

UNIVERSIDADE ESTADUAL DE MARINGÁ

Algoritmo Guloso para Busca em grafo
implementado em PROLOG

Aluno: Thiago Rodrigo Bucalão

RA: 68962

2014 Maringá

1. Definição da estratégia gulosa:

Segundo Cormen um algoritmo guloso obtém a solução ótima para um problema fazendo uma sequência de escolhas. Para cada iteração o algoritmo escolhe a opção que parece ser a melhor no momento, nem sempre é possível chegar em uma solução ótima utilizando essa técnica.

A pergunta a ser feita é como obter um algoritmo guloso para um problema? Esse tipo de heurística funciona para qualquer tipo de problemas encontrados na computação?

Portanto para se desenvolver um algoritmo guloso de modo geral devemos seguir as seguintes etapas:

1. Modelar o problema de otimização para um problema no qual é possível fazer uma escolha e ficamos com um único subproblema para resolver;
2. Provar que sempre existe uma solução ótima para o problema original que torna a escolha gulosa, de modo que essa escolha é sempre a melhor;
3. Demonstrar que tendo feito essa escolha, o que resta é um subproblema que se combinarmos a solução ótima para o subproblema com a escolha que acabamos de tomar no final teremos chegado a uma solução ótima para o problema como um todo.

Portanto não há como saber se um determinado problema é capaz de ser resolvido utilizando o método guloso ou não, porém, se conseguirmos mostrar essas três propriedades pode se dizer que estamos no caminho certo.

O ponto chave para saber se é possível desenvolver um algoritmo guloso é a **propriedade de escolha gulosa**: é possível chegar em uma solução ótima global a partir de decisões ótimas locais. Dessa forma dizemos que, a escolha gulosa é feita de cima para baixo pois essa escolha no momento depende de escolhas feitas anteriormente porém, não depende de escolhas futuras.

Outra característica do método guloso é a **subestrutura ótima** ao qual se uma solução ótima é encontrar solução ótima para problema a partir de soluções ótimas dos subproblemas.

Para poder facilitar o entendimento vamos demonstrar dois problemas que é possível ser aplicado um algoritmo guloso, o problema do troco e problema da mochila.

2. Problema do Troco:

Suponha que tenhamos disponíveis moedas com valores de 100, 25, 10, 5 e 1. O problema é criar um algoritmo que para conseguir obter um determinado valor com o menor número de moedas possível (problema do troco).

Troco(n)

1. $C \leftarrow \{ 100, 50, 10, 5, 1 \}$
2. $S \leftarrow \emptyset$ //Conjunto de moedas escolhida
3. $soma \leftarrow 0$ //Soma obtida
4. $i \leftarrow 0$
5. enquanto $soma \neq n$ faça
6. //Faça a escolha gulosa
7. $x \leftarrow$ é o maior item em C tal que $x + s \leq n$
8. se não existe tal item então
9. retorne “solução não encontrada”
10. fim-se
11. //Dependendo dos valores escolhidos
12. //alguns valores poderiam não ser construídos
13. $S = S \cup \{ \text{valor da moeda } x \}$
14. $soma = soma + x$
15. fim-enquanto

Simplificando ficaria:

Troco(n)

1. $C \leftarrow \{ 100, 50, 10, 5, 1 \}$
2. $S \leftarrow \emptyset$ //Conjunto de moedas escolhida
3. $soma \leftarrow 0$ //Soma obtida
4. $i \leftarrow 1$
5. enquanto $i \leq 5$ e $soma \neq n$ faça
6. se $s + C[i] \leq n$ então
16. $S = S \cup \{i\}$
7. $soma = soma + C[i]$
8. Senão
9. $i = i + 1$

Entretanto, ressalta-se que para diferentes valores de moedas, ou então quando se tem uma quantidade limitada de cada uma, o algoritmo guloso pode vir a não chegar em uma solução ótima, ou até mesmo não chegar a solução nenhuma (mesmo esta existindo). Suponha que os valores disponíveis seja $C = \{10, 7, 2, 1\}$ e precisamos dar o troco de 14 centavos. A solução dada pelo algoritmo guloso é 3 moedas: 1 de valor 10 e 2 de valor 2. Porém, o menor número de moedas é 2 moedas: 2 de valor 7.

Teorema: O algoritmo guloso produz a solução com o menor número de moedas possível. *Suponha por absurdo que existe uma solução ótima S' que utiliza menos moedas que a solução encontrada pelo algoritmo S . Seja q' o número de moedas de 100 utilizada na solução S' e q o número de moedas 100 encontrada pelo algoritmo guloso. Como a solução S' é melhor que S temos que $q' < q$. Se $q' < q$ então a solução S' utiliza um menos de moedas de 100 do que o permitido. Logo, a solução S' utiliza moedas de 50, 10, 5, 1 para obter cada moeda de 100. Podemos encontrar uma solução $S' + \{ \text{um moeda de 100} \} - \{x+y+z+w \mid 50x+10y+5z+w=100\}$. Essa solução é melhor do que a solução ótima S' . Absurdo!*

GulosoGeral(P)

1. $S \leftarrow \emptyset$
2. enquanto S não for uma solução de P faça
3. $x \leftarrow$ é um elemento selecionado
4. se $S \cup \{x\}$ é viável para P então
5. $S = S \cup \{x\}$
6. fim-se
7. fim-enquanto

3. O problema da mochila fracionária

O problema da mochila fracionária também é conhecido como problema fracionário da mochila e como problema da mochila contínua. O problema consiste em supondo que um ladrão queira roubar uma loja e quer levar a maior quantidade em valor de produtos, porém, pode ocorrer que apesar do produto A ter um valor econômico mais alto que o produto B não seja vantajoso levar o A pois, ele tem três vezes mais o peso do B.

Resumindo tem que encher a mochila com os produtos de maior valor sem que não ultrapasse a quantidade de peso que é possível levar. Desta forma dados vetores (x_1, x_2, \dots, x_n) e (p_1, p_2, \dots, p_n) , denotaremos por $x \cdot p$ o produto escalar de x por p . Portanto, $x \cdot p = x_1 p_1 + x_2 p_2 + \dots + x_n p_n$. Diremos que um vetor (x_1, x_2, \dots, x_n) é *racional* se todos os seus componentes forem números racionais e *natural* se todos os seus componentes forem números naturais.

Diremos que (p_1, \dots, p_n) é o vetor de *pesos*, que (v_1, \dots, v_n) é o vetor de *valores* e que c é a *capacidade* do problema. Diremos que uma *mochila fracionária viável* é qualquer vetor racional (x_1, \dots, x_n) tal que $x \cdot p \leq c$ e $0 \leq x_i \leq 1$ para todo i . O *valor* de uma mochila x é o número $x \cdot v$. Nosso problema é encontrar uma mochila viável de valor máximo.

EXEMPLO: Suponha $c = 50$ e $n = 4$. A figura abaixo dá os valores de p e v . Mais abaixo, temos uma solução x do problema da mochila fracionária. O valor dessa solução é $x \cdot v = 1040$. (A solução do correspondente problema booleano tem valor 1000.)

p	40	30	20	10	20
v	840	600	400	100	300
x	1	1/3	0	0	0

O seguinte algoritmo guloso resolve o problema da mochila fracionária. O algoritmo exige que os dados estejam em ordem crescente de valor específico (ou seja, valor por unidade de peso):

$$v_1/p_1 \leq v_2/p_2 \leq \dots \leq v_n/p_n$$

(se $p_i = 0$ então v_i/p_i é infinito). É nesta ordem "mágica" que está o segredo do funcionamento do algoritmo.

MOCHILA-FRACIONÁRIA (p, v, n, c)

```
1    $j \leftarrow n$ 
2   enquanto  $j \geq 1$  e  $p_j \leq c$  faça
3        $x_j \leftarrow 1$ 
4        $c \leftarrow c - p_j$ 
5        $j \leftarrow j - 1$ 
6   se  $j \geq 1$  então
7        $x_j \leftarrow c/p_j$ 
8       para  $i \leftarrow j-1$  decrescendo até 1 faça
9            $x_i \leftarrow 0$ 
10  devolva  $x$ 
```

O algoritmo é guloso porque, em cada iteração, abocanha o objeto de maior valor específico dentre os disponíveis, sem se preocupar com o que vai acontecer depois. O algoritmo jamais se arrepende do valor atribuído a um componente de x .

4. Vantagens e Desvantagens do algoritmo guloso:

4.1. Pontos positivos dos algoritmos gulosos:

- Geralmente são algoritmos simples de se construir;
- Baixa complexidade;

4.2. Pontos negativos dos algoritmos gulosos:

- Nem sempre conduz à soluções ótimas globais. Podem efetuar cálculos repetitivos;
- Escolhe o caminho que é mais econômico à primeira vista;
- Pode entrar em loop se não detectar a expansão de estados repetidos;
- Pode tentar desenvolver um caminho infinito.

5. Conceito de *straight line distance* (distância em linha reta):

A distância mínima entre dois pontos em um plano é por definição uma reta, isso é mostrado da seguinte forma:

O menor caminho ligando dois pontos $X = (x_1, x_2)$ e $Y = (y_1, y_2)$ é igual a:
 $d'(X, Y) = |x_1 - y_1| + |x_2 - y_2|$.

Portanto essa heurística é utilizada no nosso problema para se obter o melhor caminho saindo de Arad indo até Bucharest.

6. Explicação do trabalho

No desenvolvimento do trabalho devíamos utilizar busca gulosa para sair de Arade até Bucharest, portanto temos:



Figura 1: Mapa simbolizado por estado

- Estado Inicial = Arad.

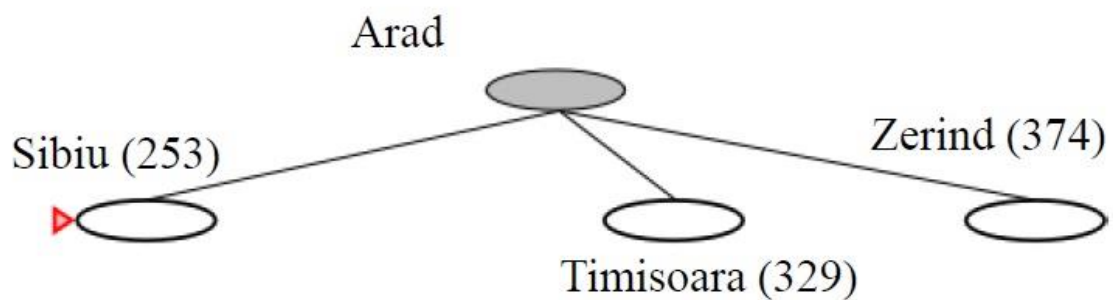


Figura 2: Mapa simbolizado por estado

O primeiro passo de expansão produz: Sibiu, Timisoara e Zerind. A busca gulosa irá selecionar Sibiu pois essa é a cidade mais próxima no momento para Bucharest.

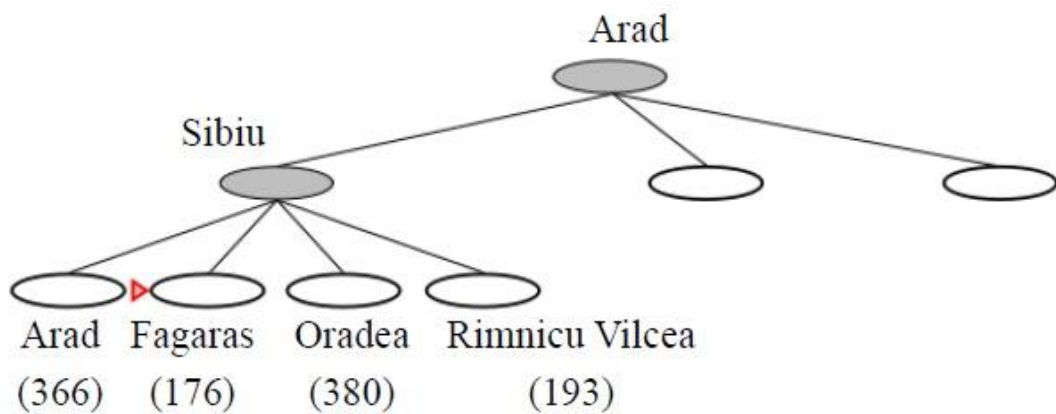


Figura 3: Mapa simbolizado por estado

Quando Sibiu for selecionada então vai trazer como opção as suas cidade vizinhas que são: Arad, Fagara, Oradea e Rimnicu Vilcea. A busca gulosa deverá selecionar Fagaras.

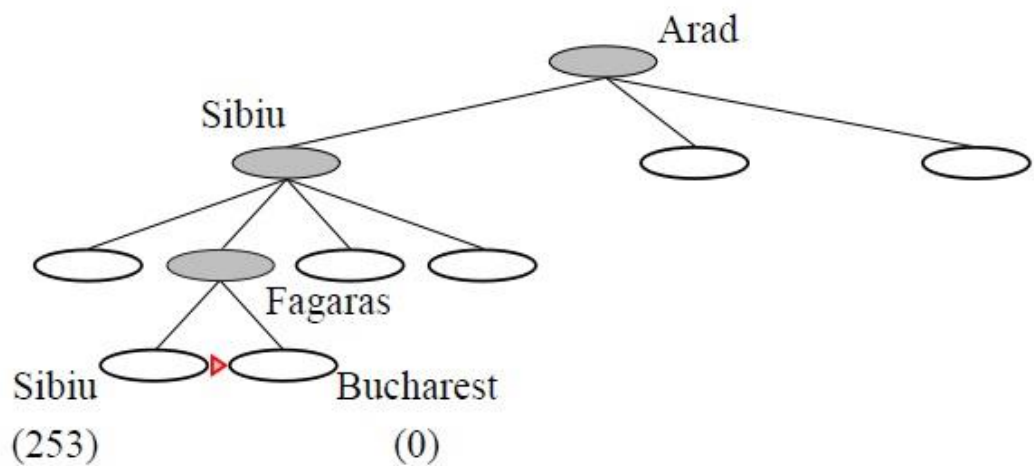
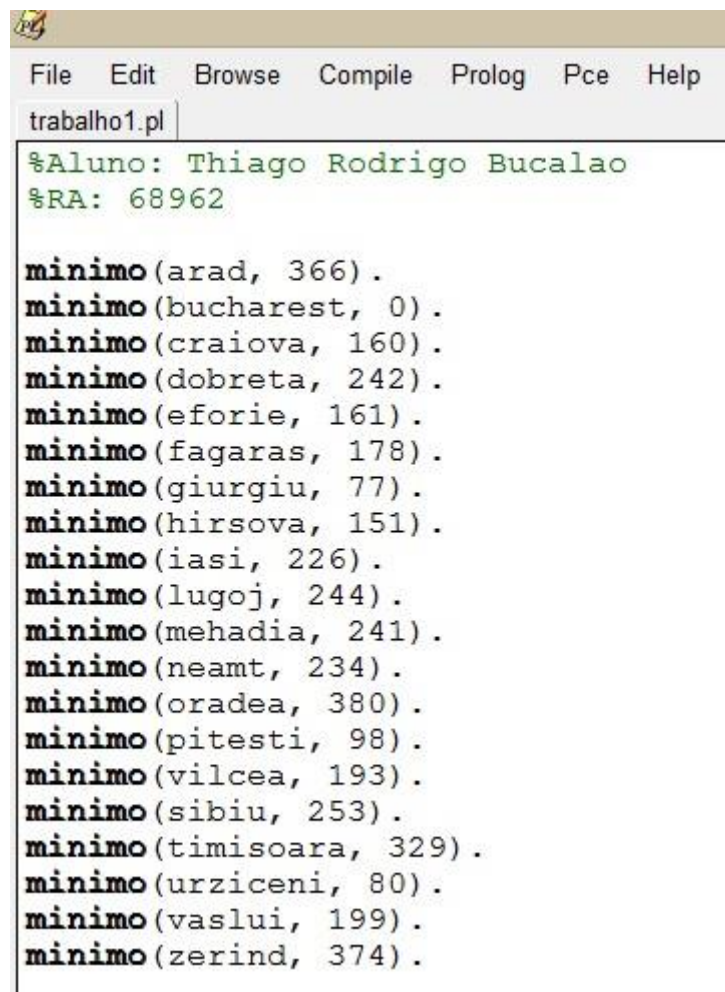


Figura 4: Mapa simbolizado por estado

E por fim, estando em Fagaras posso ir para Sibiu ou Buchares, neste caso vou para Bucharest com o objetivo alcançado. Entretanto esse não é o caminho ótimo.

6.1. Código:



```
File Edit Browse Compile Prolog Pce Help
trabalho1.pl
%Aluno: Thiago Rodrigo Bucalao
%RA: 68962

minimo(arad, 366).
minimo(bucharest, 0).
minimo(craiova, 160).
minimo(dobreta, 242).
minimo(eforie, 161).
minimo(fagaras, 178).
minimo(giurgiu, 77).
minimo(hirsova, 151).
minimo(iasi, 226).
minimo(lugoj, 244).
minimo(mehadia, 241).
minimo(neamt, 234).
minimo(oradea, 380).
minimo(pitesti, 98).
minimo(vilcea, 193).
minimo(sibiu, 253).
minimo(timisoara, 329).
minimo(urziceni, 80).
minimo(vaslui, 199).
minimo(zerind, 374).
```

Figura 5: Base de Dados do Código em Prolog

Os fatos apresentados acima são os do nome da cidade e a distância que elas estão para se chegar a Bucharest.

```
idadesvizinhas(zerind, [arad, oradea]).
idadesvizinhas(oradea, [zerind, sibiu]).
idadesvizinhas(timisoara, [arad, lugoj]).
idadesvizinhas(lugoj, [timisoara, mehadia]).
idadesvizinhas(mehadia, [dobreta, lugoj]).
idadesvizinhas(dobreta, [mehadia, craiova]).
idadesvizinhas(craiova, [vilcea, pitesti]).
idadesvizinhas(vilcea, [sibiu, pitesti, craiova]).
idadesvizinhas(pitesti, [bucharest, vilcea, craiova]).
idadesvizinhas(sibiu, [arad, oradea, vilcea, fagaras]).
idadesvizinhas(fagaras, [bucharest, sibiu]).
idadesvizinhas(bucharest, [pitesti, fagaras, giurgiu, urziceni]).
idadesvizinhas(giurgiu, [bucharest]).
idadesvizinhas(urziceni, [bucharest, hirsova, vaslui]).
idadesvizinhas(hirsova, [eforie, urziceni]).
idadesvizinhas(eforie, [hirsova]).
idadesvizinhas(vaslui, [urziceni, iasi]).
idadesvizinhas(iasi, [vaslui, neamt]).
idadesvizinhas(neamt, [iasi]).
idadesvizinhas(arad, [sibiu, timisoara, zerind]).
```

Figura 6: Base de Dados do Código em Prolog

A figura 6 mostra os fatos das cidades e uma lista com as suas vizinhas.

```
pesquisa(CidadeAtual):-
    write('Esta em: '),
    write(CidadeAtual),
    write('\n'),
    melhorvizinho(CidadeAtual, ProximaCidade2),
    write('Vai para: '),
    write(ProximaCidade2),
    write('\n'),
    write('Continuar? (s/sim | n/nao) '),

    read(Condicao),
    continuacao(Condicao),
    write('\n'),

    pesquisa(ProximaCidade2).
```

Figura 6: Código em Prolog

Acima temos a função que dará início a busca pelo mapa portanto é necessário que passe na linha de comando do PROLOG **pesquisa(arad)**. para dar início a pesquisa.

O restante do código não será está no anexo junto com o trabalho

7. Referências bibliográficas:

<http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/guloso.html> visitado em 25/09/14 às 13:00 h.

<http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-frac.html> visitado em 25/09/14 às 10:40 h.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Algoritmos Teoria e Prática 2ª edition, MIT Press & McGraw-Hill, 2001;

Jon Kleinberg, Éva Tardos, *Algorithm Design*, Addison-Wesley, 2005