0. Bài toán đặt ra, mục tiêu:

- Tri thức con người gồm nhiều loại khác nhau như kiến thức về toán học, vật lý, cuộc sống hằng ngày… Bài toán đặt ra là làm sao để có thể lưu trữ và biểu diễn tri thức con người một cách tự nhiên và dễ dàng nhất trong máy tính.
- Việc xây dựng cấu trúc dữ liệu cho tri thức còn phụ thuộc vào đặc thù của loại tri thức đó như thế nào vì tri thức có nhiều loại khác nhau. Dưới đây là ý tưởng cơ bản cũng như xây dựng thuật giải, code để hiện thực hóa vấn đề trên một miền tri thức nhỏ đó là dạng cấu trúc **Nếu … Thì**.

1. Các cấu trúc dữ liệu cần thiết cho bài toán:

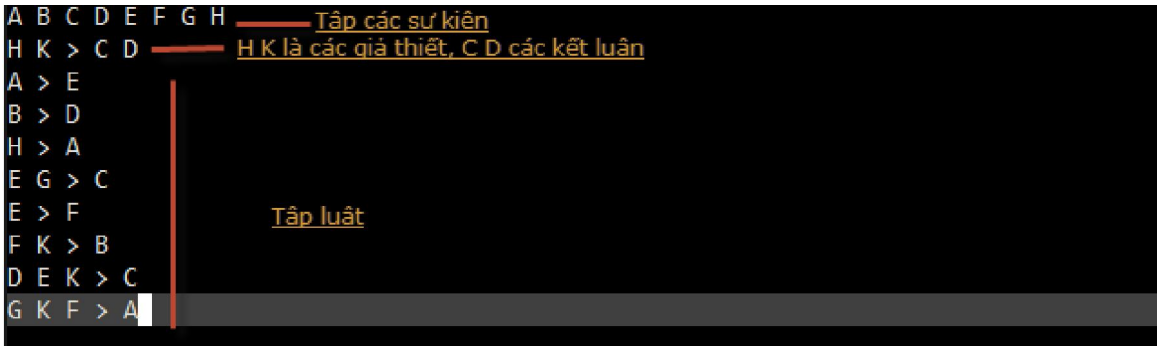
- Cấu trúc cơ bản nhất cho đối tượng tập luật dưới dạng AB -> CD
- Lớp cần xây dựng có tên là Rule gồm có 2 thành phần chính là string leftSide, và string rightSide.
- Ngoài ra còn có một số hàm nhập xuất cơ bản, so sánh đối tượng.

Đoạn code được thể hiện như sau:

```
class Rule {
public:
  string leftSide;
  string rightSide;
  Rule(void) : leftSide(""), rightSide("") {}
  ~Rule(void) {}
  void input(string left, string right) {
    leftSide = left;
    rightSide = right;
  }
  void output() {
    cout << leftSide << " > " << rightSide << endl;
  }
  bool isEqual(Rule anotherRule) {
    if (this->leftSide == anotherRule.leftSide && this->rightSide ==
anotherRule.rightSide)
      return true;
    else
      return false;
  }
};
```

2. Cấu trúc file cho hệ cơ sở tri thức

- Cấu trúc file input cho tập cơ sở tri thức có dạng:



- Với tương ứng mỗi khoảng trắng sẽ là phép AND.
- Ta sẽ xây dựng hàm có khả năng đọc file theo câu trúc trên

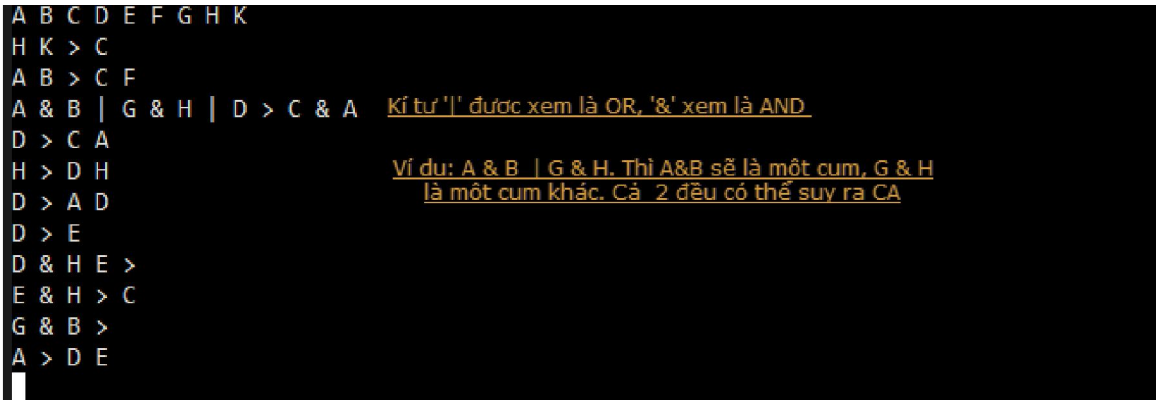
- **<u>Ý tưởng cơ bản</u>**:
    - Đọc tất cả các dòng có trong file theo cấu trúc quy định. Ví dụ với 2 dòng đâu, ta sẽ đọc và lưu string Fact (chuỗi sự kiện), string hypos(chuỗi cần hiện thực). Sau đó sẽ lần lượt đọc toàn các tập luật rồi lưu vào mảng các chuỗi, hàm này được gọi **loadFileString().**
    - Tiếp đến ta sẽ chuyển các chuỗi này thành các đối tượng thông qua hàm **convertFileStringToObject()**. Các chuỗi giả thiết, và tập luật đều chuyển thành đối tượng Rule.

Dưới đây là cách cái đặt cơ bản của 2 hàm trên.

```
void loadFileString(string path, string &fact, string &hypos, string listRule[], int
&m) {
  ifstream f(path);
  m = 0;
  if(f.is_open()) {
    getline(f, fact);
    getline(f, hypos);
    while (!f.eof()) {
      getline(f, listRule[m]);
      m++;
    }
    f.close();
  }
  else cout << "Can not open the file !";
}
```

```
void convertFileStringToObject(string &fact, string &hypos, string List[], int n,
vector<Rule> &listRule) {
  string::iterator end_pos = remove(fact.begin(), fact.end(), ' ');
  fact.erase(end_pos, fact.end());
  end_pos = remove(hypos.begin(), hypos.end(), ' ');
  hypos.erase(end_pos, hypos.end());

 for(int i = 0; i < n-1; i++) {
   end_pos = remove(List[i].begin(), List[i].end(), ' ');
   List[i].erase(end_pos, List[i].end());          // remove all of space exsist in
string rule

   end_pos = remove(List[i].begin(), List[i].end(), '&');
   List[i].erase(end_pos, List[i].end());         // remove all of & exsist in
string rule

   convertStringToRule(List[i], listRule);

 }
}
```

- **Bổ sung**: Ta có thể khai báo cấu trúc file như sau trong trường hợp vế trái có nhiều phép toán AND và OR



3. Chức năng tối ưu hóa luật sinh.

a. Ý tưởng cơ bản:

B1: Rút gọn vế phải

Với mỗi luật r trong R

Với mỗi sự kiện A $\in$ VếPhải(r)

Nếu A $\in$ VếTrái(r) thì Loại A ra khỏi vế phải của R.

Nếu VếPhải(r) rỗng thì loại bỏ r ra khỏi hệ luật dẫn : R = R \{r}

B2: Phân rã các luật

Với mỗi luật r: $X_1 \vee X_2 \vee \dots \vee X_n \rightarrow Y$ trong R

Với mỗi i từ 1 đến n, R:= R + { $X_i \rightarrow Y$ }

R:= R \ {r}

B3: Loại bỏ luật thừa

Với mỗi luật r thuộc R

Nếu VếPhải(r) $\in$ BaoĐóng(VếTrái(r), R-{r}) thì R := R \ {r}

B4: Rút gọn vế trái

Với mỗi luật dẫn r: X: $A_1 \wedge A_2, \dots, A_n \rightarrow Y$ thuộc R

Với mỗi sự kiện $A_i \in r$

Gọi luật r_1: X - $A_i \rightarrow Y$

S = (R - {r}) $\cup$ {r_1}

Nếu BaoĐóng (X - A_i, S) $\equiv$ BaoĐóng (X, R) thì loại sự kiện A ra khỏi X

b. Quá trình thực hiện:

- Chức năng trên được xây dựng thông qua hàm **handleOptimalListRule**().
- Ta sẽ chia nhỏ hàm thành 4 giai đoạn tương ứng 4 bước trên.

- **Bước đầu tiên**: Quét từng tập luật trong listRule, kiểm tra các điều kiện thỏa mãn như vế phải không được rỗng, nếu rỗng sẽ xóa khỏi tập luật. Nếu có phần tử vế phải nằm ở vế trái thì xóa phần tử đó.

```
int i = 0;
while (i < listRule.size()) {
  if (listRule[i].rightSide.empty()) {
    listRule.erase(listRule.begin() + i);
  }
  else {
    string leftSide = listRule[i].leftSide;
    for(int j = 0; j < leftSide.length(); j++) {
      size_t found = listRule[i].rightSide.find(leftSide[j]);
      if (found != string::npos) {  // if leftSide character exists in rightSide,
hen remove
        listRule[i].rightSide.erase(listRule[i].rightSide.begin() + found);
      }
    }
    i++;
  }
}
```

Trường hợp vế phải rỗng, ta loại vế phải ra khỏi tập luật

Scan từng phần tử trong leftSide

- **Bước 2**: Phân rã các luật, ở bước này ta đã thực hiện việc phân rã ở hàm **convertStringToRule(),** hàm này xử lí việc phân ra các dấu '|' thành các cặp phần tử luật .
- **Bước 3**: Loại bỏ luật thừa, ta tìm bao đóng của vế trái trong tập luật đang xét, nếu vế phải tập luật đó nằm trong bao đóng thì sẽ loại tập luật đó.

```
i = 0;       // step 3: remove the redunt rule
while (i < listRule.size()) {
  string tapBaoDong;
  tapBaoDong = baoDong(listRule[i].leftSide, i, listRule);

  if (checkElementInFact(listRule[i].rightSide, tapBaoDong)) {
    listRule.erase(listRule.begin() + i);
  }
  else {
    i++;
  }
}
```

- **Bước 4**: Loại bỏ phần tử dư thừa vế trái, tìm phần bao đóng của các phân tử vế trái với điều kiện các phần tử không có phần từ đang xét, Nếu nằm trong bao đóng thì loại phần tử đó ra khỏi vế trái.

```
  // step 4: remove the redunt fact in left side
  for(int i = 0; i < listRule.size(); i++) {
    string leftSide;
    leftSide = listRule[i].leftSide;
    if (leftSide.length() > 1) {
      int j = 0;
      while (j < leftSide.length()) {
        string temp;
        string removedCharacter;
        temp = leftSide;
        removedCharacter = temp[j];
        temp.erase(temp.begin() + j);  // remove the current character, we do not
need to use BaoDong
        string tapBaoDong;
        tapBaoDong = baoDong(temp, i, listRule);
        if (tapBaoDong.find(removedCharacter) != string::npos) { // find out the
character in tapBaoDong
          leftSide.erase(leftSide.begin() + j);
        }
        else {
          j++;
        }
      }
      listRule[i].leftSide = leftSide;
    }
  }
```

c. Ví dụ minh họa:

- File dữ liệu mẫu

```
A B C D E F G H K
H K > C
A B > C F
A & B | G & H | D > C & A
D > C A
H > D H
D > A D
D > E
D & H E >
E & H > C
G & B >
A > D E
```

- Kết quả

```
AB > CF
D > CA
H > D
EH > C
A > DE
```

- **Giải thích**: Hai dòng đầu ta không xét đến. Xét AB > CF, nhận luật, vì chưa có sự trùng lặp. Xét AB | GH | D > CA, vì AB > CA đã có, xét GH > CA, vì ta có H > D H, tức H > D mà D > CA, nên GH > CA dư thừa, ta chỉ lấy D > CA. Xét D > CA, đã có nên loại. Xét H > DH, loại H ở vế phải còn H > D, Xét D > AD, dư thừa vì D > CA ở trên. Xét D & H & E > rỗng, loại. Xét EH > C, nhận và A > DE nhận.

4. Thuật toán suy diễn tiến, suy diễn lùi.

a. Suy diễn tiến:

Bước 1: **facts** =Hypos;

rules= [];

Bước 2:

while Goals⊄**facts** do

Bước 2.1: Tìm luật r ∈R có thể áp dụng được trên **facts**để sinh ra sự kiện mới.

Bước 2.2: Nếu tìm được r thì:

+ Bổ sung r vào **rules**

+ Bổ sung sự kiện mới vào **facts**

Ngược lại: Kết thúc suy luận, không tìm được lời giải

Bước 3: Kết luận tìm được mục tiêu

- **<u>Ý tưởng cơ bản</u>**: Tìm kiếm trong các tập luật, nếu có luật dẫn ra được một sự kiến mới khác (chưa có trong tập fact) thì sẽ tiếp tục tìm đến khi ra được lời giải hoặc không còn suy diễn được nữa. Chức năng trên được cài đặt bởi hàm **handleForwardChaining()**
- **<u>Code cài đặt:</u>**

```
void handleForwardChaining(string fact, string hypos, vector<Rule> listRule) {
  Rule hyposRule = convertStringToRule(hypos);

  fact = hyposRule.leftSide;
  string goal = hyposRule.rightSide;

  vector<Rule> sequence;

  while (!checkElementInFact(goal, fact)) {
    vector <int> index;
    int flag = 0;
    for (int i = 0; i < listRule.size(); i++) {
      if (checkElementInFact(listRule[i].leftSide, fact)) {
        sequence.push_back(listRule[i]);
        fact += listRule[i].rightSide;
        flag = 1;
        index.push_back(i);
      }
    }
    if (flag == 1) {
      int distance = 0;
      for(auto & element : index) {
        listRule.erase(listRule.begin() + element - distance);
        distance++;
      }
    }
    else {
      cout << "Not found" << endl;
      return;
    }
  }
```

Thiết lập các biến và đối tượng

Kiểm tra nếu goal đã nằm trong fact thì không cần làm, ngược lại thì vào vòng loop

Kiểm tra các phần tử bên trái tập luật có nằm trong fact không, nếu có hoàn toàn thì sẽ chấp nhận tập luật đó

Trường hợp tập luật đã được chọn, thì ta sẽ xóa luật đó trong listRule không xét nữa

Ngược lại, không tìm thấy tập luật thỏa , thì sẽ thoát khỏi vòng lặp

b. Suy diễn lùi:

- **Ý tưởng cơ bản**: Ta sẽ đi ngược từ kết luận đến giả thiết được cho. Bằng cách từ kết luận ta sẽ xem các tập luật ở vế phải có suy ra được phần kết luận không, nếu có thì chọn phần bên trái tập luật, rồi tiếp túc xét đến các tập luật khác. Cho đến khi chỉ còn duy nhất phần giả thiết trong tập sự kiện.
- **Code cài đặt**:

```
void handleBackwardChaining(string fact, string hypos, vector<Rule> listRule) {
  Rule hyposRule = convertStringToRule(hypos);
  string goal = hyposRule.leftSide;

  SetFact realFact;
  vector<SetFact> listRealFact;
  for(int i = 0; i < hyposRule.rightSide.length(); i++) {

    fact = hyposRule.rightSide[i];
    vector<SetFact> sequence;
    int flag = 0;
    SetFact* newFact = new SetFact(fact);
    vector<SetFact> generatedSet = generateSetOfFact(*newFact, listRule);
    sequence.insert(sequence.end(), generatedSet.begin(), generatedSet.end());
```

Thiết lập các giá trị và đối tượng

Quét từng phần tử ở kết luận, để tìm giả thiết suy ra nó

Tạo ra các khả năng có thể dẫn đến giả thiết

```
    sequence.insert(sequence.end(), generatedSet.begin(), generatedSet.end());
    while (sequence.size() != 0 && flag == 0) {

      SetFact hyposObj;
      hyposObj = sequence.front();
      realFact = hyposObj;
      sequence.erase(sequence.begin());
      if (isEqualValueString(hyposObj.factNow, goal)) {
        flag = 1;
      }
      else {
        generatedSet = generateSetOfFact(hyposObj, listRule);
        sequence.insert(sequence.end(), generatedSet.begin(), generatedSet.end());
      }

    }
    listRealFact.push_back(realFact);

}
```

Sau đó lưu vào vector sequence

Thực hiện vòng lặp tìm các khả năng trong tập sequence có thể dẫn đến giả thiết

Chọn phần tử đầu ra khỏi mảng, để kiểm tra

Nếu hyposObj bằng chuỗi giả thiết, thì kết luận tìm được

Ngược lại, từ hyposObj, tiếp tục sinh ra các khả năng có thể khác, lưu vào tập sequence

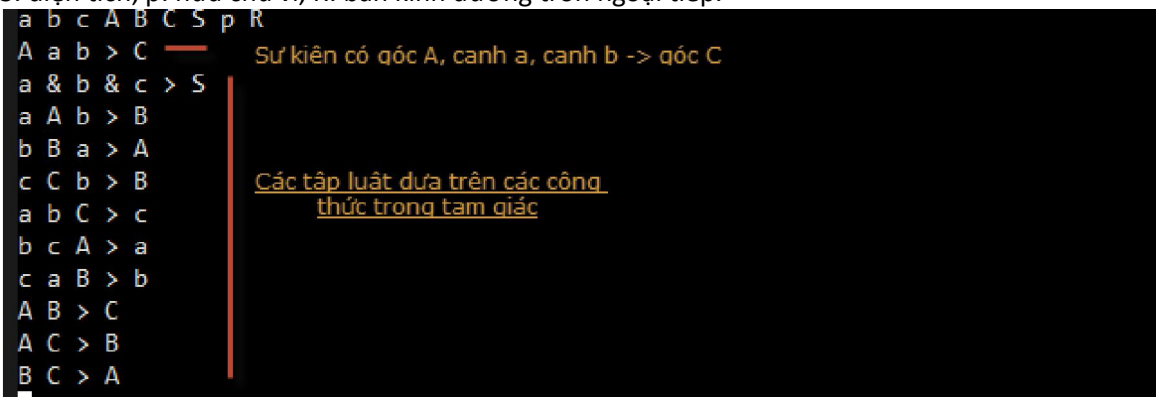
Sau khi tìm được, thì sẽ lưu vào mảng kết quả

- Ý tưởng trên giống với việc ta sẽ đi từ kết luận đến giả thiết, như một dạng cấu trúc cây đi từ gốc sẽ rẽ ra các nhánh con, rồi đến kết quả.

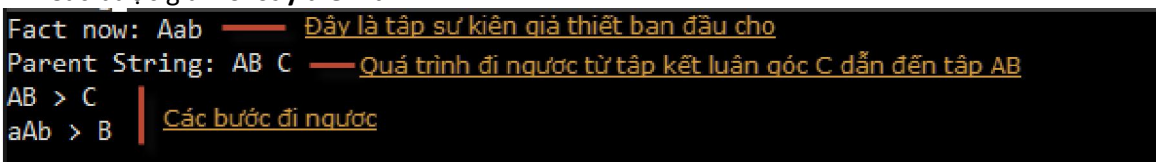
5. Ví dụ minh họa cho CSTT

- Cho các sự kiện của một tam giác ABC gồm có:

A: góc A, a: cạnh a,
B: góc B, b: cạnh b,
C: góc C, c: cạnh c,
S: diện tích, p: nữa chu vi, R: bán kính đường tròn ngoại tiếp.

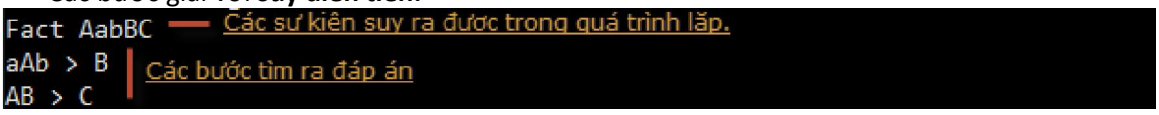


- Các bước giải với **suy diễn lùi**:



- Tập fact now là các sự kiện đề cho gồm có góc A, cạnh a, cạnh b, yêu cầu suy ra góc C. Parent string là chuỗi sự kiện đi ngược từ tập kết luận đến tập giả thiết có thể suy ra.
- Xét bước đi AB > C: muốn có góc C, thì phải có góc A, B. A đã có, ta áp dụng công thức A + B + C = 180, ta chỉ còn tìm góc B.
- Bước kế tiếp để tìm góc B: ta sẽ có cạnh a, góc A, cạnh b. Ta áp dụng công thức a/sinA = b/sinB -> góc B. Vậy ta đã giải quyết được bài toán.

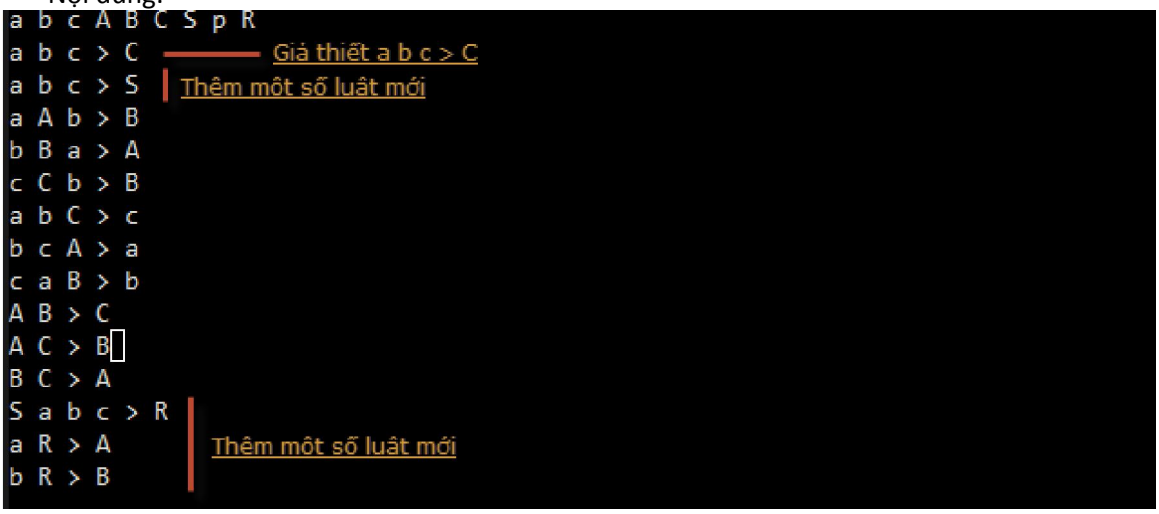
- Các bước giải với **suy diễn tiến:**



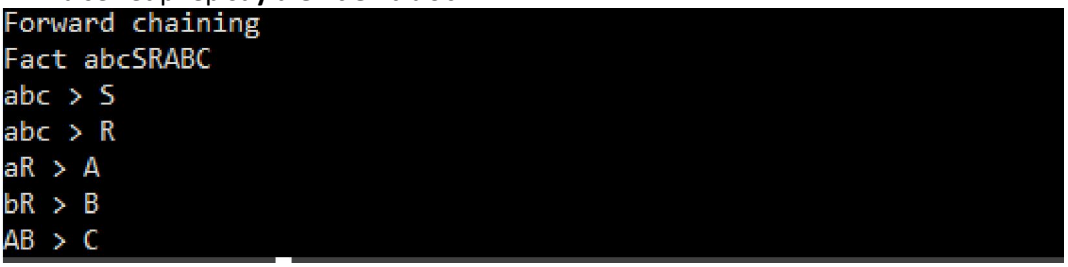
- Từ tập fact ta có 3 sự kiện từ giả thiết cho: A a b. Sau đó sẽ sinh ra các tập sự kiện mới.
- Từ a A b > B, áp dụng công thức sin, a/sinA = b/sinB -> góc B. Từ góc B và A đã có suy ra được góc C qua tập luật A B > C.

- Trên đây là ví dụ đơn giản, bây giờ ta cũng xét với bài toán tam giác và thêm một số yếu tố để làm phức tạp hơn.

- Nội dung:



- Ta sẽ xét phép **suy diễn tiến** trước:



- Ta có các cạnh a, b, c. Mục tiêu suy ra góc C.
- Ta có a b c -> diện tích S, do áp dụng công thức herong.
- Tiếp theo a b c -> bán kính đường tròn ngoại tiếp R, ta thây chỉ có abc > R, mất S. Nguyên nhân là ta sử dụng luật tối ưu hóa tập luật bằng cách loại bỏ vế trái thừa. Vì a b c -> S, thì chắc chắn S a b c -> R. S sẽ là thừa. Tại đây ta áp dụng công thức S= abc/4R -> R.
- Ta có cạnh a R -> A, áp dụng định lí sin, a/sinA = 2R.
- Tương tự đối với b R -> B, b/sinB = 2R.
- Từ A B -> C. Bằng công thức A + B + C = 180.

- Ta xét tiếp phép **suy diễn lùi**:

- Đầu tiên ta cũng có tập giả thiết cạnh a, b, c.
- Muốn có góc C, theo luật A B > C. Tìm góc A, B.

- Góc B, được suy ra theo a A b > B. Áp dụng định lí sin a/sinA = b/sinB -> góc B.
- Ta cần tìm góc A, góc A được tính theo luật a R > A. Do a/sinA = 2R. Nên suy ra được góc A. Vậy ta cần tìm R.
- R được tìm theo luật a b c > R. Do có công thức S=abc/4R. Mà S được tính qua công thức herong, chỉ phụ thuộc vào a, b, c. Nên R được suy ra.
- Kết luận cho thấy khi thực hiện phép suy diễn lùi thì các bước đưa ra không hoàn toàn theo các bước của suy diễn tiến.

6. Ứng dụng CSTT vào giải các bài toán

- Nhờ vào việc xây hệ CSTT cũng như các phép suy diễn, ta có thể xây dựng được ứng dụng giải các bài toán cơ bản như tam giác thông qua các công thức đã có, và các giá trị số giả thiết ban đầu được đưa vào để suy ra đáp án cần tìm.
- Ta sẽ xây dựng lại cấu trúc file test.txt để phù hợp cho ngữ cảnh sau

```
a b c A B C S p R
a R c B > S
4
a = 3
R = 6
c = 4
B = PI/6
a b c > p
p = (a + b + c)/2
p a b c > S
S = SQRT(p*(p-a)*(p-b)*(p-c))
a A b > B
B = ASIN((b * SIN(A))/a)
b B a > A
A = ASIN((a * SIN(B))/b)
c C b > B
B = ASIN((b * SIN(C))/c)
a b C > c
c = SQRT(a^2 + b^2 - 2*a*b*COS(C))
b c A > a
```

- Dữ liệu đưa vào có chút thay đổi: đó là thêm các giá trị số như cạnh a=3, bán kính R=6, cạnh c=4, góc B=PI/6, để tính ra diện tích S. Trong đó ta sẽ thêm một số công thức cho các tập luật suy diễn như: a b c > p, tức 3 cạnh suy ra nữa chu vi thông qua công thức p = (a+b+c)/2.

- **Ý tưởng cơ bản:** Đầu tiên ta sẽ tạo ra 2 tập, tập 1 chưa các tập luật suy diễn, tập 2 chưa các công thức. Khi đó từ mỗi luật suy diễn ở tập 1 sẽ có thể ánh xạ sang công thức ở tập 2. Thông qua hàm suy diễn tiến forwardChaining, ta sẽ có các bước tập luật cần suy dẫn để đi đến kết quả, từ những tập suy dẫn này, ta sẽ ánh xạ sang tập công thức để tính toán và đưa ra giá trị ở từng bước.

- Ở đây ta sẽ gặp vấn đề đó là làm sao có thể xử lý được các công thức dạng chuỗi sang thành các công thức cho máy tính có thể hiểu và tính toán. Việc xử lý chuỗi công thức đó cực kỳ phức tạp và tốn nhiều thời gian, nên để hiện thực được tính ứng dụng của CSTT, nên đã mượn phần xử lý chuỗi công thức từ bài viết này: **http://www.speqmath.com/tutorials/expression_parser_cpp/index.html**

- Ta sẽ sử dụng source code và một số hàm từ đường link trên, có tùy biến chỉnh sửa để phù hợp ngữ cảnh trong bài toán này. Source code được nằm chung trong thư mục của file này.

- Dưới đây là quá trình hiện thực hóa bài toán.

- Nội dung file test gồm có như sau:

```
a b c A B C S p R
a R c B > S
4
a = 3
R = 6
c = 4
B = PI/6
a b c > p
p = (a + b + c)/2
p a b c > S
S = SQRT(p*(p-a)*(p-b)*(p-c))
a A b > B
B = ASIN((b * SIN(A))/a)
b B a > A
A = ASIN((a * SIN(B))/b)
c C b > B
B = ASIN((b * SIN(C))/c)
a b C > c
c = SQRT(a^2 + b^2 - 2*a*b*COS(C))
b c A > a
```

```
b c A > a
a = SQRT(b^2 + c^2 - 2*b*c*COS(A))
c a B > b
b = SQRT(a^2 + c^2 - 2*a*c*COS(B))
A B > C
C = PI - A - B
A C > B
B = PI - A - C
B C > A
A = PI - B - C
S a b c > R
R = a*b*c/(4*S)
a R > A
A = ASIN(a/(2*R))
b R > B
B = ASIN(b/(2*R))
a b c R > S
S = a*b*c/(4*R)
```

- **Kết quả tính toán:**

```
Solving
Forward chaining
Fact aRcBbAS
caB > b
aR > A
abcR > S
b = SQRT(a^2 + c^2 - 2*a*c*COS(B))
        Ans = 2.05314
A = ASIN(a/(2*R))
        Ans = 0.25268
S = a*b*c/(4*R)
        Ans = 1.02657
```

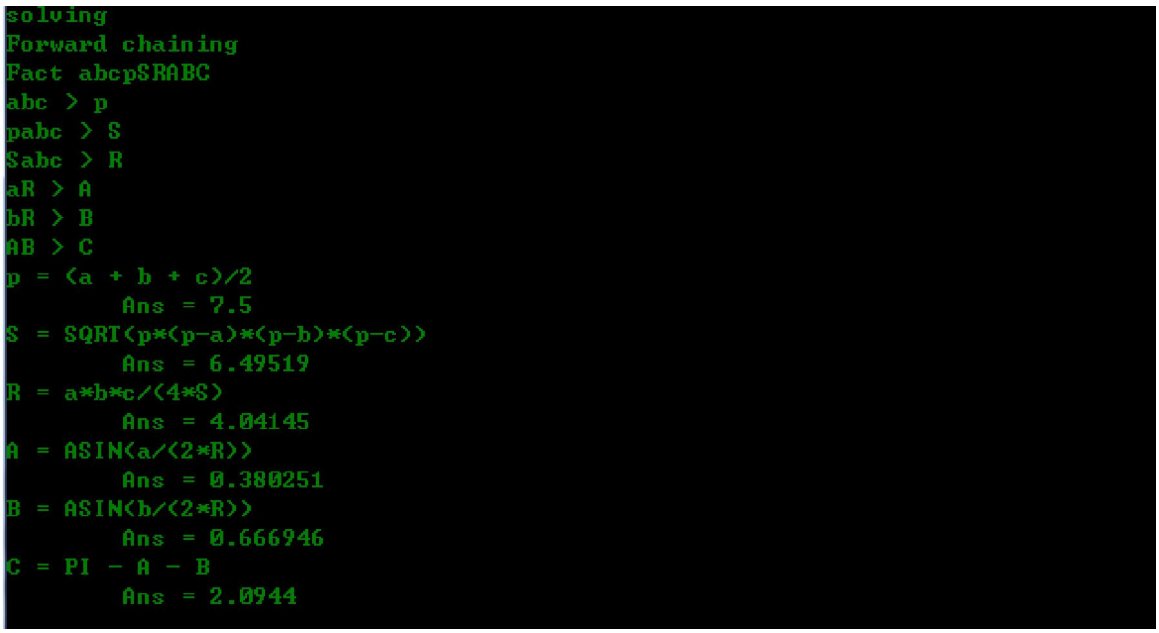
- **Quan sát:** Ban đầu ta có các giá trị cạnh a=3, c=4, góc B=pi/6, ta sẽ tính giá trị b thông qua công thức trên ra kết quả Ans=2.05314, tiếp đến lần lượt tính các giá trị khác. Tuy nhiên trong qua trình tính, vẫn có một số giá trị được tính ra, nhưng không cần thiết cho đề bài ví dụ như trong trường hợp này là góc A.

- Dưới đây ta sẽ quan sát thêm một ví dụ. Nội dung file test:

```
a b c A B C S p R
a b c > C
3
a = 3
b = 5
c = 7
a b c > p
p = (a + b + c)/2
p a b c > S
S = SQRT(p*(p-a)*(p-b)*(p-c))
a A b > B
B = ASIN((b * SIN(A))/a)
b B a > A
A = ASIN((a * SIN(B))/b)
c C b > B
B = ASIN((b * SIN(C))/c)
a b C > c
c = SQRT(a^2 + b^2 - 2*a*b*COS(C))
b c A > a
a = SQRT(b^2 + c^2 - 2*b*c*COS(A))
c a B > b
b = SQRT(a^2 + c^2 - 2*a*c*COS(B))
A B > C
C = PI - A - B
A C > B
B = PI - A - C
B C > A
A = PI - B - C
S a b c > R
R = a*b*c/(4*S)
a R > A
A = ASIN(a/(2*R))
b R > B
B = ASIN(b/(2*R))
```

- Từ 3 cạnh a, b, c cho trước hãy suy ra giá trị góc C

- **Kết quả bài toán:**



- Ban đầu từ 3 cạnh a, b, c sẽ suy ra được nữa chu vi p = 7.5. Từ đó sẽ tính được diện tích S có giá trị 6.49519. Lần lượt như vậy suy ra được góc C = 2.0944 độ radian.

7. Nhận xét:

- Nhờ cấu trúc tri thức dưới dạng Nếu … thì …, ta có thể xây dựng được nhiều cơ sở tri thức và áp dụng thực tiễn.
- Nhờ việc xây dựng hệ CSTT và các phép lập luận suy diễn, ta có thể biến các bài toán ứng dụng như giải tam giác trở nên đơn giản hơn, thân thuộc hơn và gắn liền với thực tế.

Lưu ý về Source code:

- Có 2 thư mục chính trong project này:
    - Phan yeu cau chinh: thư mục này chứa code cơ bản: Xây dựng hệ CSTT, Cải tiến hệ luật dẫn, Trình bày suy diễn tiến, suy diễn lùi. Đây là phần yêu cầu chính.
    - Phan them tinh ung dung: thư mục này cũng chưa các chức năng cơ bản để xây dựng hệ CSTT, cải tiến luật dẫn, suy diễn tiến lùi, như có thềm phần đưa các thông số giá trị, công thức để tính toán và giải ra đáp số bài toán.

- Tất cả các code đều được viết bằng ngôn ngữ C++ và được compile bằng phần mềm g++ (có thể dùng Codeblock, hoặc Visual Studio để compile) trên Windows 7. Nếu không thể chạy trực tiếp bằng file exe (có thể lỗi do không tương thích hệ diều hành), có thể dùng source code để compile lại.
- Sử dụng g++ compiler để compile source code (có thể sử dụng Visual Studio), thông qua lệnh: g++ main.cpp -o main -std=c++11