الغاية من الجلسة: تطبيق خوارزمية الـDFS على مسألة الكرات والصناديق.

مقدمة:

يُعتبر استخدام خوارزميات البحث الذكية مثل البحث بالعمق (DFS) والبحث بالعرض (BFS) وخوارزمية A* أمراً حيوياً في حل مشاكل معقدة، وخاصةً عندما يتعلق الأمر بالبرمجة اللوجيكية باستخدام لغة البرولوغ (Prolog). يتيح استخدام هذه الخوارزميات الذكية تحقيق تفوق في فهم وحل مجموعة متنوعة من المسائل التي قد تكون صعبة بشكل كبير باستخدام أساليب أخرى.

البحث بالعمق (DFS):

يستخدم DFS في استكشاف الشجرة أو الغراف بالتحرك عميقاً في اتجاه واحد حتى يصل إلى الهدف أو يستنفد جميع الخيارات.

في Prolog، يمكن تمثيل عمليات البحث بشكل فعال باستخدام القواعد والحقائق.

البحث بالعرض (BFS):

يعتمد BFS على استكشاف الشجرة أو الغراف بالتحرك في جميع الاتجاهات على نفس المستوى قبل الانتقال إلى المستوى التالي.

في Prolog، يمكن تنظيم عمليات BFS باستخدام مفهوم القواعد والاستعلامات.

خوارزمية (A-star):

تجمع A* بين فعالية البحث بالعرض وذكاء البحث بالعمق.

تتميز A* بحساب التكلفة بدءاً من المصدر وكذلك التكلفة المتوقعة للوصول إلى الهدف وتستخدم هذه المعلومات في اتخاذ القرارات.

ويمكن ايضاً استخدام الـProlog لتنفيذ خوارزمية A* بشكل فعّال.

التكامل بين البحث الذكي وبرمجة اللوجيك:

يمكن تنفيذ خوارزميات البحث الذكية في برولوغ بشكل طبيعي، مما يتيح للمبرمجين استفادة من الخصائص الفريدة لكلا النهجين.

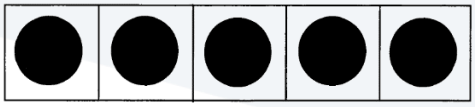
تستفيد Prolog من تمثيل المشكلات بشكل منطقي ومن التفاعل السهل مع القواعد والحقائق.

باستخدام خوارزميات البحث الذكية مع برمجة اللوجيك في Prolog، يصبح من الممكن تحقيق حلول فعّالة وذكية لمشاكل تتطلب استدلالاً لوجيكياً وتحليلاً عميقاً للمعلومات.

السودوكود الخاص بخوارزمية الـDFS:

```
begin
  open := [Start];                                  % initialize
  closed := [ ];
  while open ≠ [ ] do                               % states remain
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS            % goal found
        else begin
          generate children of X;
          put X on closed;
          discard children of X if already on open or closed;   % loop check
          put remaining children on left end of open            % stack
        end
    end;
  return FAIL                                       % no states left
end.
```

مسألة الكرات والصناديق Balls And Boxes Puzzle:

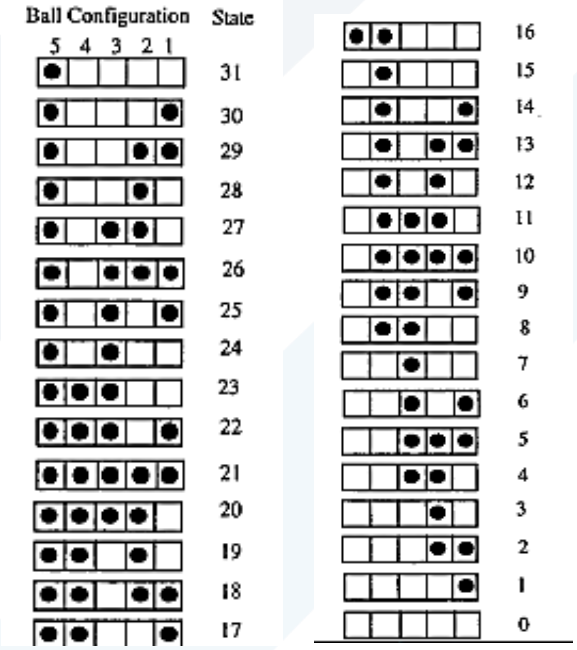


المسألة تبدأ من خمسة صناديق وفي كل صندوق توجد كرة.

الهدف هو إزالة جميع الكرات من الصناديق لكن العمليات الممكنة هي:

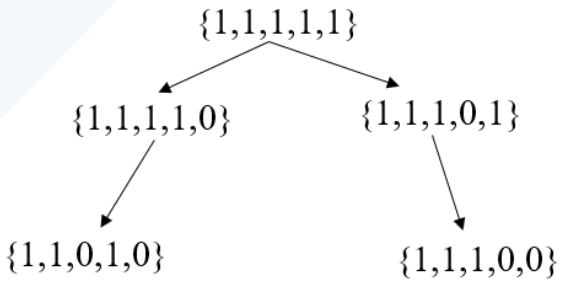
يمكنك أن تضع/تُخرِج الكرة من صندوقها عندما يكون الصندوق الذي على يمينها مباشرة توجد فيه كرة بينما باقي الصناديق التي على اليمين فارغة، باستثناء الصندوق أقصى اليمين يمكنك أن تضع/تُخرِج الكرة فيه بحرية تامة.

الشكل الآتي يمثل فضاء الحالات أي كل الحالات الممكنة:



لاحظ من الحالة 21 وصعوداً يمكنك ملاحظة الحل الذي يبدأ بإخراج الكرة أقصى اليمين ثم الكرة الثالثة من جهة اليمين ثم ... وهكذا حتى نصل إلى الهدف الذي هو صناديق خمسة فارغة.

لو جربنا أن نحل باستخدام الـDFS ستكون بداية شجرة البحث كالشكل الآتي (حيث أن 1 يعني وجود كرة و 0 يعني غياب كرة):



الكود الآتي يمثل توصيف المسألة وكذلك خوارزمية الـ DFS في البرولوغ:

```
addFacts(State,[]):-!.
addFacts(State,[H|T]):-assert(father(State,H)),write(father(State,H)),nl,addFacts(State,T).

findPath(Start,Start,ReverseAccumulator,ReverseAccumulator):-!.
                findPath(H,Start,ReverseAccumulator,Solution):-
        not(H=Start),father(X,H),findPath(X,Start,[X|ReverseAccumulator],Solution).


removeChildrenWhoAreInOpen([], Stack, []):-!.
removeChildrenWhoAreInOpen([H|T],Stack,[H|Tresult]):- ot(member(H,Stack)), removeChildrenWhoAreInClosed(T,Stack,
Tresult).
removeChildrenWhoAreInOpen([H|T], Stack, Result):- member(H,Stack), removeChildrenWhoAreInClosed(T,Stack, Result).


removeChildrenWhoAreInClosed([], Closed, []):-!.
removeChildrenWhoAreInClosed([H|T],Closed,[H|Tresult]):-not(member(H,Closed)),
removeChildrenWhoAreInClosed(T,Closed, Tresult).
removeChildrenWhoAreInClosed([H|T], Closed, Result):- member(H,Closed), removeChildrenWhoAreInClosed(T,Closed,
Result).


generateChildren(Start,Children):-putBall(Start,[],[],Children1),                pullBall(Start,[],[],Children2),
append(Children1,Children2,Children).



putBall([],Pres,Acc,Acc):-!.
putBall([H|T],Pres, Acc,Succ):- H=:=0, checkRight(T,Res), Res=true,
                                append(Pres,[1],S),append(S,T,L), append([L],Acc,Acc1),
                                append(Pres,[H],Pres1), putBall(T,Pres1,Acc1,Succ).
putBall([H|T],Pres, Acc,Succ):- H=:=0, checkRight(T,Res), Res=false,
                                append(Pres,[H],Pres1), putBall(T,Pres1,Acc,Succ).
putBall([H|T],Pres, Acc,Succ):- H=\=0,
                                append(Pres,[H],Pres1), putBall(T,Pres1,Acc,Succ).


pullBall([],Pres,Acc,Acc):-!.
```

```
pullBall([H|T],Pres, Acc,Succ):- H=:=1, checkRight(T,Res), Res=true,
                                    append(Pres,[0],S),append(S,T,L), append([L],Acc,Acc1),
                                    append(Pres,[H],Pres1), pullBall(T,Pres1,Acc1,Succ).
pullBall([H|T],Pres, Acc,Succ):- H=:=1, checkRight(T,Res), Res=false,
                                    append(Pres,[H],Pres1), pullBall(T,Pres1,Acc,Succ).
pullBall([H|T],Pres, Acc,Succ):- H=\=1,
                                    append(Pres,[H],Pres1), pullBall(T,Pres1,Acc,Succ).

checkRight([],true):-!.
checkRight([1],true):-!.
checkRight([H|T],true):- H=1, not(member(1,T)),!.
checkRight([H|T],false):- H=0; member(1,T).
```

القاعدة الآتية تتيح للمستخدم بتحديد حالة البداية وحالة الهدف ونوع الخوارزمية وكذلك متحول Solution والذي يحتوي على الحل الناتج.

```
solve(Start,Goal,Algorithm,Solution):-Algorithm=dfs, dfs(Start,Goal,[Start],[],[],Solution).
```

مثلاً يمكن كتابة الاستعلام الآتي:

```
?- solve([1,1,1,1,1], [0,0,0,0,0], dfs, Solution).


dfs(Start,Goal,[],Closed,ReverseAccumulator,Solution):-!.
```

الجزء الآتي يمثل الخوارزمية المستخدمة وهي DFS والتي نمرر لها حالة البداية والهدف والمكدس (بنية تخزين الأوراق) ومجموعة الحالات الموسعة أو المزارة Closed ومتحول من اجل بناء الحل بشكل عودي ReverseAccumulator والحل Solution.

```
dfs(Start,Goal,[H|T],Closed,ReverseAccumulator,Solution):-not(H=Goal),append(Closed,[H],Closed1),
                                    generateChildren(H,Children),
                                    removeChildrenWhoAreInClosed(Children, Closed1, FinalChildren),
                                    removeChildrenWhoAreInOpen(FinalChildren, T, FinalChildren1),
                                    addFacts(H,FinalChildren1),
                                     append(FinalChildren1,T,NewStack),
                                    dfs(Start,Goal,NewStack,Closed1,ReverseAccumulator,Solution).
dfs(Start,Goal,[H|T],Closed,ReverseAccumulator,Solution):-
H=Goal,write(H),write(','),findPath(H,Start,[H|ReverseAccumulator],Solution),dfs(Start,Goal,[],Closed,ReverseAccumulator,
Solution).
```