Backtracking

Wladimir Araújo Tavares [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

8 de março de 2020

O que é Backtracking?

- Backtracking é um algoritmo genérico para encontrar todas (ou algumas) soluções para algum problema computacional.
- O algoritmo constrói incrementalmente possíveis candidatos para a solução e abandona candidatos que não podem gerar soluções para o problema.
- É um refinamento do método de enumeração bruta.
- É uma importante ferramenta para resolver problemas de satisfação de restrições.

Algoritmo Genérico

Algorithm 1 Algoritmo Genérico de Backtracking

```
1:  function solve(c)
2:      if completo(c) then
3:          if aceita(c) then
4:              imprima(c)
5:              return true
6:          else
7:              return false
8:      for cada filho c′ de c do
9:          if solve(c') then
10:             return true
```

Problema 1: Soma de subconjuntos

P1 Enoque e Leonardo encontraram um tesouro secreto no campus de Quixadá. O tesouro é composto por n itens de valores diferentes $v_1, v_2, \ldots, v_n$. Eles querem saber se é possível particionar os itens do tesouro em dois subconjuntos de tal maneira que ambos recebam o mesmo valor.

Problema 1: Soma de subconjuntos

```
#include <stdio.h>
#include <iostream>
#include <vector>
using namespace std;
int solve(vector<int> & objetos, int meta, int i, int soma);
int main(){
    int n,meta;
    vector<int> objetos;
    cin >> n;
    objetos.resize(n);
    meta = 0;
    for(int i = 0; i < n; i++){
        cin >> objetos[i];
        meta += objetos[i];
    }
    if(meta%2!= 0) cout << "N" << endl;
    else {
        meta = meta/2;
        cout << (solve(objetos,meta,0,0) ? "S" : "N") << endl;
    }
}
```

Problema 1: Soma de subconjuntos

```
int solve(vector <int> & objetos, int meta, int i, int soma){
    if( i == (int)objetos.size() ){
        if( soma == meta ) return 1;
        else return 0;
    }
    if( solve(objetos, meta, i+1, soma) )
        return true;
    if( solve(objetos, meta, i+1, soma + objetos[i]) )
        return true;
    return false;
}
```

Problema 2: N-Queen

P2 Determine quantas maneiras podemos colocar 8 rainhas em tabuleiro de xadrez de maneira que nenhuma rainha ataque as outras rainhas do tabuleiro.

Problema 2: N-Queen

Variáveis booleanas:

- diagonais principais: $d1[-(N-1)\ldots N-1]$
- diagonais secundarias: $d2[0\ldots 2\times(n-1)]$
- linha: $linha[0\ldots n]$

Problema 2: N-Queen

Algorithm 2 N-Queen

```
1:  function queen(j)
2:      if Se solução completa válida then
3:          registre solução
4:      for  cada i tal que linha[i] == true  do
5:          if  d1[i − j] == true E d2[i + 2] == true then
6:              d1[i − j] ← false
7:              d2[i + j] ← false
8:              linha[i] ← false
9:              queen(j+1)
10:             d1[i − j] ← true
11:             d2[i + j] ← true
12:             linha[i] ← true
```

Problema 2: N-Queen

```
int contador;
vb linha, d1, d2;
vector<int> x;
void queen(int j, int n);
int main(){
   int n;
   cin >> n;
   linha.assign(n, true);
   d1.assign(2*n-1, true);
   d2.assign(2*n-1 , true);
   x.resize(n);
   contador = 0;
   clk = clock();
   queen(0, n);
}
```

Problema 2: N-Queen

```
void queen(int j, int n){
  if( j == n ){
        contador++;
    }else{
        for(int i = 0; i < n; i++){
            if(  linha[i] && d1[i-j+n-1] && d2[i+j]){
                linha[i] = false;
                d1[i-j+n-1] = false;
                d2[i+j] = false;
                x[j] = i;
                queen(j+1, n);
                linha[i] = true;
                d1[i-j+n-1] = true;
                d2[i+j] = true;
            }
        }
    }
}
```

Problema 2: N-Queen

```
4
--------------------------------------------------
..Q.
Q...
...Q
.Q..
[2 , 0 , 3 , 1]
--------------------------------------------------
--------------------------------------------------
.Q..
...Q
Q...
..Q.
[1 , 3 , 0 , 2]
--------------------------------------------------
2
```

Problema 3 - Resta um

P3 Resta um é jogo interessante. Neste jogo, você tem um tabuleiro com dozes buracos em uma linha. No começo de cada jogo, alguns buracos estão ocupados por pedrinhas. Um movimento é possível se há uma linha reta de três buracos adjacentes, chamados A, B e C, onde A e B estão com uma pedrinha e C está vazio. O movimento consiste em mover a pedrinha de A para C e retirar a pedrinha que está em B. Sua missão é encontrar uma sequência de movimentos tal que reste o menor número de pedrinhas no tabuleiro.

Problema 3 - Resta um

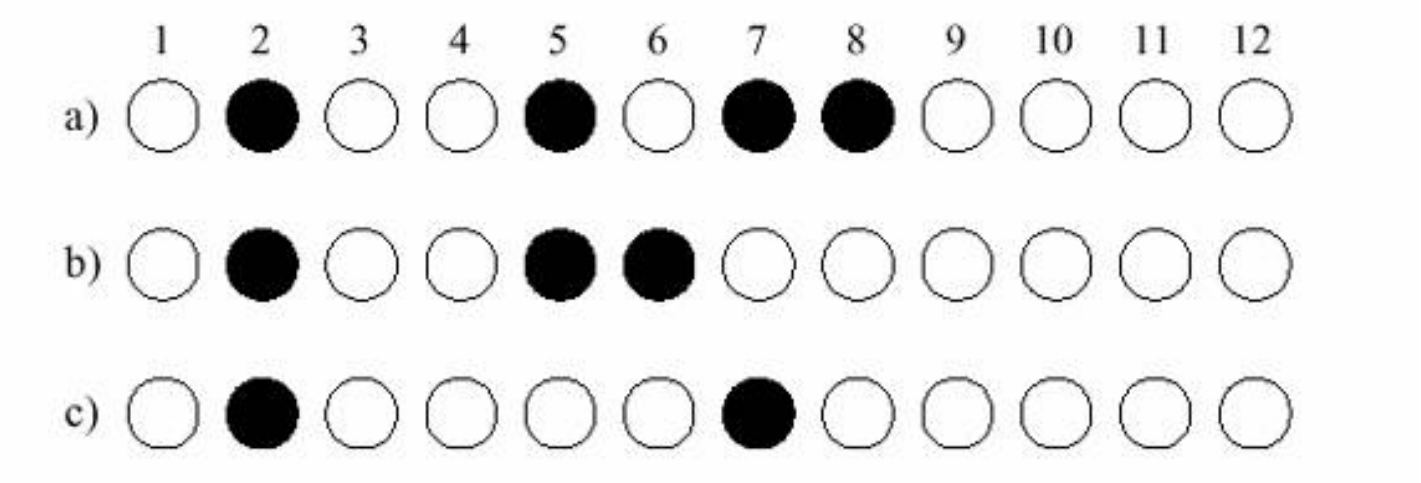

Figura: Em a), dois movimentos são possíveis $8 \rightarrow 6$ e $7 \rightarrow 9$. Em b), o resultado do movimento de $8 \rightarrow 6$ sobre tabuleiro em a) pode ser visto. Em c), o resultado do movimento $5 \rightarrow 7$ sobre o tabuleiro b) é mostrado.

Problema 3 - Resta um

Entrada	Saída
5	
---oo-------	1
-o--o-oo----	2
-o----ooo---	3
oooooooooooo	12
ooooooooooo-o	1

Problema 3 : RestaUm

```
int cnt;
void restaUm(char * aux){
  int n = strlen(aux);
  int contador = 0;
  for( int i = 0; i < n; i++){
      if( aux[i] == 'o'){
          contador++;
      }
  }
  if(contador < cnt) cnt = contador;
  for(int i = 0; i < n; i++){
    if( aux[i] == 'o'){
      if( i+2 < n && aux[i+1] == 'o' && aux[i+2]=='-'){
        aux[i]   = '-'; aux[i+1] = '-'; aux[i+2] = 'o';
        restaUm(aux);
        aux[i]   = 'o'; aux[i+1] = 'o'; aux[i+2] = '-';
      }
      if(i-2 >= 0 && aux[i-1]=='o' && aux[i-2]=='-'){
        aux[i]   = '-'; aux[i-1] = '-'; aux[i-2] = 'o';
        restaUm(aux);
        aux[i]   = 'o'; aux[i-1] = 'o'; aux[i-2] = '-';
      }
    }
  }
```

Problema 4: Passeio do Cavalo

P4 Faça um cavalo percorrer todas as casas de um tabuleiro de xadrez de forma a não repetir nenhuma posição pela qual já passou começando da posição (0,0).

Problema 4: Passeio do cavalo

Algorithm 3 Passeio do cavalo

```
1:  function cavalo
2:      if tabuleiro está completo then
3:          imprima tabuleiro
4:          return sucesso
5:      while existe movimento não realizado do
6:          selecione um movimento
7:          if movimento aceitável then
8:              registre movimento
9:              chamada recursiva cavalo
10:             if chamada não sucedida then
11:                 desfaz movimento
12:             else
13:                 return sucesso
14:     return fracasso
```

Problema 4 : Passeio do Cavalo

```
int t[8][8];
int dx[8] = {2, 1,-1,-2, -2, -1, 1,  2};
int dy[8] = {1, 2, 2, 1, -1, -2,-2, -1};
int cavalo(int i, int x, int y){
 int u,v,k,q;
 if(i==65){ imprime(); return 1;}
 for(k=0;k<8;k++){
  u = x + dx[k];  v = y + dy[k];
  if( (u>=0 && u<=7) && (v>=0 && v<=7)){ //testa limites
   if(t[u][v]==0){ //posicao livre
    t[u][v]=i; //registre o movimento
    q = cavalo(i+1,u,v);
    if(q==0) t[u][v]=0; //se ãno çalcanou todos, çdesfaa
    else return 1; // se çalcanou todos, retorne 1
   }
  }
 }
 return 0;
}
int main(){
 memset(t,0,sizeof(t));
 t[0][0]=1;
 cavalo(2,0,0);
}
```

Problema 4 : Passeio do Cavalo

```
 1 60 39 34 31 18  9 64
38 35 32 61 10 63 30 17
59  2 37 40 33 28 19  8
36 49 42 27 62 11 16 29
43 58  3 50 41 24  7 20
48 51 46 55 26 21 12 15
57 44 53  4 23 14 25  6
52 47 56 45 54  5 22 13
```

Problema 5: Flood Fill

P5 Você deve analisar uma imagem de uma zona rural, gerada por satélite, para determina quantas construções existem na área da imagem. Uma célula escura adjacentes pertecem à uma mesma construção. Células adjacentes são vizinhas imediatas nas direções horizontal, vertical ou diagonal.

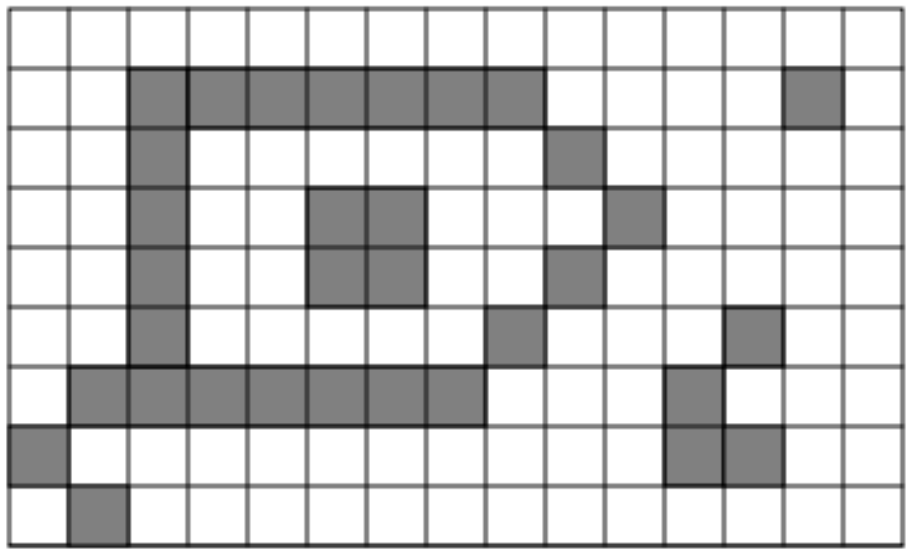

Figura: Imagem com 4 construções.

Problema 5: Flood Fill

```
Entrada
9 15
000000000000000
007677888000080
008000000900000
006003300080000
005003300800000
004000009000600
034556780005000
300000000004400
020000000000000í

Sada
4
```

Problema 5: Flood Fill

Algorithm 4 FloodFill

```
1:  function construções
2:      cont ← 0
3:      for cada construção na posição (i,j) do
4:          apaga(i,j)
5:          cont ← cont + 1
        return cont
6:  function apaga(int i, int j)
7:      imagem[i][j] ← 0
8:      for cada (u, v) ← adjacente(i, j) do
9:          if (u, v) está no tabuleiro then
10:             if imagem[u][v] ≠ 0 then
11:                 apaga(u,v)
```

Problema 6: Maior clique

P6 Encontre a maior clique em um grafo.

Problema 6: Maior Clique

```
Entrada
5 6
0 1
0 3
1 2
2 3
2 4
3 4í
Sada
3
```

Problema 6: Maior Clique

Algorithm 5 Maior Clique

```
1: function CLIQUE(C, P)
2:     if |C| > |best| then
3:         best ← C
4:     if |C| + |P| > |best| then
5:         Escolha v ∈ P
6:         CLIQUE(C ∪ {v}, P ∩ N(v))
7:         CLIQUE(C, P \ {v})
```