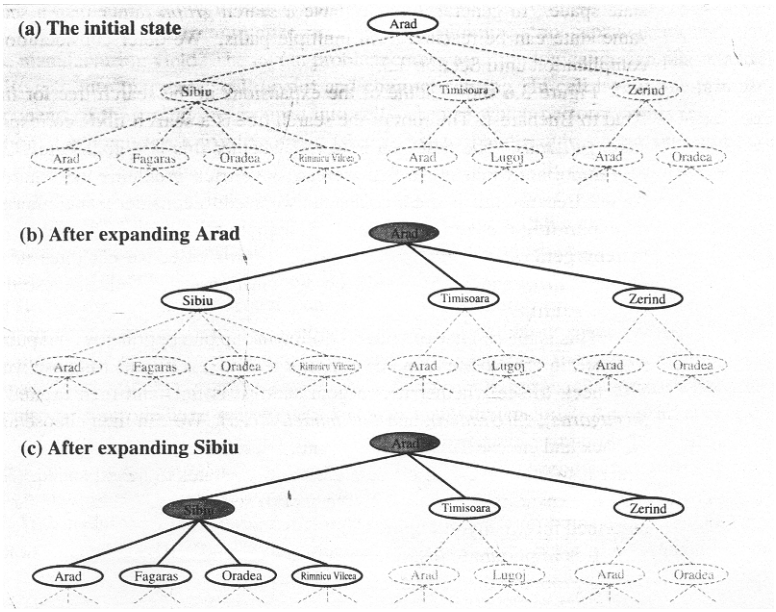
3.3. SOLUÇÕES POR BUSCA:

Tendo visto como formular um problema, é preciso definir como resolvê-lo. Isso é feito por meio de um processo de busca em um espaço de estados[2].

A figura a seguir ilustra algumas expansões na árvore de busca para encontrar a rota de *Arad* para *Bucharest*.



A raiz da árvore é o nodo de busca (*search node*) corresponde ao estado inicial, *In(Arad)*. O primeiro passo é testar se esse é o objetivo. Como não é o objetivo , é preciso considerar outros estados. Isso é feito expandindo o estado corrente, ou seja, aplicando a função sucessor no estado corrente, assim gerará um novo conjunto de estados. Neste caso, três novos estados são criados: *In(Sibiu), In(Timisoara)* e *In(Zerind).* Agora, deve-se escolher qual das três possibilidades deve ser considerada.

Essa é a essência do algoritmo de busca: fazer expansão sucessiva de nodos da árvore de busca, onde expansão de um estado é a geração de seus sucessores, enquanto que estratégia de busca consiste em determinar a ordem das expansões, ou seja, qual estado será expandido.

O algoritmo a seguir mostra a idéia básica de um processo de busca:

```
function TREE_SEARCH(problem, strategy) returns a solution, or failure
        initialize the search tree using the initial state of problem
        loop do
                if there are no candidates for expansion then
```

[2] Espaço de estados(finito) é diferente de árvore de busca(infinita).

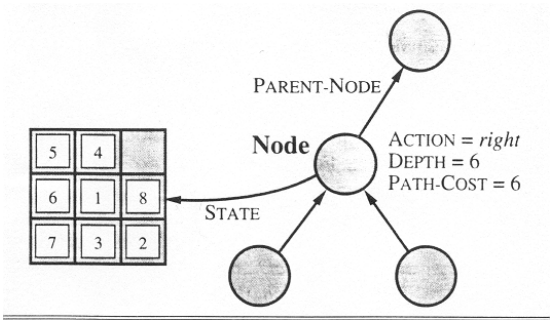
```
            return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then
            return the corresponding solution
        else expand the node and
            add the resulting nodes to the search tree
end
```

ÁRVORE DE BUSCA:

A cada passo, o algoritmo de busca deve escolher um nodo folha para expandir. Os nodos de fronteiras (*fringe*) são os nodos que não têm sucessores (ou por que não expandiu ainda ou já expandiu mas gerou um conjunto vazio de estados), e podem ser expandidos em um determinado momento. Estes nós podem estar em uma fila ou pilha, dependendo da estratégia de busca utilizada.

Existem diversas maneiras de representar um nodo de uma árvore de busca, mas assumiremos que o nodo[3] é uma estrutura de dados com cinco componentes:

- STATE: descrição do estado corrente;
- PARENT_NODE: referências ao nodo na árvore de busca que gerou o estado corrente;
- ACTION: ação que aplicada ao nodo pai gerou o estado corrente;
- PATH_COST: custo do caminho do estado inicial ao nodo corrente, denotado por *g(n);*
- DEPTH: número de passos do estado inicial ao estado corrente.



De modo genérico podemos dizer que o nó está em uma fila e são inseridos de diferentes maneiras pelos algoritmos de busca:

```
function GENERAL_SEARCH (problem, QUEUING-FN) returns a solution, or failure
    Nodes ← MAKE-QUEUE(Make-Node(Initial-State[problem]))
    loop do
```

[3] Diferença entre nodo (estrutura de dados usada para representar a árvore de busca) e estado (corresponde a uma configuração do mundo num dado momento).

```
        if  EMPTY(nodes) then return failure
        node ← REMOVE-FRONT(nodes)
        if Goal-Test[problem] applied to State(node) succeeds then
                return node
        nodes ← QUEUING-FN(nodes, Expand(node, Operators([problem]))
end
```

Onde:

MAKE-QUEUE(*elems*) cria uma fila com os elementos *elems*;
EMPTY (*Queue)* retorna verdadeiro se a fila *Queue* está vazia;
REMOVE-FRONT *(Queue)* remove o elemento na frente da fila *Queue*;
QUEUING-FN *(elems, Queue)* insere elementos *elems* na fila *Queue*.

CRITÉRIOS DE AVALIAÇÃO DAS ESTRATÉGIAS DE BUSCA:

Avaliar a estratégia segundo os seguintes critérios:

- ***Completeness*:** o algoritmo garante encontrar uma solução quando esta existe?
- **Complexidade de tempo:** quanto tempo a busca demora para achar a solução?
- **Complexidade de espaço:** quanta memória é necessária para realizar a busca?
- ***Optimality*:** a busca acha a solução de menor custo quando existem várias soluções diferentes?

Complexidades de tempo e espaço são medidas em termos de:

- *b*: fator de ramificação máximo da árvore de busca, ou seja, o número máximo de sucessores por nó (*branching fator)*;
- *d*: profundidade da solução de menor custo;
- *m*: profundidade máxima (pode ser ∞).

PROCESSOS DE BUSCA:

Busca cega (não informada): não tem informação sobre o número de passos e custo do estado atual para o estado final;

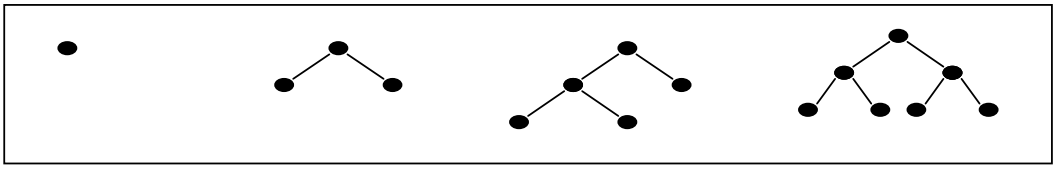
Busca heurística: a estratégia de busca está baseada em informações sobre o domínio, ou seja, tem informações suficientes para perceber que um estado não final é mais promissor que o outro.

ESTRATÉGIAS DE BUSCA NÃO INFORMADAS:

- Busca por amplitude (*breadth-first*);
- Busca por custo uniforme;
- Busca por profundidade (*depth-first*);
- Busca por profundidade limitada (*depth-limited*);
- Busca por aprofundamento iterativo (*iterative deepening depth-first*).

BUSCA POR AMPLITUDE (LARGURA)

Expande todos os nós do primeiro nível, depois do segundo, do terceiro e assim por diante. Pode ser implementado utilizando uma fila FIFO ⇒ os nodos visitados primeiro devem ser explorados primeiro ⇒ nodos mais rasos são explorados primeiro que os mais profundos.



Onde:

- O número máximo de nós em cada nível: b^d sendo que *d* é o nível e b é o número de filhos por nó (em uma árvore binária b=2).
- O número total de nós é dado pela soma: $1+b+b^2+b^3+...+b^d$

Propriedades:

- **Completo:** sim (se *b* é finito);
- **Tempo:** $O(b^d) \Rightarrow$ exponencial
- **Espaço:** $O(b^d) \Rightarrow$ nodos do último nível na memória
- **Ótimo:** sim, se custo fixo por expansão.

Notem que existem situações onde a solução mais rasa não é necessariamente a melhor.

Pontos críticos da busca em largura são os requisitos de memória elevados e o tempo de execução ⇒ a tabela a seguir[4] mostra os recursos necessários de tempo e memória para a busca em largura para b = 10, considerando 10.000nodos/segundo e 1.000 bytes/nodo:

[4] A tabela completa pode ser encontrada na figura 3.11 da página 74 do livro *"Artificial Intelligence: A Modern Approach"* de Stuart Russell e Peter Norvig, 2ª. edição.