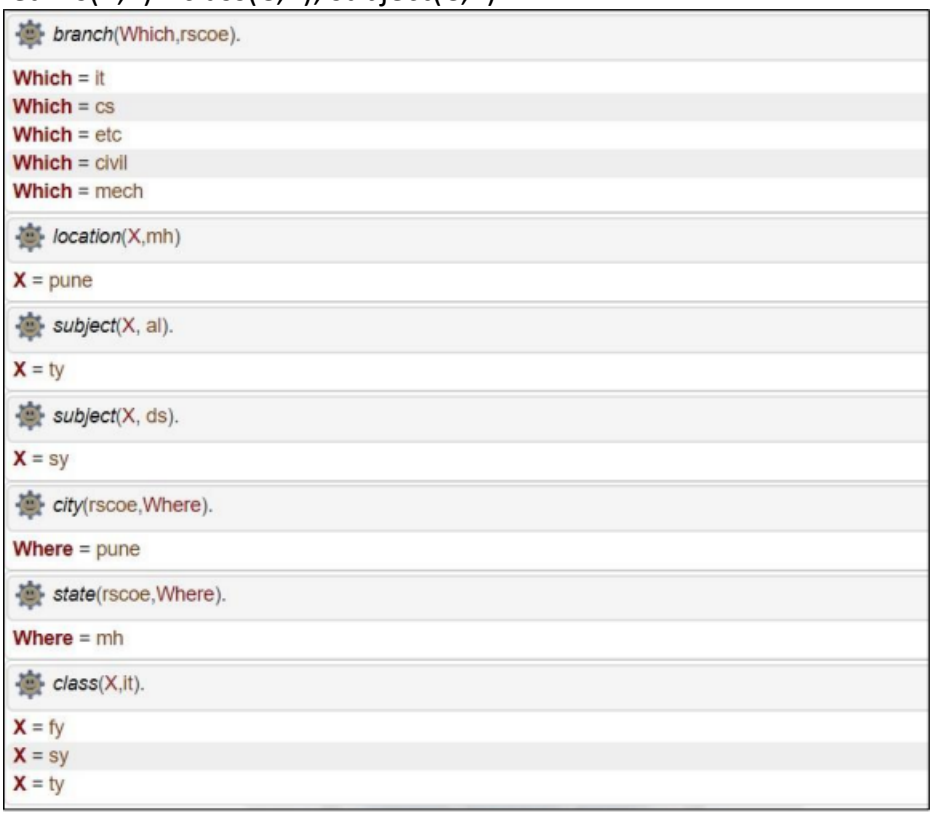
PROGRAM 1: PROLOG PROGRAM FOR COLLEGE KNOWLEDGE

```
class(fy,it).
class(sy,it).
class(ty,it).
college(rscoe).
city(rscoe,pune).
state(rscoe,mh).
branch(it,rscoe).
branch(cs,rscoe).
branch(etc,rscoe).
branch(civil,rscoe).
branch(mech,rscoe).
subject(fy,cpp).
subject(sy, ds).
subject(ty, al).
location(X,Y):-city(C,X), state(C,Y).
learns(X,Y):- class(C,X), subject(C,Y).
```



PROGRAM 2: PROLOG PROGRAM FOR RELATIONS KNOWLEDGE

```
parent(x,y).
parent(z,x).
child(X,Y):-parent(Y,X).
grandparent(Z,Y):-parent(Z,X),parent(X,Y).
friend(p,y).
friend(X,Y):-friend(Y,X).
likes(p,sing).
likes(y,cricket).classmates(p,y).
classmates(X,Y):-classmates(Y,X).
```

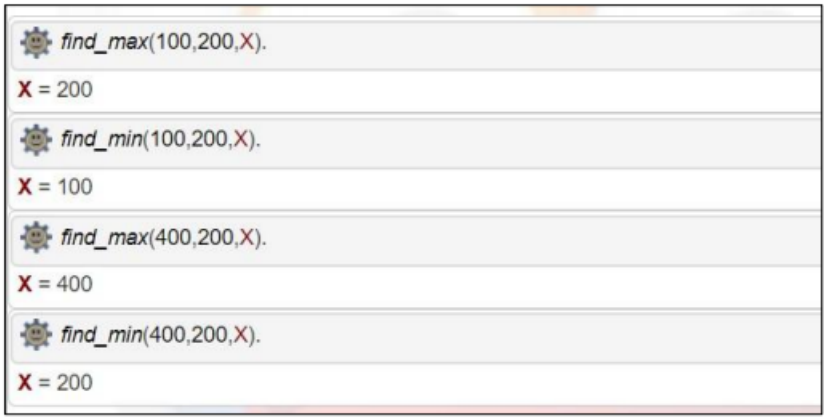
PROGRAM 3: PROLOG PROGRAM FOR TEACHER STUDENT KNOWLEDGE

```
studies(charlie, csc135).
studies(olivia, csc135).
studies(jack, csc131).
studies(arthur, csc134).
teaches(kirke, csc135).
teaches(collins, csc131).
teaches(collins, csc171).
teaches(juniper, csc134).
professor(X, Y) :- teaches(X, C), studies(Y, C).
```

PROGRAM 4: PROLOG PROGRAM FOR MIN MAX

```
find_max(X,Y,X):-X>Y , ! .
find_max(X,Y,Y):-Y>X.
find_min(X,Y,X):-X<Y,!.
```

find_min(X,Y,Y):-Y<X.



PROGRAM 5: PROLOG PROGRAM FOR BIKES

```
bike(ktm).
bike(bike1).
bike(bike2).
bike(bike3).
location(bike1,city1).
location(bike1,city2).
location(bike2,city2).
location(bike3,city3).
category(bike1,electric).
category(bike2,petrol).
category(bike3,pertol).
price(bike1,80000).
price(bike2,70000).
price(bike3,60000).
find_max(A,B,A):-price(A,X),price(B,Y),X>=Y, ! .
find_max(A,B,B):-price(A,X),price(B,Y),Y>X.
find_min(A,B,A):-price(A,X),price(B,Y),X<Y, ! .
find_min(A,B,B):-price(A,X),price(B,Y),Y<X.
```


```
======================BFS====================================
#include<iostream>
#include<map>
#include<queue>
#include<list>
using namespace std;
template <typename T>  /*generic type , template class
                    to work with graph of integers or variables */

class Graph{
  map<T, list<T>> l;    //map<int,list<int>> // 2->(1,0,3)

public:
  void addEdge(int x, int y){ //edges are bi-directional //Adds a new element at the end of the vector, after its current last
element
    l[x].push_back(y);
    l[y].push_back(x);
  }

  void bfs(T src){
    map<T,int> visited;    //we created visited array
    queue<T> q;

    q.push(src);
    visited[src]=true;

    while(!q.empty()){
      T node = q.front();
      q.pop();
      cout<<node<<" ";
```

```
    for(auto nbr : l[node])
      {
        if(!visited[nbr]){
          q.push(nbr);
          //mark that nbr as visited
          visited[nbr]= true;

        }
      }
   }

 }
};

int main()
{
  Graph<int> g;
  g.addEdge(0,1);
  g.addEdge(1,2);
  g.addEdge(2,3);
  g.addEdge(3,4);
  g.addEdge(4,5);
  g.addEdge(3,0);

  g.bfs(0);
  return 0;

}
```

==========DFS==========

```
#include<iostream>
#include<map>
#include<queue>
#include<list>

using namespace std;

template <typename T>

class Graph
{
  map<T,list<T>> l; //auxiliary

public:
void addEdge(int x,int y)
{
  l[x].push_back(y);
  l[y].push_back(x);
}

void dfs_helper(T src , map<T,bool> &visited)
{
  cout<<src<<" ";
  visited[src]=true;

  for(T nbr:l[src])
    {
      if(!visited[nbr])
      {
        dfs_helper(nbr,visited);
```

```
    }
   }
}
void dfs(T src)
{
 map<T,bool> visited;
 for(auto p:l)
  {
    T node = p.first;
    visited[node]=false;

  }
 dfs_helper(src,visited);

}
};

int main()
{
 Graph <int> g;
 g.addEdge(0,1);
 g.addEdge(1,2);
 g.addEdge(2,3);
 g.addEdge(3,4);
 g.addEdge(4,5);
 g.addEdge(3,0);

 g.dfs(0);
}
```

==========best==========

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;
vector<vector<pi> > graph;
// Function for adding edges to graph
void addedge(int x, int y, int cost)
{
graph[x].push_back(make_pair(cost, y));
graph[y].push_back(make_pair(cost, x));
}
// Function For Implementing Best First Search
// Gives output path having lowest cost
void best_first_search(int source, int target, int n)
{
vector<bool> visited(n, false);
// MIN HEAP priority queue
priority_queue<pi, vector<pi>, greater<pi> > pq;
// sorting in pq gets done by first value of pair
pq.push(make_pair(0, source));
int s = source;
visited[s] = true;
while (!pq.empty()) {
int x = pq.top().second;
// Displaying the path having lowest cost
cout << x << " ";
pq.pop();
if (x == target)
break;
for (int i = 0; i < graph[x].size(); i++) {
if (!visited[graph[x][i].second]) {
```

```
visited[graph[x][i].second] = true;
pq.push(make_pair(graph[x][i].first,graph[x][i].second));
}
}
}
}
// Driver code to test above methods
int main()
{2
// No. of Nodes
int v = 14;
graph.resize(v);
// The nodes shown in above example(by alphabets) are
// implemented using integers addedge(x,y,cost);
addedge(0, 1, 3);
addedge(0, 2, 6);
addedge(0, 3, 5);
addedge(1, 4, 9);
addedge(1, 5, 8);
addedge(2, 6, 12);
addedge(2, 7, 14);
addedge(3, 8, 7);
addedge(8, 9, 5);
addedge(8, 10, 6);
addedge(9, 11, 1);
addedge(9, 12, 10);
addedge(9, 13, 2);
int source = 0;
int target = 9;
// Function call
best_first_search(source, target, v);
return 0;
}
```

==========water jug==========

```
#include<bits/stdc++.h>
using namespace std;
int x;
int y;
void show(int a, int b);
int min(int w, int z)
{
if (w < z)
return w;
else
return z;
}
void show(int a, int b)
{
cout << setw(12) << a << setw(12) << b<<endl;
}
void s(int n)
{
int xq = 0, yq = 0;
int t;
cout << setw(15) <<"FIRST JUG"<< setw(15) <<"SECOND JUG"<<endl;
while (xq != n && yq!=n )
{
  if (xq == 0)
  {
    xq = x;
```

```
   show(xq, yq);
  }
  else if (yq == y)
  {
   yq = 0;
   show(xq, yq);
  }
  else
  {
   t = min(y - yq, xq);
   yq= yq + t;
   xq = xq - t;
   show(xq, yq);
    }
}
}
int main()
{
int n;
cout << "Enter the liters of water required out of the two jugs: ";
cin >> n;
cout << "Enter the capacity of the first jug: ";
cin >> x;
cout << "Enter the capacity of the second jug: ";
cin >> y;
if(n<x || n<y)
{ if(n%(__gcd(x,y))==0)
   s(n);
  else
  cout<<"This is not possible....\n";
}
else
  cout<<"This is not possible....\n";
}
```

==========travelling salesman==========

```
#include <iostream>

using namespace std;

int array[5][5],visited[5],n,cost=0;

int findSmallest(int c){
   int nc=99999;
   int min=99999, city_min;

   for(int i=0; i < n; i++){
      if((array[c][i]!=0) && (visited[i] == 0)){
         if(array[c][i]+array[i][c] < min){
            min=array[i][0]+array[c][i];
            city_min=array[c][i];
            nc=i;
         }
      }
   }

   if(min!=99999)
   cost+=city_min;
```

```
    return nc;
}

void totalCost(int city){
    int ncity;

    visited[city]=1;
    cout<<city+1<<" --> ";
    ncity = findSmallest(city);
    if(ncity==99999){
        ncity=0;
        cout<<ncity+1;
        cost+=array[city][ncity];
        return;
    }
    totalCost(ncity);
}


int main(){
    cout<<"\nEnter the number of cities : ";
    cin>>n;

    for(int i=0; i < n; i++){
        cout<<"\nEnter Elements of Row :"<<i+1;
        for(int j=0; j < n; j++){
            cout<<"\nEnter Elements from "<<i+1<<" to "<<j+1<<" : ";
            cin>>array[i][j];
        }
        visited[i]=0;
    }

    cout<<"\n\nThe cost list is:";

    for(int i=0;i < n;i++){
        cout<<"\n";
        for(int j=0;j < n;j++){
            cout<<array[i][j]<<"\t";
        }
    }

    cout<<"\n\nThe Path is : \n";
    totalCost(0);
    cout<<"\n\nTotal cost is : "<<cost;

    return 0;
}
```