// visit all childs

for (Node child : u.getChilds().values()){

DFS(child);

}

}

}

if (found) return;

// Chọn một chất trong template để phân biệt coi nào.

// Chọn chất nào mà có thể phân hoạch ra nhiều nhóm nhất.

Map <String, List<String>> choose = null;

int Max = 0;

String edge = "";

for (String spt : templates){

List <String> label = knowledge.productLabelList(chats, spt);

Map <String, List<String>> ph = PwL.phanHoach(label, chats);

if (Max < ph.keySet().size()){

Max = ph.keySet().size();

choose = ph;

edge = spt;

}

}

if (Max < 2){

// Hết cách

for (String st : chats) hetcach.add(st);

return;

}

u.setEdge(edge);

for (String key : choose.keySet()){

u.addChild(key, new Node(choose.get(key)));

}

found = true;

for (Node child : u.getChilds().values()){

DFS(child);

}

}

Cấu trúc dữ liệu Node:

private List <String> chats;

private String edge;

private Map<String, Node> childs;

Cấu trúc dữ liệu này như một node trên cây đã mô tả.

Biến chats lưu các chất mà tại nút đó đang cần phải phân biệt.

Biến edge lưu cách thức phân biệt

Biến childs lưu lại các phân hoạch của các phân biệt đó áp dụng trên các chats

Các hàm:

// Để đặt giá trị cho cạnh là X

public void setEdge(String X){

edge = X;

}

// thêm đứa con X vào trong key

public void addChild(String key, Node X){

childs.put(key, X);

return;

}

public List<String> getChats(){

return chats;

}

public Map<String, Node> getChilds(){

return childs;

}

public String getEdge(){

return edge;

}

Cấu trúc dữ liệu Tree:

Gồm có các biến sau:

private Node root;

private List<Node> all;

Biến root để lưu nút gốc của cây.

Biến all để lưu lại trình tự các nút theo thứ tự duyệt sâu từ root của cây.

Các hàm:

// Hàm gán tạo một Node mới cho Root

public void setRoot(List <String> L){

root = new Node(L);

}

// Hàm thêm một Node vào nhãn label của Node parent

public void addNode(Node X, String label, Node parent){

parent.addChild(label, X);

}

public Node getRoot(){

return root;

}

public void DFS(Node u){

for (Node child: u.getChilds().values()){

if (child.getChats().size() == 1) continue;

DFS(child);

}

}

public void DFS2(Node u){

all.add(u);

for (Node child: u.getChilds().values()){

if (child.getChats().size() == 1) continue;

DFS2(child);

}

}

public void travel(){

DFS(root);

}

public List<Node> getTravel(){

all = new Vector<Node>();

DFS2(root);

return all;

}

* 1. **Tìm chất:**

Bot nhận thông tin và sự kiện:

Void waitBot(){

Boolean check = false; // Biến check thể hiện người dùng chưa / /truyền thông tin gì vào cả.

While( !check ){

‘không làm gì cả’

}

}

Bot xử lý thông tin:

Void Bot(){

- Bot sẽ lần lượt hỏi những câu hỏi để làm rõ vấn đề, sau mỗi câu hỏi Bot sẽ chờ và bắt thông tin từ người dùng (ở đây sẽ dùng waitBot() để chờ).

- Khi nhận được tín hiệu từ người dùng, nếu tín hiệu đó sai ( nghĩa là người dùng đã nhập câu trả lời không như ý muốn của Bot) sẽ bắt người dùng nhập lại đến khi tin hiệu đó đúng.

- Tín hiệu đúng thì Bot sẽ thực hiện khoanh vùng các chất lại

- Bot sẽ hỏi cho đến khi vùng chất được khoanh vùng chỉ còn lại 1 chất hoặc không có chất nào hoặc là số câu hỏi của Bot đã hết. Nếu câu hỏi hết mà vùng chất được khoanh vùng còn rất nhiều chất thì sẽ xuất các chất ra cho người dùng đoán tiếp.

}

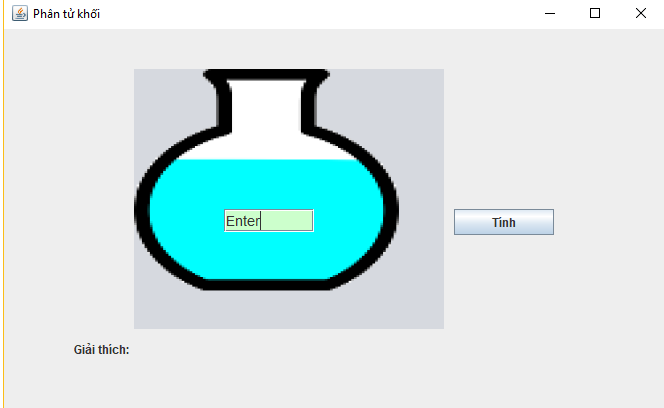
1. **Phân công công việc**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ tên | Chức vụ | Công việc cụ thể được phân công |
| 1 | Nguyễn Quốc Tài | Trưởng nhóm | - Phân tích và xây dựng kiến thức ứng dụng hóa học.  - Xây dựng chức năng tính phân tử khối.  - Xây dựng chức năng chuỗi phản ứng.  - Viết báo cáo. |
| 2 | Vũ Gia Khiêm  Huỳnh Gia Phát | Thành viên | - Xây dựng chức năng cân bằng phương trình hóa học.  - Xây dựng chức năng điều chế chất.  - Xây dựng chức năng mô tả hiện tượng.  - Xây dựng chức năng phân biệt các chất.  - Xây dựng chức năng chatbot tìm chất.  - Viết báo cáo. |

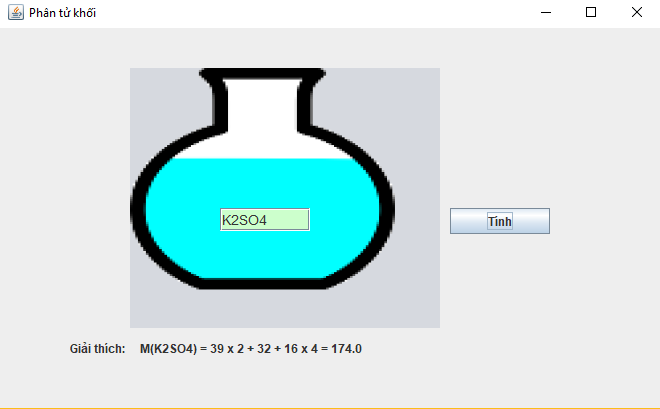
1. **Demo.**
   1. **Giao diện chính**



* 1. **Tính phân tử khối của một chất**



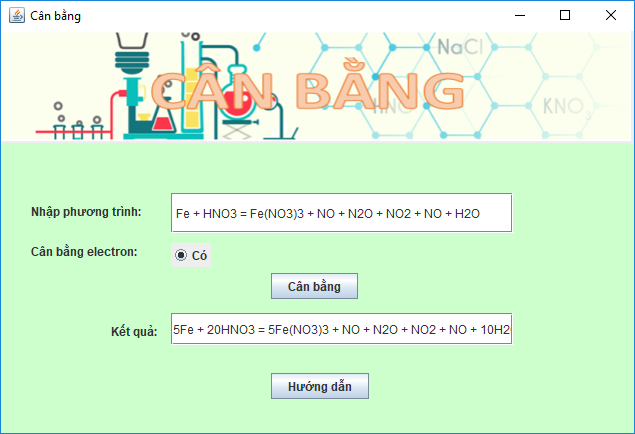
Kết quả:



* 1. **Cân bằng phương trình hóa học**



Kết quả:



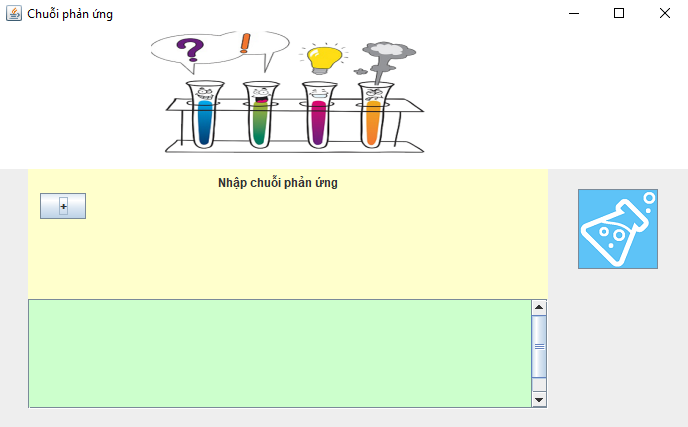
* 1. **Điều chế**



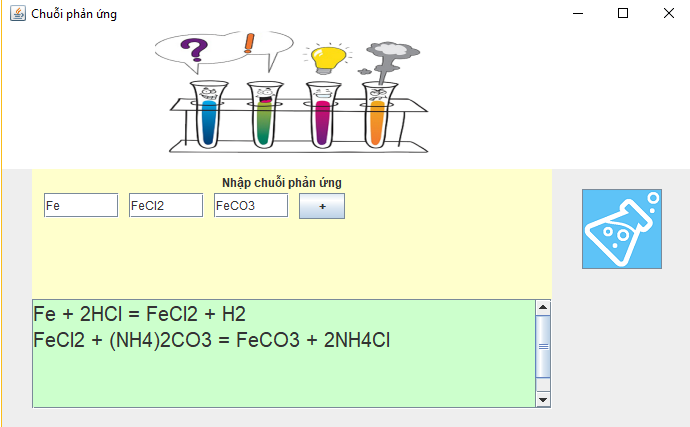
Kết quả:



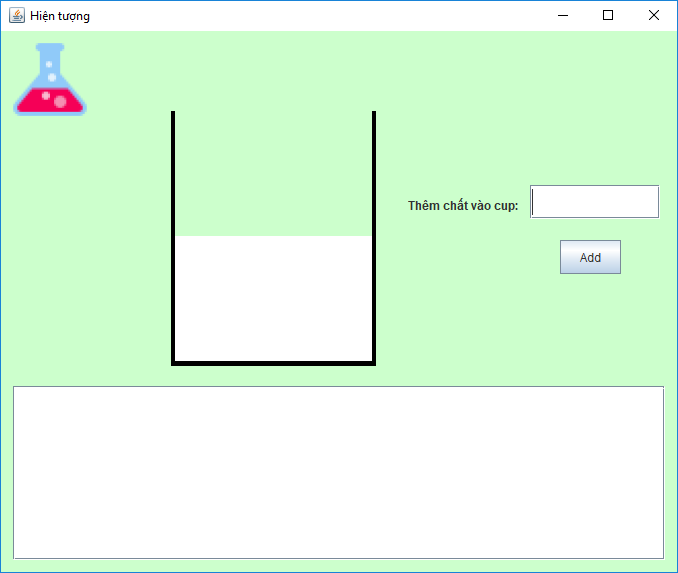
* 1. **Chuỗi phản ứng**



Kết quả:



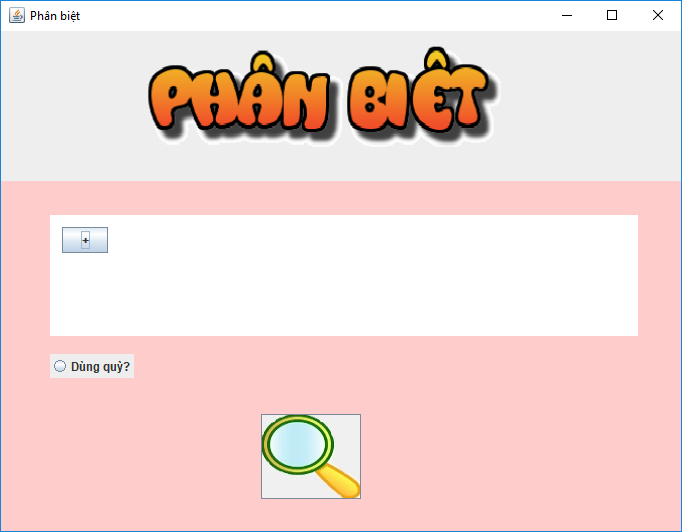
* 1. **Mô tả hiện tượng**



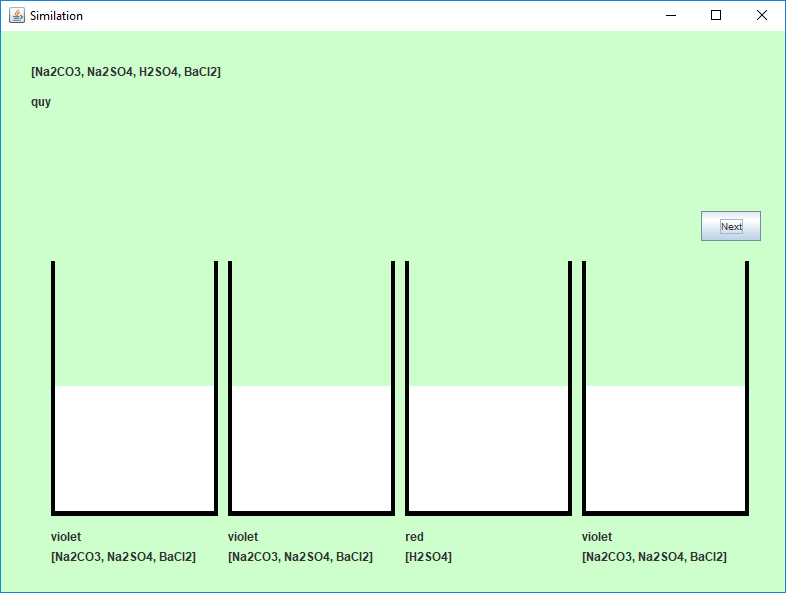
Kết quả:

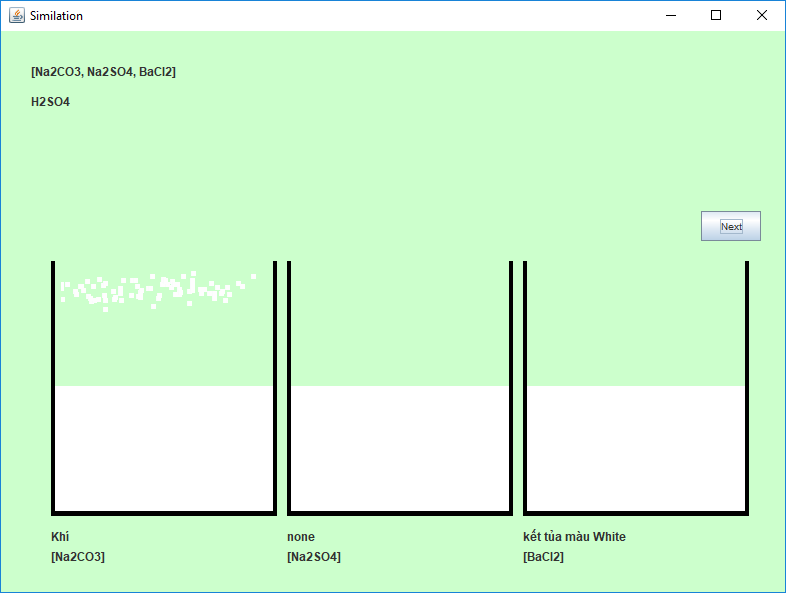


* 1. **Phân biệt các chất**

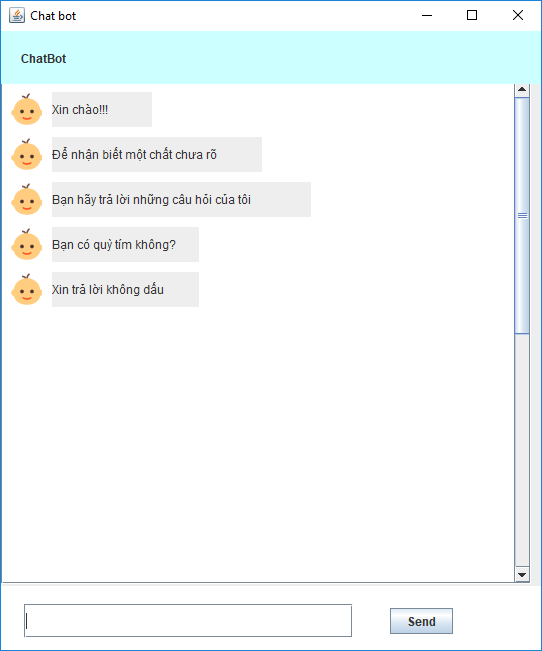


Kết quả:





* 1. **Tìm chất**



Kết quả:

