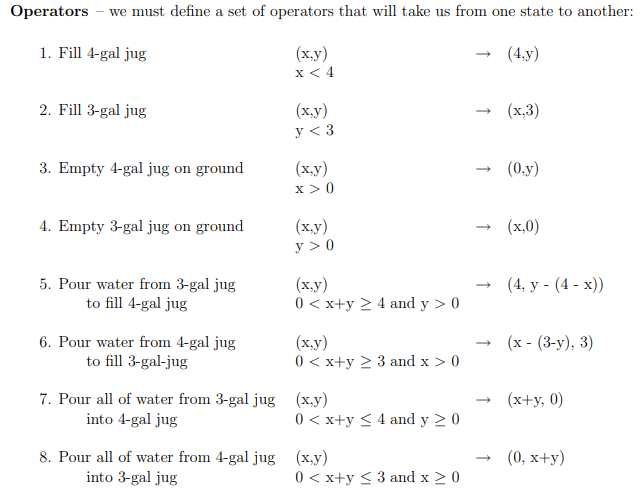
**State Representation and Initial State** – we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note 0 *≤* x *≤* 4, and 0 *≤* y *≤* 3. Our initial state: (0,0)

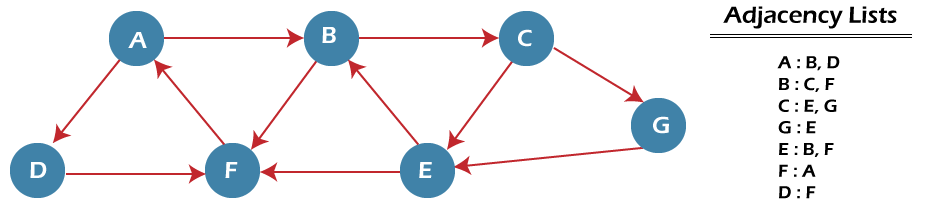
**Goal Predicate** – state = (2,y) where 0 *≤* y *≤* 3.



Through Graph Search, the following solution is found:

|  |  |  |
| --- | --- | --- |
| **Gals in 4-gal jug** | **Gals in 3-gal jug** | **Rule Applied** |
| 0 | 0 |  |
|  |  | 1. Fill 4 |
| 4 | 0 |  |
|  |  | 6. Pour 4 into 3 to fill |
| 1 | 3 |  |
|  |  | 4. Empty 3 |
| 1 | 0 |  |
|  |  | 8. Pour all of 4 into 3 |
| 0 | 1 |  |
|  |  | 1. Fill 4 |
| 4 | 1 |  |
|  |  | 6. Pour into 3 |
| 2 | 3 |  |

1. 1. Bread-first search



In the above graph, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

Now, let's start examining the graph starting from Node A.

**Step 1** - First, add A to queue1 and NULL to queue2.

1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

**Step 2** - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

1. QUEUE1 = {B, D}
2. QUEUE2 = {A}

**Step 3** - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

1. QUEUE1 = {D, C, F}
2. QUEUE2 = {A, B}

**Step 4** - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

1. QUEUE1 = {C, F}
2. QUEUE2 = {A, B, D}

**Step 5** - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

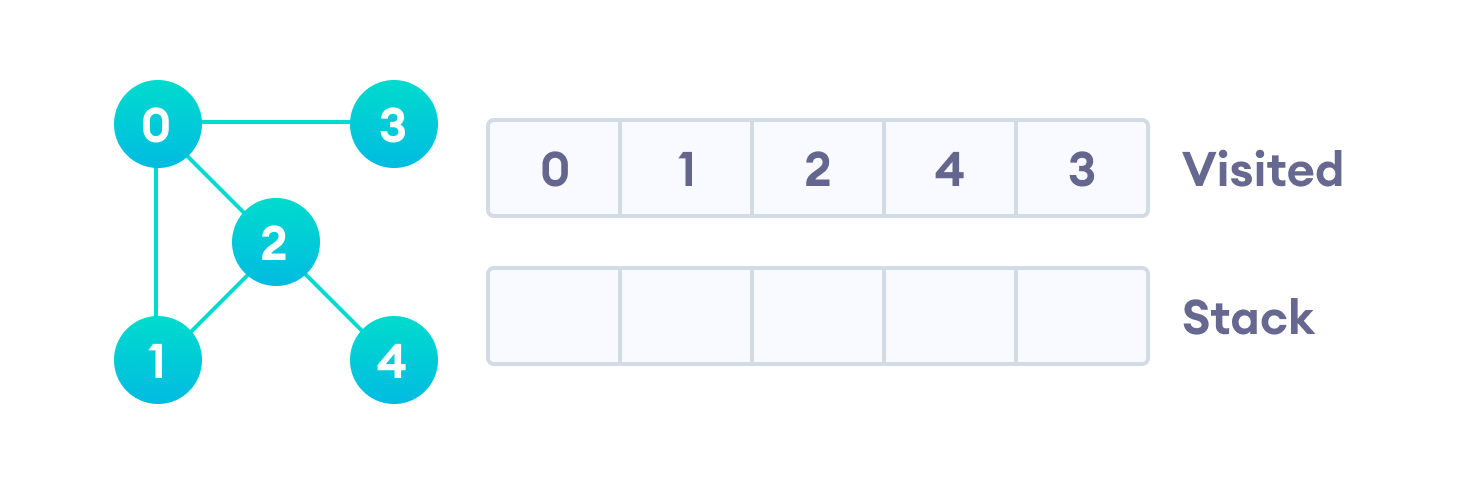
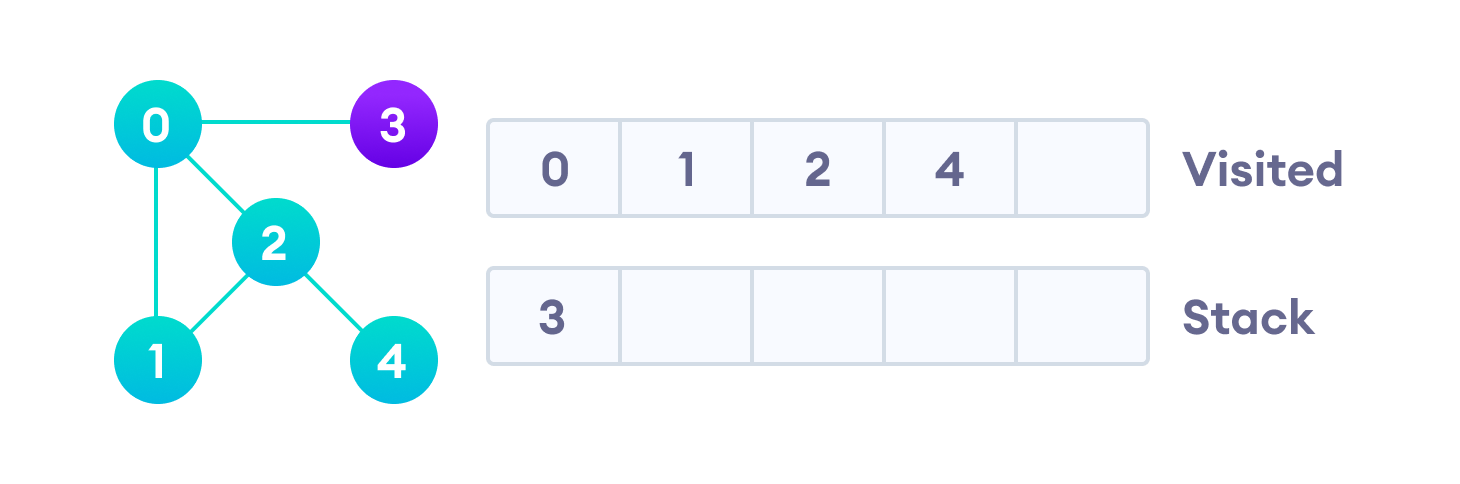
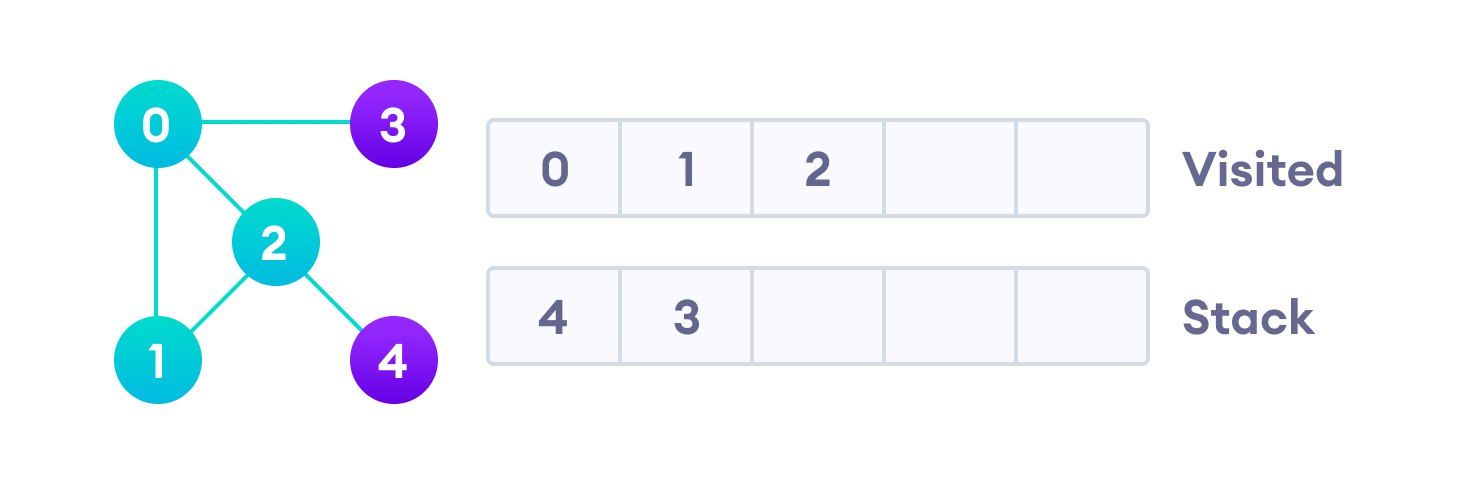
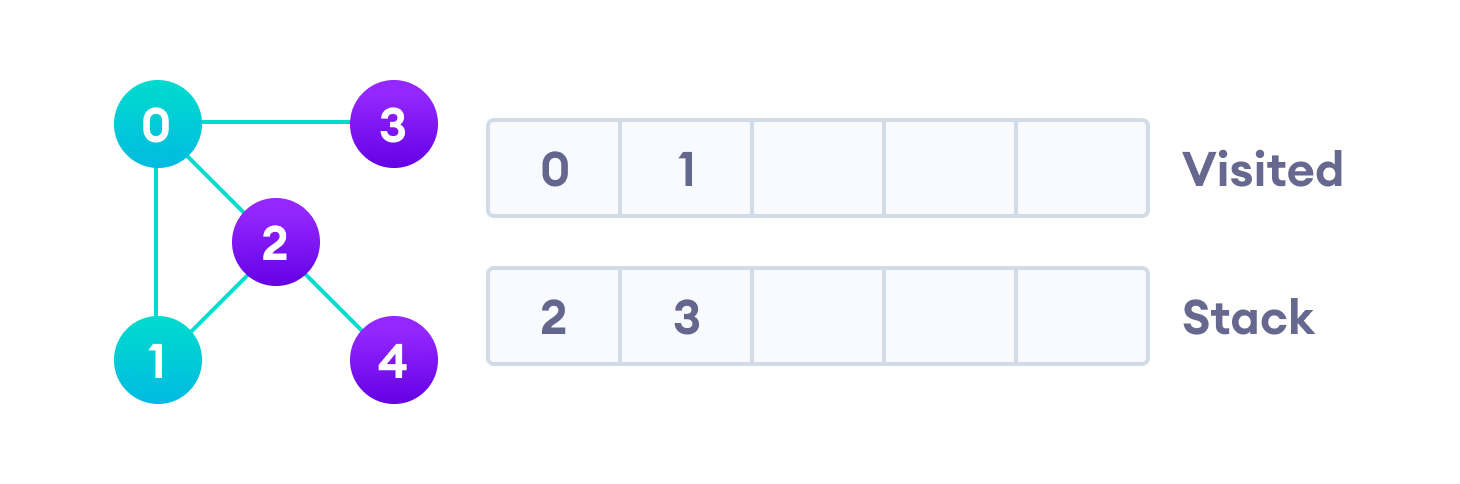
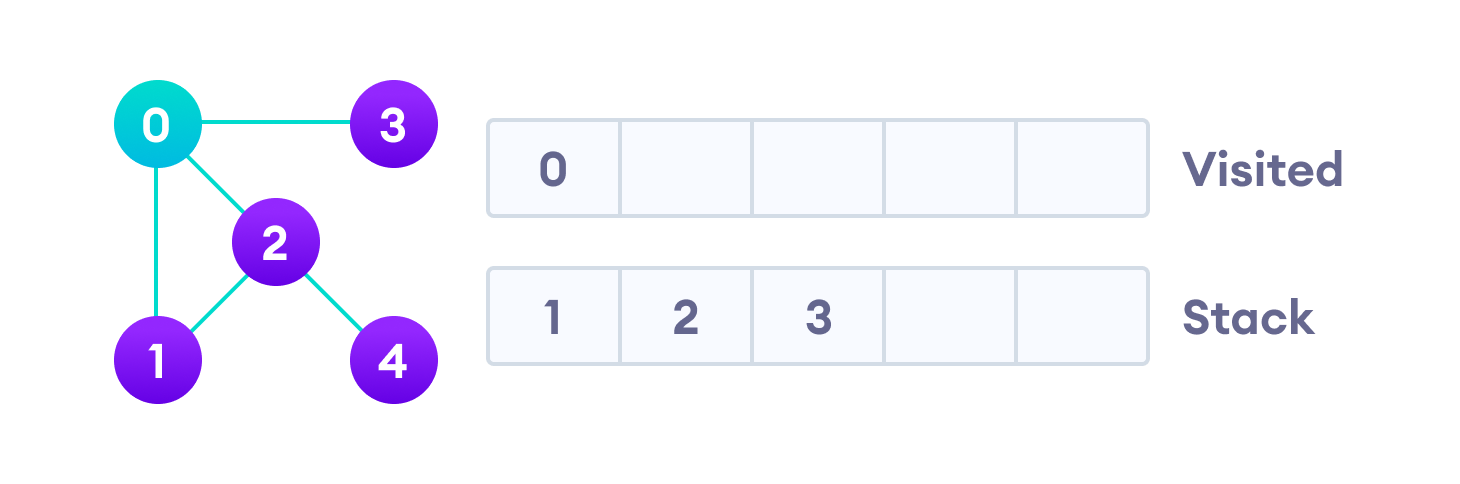
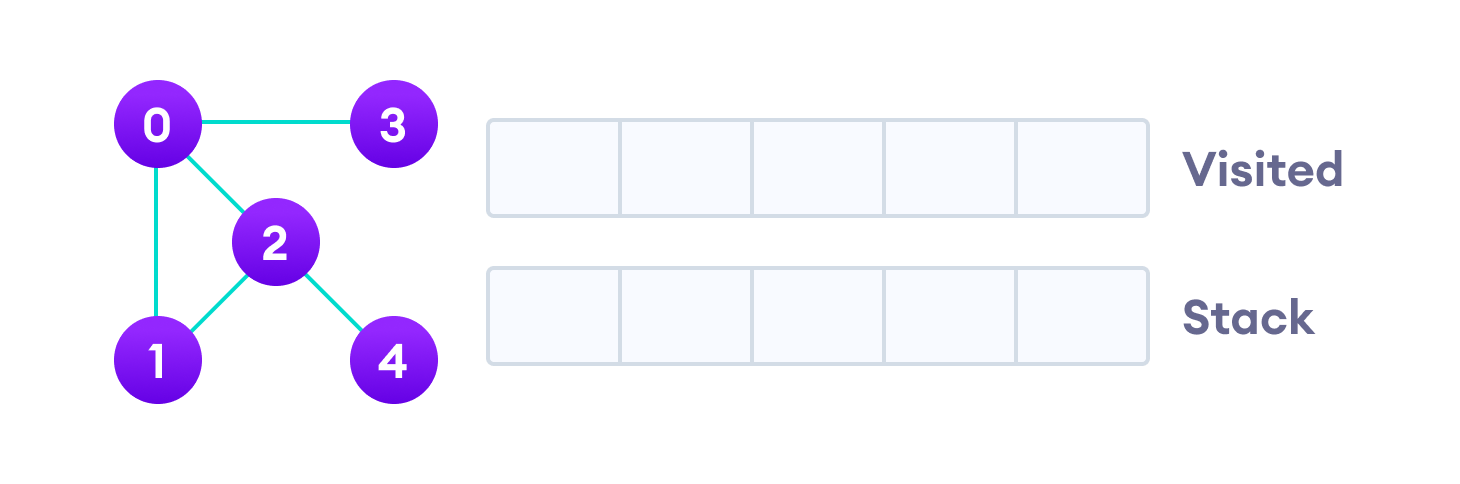
1. QUEUE1 = {F, E, G}
2. QUEUE2 = {A, B, D, C}

**Step 5** - Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

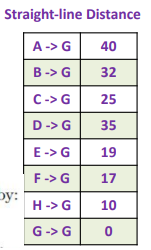
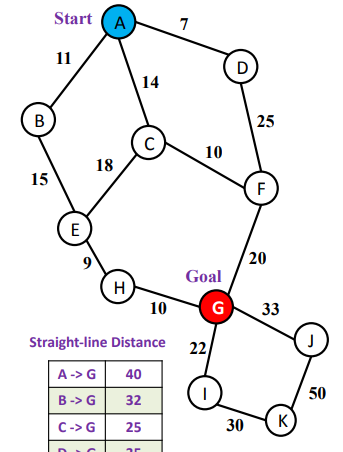
1. QUEUE1 = {E, G}
2. QUEUE2 = {A, B, D, C, F}

**Step 6** - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

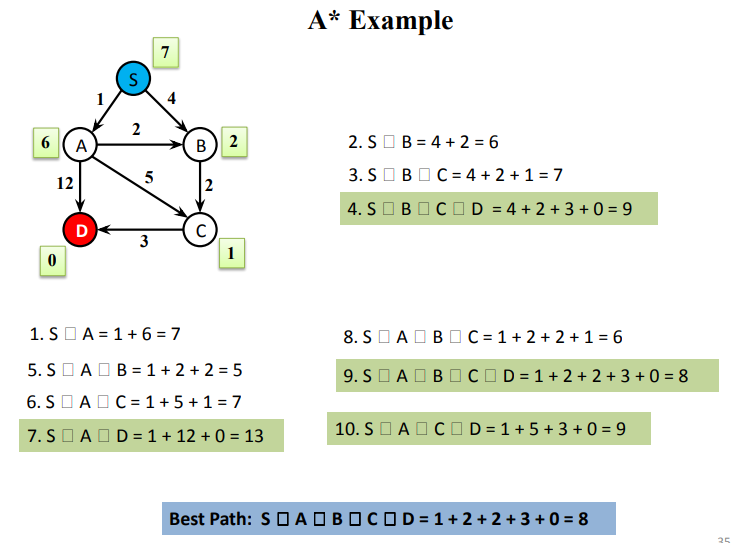
1. QUEUE1 = {G}
2. QUEUE2 = {A, B, D, C, F, E}
   1. Depth-first search



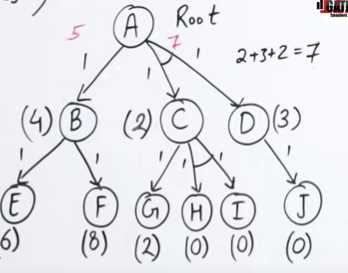
* 1. Local beam search
  2. Best first search



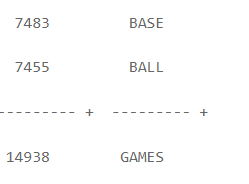
* 1. A\* algorithm



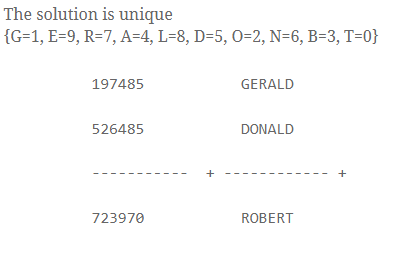
* 1. AO\* Algo



1. Solve following CSP problems using crypt-arithmetic.
   1. B A S E + B A L L = G A M E S



* 1. D O N A L D + G E R A L D = R O B E R T

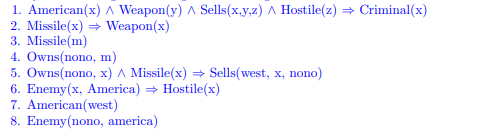




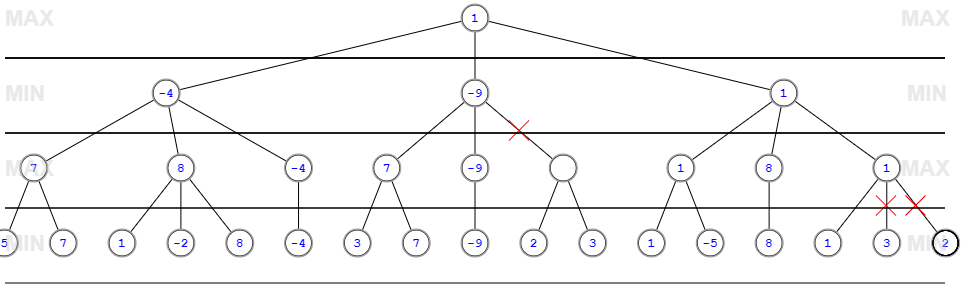
The conclusion can be proved using Resolution as shown below. The first step is to write each axiom as a well-formed formula in first-order predicate calculus. The clauses written for the above axioms are shown below, using LS(x) for `light sleeper'.

1. *∀ x (HOUND(x) → HOWL(x))*
2. *∀ x ∀ y (HAVE (x,y) ∧ CAT (y) → ¬ ∃ z (HAVE(x,z) ∧ MOUSE (z)))*
3. *∀ x (LS(x) → ¬ ∃ y (HAVE (x,y) ∧ HOWL(y)))*
4. *∃ x (HAVE (John,x) ∧ (CAT(x) ∨ HOUND(x)))*
5. *LS(John) → ¬ ∃ z (HAVE(John,z) ∧ MOUSE(z))*

5.







1. 1. All students are smart.

" x ( Student(x) Þ Smart(x) )

* 1. There exists a student.

$ x Student(x).

* 1. There exists a smart student.

$ x ( Student(x) Ù Smart(x) )

* 1. Every student loves some student.

" x ( Student(x) Þ $ y ( Student(y) Ù Loves(x,y) ))

* 1. Every student loves some other student.

" x ( Student(x) Þ $ y ( Student(y) Ù Ø (x = y) Ù Loves(x,y) ))

* 1. There is a student who is loved by every other student.

$ x ( Student(x) Ù " y ( Student(y) Ù Ø(x = y) Þ Loves(y,x) ))

* 1. Bill is a student.

Student(Bill)

1. Bill takes either Analysis or Geometry (but not both)

Takes(Bill, Analysis) Û Ø Takes(Bill, Geometry)

1. Bill takes Analysis or Geometry (or both).

Takes(Bill, Analysis) Ú Takes(Bill, Geometry)

1. Bill takes Analysis and Geometry.

Takes(Bill, Analysis) Ù Takes(Bill, Geometry)

1. Bill does not take Analysis.

Ø Takes(Bill, Analysis).

1. No student loves Bill.

Ø $ x ( Student(x) Ù Loves(x, Bill) )

1. Bill has at least one sister.

$ x SisterOf(x,Bill)

1. Bill has no sister.

Ø $ x SisterOf(x,Bill)

1. Bill has at most one sister.

" x, y ( SisterOf(x, Bill) Ù SisterOf(y, Bill) Þ x = y )

1. Bill has exactly one sister.

$ x ( SisterOf(x, Bill) Ù " y ( SisterOf(y, Bill) Þ x = y ))

1. Bill has at least two sisters.

$ x, y ( SisterOf(x, Bill) Ù SisterOf(y, Bill) Ù Ø (x = y) )

1. Every student takes at least one course.

" x ( Student(x) Þ $ y ( Course(y) Ù Takes(x,y) ))

1. Only one student failed History.

$ x ( Student(x) Ù Failed(x, History) Ù " y ( Student(y) Ù Failed(y, History) Þ x = y ))

1. No student failed Chemistry but at least one student failed History.

Ø $ x ( Student(x) Ù Failed(x, Chemistry) ) Ù $ x ( Student(x) Ù Failed(x, History) )

1. Every student who takes Analysis also takes Geometry.

" x ( Student(x) Ù Takes(x, Analysis) Þ Takes(x, Geometry) )

1. No student can fool all the other students.

Ø $ x ( Student(x) Ù " y ( Student(y) Ù Ø (x = y) Þ Fools(x,y) ))