DACC | Departamento Acadêmico de Ciência da Computação

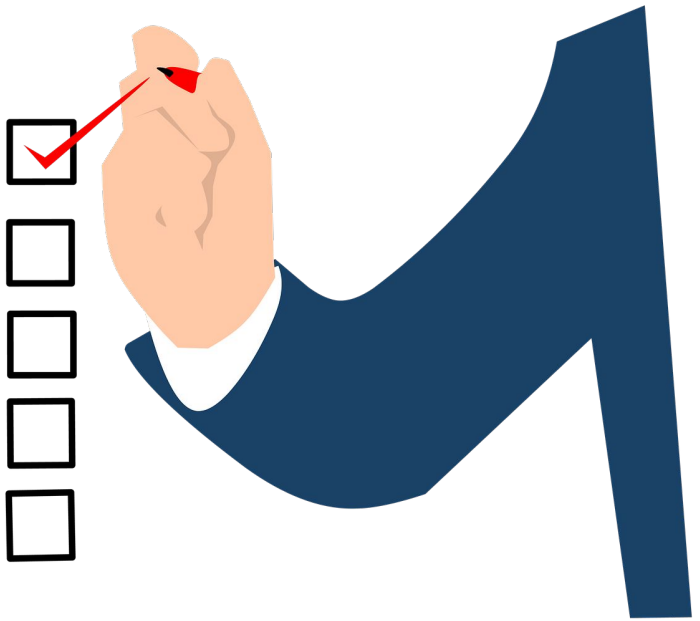
FUNDAÇÃO UNIVERSIDADE FEDERAL DE RONDÔNIA

Algoritmos Avançados

Professor:
Me. Lucas Marques da Cunha
lucas.marques@unir.br

Roteiro

1. **Busca Backtracking**
   a. Adicionando inteligência à busca;
   b. Problema: Jogo do Cadeado!
   c. Heurísticas
   d. Greedy Best-first Search
   e. Busca A*
2. **Desafios**

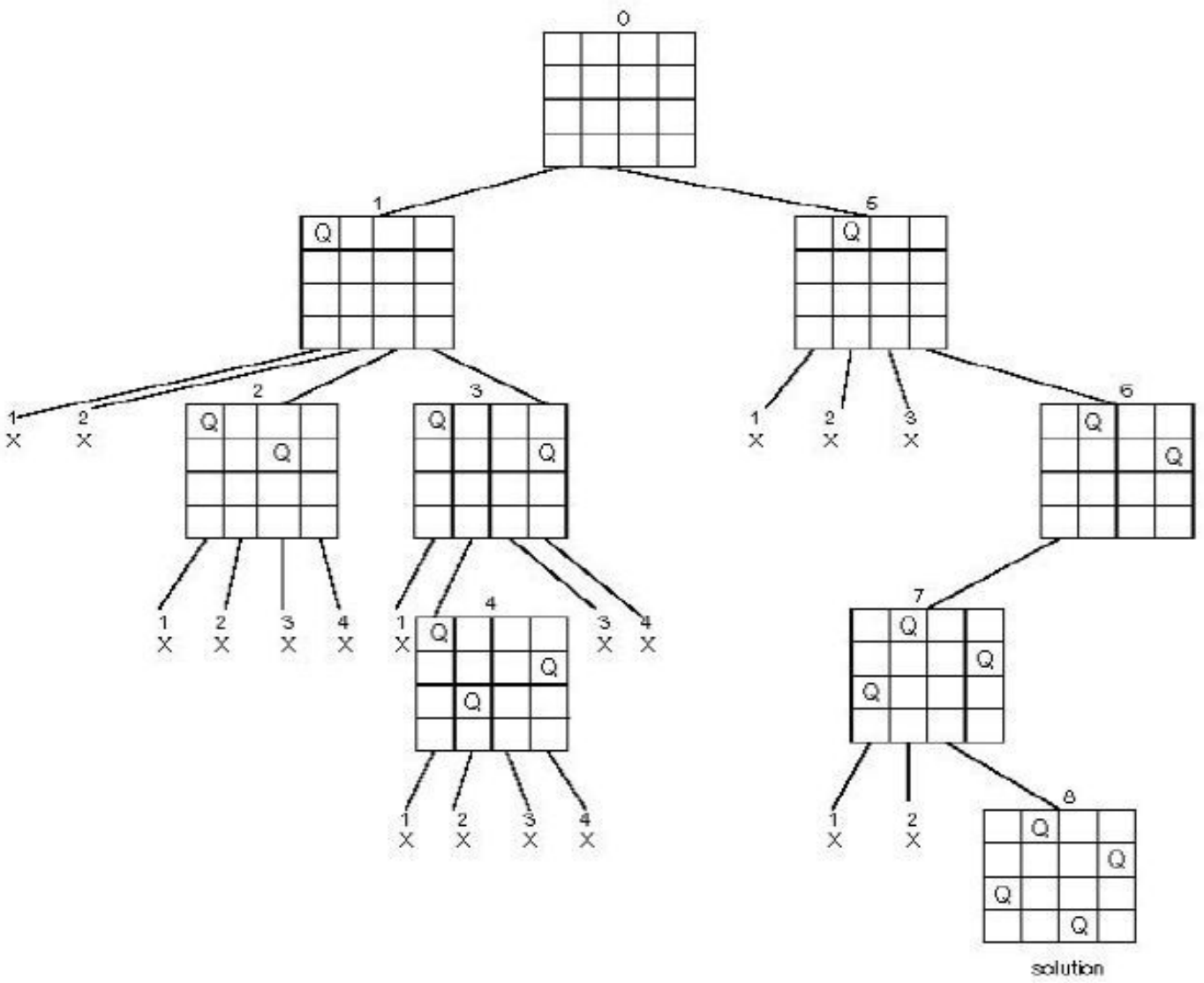


Relembrando : Backtracking

UNIR

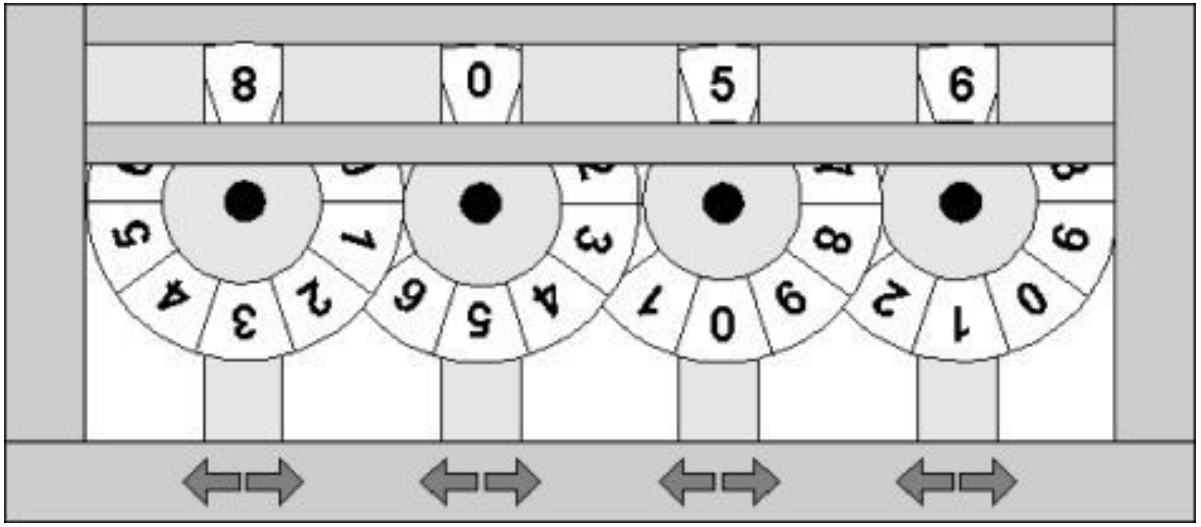
- Técnicas de **busca exaustiva** geram todas as soluções candidatas e então identificam aquela (ou aquelas) com a propriedade desejada;
- A ideia do ***Backtracking***, ao contrário, é construir soluções parciais e avaliá-las da seguinte maneira:
  - Se a solução parcial pode ser continuada, sem violar os objetivos, então, faça-o, incorporando um próximo componente legítimo;
  - Se não há nenhuma opção legítima, nenhuma alternativa restante precisa ser considerada.
  - **Backtracking usa o conceito de árvore de estado-espaço.**

Backtracking: 4 Rainhas!



Problema: Jogo do Cadeado!

- A partir de uma configuração inicial ($S_1S_2S_3S_4$), um conjunto de configurações proibidas ($F_{i1}F_{i2}F_{i3}F_{i4}$) ($1 \leq i \leq n$) e uma final ($T_1T_2T_3T_4$), escreva um programa que calcule a menor quantidade de movimentações nas rodas para sair de S e chegar em T.



Problema: Jogo do Cadeado!

- Qual é o grafo por trás deste problema?
  - Vértices: ?
  - Arestas: ?
- Qual é o grau de cada vértice deste grafo?
- Como é possível encontrar o caminho mínimo?

Problema: Jogo do Cadeado!

- É necessário representar o grafo explicitamente?
- Como saber por quais estados já passou?

Problema: Jogo do Cadeado!

UNIR

- **É necessário representar o grafo explicitamente?**
  - Não. Podemos construir uma função que retorna os próximos estados dado o estado atual;
  - Backtracking com **fringe (lista aberta)** organizado em uma fila.
- **Como saber por quais estados já passou?**
  - Os estados já visitados podem ser marcados em uma matriz.

Problema: Jogo do Cadeado!

UNIR

- **Qual é o grafo por trás deste problema?**
  - Vértices: estados (números de 4 dígitos);
  - Arestas: possíveis transições entre estados.
- **Qual é o grau de cada vértice deste grafo?**
  - Exatamente oito, pois existem oito possíveis transições a partir de cada estado.
- **Como é possível encontrar o caminho mínimo?**
  - A forma mais simples é utilizar uma busca em largura, pois o grafo é não-ponderado.

Busca X BackTracking

UNIR

- Mesmo conceito:
  - **Busca em grafos:** grafo explícito;
  - **Backtracking:** grafo implícito;
- Ou seja:
  - **Busca em grafos:** pode-se consultar o grafo para se saber os vértices adjacentes;
  - **Backtracking:** deve-se ter uma sub-rotina que gera os próximos “vértices” e verifica se são válidos.

Problema: Jogo do Cadeado!

UNIR

```
1  #include <stdio.h>
2  #include <queue>
3  #include <cstring>
4
5  using namespace std;
6
7  struct state{
8      int digit[4];
9      int depth;
10 };
11
12 int moves [8][4]= { {-1, 0, 0, 0 },
13                     { 1, 0, 0, 0 },
14                     { 0,-1, 0, 0 },
15                     { 0, 1, 0, 0 },
16                     { 0, 0,-1, 0 },
17                     { 0, 0, 1, 0 },
18                     { 0, 0, 0,-1 },
19                     { 0, 0, 0, 1 } };
20
```

Problema: Jogando com Rodas!

UNIR

```
22  void next_states(state s, state next[8]){
23      int i,j;
24
25      for(i=0; i<8; i++){
26          next[i] = s;
27          next[i].depth = s.depth+1;
28          for(j=0; j<4; j++){
29              next[i].digit[j] += moves[i][j];
30              if (next[i].digit[j]<0)
31                  next[i].digit[j] = 9;
32              if (next[i].digit[j]>9)
33                  next[i].digit[j] = 0;
34          }
35      }
36
37  }
```

Problema: Jogando com Rodas!

UNIR

```
78  int main (){
79      int nr_testes, test, forbidden, i;
80      int visited[10][10][10][10];
81      state initial, final, aux;
82
83      scanf("%d", &nr_testes);
84      for (test=0; test<nr_testes; test++){
85          scanf("%d %d %d %d", &initial.digit[0], &initial.digit[1], &initial.digit[2], &initial.digit[3]);
86          scanf("%d %d %d %d", &final.digit[0], &final.digit[1], &final.digit[2], &final.digit[3]);
87          scanf("%d", &forbidden);
88
89          memset(visited, 0, sizeof visited);
90
91          for(i=0; i<forbidden; i++) {
92              scanf("%d %d %d %d", &aux.digit[0], &aux.digit[1], &aux.digit[2], &aux.digit[3]);
93              visited[aux.digit[0]][aux.digit[1]][aux.digit[2]][aux.digit[3]] = 1;
94          }
95          initial.depth=0;
96          printf("%d\n", bfs(initial, final, visited));
97      }
98      return 0;
99  }
```

Problema: Jogando com Rodas!

UNIR

```
48
49  int bfs(state current, state final, int visited[10][10][10][10]){
50      state next[8];
51      int i;
52      queue<state> q;
53      // o estado inicial pode ser invalido !!!!
54      if (!visited[current.digit[0]][current.digit[1]][current.digit[2]][current.digit[3]]){
55          visited[current.digit[0]][current.digit[1]][current.digit[2]][current.digit[3]] = 1;
56          q.push(current);
57          while(!q.empty()) {
58              current = q.front();
59              q.pop();
60              // chegou no estado final.. basta terminar (este é o nivel minino!!!) e mostrar o nivel que saiu da árvore...!
61              if (equal(current, final))
62                  return current.depth;
63              // nao é igual.. entao, calcula todos os proximos estados e continua a busca em largura.....
64              next_states(current, next);
65              for (i=0; i<8; i++)
66                  if (!visited[next[i].digit[0]][next[i].digit[1]][next[i].digit[2]][next[i].digit[3]]){
67                      visited[next[i].digit[0]][next[i].digit[1]][next[i].digit[2]][next[i].digit[3]] = 1;
68                      q.push(next[i]);
69                  }
70          }
71      }
72      return -1;
73  }
```

Problema: Jogando com Rodas!

```
48
49  int bfs(state current, state final, int visited[10][10][10][10]){
50      state next[8];
51      int i;
52      queue<state> q;
53      // o estado inicial pode ser invalido !!!!
54      if (!visited[current.digit[0]][current.digit[1]][current.digit[2]][current.digit[3]]){
55          visited[current.digit[0]][current.digit[1]][current.digit[2]][current.digit[3]] = 1;
56          q.push(current);
57          while(!q.empty()) {
58              current = q.front();
59              q.pop();
60              // chegou no estado final.. basta terminar (este é o nivel minino!!!) e mostrar o nivel que saiu da árvore...!
61              if (equal(current, final))
62                  return current.depth;
63              // nao é igual.. entao, calcula todos os proximos estados e continua a busca em largura.....
64              next_states(current, next);
65              for (i=0; i<8; i++)
66                  if (!visited[next[i].digit[0]][next[i].digit[1]][next[i].digit[2]][next[i].digit[3]]){
67                      visited[next[i].digit[0]][next[i].digit[1]][next[i].digit[2]][next[i].digit[3]] = 1;
68                      q.push(next[i]);
69                  }
70          }
71      }
72      return -1;
73  }
```

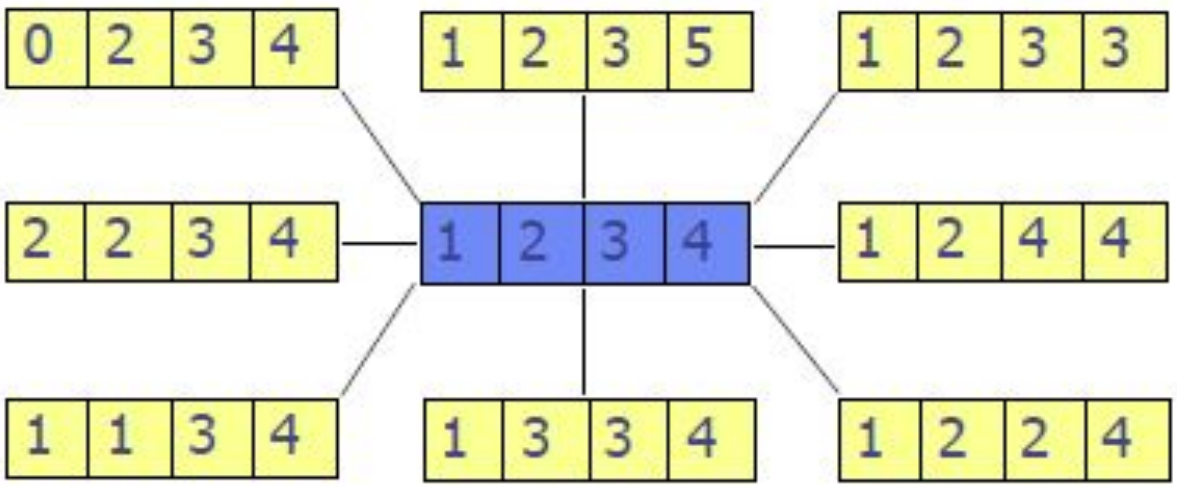
```
39
40  int equal(state s, state e){
41      int i;
42      for (i=0; i<4; i++)
43          if (s.digit[i] != e.digit[i])
44              return 0;
45
46      return 1;
47  }
```

Problema: Jogo do Cadeado!

- Você consegue explicar porque este algoritmo funciona e retorna a "*shortest path*" para o problema?
- Tem como tornar a solução mais inteligente?

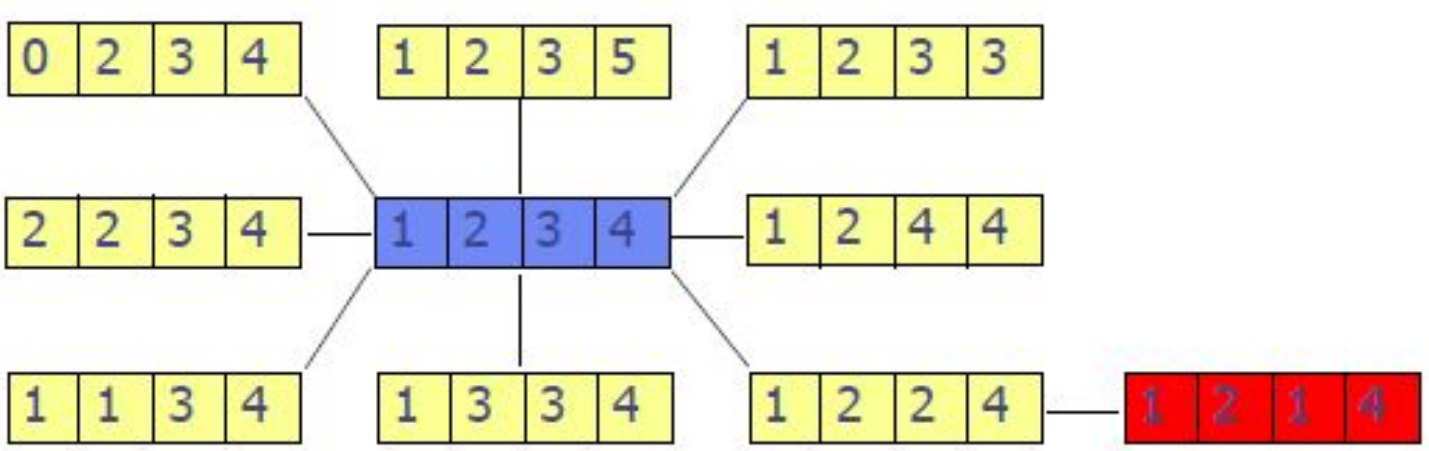
Jogo do Cadeado: Heurística

- Com a busca em largura, dado um certo estado (vértice - [1,2,3,4]), todos os vértices adjacentes não visitados são inseridos na fila q.



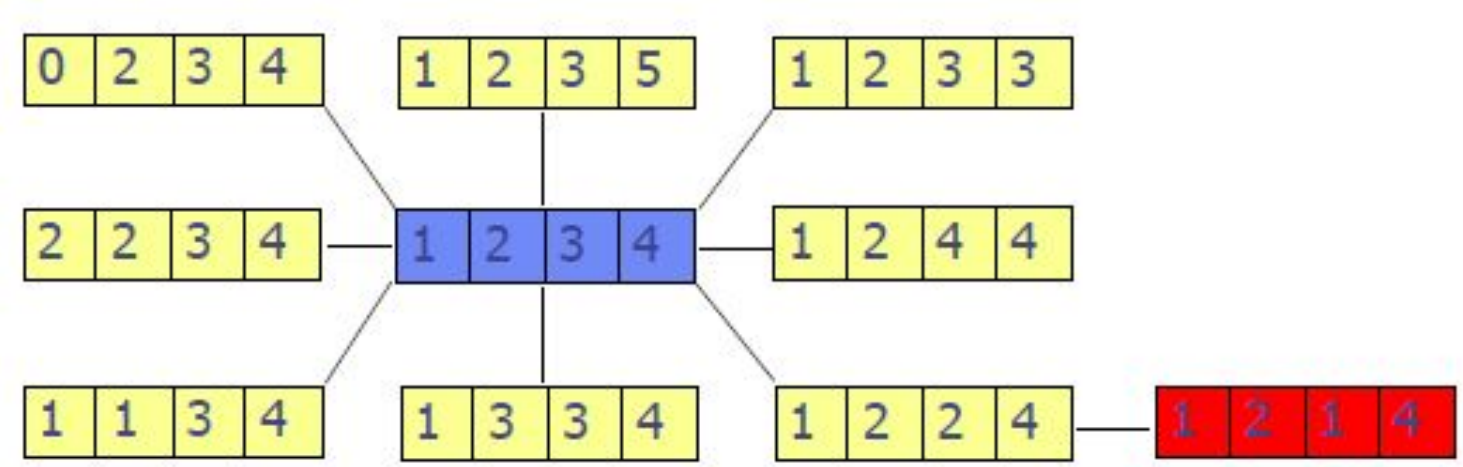
Jogo do Cadeado: Heurística

- Com a busca em largura todos os vértices adjacentes não visitados são inseridos na fila q.
- Sendo [1,2,3,4] o estado inicial, imagine que o estado final é [1,2,1,4]. Então é melhor seguir pelo vértice [1,2,2,4].



Jogo do Cadeado: Heurística

- A função heurística pode ser então a distância entre o estado analisado e o estado final:
  - D([1,2,2,4], [1,2,1,4]) = 1 é a menor dentre todos os 8 possíveis vizinhos.



Jogo do Cadeado: Heurística

- Um detalhe importante neste caso:
  - A existência de estados proibidos faz com que a função heurística seja uma estimativa otimista.
  - O que significa ser “otimista” ?

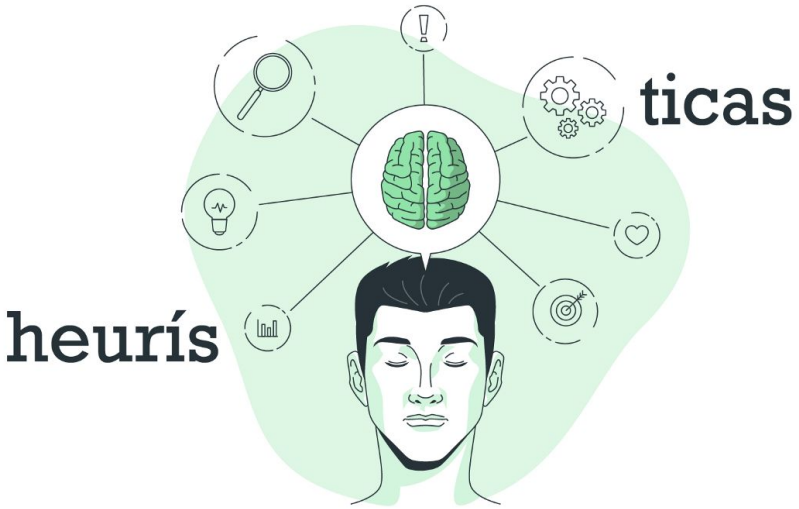
Jogo do Cadeado: Heurística

UNIR

- Um detalhe importante neste caso:
  - A existência de estados proibidos faz com que a função heurística seja uma estimativa otimista.
  - O que significa ser “otimista” ?
    - Nunca superestima o custo real de se chegar ao destino. Seja n um estado atual e h(n) a função heurística -> h(n) é sempre > ou = ao real valor para se chegar até o destino final.

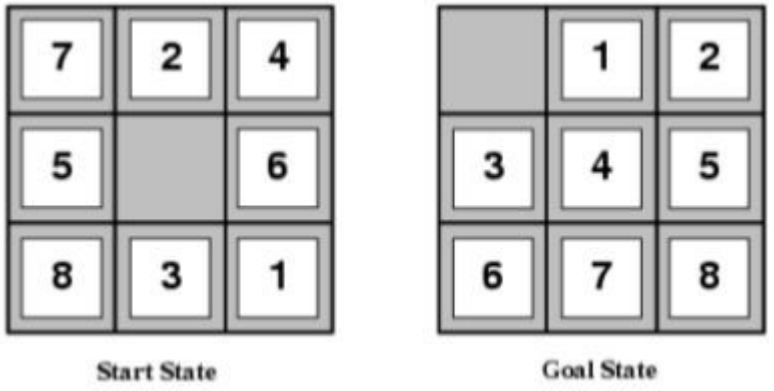
Heurísticas

- Intuição
- Uma maneira rápida de estimar o quão próximo estamos do objetivo.
- “Aquela observação que muita gente faz de maneira simplista, baseada na intuição!

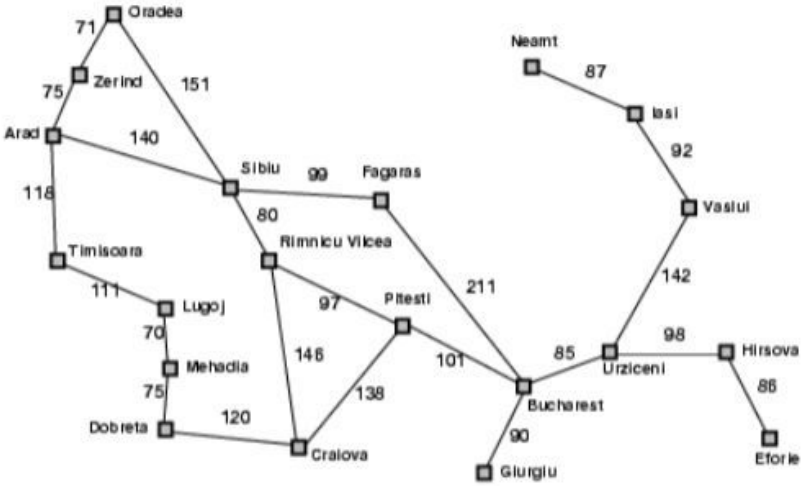


Heurísticas

- Para o quebra-cabeças de oito peças:
  - h(n): número de peças fora do lugar;
  - h(n): distância Manhattan.



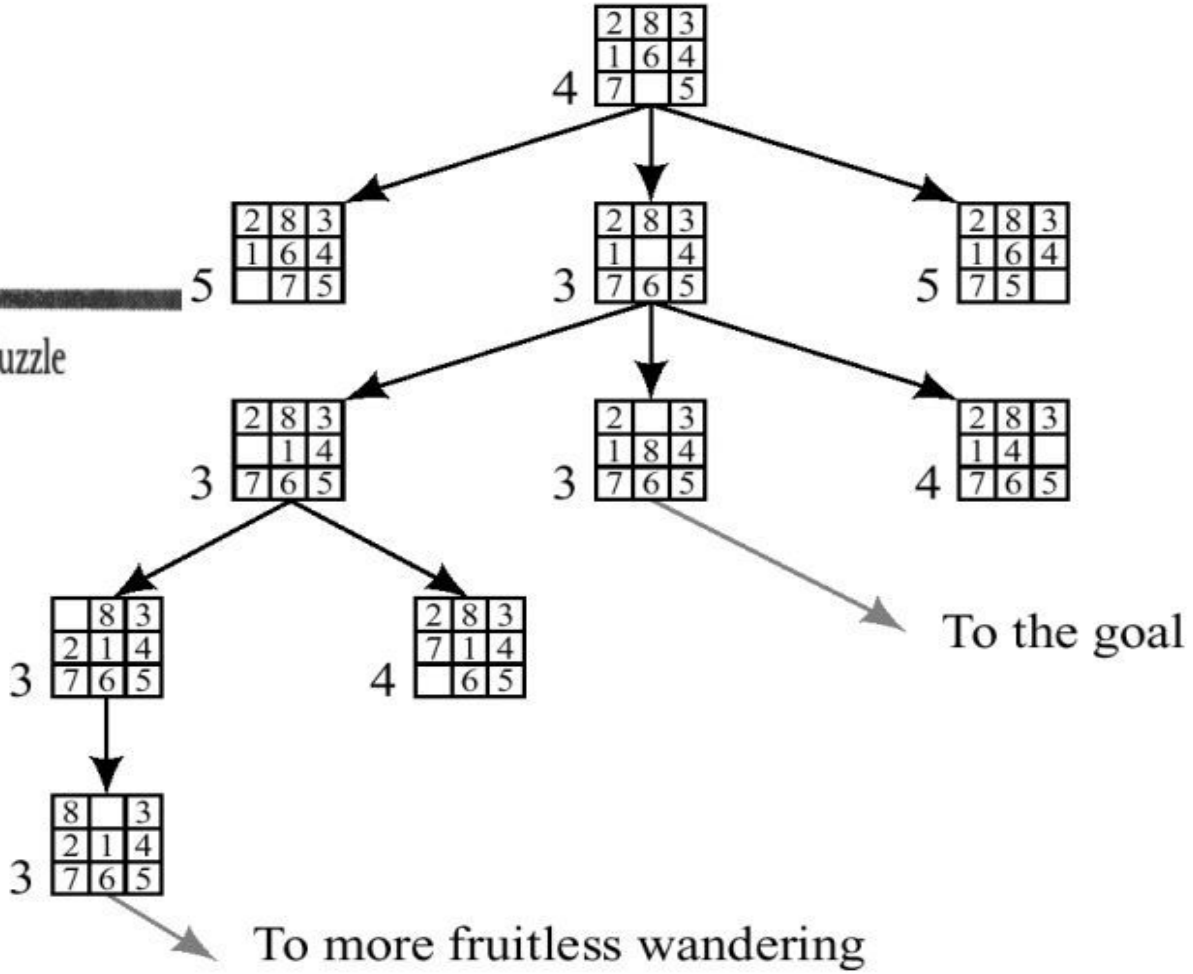
- Para o mapa:
  - A Dist. Euclidiana (linha reta entre cada cidade à Bucareste.



Mas será que só isso resolve?

Figure 8.1

Start and Goal Configurations for the Eight-Puzzle

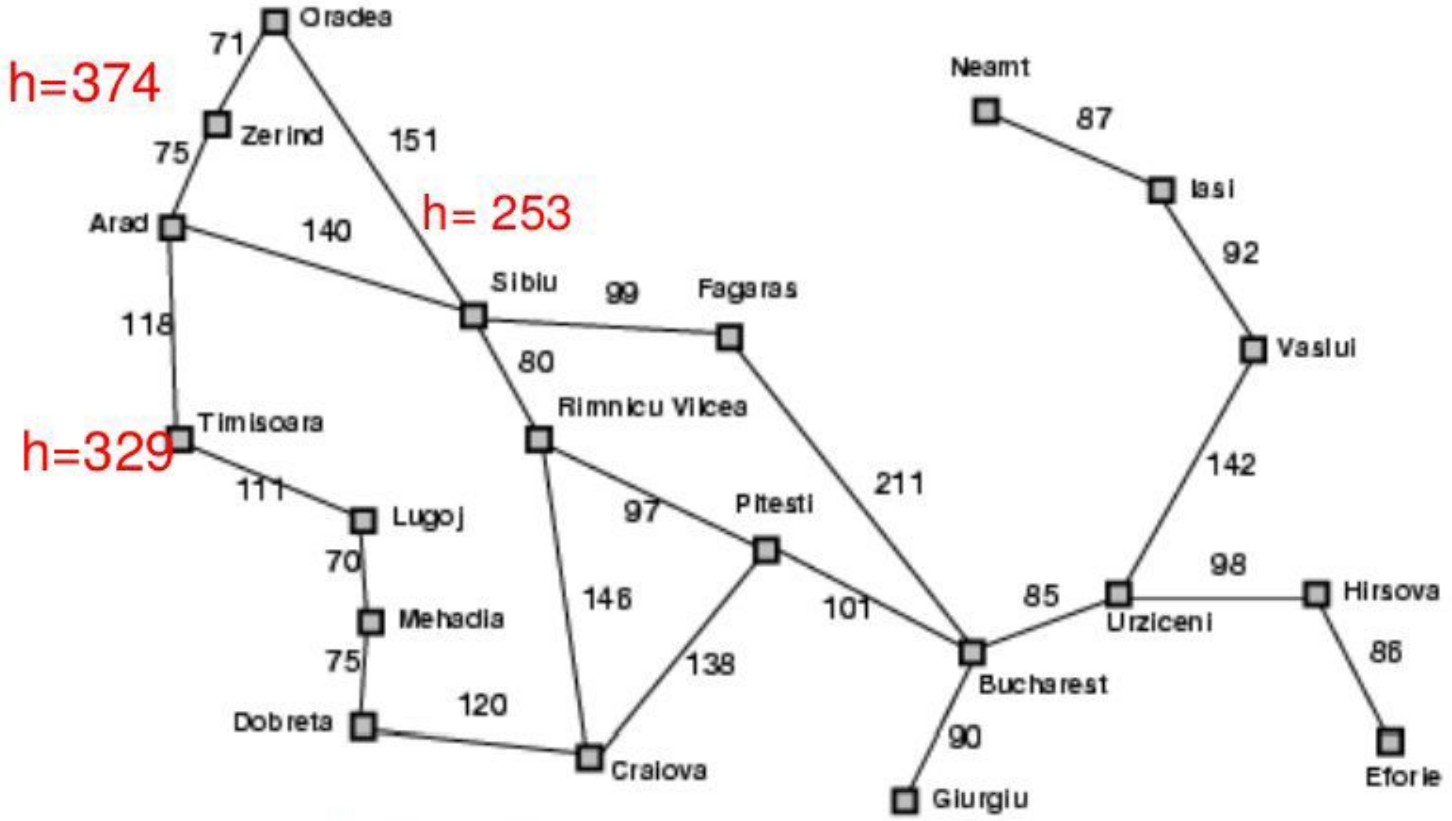


Mas será que só isso resolve?

UNIR

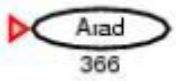
- Vimos que não.
- O que acabamos de fazer é uma técnica conhecida como “Greedy best-first search” (Busca gulosa de melhor escolha)
  - tenta expandir o nó que está mais próximo do objetivo, com o fundamento de que isso pode conduzir a uma solução rapidamente.
- **Função de avaliação f(n) = h(n)**
  - Estimativa do custo de n até o destino;
- No caso do mapa, h(n) = hSLD(n) -> *Straight Line Distance* de n até Bucareste.

Greedy Best-first Search



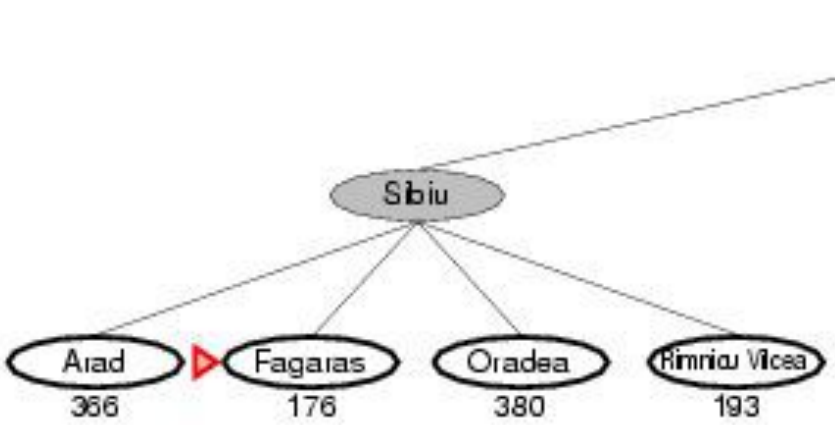
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-first Search



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

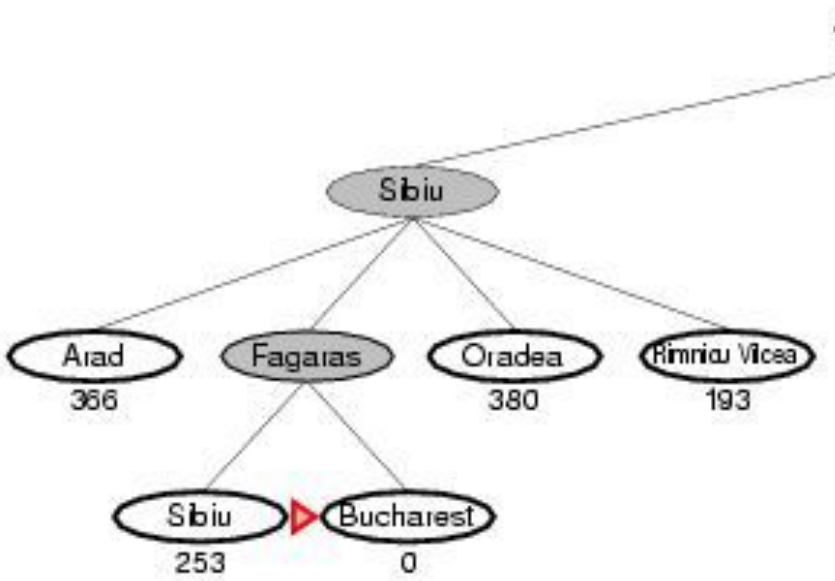
Greedy Best-first Search



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-first Search

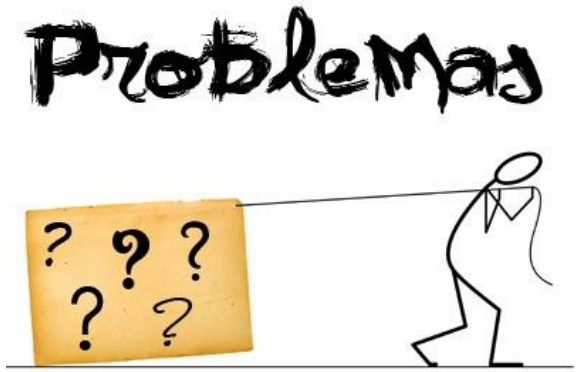


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-first Search

- Busca não completa;
  - Nem sempre acha uma solução, se existe.
- **Loops infinitos**
- Não é ótima;
  - O caminho via Sibiu e Fagaras para Bucareste é 32 quilômetros mais longo que o caminho através de Rimnicu Vilcea e Pitesti.

Isso mostra por que o algoritmo é chamado de "ambicioso"; a cada passo ele tenta chegar o mais próximo do objetivo que puder.



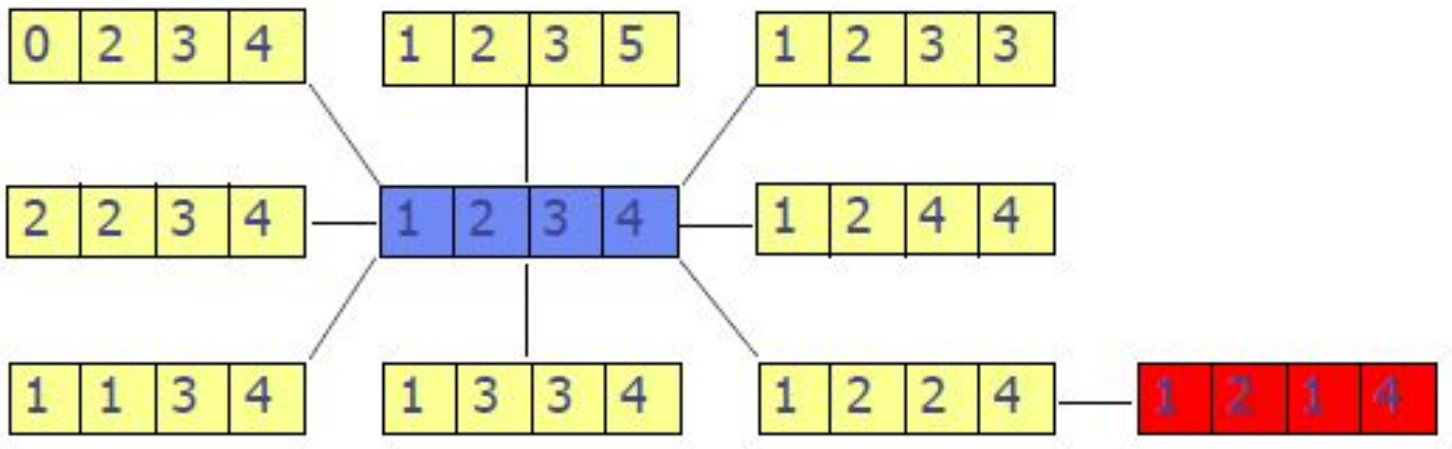
Greedy Best-first Search

UNIR

```
função BUSCA-DE-CUSTO-UNIFORME(problema) retorna uma solução ou falha
  nó ← um nó com ESTADO = problema.ESTADO-INICIAL, CUSTO-DE-CAMINHO = 0
  borda ← fila de prioridade ordenada pelo CUSTO-DE-CAMINHO, com nó como elemento
único
  explorado ← um conjunto vazio
  repita
    se VAZIO?(borda), então retornar falha
    nó ← POP(borda) / * escolhe o nó de menor custo na borda */
    se problema.TESTE-OBJETIVO(nó.ESTADO) então retornar SOLUÇÃO(nó)
    adicionar (nó.ESTADO) para explorado
    para cada ação em problema. AÇÕES(nó.ESTADO) faça
      filho ← NÓ-FILHO (problema, nó, ação)
      se (filho.ESTADO) não está na borda ou explorado então
        borda ← INSIRA (filho, borda)
      senão se (filho.ESTADO) está na borda com o maior CUSTO-DE-CAMINHO então
        substituir aquele nó borda por filho
```

Jogando com Rodas: O retorno

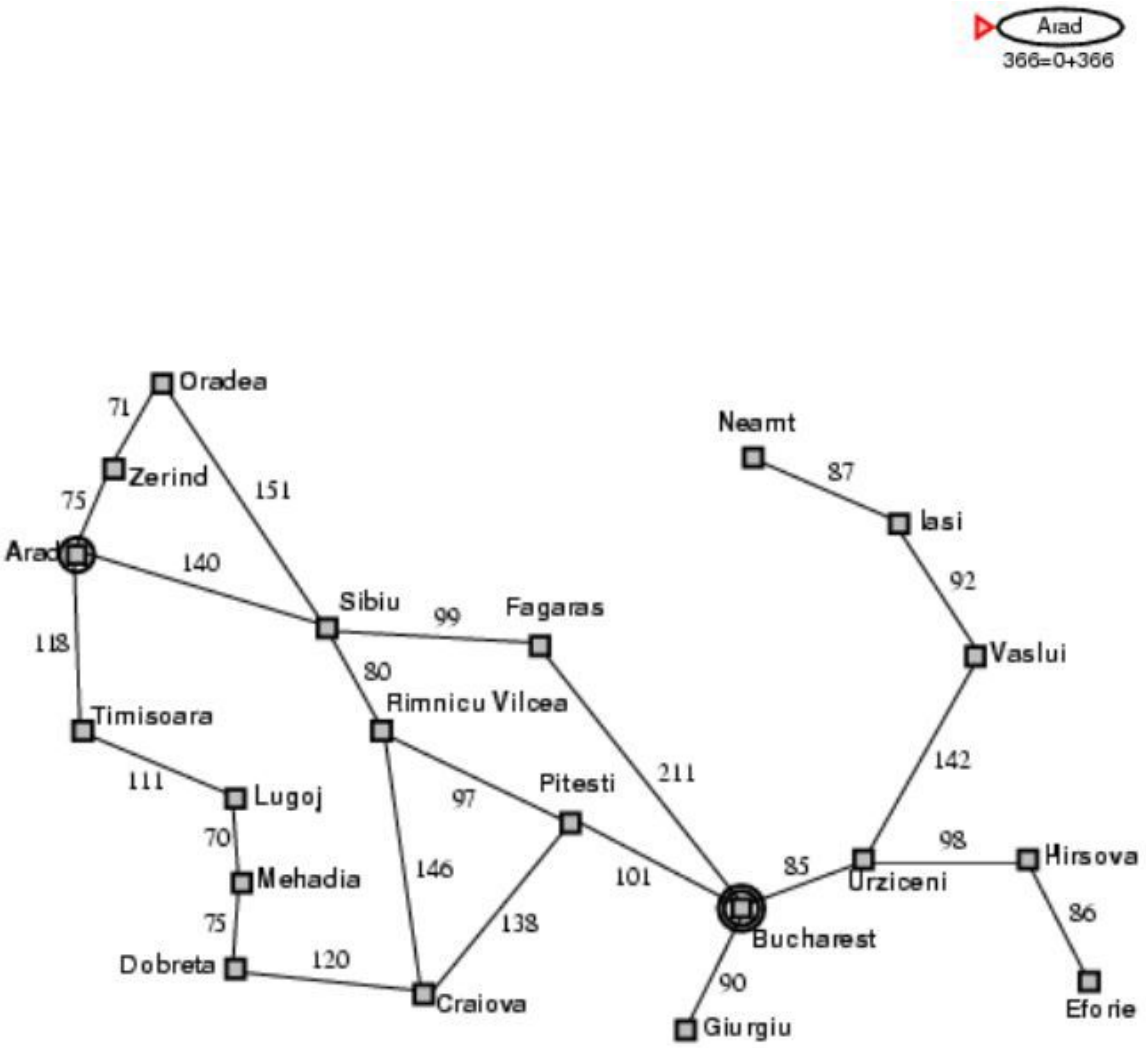
- A função heurística pode ser incorporada alterando-se a fila da busca em largura por uma fila de prioridade (ordenada pela função heurística).
  - pq = (1:[1,2,2,4], 3:[1,2,4,4], 3:[1,2,3,3], 3:[1,2,3,5], 3:[0,2,3,4], 3:[2,2,3,4], 3:[1,1,3,4], 3:[1,3,3,4])



Busca A*

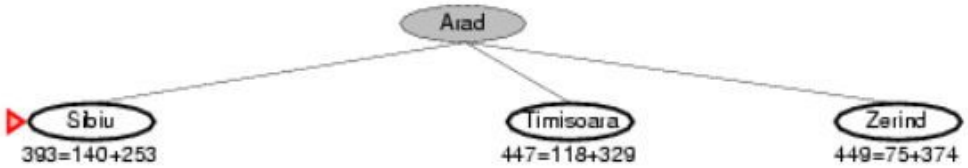
- Ideia
  - evitar expandir trilhas de custo alto;
  - dar prioridade àquelas que são promissoras;
- Função de avaliação f(n) = g(n) + h(n);
- g(n) = custo da origem até n;
- h(n) = custo estimado de n até o destino;

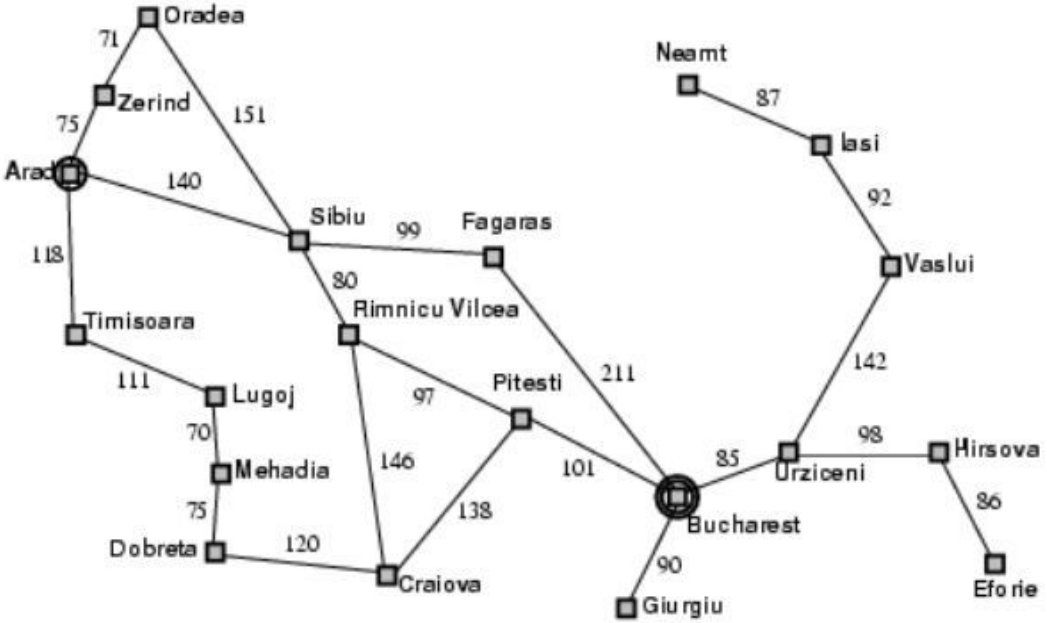
Busca A*



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

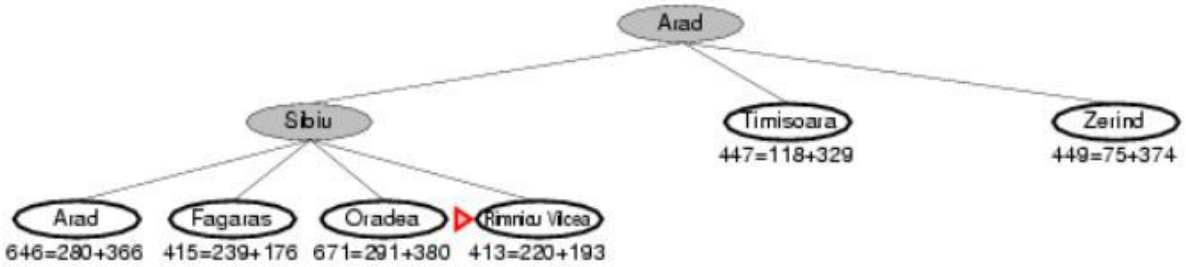
Busca A*

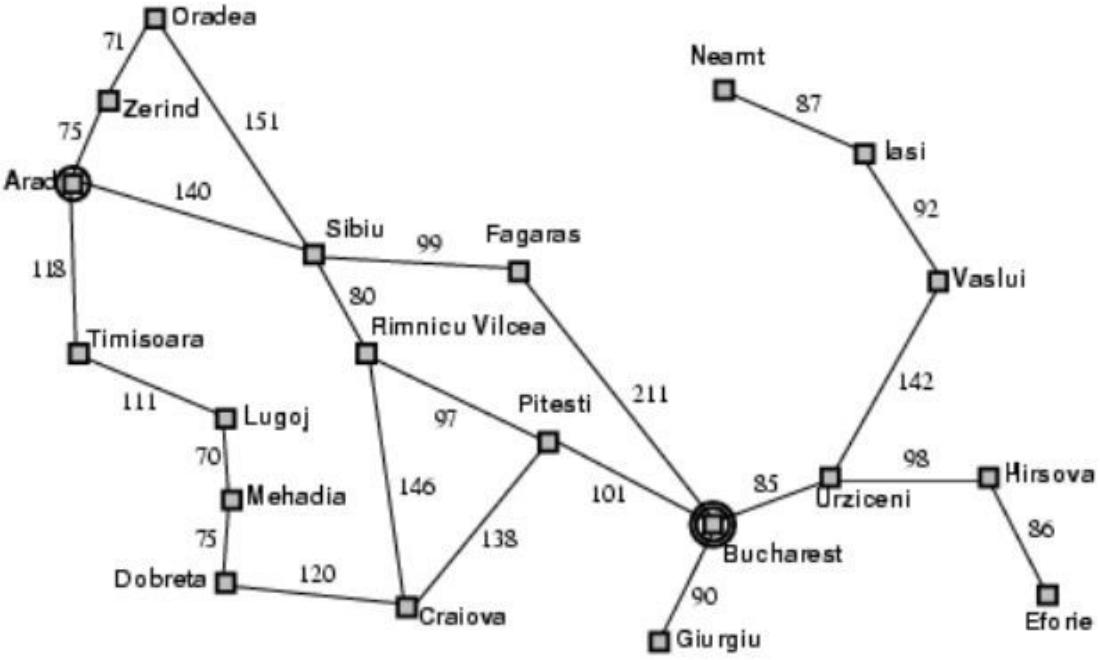




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

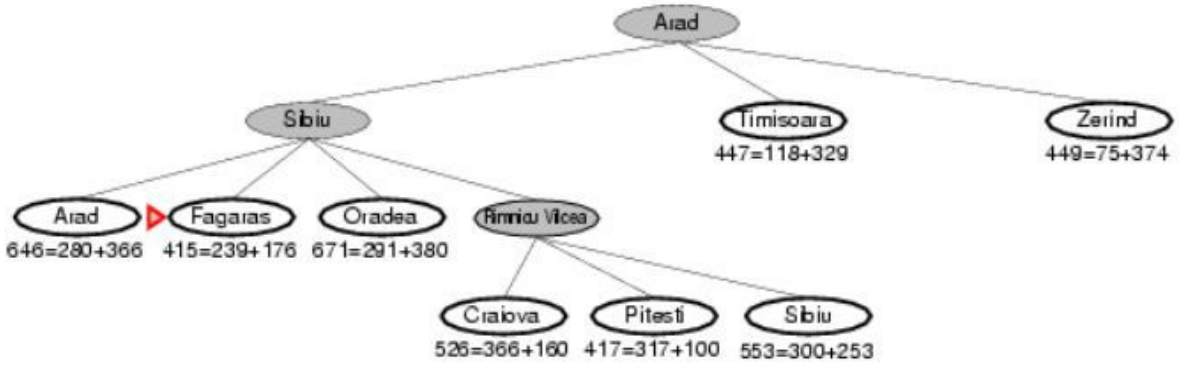
Busca A*

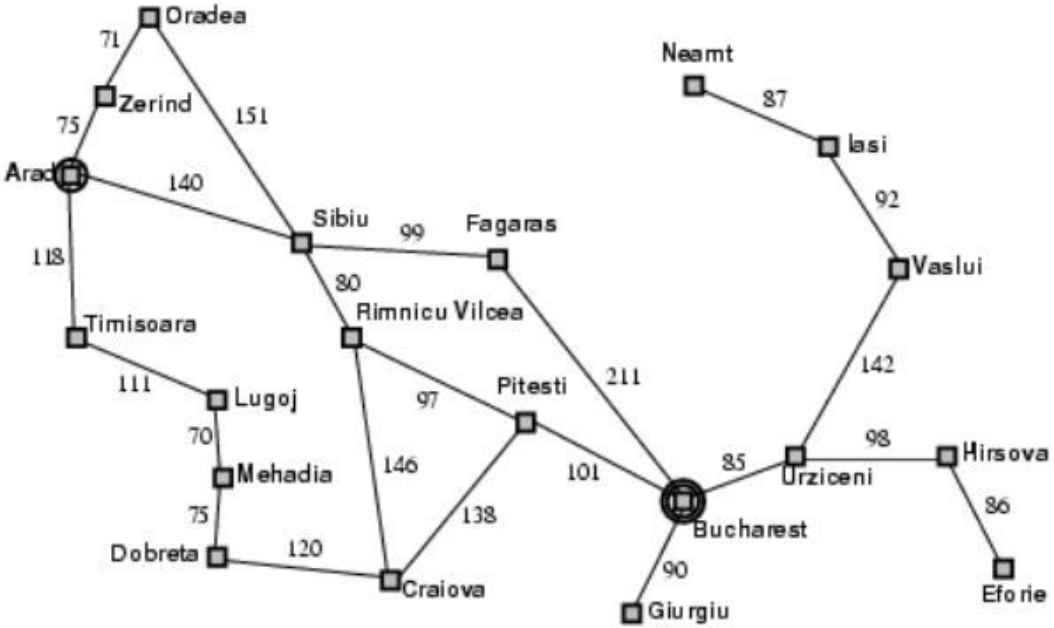




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

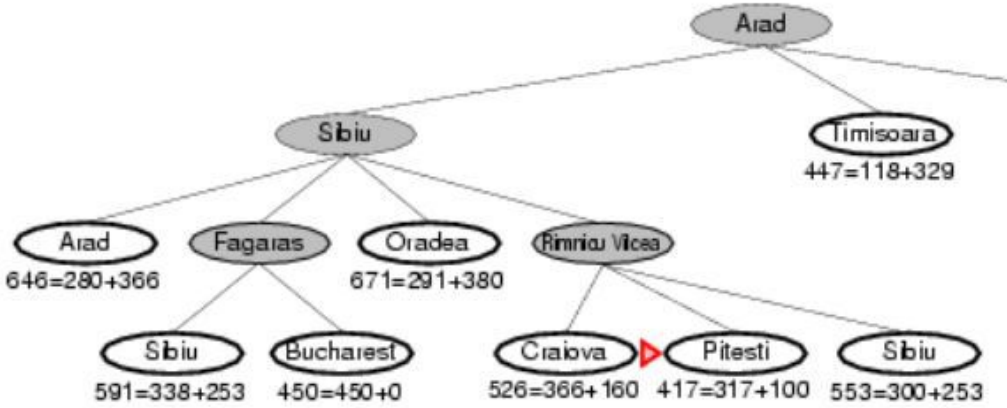
Busca A*

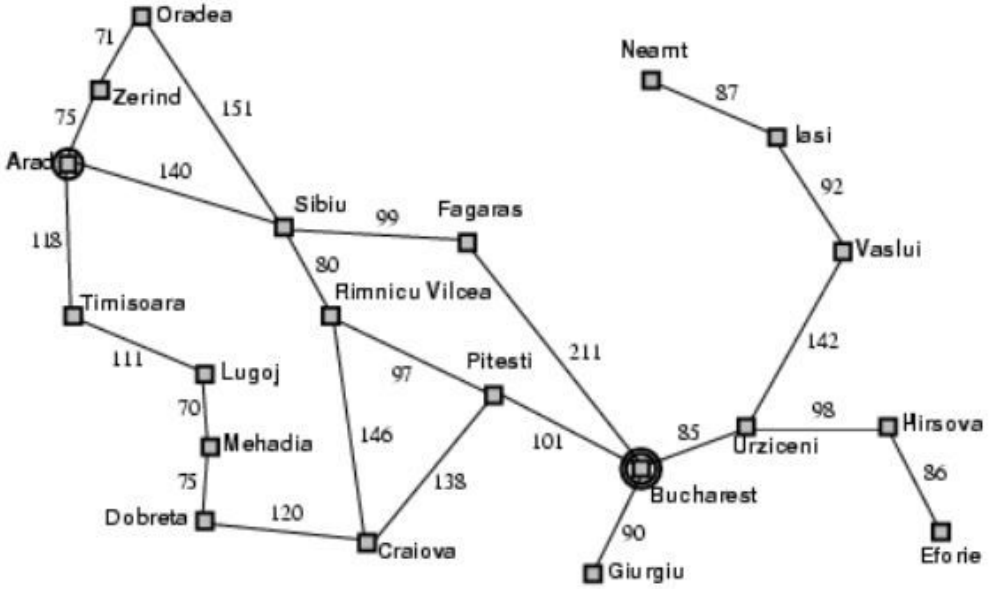




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca A*

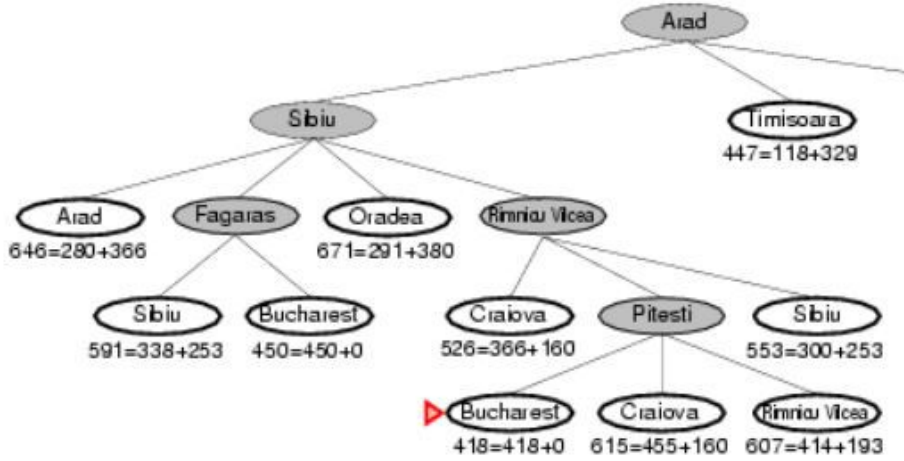


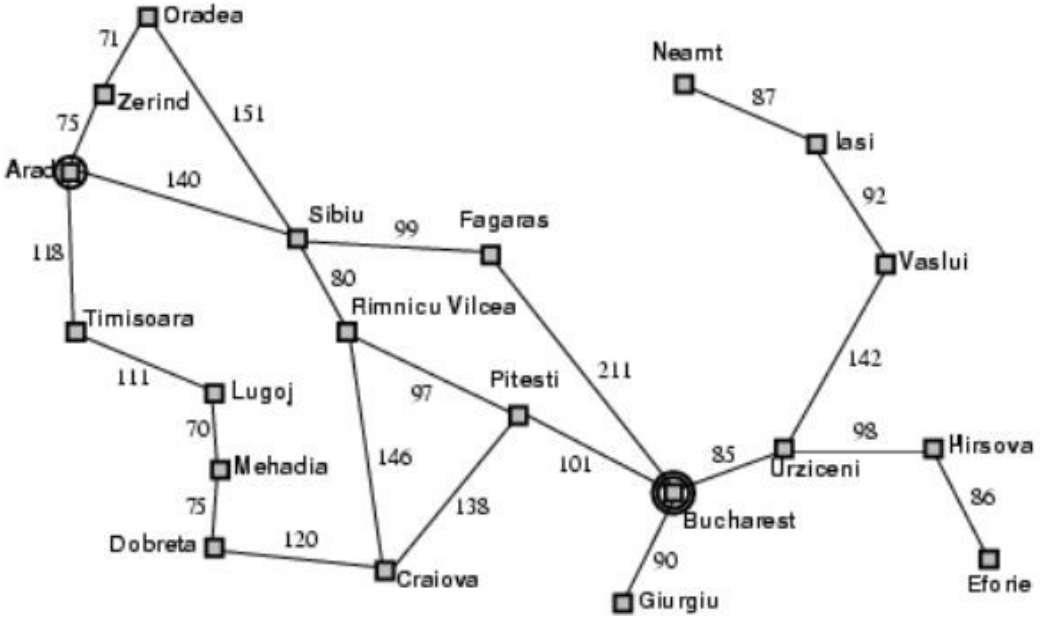


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

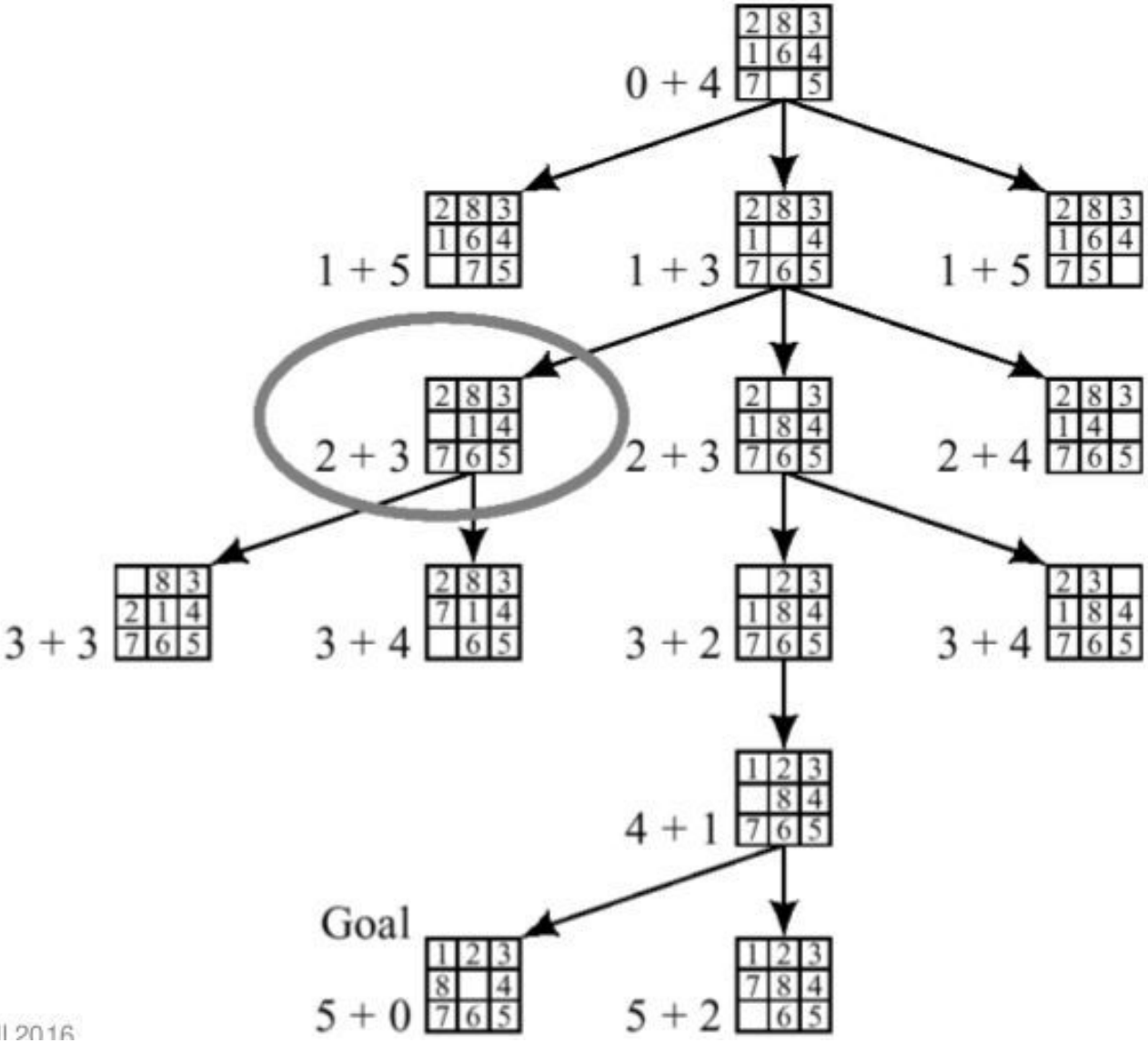
Busca A*





Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca A* para o Jogo



Formalizando A*

UNIR

- Notação:
  - c(n, n') = custo real da aresta (n,n')
  - g(n) = custo do caminho atual do início até n
  - h(n) = estimativa do menor custo de uma trilha de n até o destino
  - função de avaliação: f(n) = g(n) + h(n)
  - h*(n) é o menor custo real de n até o destino
- A heurística h(n) é **admissível** se, para todo nó n $h(n) \leq h^*(n)$
- Uma heurística admissível jamais superestima a distância real, ou seja, é **otimista**.

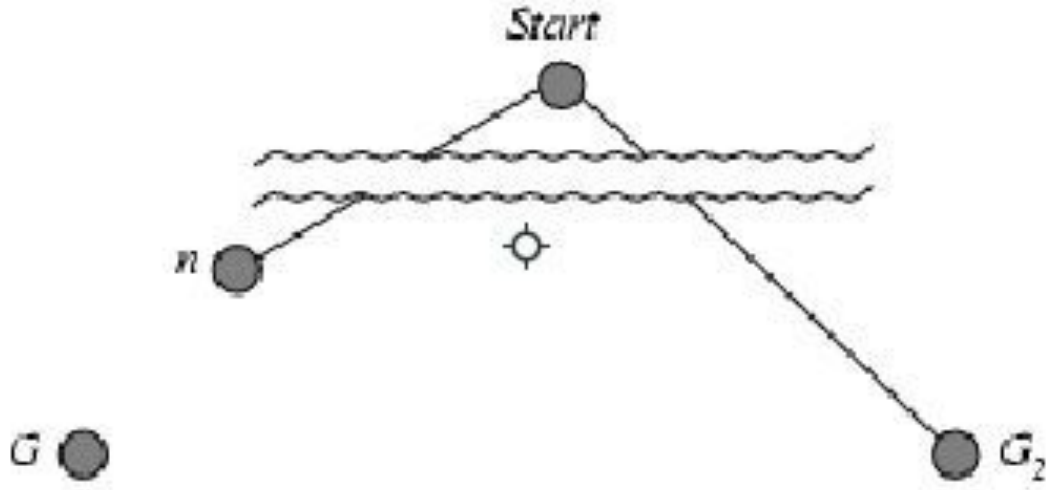
Algoritmo A*

UNIR

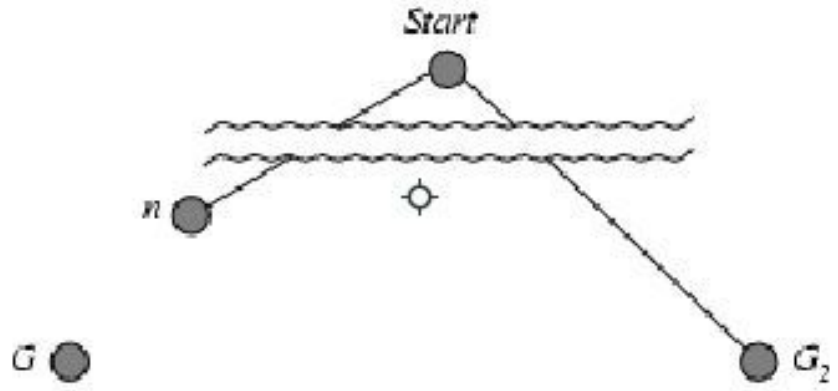
- Input: an implicit search graph problem with cost on the arcs
- Output: the minimal cost path from start node to a goal node.
  - 1. Put the start node s on OPEN.
  - 2. If OPEN is empty, exit with failure
  - 3. Remove from OPEN and place on CLOSED a node n having minimum f.
  - 4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s.
  - 5. Otherwise, expand n generating its children and directing pointers from each child node to n.
    - For every child node n' do
      - evaluate h(n') and compute $f(n') = g(n') + h(n') = g(n) + c(n,n') + h(n')$
      - If n' is already on OPEN or CLOSED compare its new f with the old f. If the new value is higher, discard the node. Otherwise, replace old f with new f and reopen the node.
      - Else, put n' with its f value in the right order in OPEN
  - 6. Go to step 2.

Prova da otimalidade de A*

- Suponha que um destino subótimo G_2 foi gerado e está na fringe (fila prioridade, nó aberto). Seja n um nó não expandido na fringe tal que n está na trilha mais curta de um nó G.



Prova da otimalidade de A*



- $f(G_2) = g(G_2)$ ->> porque $h(G_2) = 0$
- $g(G_2) > g(G)$ ->> lembre-se que G_2 é subótimo
- $f(G) = g(G)$ ->> porque $g(G) = 0$
- $f(G_2) > f(G)$ ->> a partir do exposto acima
- $h(n) \leq h^*(n)$ ->> h é admissível
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$
- Portanto, $f(G_2) > f(n)$ e A* jamais selecionará G_2 para expansão

Busca A*

UNIR

- A* possui as seguintes características:
  - **Completa:** garantia de encontrar uma solução quando existe;
  - **Ótima:** encontra a melhor solução entre várias soluções não ótimas;
  - **Eficiência ótima:** nenhum outro algoritmo da mesma família expande um número menor de nós que o A*.
- Por outro lado, A* pode consumir muita memória.
  - Uma possível solução: IDA* (Russel e Norvig, 2002).

A* : Jogo do Cadeado

UNIR

- Para modificar a solução atual para A* é necessário:
  - Definir uma função heurística e uma função custo;
  - Transformar a fila em uma fila de prioridades, na qual os estados são ordenados pelos valores da função heurística + função de custo;
  - Utilizar como próximo estado a ser processado o estado de menor combinação heurística + custo, fornecido pela fila de prioridades.

Dúvidas, sugestões?