#ADS06 Backtracking

Rationale of the backtracking Algorithm

使用回溯算法的理由。

A sure–fire way to find the answer to a problem is to make a list of all candidate answers, examine each, and following the examination of all or some of the candidates, declare the identified answer.

穷举法一定能找到答案一个可能的找寻答案的方法是穷举并验证。

Backtracking enables us to eliminate（pruning） the explicit examination of a large subset of the candidates while still guaranteeing that the answer will be found if the algorithm is run to termination.

回溯使我们能够消除对大量候选者子集的显式检查，同时仍然保证如果算法运行到终止，将能够找到答案。

The basic idea is that suppose we have a partial solution. We add xi+1 and check if the answer set satisfies the constrains. If the answer is yes we continue to add, else we delete and backtrack to the previous partial solution.

The basic idea is that suppose we have a partial solution $(x_1, \ldots, x_i)$ where each $x_k \in S_k$ for $1 \le k \le i < n$. First we add $x_{i+1} \in S_{i+1}$ and check if $(x_1, \ldots, x_i, x_{i+1})$ satisfies the constrains. If the answer is "yes" we continue to add the next x, else we delete x_i and backtrack to the previous partial solution $(x_1, \ldots, x_{i-1})$.

剪枝pruning

回溯是一种算法思想，主要是通过递归来构建并求解问题，当发现不满足问题的条件时，就回溯寻找其他的满足条件的解。

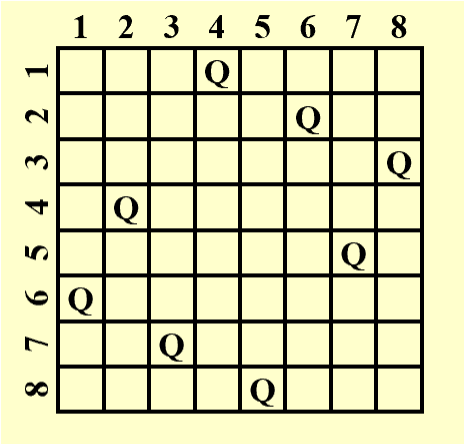
通过几道典型问题的求解，大致可以将回溯问题的求解步骤分为三个：

1. choices 即在这个问题中，我们可以做哪些事情。例如在数独空格中我们的选择有0 – 9是个数字。
2. constraint 即我们并不能随心所欲的选择，在做选择的同时，有一定的约束会限制我们的选择。比如数独空格中我们需要依据数独的规则，每个空格选填一个数字，并不是每个空格随心所欲的填0–9任意一个都可以。
3. goals 或者我们叫做递归的停止条件。 即随着选择的进行，我们什么时候就不用再做选择了，即问题求解完成

Example 1: Eight Queens

Find a placement of **8 queens on an 8 ´ 8 chessboard such that no two queens attack**

> **Two queens are said to attack iff they are in the same row, column, diagonal, or antidiagonal of the chessboard.**

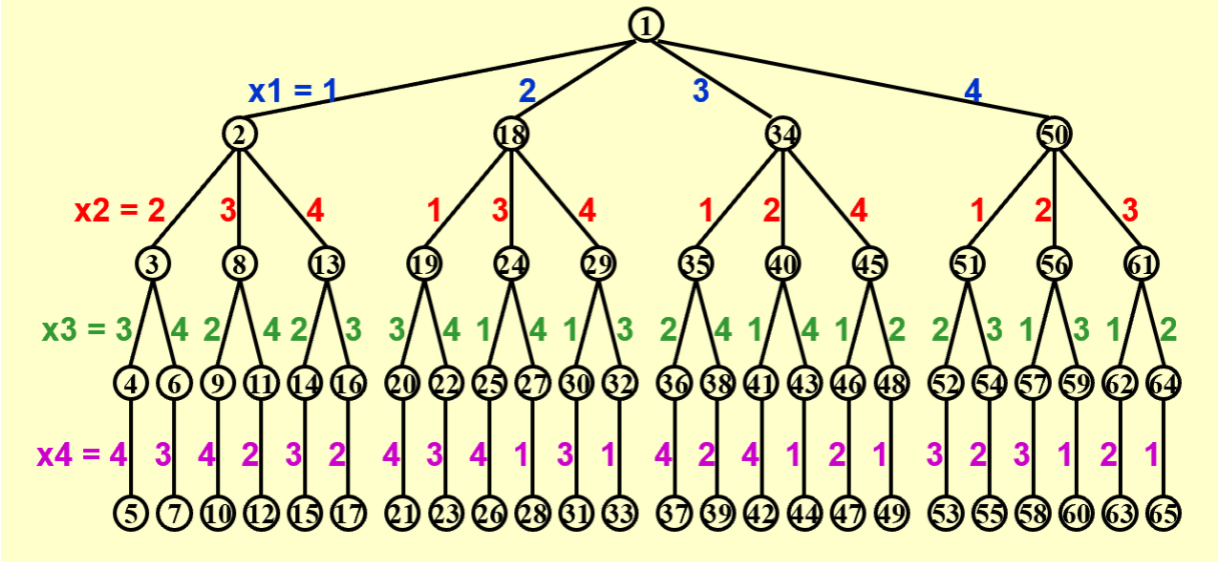


Constrains: ① $S_i = \{ 1,2,3,4,5,6,7,8 \}$ for $1 \le i \le 8$

② $x_i \neq x_j$ if $i \neq j$ ③ $(x_i - x_j) / (i - j) \neq \pm 1$

Method:

- Construct a **game tree**



Each path from the root to a leaf defines an element of the solution space.

- Perform a **depth–first** search (post–order traversal) to examine the paths

No tree is actually constructed. The game tree is just an abstract concept.

Example 2: The Turnpike Reconstruction Problem

收费公路重构问题

Given N points on the x–axis with coordinates $x_1 < x_2 < \ldots < x_N$. Assume that $x_1 = 0$. There are N (N — 1) / 2 distances between every pair of points.

Given N (N — 1) / 2 distances. Reconstruct a point set from the distances.

Find the next largest distance and check.

驱动例程：

C | 复制代码

```c
int Turnpike (int X[],DistSet D, int N) //drive
{
    X[1] = 0;
    X[N] = Delete(D);
    X[N-1] = Delete(D);
    if(X[N]-X[N-1] in D)
    {
        Remove(X[N]-X[N-1],D);
        return Place(X,D,N,2,N-2);
    }
    else
        return False;
}
```

C 复制代码

```c
//回溯
//X[Left ... Right]
//X[1 ... Left - 1] and X[Right + 1 ... N]
// are already tentativelu placed
int Place(int X[],DistSet D,int N,int left,int right)
{
    int DMax,Found=false;

    if(IsEmpty(D))
        return true;
    DMax = FindMax(D);

    if(|X[j] - DMax| in D for all 1 <= j < left, right < j <= N)
    {//已确定的点中是否满足D
        X[right]=DMax;
        for( 1 <= j < left,right < j <= N)
            Delete(|X[j] - DMax|,D);
        Found = Place(X,D,N,left,right-1);

        if(!Found){ //Backtrack
           for( 1 <= j < left,right < j <= N)
               Insert(|X[j] - DMax|,D);
        }
    }

    if(!Found && (|X[N] - DMax - X[j]| in D for all 1 <= j < left and right < j <= N){
        X[left] = X[N] - DMax;
         for(1 <= j < left,right < j <= N)
            Delete(|X[N] - DMax - X[j]|,D);
        Found = Place(X,D,N,left + 1,right);

        if(!Found){
           for(1 <=  j < left,right < j <= N)
               Insert(|X[N] - DMax - X[j]|,D);
    }

   return Found;
}
```

回溯的效率跟S的规模、约束函数的复杂性、满足约束条件的结点数相关。

约束函数决定了剪枝的效率，但是如果函数本身太复杂也未必合算。

满足约束条件的结点数最难估计，使得复杂度分析很难完成。

Example 3 : Tic–tac–toe

The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

19,683 possible board layouts (3^9 since each of the nine spaces can be X,O,blank)

9! possible games

Minimax Strategy

Use an evaluation function to quantify the goodness of a position.

使用一个赋值函数来给一个位置的好坏定值

能使计算机获胜的位置得到+1，平局0，输棋–1

$$f(P) = W_{Computer} - W_{Human}$$

where W is the number of potential wins a position P.

通过考察盘面能够确定这局棋输赢的位置叫做终端位置terminal position

如果一个位置不是终端位置，那么该位置的值通过递归地假设双方最优棋步而确定，这叫做极小极大策略。

位置P的后继位置successor position是通过从P走一步棋可以到达的任何位置P_s

The human is trying to minimize the value of the position p, while the computer is trying to maximize it.

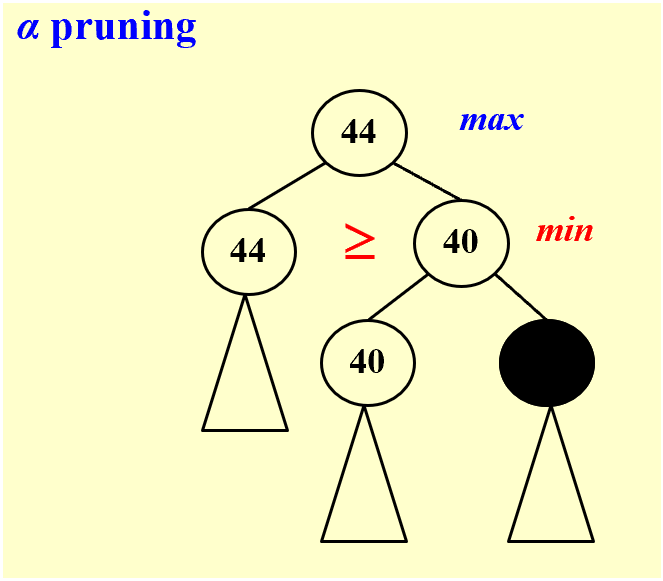
C | 复制代码

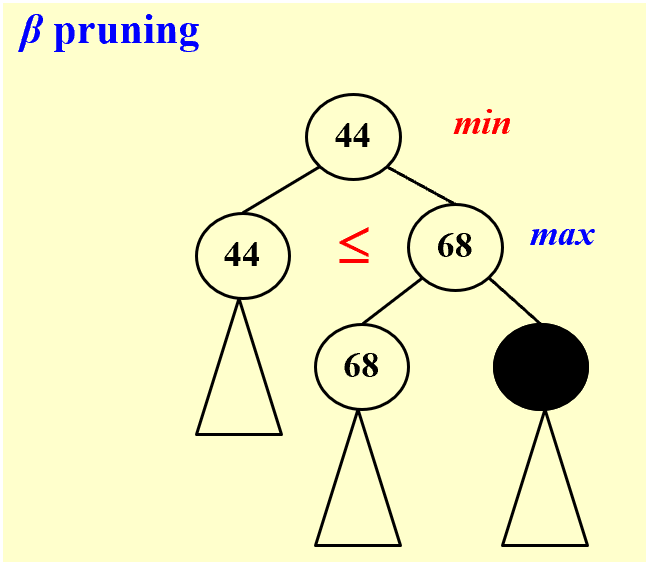
```c
void FindCompMove(BoardType Board, int *BestMove, int *Value){
    int Dc,i,Response; //Dc means don't care

    if(FullBoard(Board))  //plate
        *Value = Draw;
    else if (ImmediateCompWin(Board,BestMove)) //Win
        *Value = CompWin;
    else{ // not a terminal
        *Value = CompLoss;
        for(i=1;i<=9;i++){ //Try each square
            if(IsEmpty(Board,i))//如果是空的就放置
            {
                Place(Board,iComp);
                FindHumanMove(Board,&Dc,&Response);
                Unplace(Board,i); //restore board

                if(Response > *Value){
                    //update best value
                    *Value = Response;
                    *BestMove = i;
                }
            }
        }
    }
}
```

```c
void
    FindHumanMove(BoardType Board, int *BestMove, int *Value)
{
    int DC,i,Response;

    if(FullBoard(Board))
        *value = Draw;
    else if(ImmediateHumanWin(Board,BestMove))
        *Value = CompLoss;
    else
    {
        *value = CompWin;
        for(i = 1;i<=9;i++)
        {
            if(IsEmpty(Board,i))
            {
                Place(Board,i,Human);
                FindCompWin(Board,&Dc,$Response);
                Unplace(Board,i);

                if(Response < *Value)
                {
                    *value = Response;
                    *BestMove = i;
                }
            }
        }
    }
}
```

人们一般能够取得的最重要的改进称为 $\alpha - \beta$ 裁剪，（$\alpha - \beta$ pruning）

对于博弈树game tree





基本思想：根据倒推值的计算方法，或中取大，与中取小，在扩展和计算过程中，能剪掉不必要的分枝，提高效率

定义：

□α值：有或后继的节点，取当前子节点中的最大倒推值为其下界，称为α值。节点倒推值>=α；

□β值：有与后继的节点，取当前子节点中的最小倒推值为其上界，称为β值。节点倒推值<=β；

□α-β 剪枝：

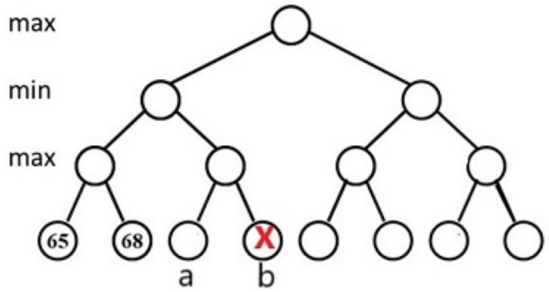
（1） β剪枝：节点x的α值不能降低其父节点的β值，x以下的分支可停止搜索，且x的倒推值为α□；

（2） α 剪枝：节点x的β值不能升高其父节点的α值，x以下的分支可停止搜索，且x的倒推值为β□ ；

2-5 分数 3

作者 何钦铭 单位 浙江大学

Given the following game tree, if node **b** is pruned with α-β pruning algorithm, which of the following statements about the value of node **a** is correct?



- A. greater than 65
- B. less than 65
- C. greater than 68
- D. less than 68

答案错误：0 分 创建提问