procura não informada

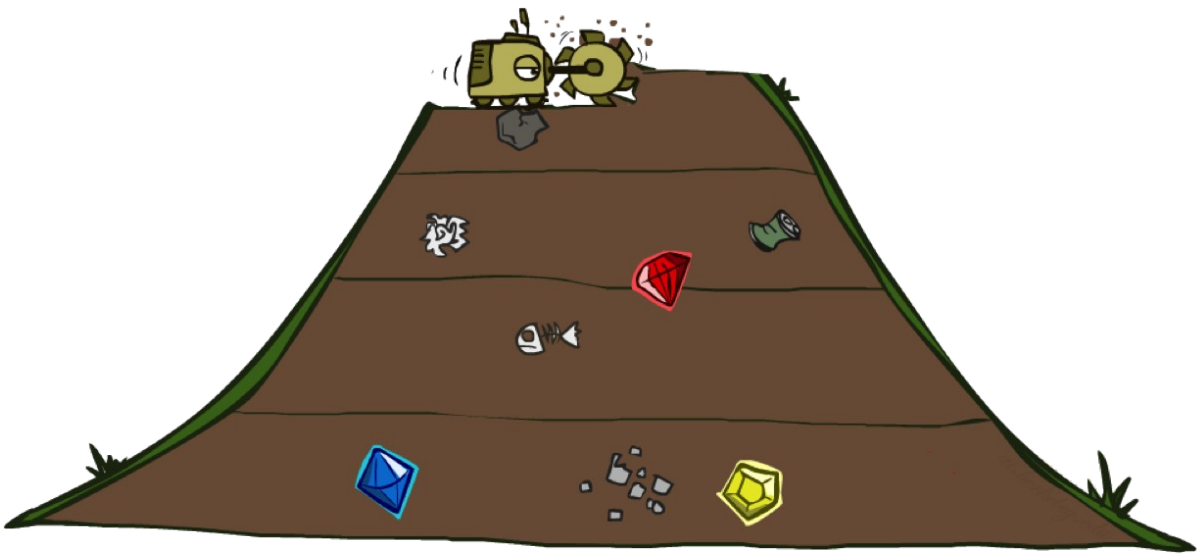
ou procura cega

- não tem informação sobre o que falta até ao objetivo
- apenas pode usar os custos até aos nós da fronteira

  (os mais avançados na expansão)
- útil em casos de informação mínima

  ou nenhuma… para além da definição do problema

procura em largura



fonte: CS188 Berkeley

procura em largura

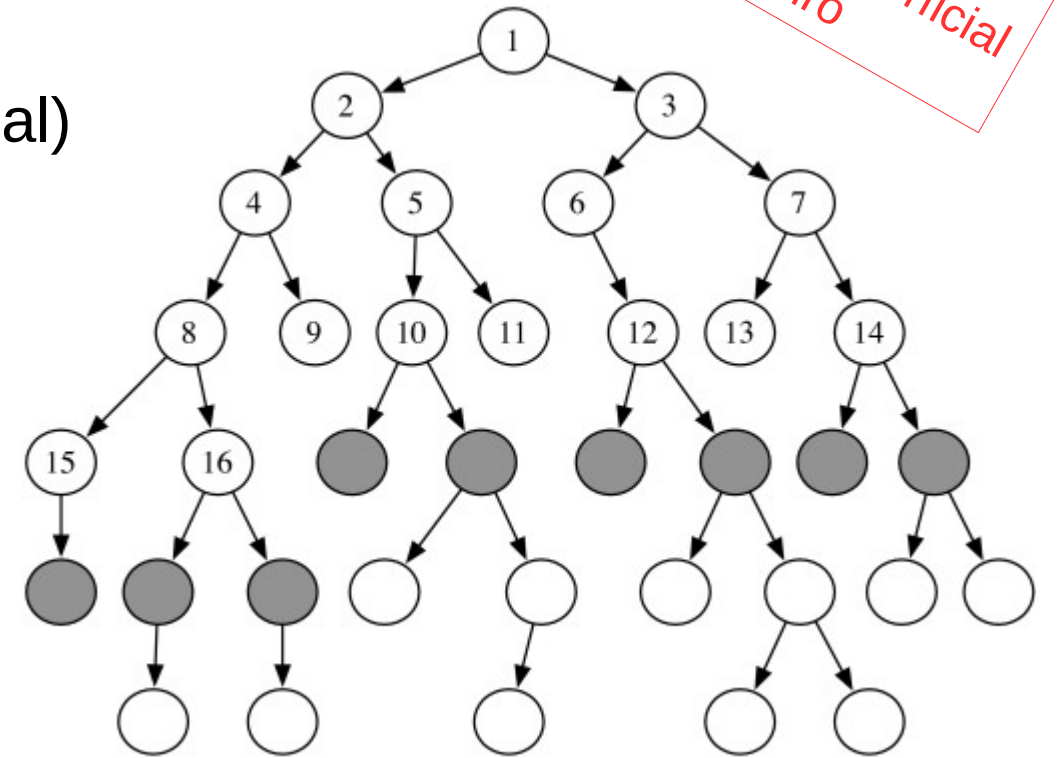
expandir o nó mais superficial primeiro

1) fronteira ← raiz (estado inicial)

2) expandir todos os sucessores da fronteira

3) fronteira ← novos nós obtidos em 2

4) repetir desde 2



fonte: AI: FCA

Ciências ULisboa

implementação

- fila FIFO

  e uns “detalhes”

- teste do objetivo, quando se gera um novo nó

  menor complexidade

  e não quando vai ser expandido
- descarta novos nós que já existam

  garante percurso mais curto até cada nó

  - na fronteira, ou expandidos

função procura em largura

função PROCURA-EM-LARGURA(*problema*) **retorna** solução, ou falha
  nó ← nó com ESTADO = *problema.*ESTADO-INICIAL, CUSTO-CAMINHO = 0
  se *problema.*TESTE-OBJETIVO(*nó.*ESTADO) **então retorna** SOLUÇÃO(*nó*)
  fronteira ← fila FIFO com *nó* como único elemento
  explorado ← conjunto vazio
  ciclo
    se VAZIO(*fronteira*) **então retorna** falha
    nó ← POP(*fronteira*) */* escolhe o nó mais superficial */*
    acrescenta *nó.*ESTADO a *explorado*
    para cada *ação* **em** *problema.*AÇÃO(*nó.*ESTADO) **faz**
      descendente ← DESCENDENTE-NÓ(*problema, nó, ação*)
      se *descendente.*ESTADO não está em *explorado* ou em *fronteira* **então**
        se *problema.*TESTE-OBJETIVO(*descendente.*ESTADO) **então retorna** SOLUÇÃO(*descendente*)
        fronteira ← INSERE(*descendente, fronteira*)

desempenho da procura em largura

- completo

  desde que objetivo a profundidade finita e *b* finito

- não garante encontrar ótimo

  encontra o objetivo mais superficial (menor caminho)

  mas pode não ser o ótimo!

garante, se custo for função não decrescente da profundidade do nó (ex. ações de custo uniforme)

complexidade

- temporal (com objetivo a profundidade *d*)

  $b + b^2 + b^3 + \ldots + b^d = O(b^d)$

  teste objetivo só quando se expande daria $O(b^{d+1})$

- espacial

  $O(b^{d-1})$ no conjunto *explorado*

  - $O(b^d)$ na *fronteira*

  => $O(b^d)$

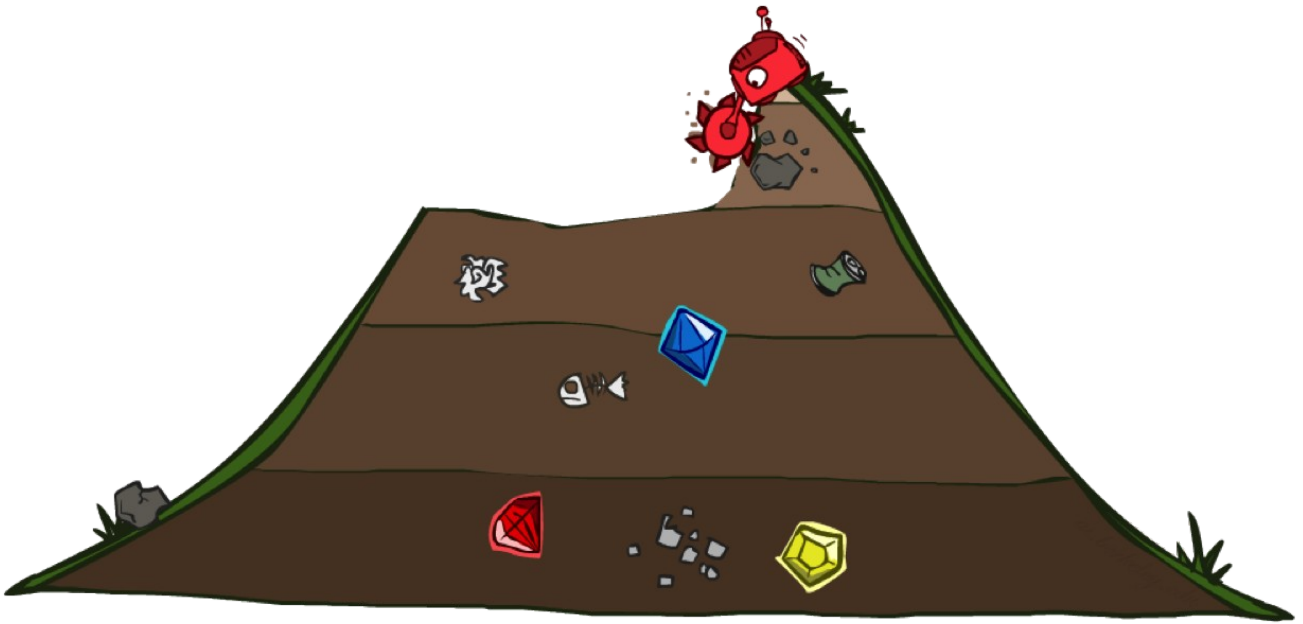
procura em largura
reality check

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

fonte: AIMA

procura de custo uniforme



fonte: CS188 Berkeley

procura de custo uniforme

ótimo com qualquer função de custo ≡ algoritmo de Dijkstra

- expande o nó *n* com menor custo acumulado *g(n)*

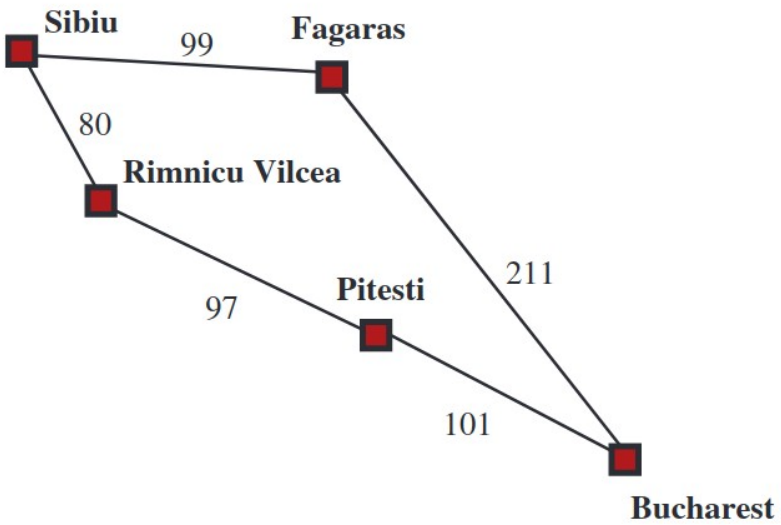
  fronteira = fila de prioridade ordenada por *g*

- teste de objetivo aplicado quando se expande o nó

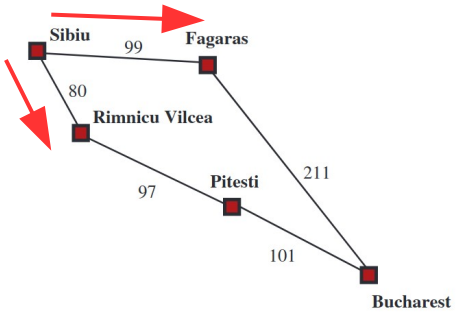
  em vez de quando é criado

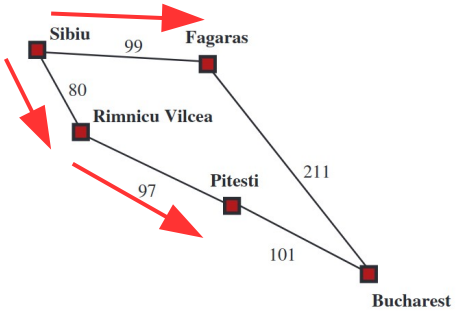
- teste no ciclo para verificar se há um melhor caminho para um nó na fronteira

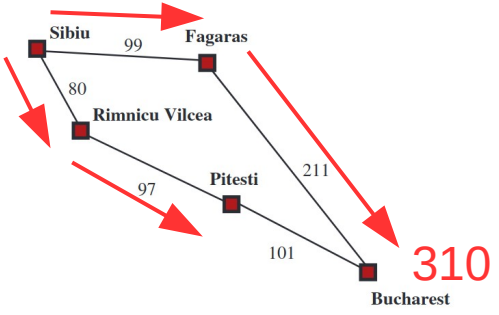
procura de custo uniforme, ex.

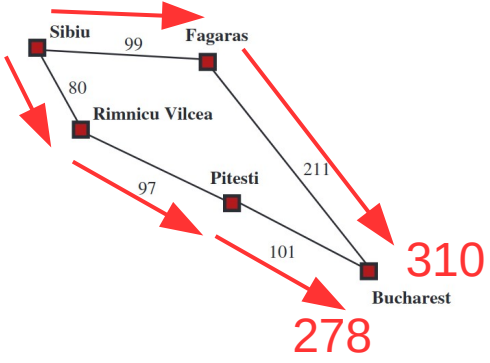


fonte: AIMA









Ciências ULisboa

procura de custo uniforme – desempenho

- ótimo e completo desde que todos os custos >0

- sendo C^* o custo do percurso ótimo
  e ϵ o custo mínimo de uma ação

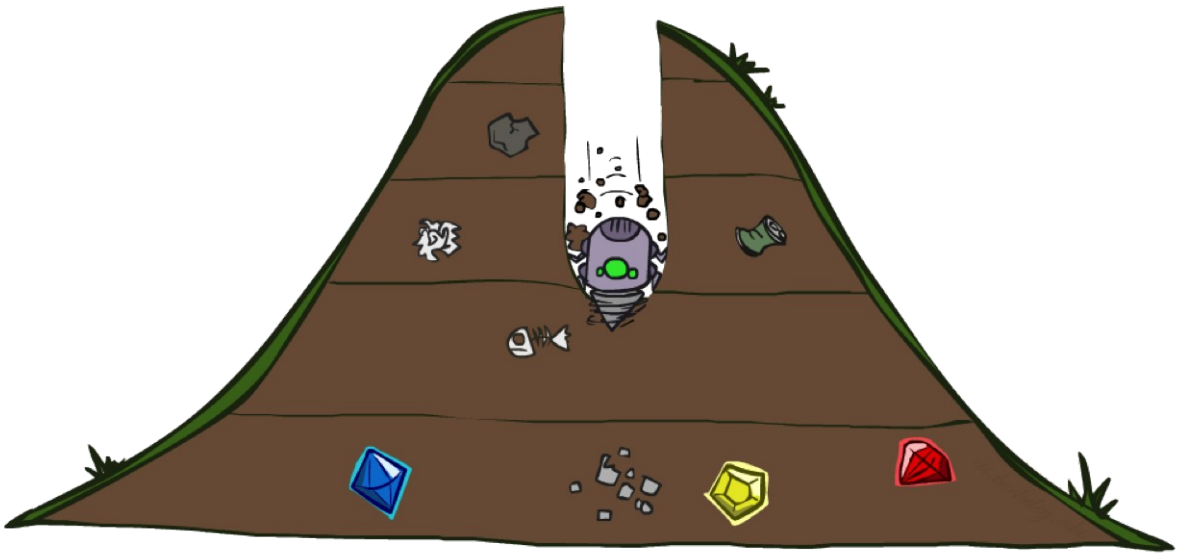
  a complexidade temporal e espacial é

$$O(b^{1+\lfloor C^*/\epsilon \rfloor})$$

  pode ser muito maior do que b^d caso ϵ pequeno

  pode explorar árvores grandes de passos com pouco custo...

procura em profundidade



fonte: CS188 Berkeley

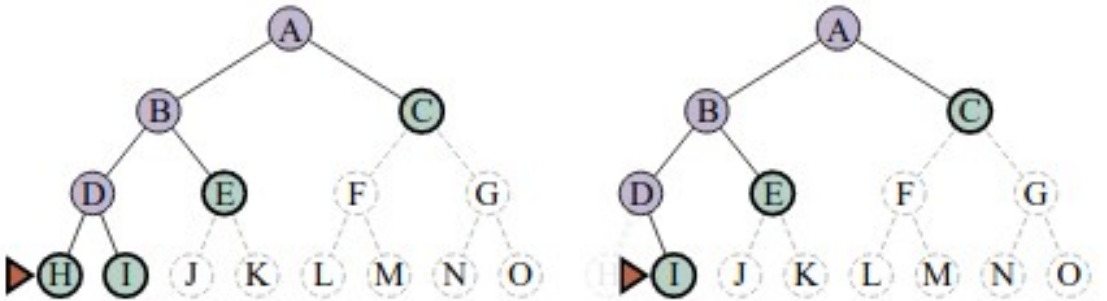
procura em profundidade

- explora primeiro o nó mais profundo na fronteira

  fila LIFO

  implementação recursiva é frequente

dois passos de uma procura em profundidade



fonte: AIMA

procura em profundidade - propriedades

- não garante solução

  pode ficar em ciclos em em espaços infinitos

- não garante o ótimo

  devolve primeira solução
  – que pode ser mais profunda do que a solução ótima

procura em profundidade – complexidade

- **complexidade temporal**

  sendo *m* a profundidade máxima da árvore de procura

  pode gerar toda a árvore

  $$O(b^m)$$

  que pode ser muito maior do que a dimensão do espaço de estados!

  m pode ser muito maior do que *d* (prof. da solução mais superficial)

procura em profundidade – complexidade

- **complexidade espacial**

  pesquisa em árvore (não usa conjunto *explorado*)

  só armazena no máximo caminho da raiz a uma folha

  $O(bm)$ nós expandidos

  comparando com $d = 16$ na tabela da pg. 9

  156×10^3 vs. 10×10^{18}

  variante com retrocesso só expande 1 descendente de cada vez

  $O(m)$

aprofundamento progressivo

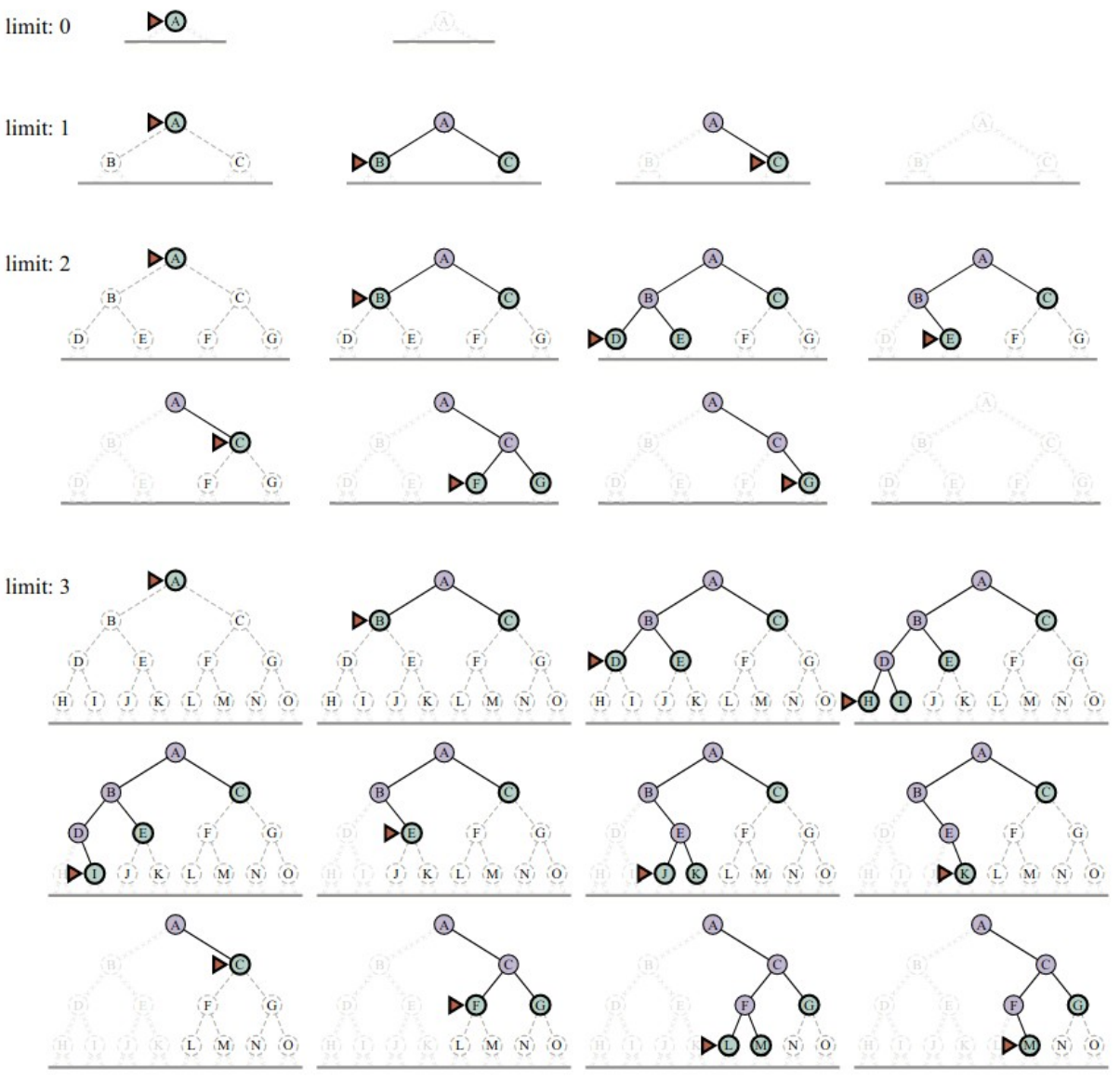
- faz sucessivas procuras em profundidade limitada aumentando o limite após cada procura sem sucesso

função APROFUNDAMENTO-PROGRESSIVO(*problema*) **retorna** uma solução, ou falha

  para *profundidade*=0 **até** ∞ **faz**

    resultado ← PROCURA-PROFUNDIDADE-LIMITADA(*problema, profundidade*)

    se *resultado* ≠ corte **então retorna** *resultado*



fonte: AIMA

Ciências ULisboa

procura em profundidade limitada (recursiva)

função PROCURA-PROFUNDIDADE-LIMITADA(*problema, limite*) **retorna** solução, ou falha/corte
  retorna PPL-RECURSIVA(GERA-NÓ(*problema*.ESTADO-INICIAL),*problema, limite*)

função PPL-RECURSIVA(*nó, problema, limite*) **retorna** solução, ou falha/corte
  se *problema*.TESTE-OBJETIVO(*nó*.ESTADO) **então retorna** SOLUÇÃO(*nó*)
  senão se *limite* = 0 **então retorna** corte
  senão
    ocorreu_corte ← falso
    para cada *ação* **em** *problema*.AÇÃO(*nó*.ESTADO) **faz**
      descendente ← DESCENDENTE-NÓ(*problema, nó, ação*)
      resultado ← PPL-RECURSIVA(*descendente, problema, limite-1*)
      se *resultado* = corte **então** *ocorreu_corte* ← verdadeiro
      senão se *resultado* ≠ falha **então retorna** *resultado*
    se *ocorreu_corte* **então retorna** corte **senão retorna** falha

aprofundamento progressivo - desempenho

- combina vantagens de procura em largura com procura em profundidade

  $O(bd)$ - poucos requisitos de memória, como pr. profundidade

  completo – como pr. largura

    desde que b finito

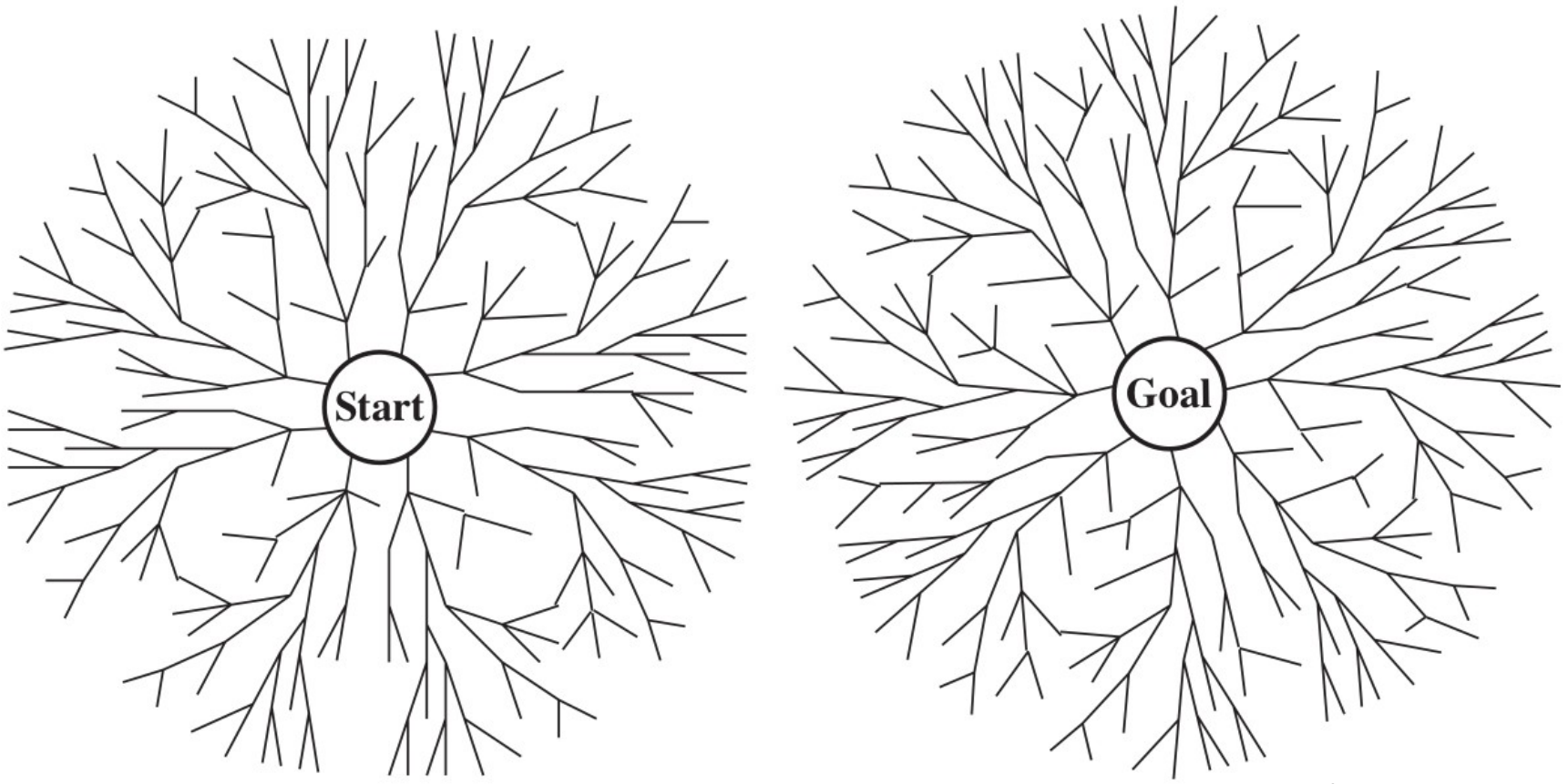
  ótimo garantido – como pr. largura

    desde que custo do caminho seja uma função não decrescente da profundidade do nó

repetição de expansão de nós a partir da raiz pouco significativa. ex: $b=10, d=5$ $N(\text{AP})=123.450$, $N(\text{PL})=111.110$

método preferível de procura não informada quando
espaço de procura é grande e
a profundidade da solução é desconhecida

procura bidirecional



fonte: AIMA

procura bidirecional

- reduz a complexidade temporal para

  $O(b^{d/2})$

- uma das fronteiras tem de estar em memória

  pode reduzir-se a outra árvore com aprofundamento progressivo

  ainda assim requisitos de espaço podem ser grandes

- e tem de haver uma função (simples) de obter predecessores

  ações inversas (nem sempre fáceis de calcular...)