Projeto de LabIA

Leliane Nunes de Barros

Algoritmo geral de busca

função Busca-Geral(*problema*, *Queing-FN*) devolve uma solução, ou falha

/* inicializa a árvore de busca com o estado inicial do *problema */*

nós ← CRIA-LISTA(Estado-Inicial[problema])

laço faça

se a lista de *nós* estiver vazia então devolva falha

nó ← Remove-Primeiro (*nós*)

se Testa-Meta(*nó*) então devolva *nó* (e o caminho da raiz até o nó)

nós ← *Queing-FN* (nós, EXPANDA(nó, Ações[problema]))

fim

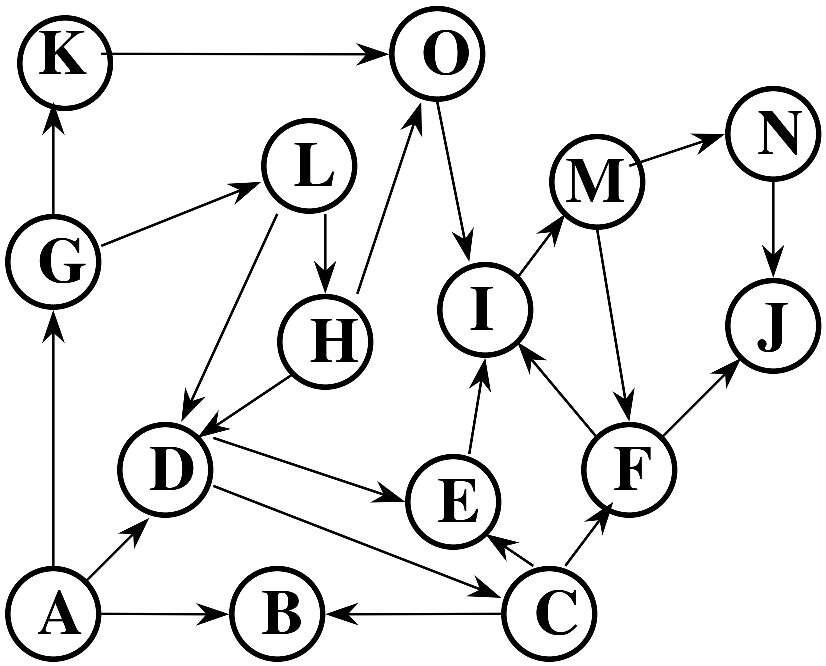
Estratégias em *Queing-FN*:

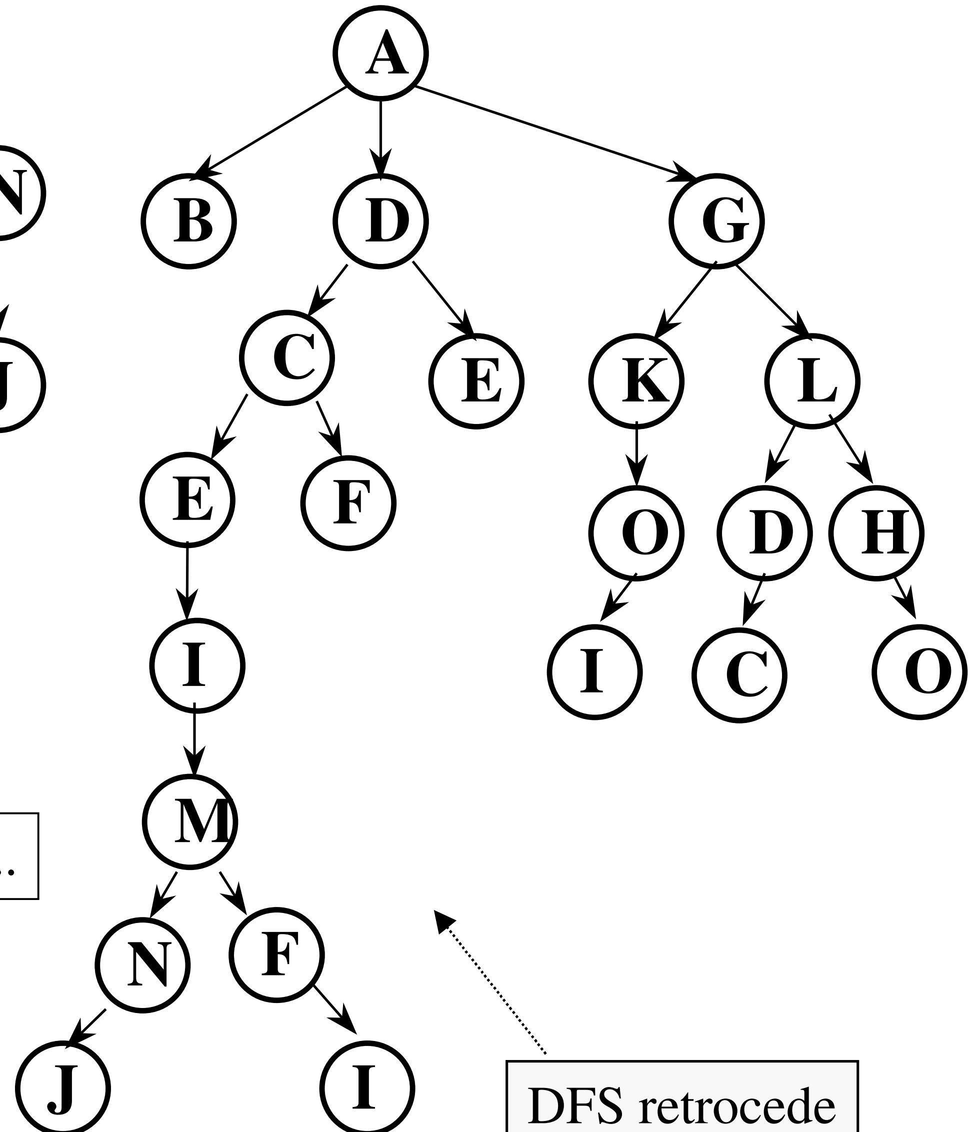
inserção numa fila: busca em largura

inserção numa pilha: busca em profundidade

fila de prioridade: A* (ordenar a fila de acordo uma função de avaliação $f(n)= g(n) + h(n)$)

espaço de estados X espaço de busca





Busca em Largura (BFS)

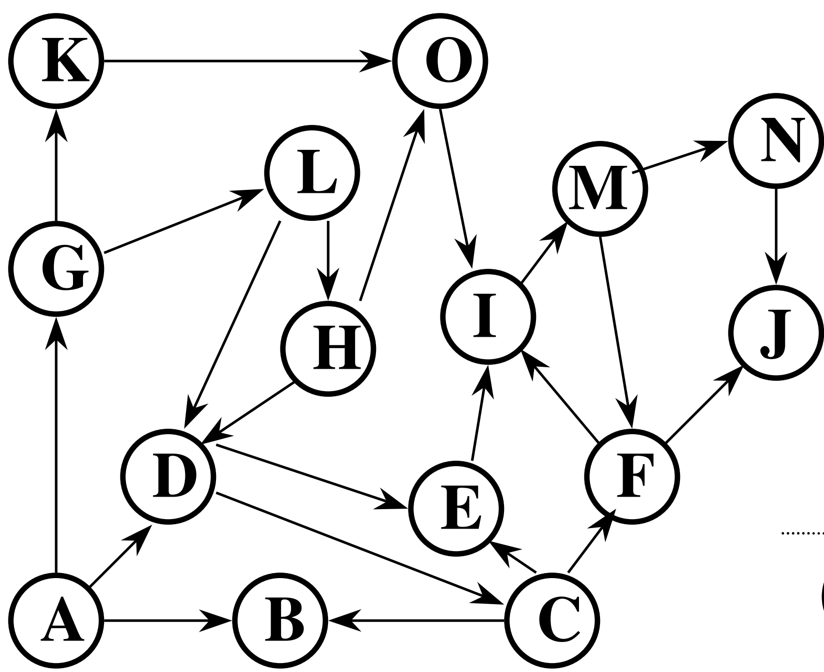
A BDG CEKL EFODH IICO M NF JI ...

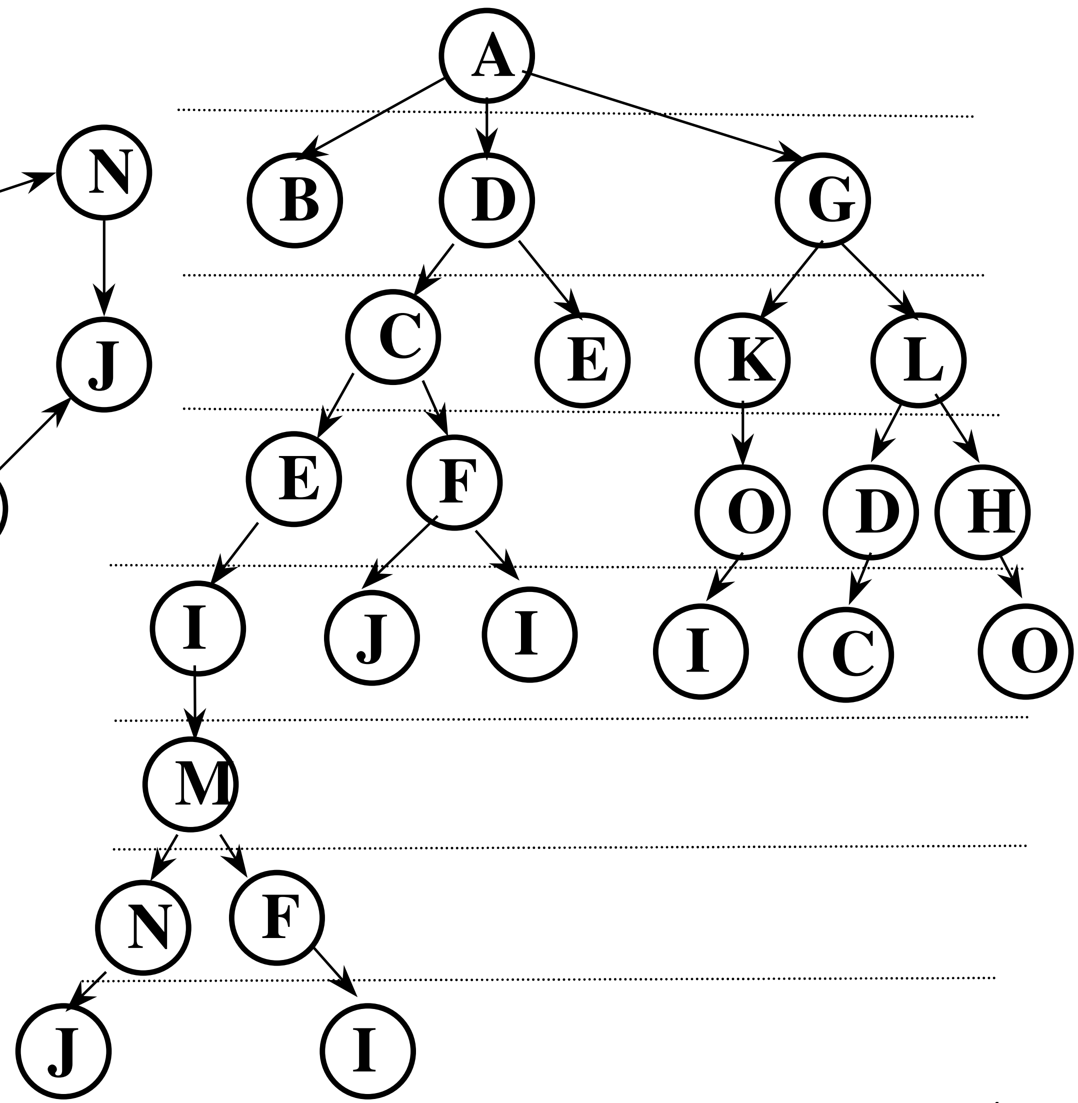
Busca em Profundidade (DFS)

AB DCEIMNJ FI F E GKOI ...

DFS retrocede

Busca em profundidade iterativa





A
A B D G
A B D C E G K L
A B D C E F E I G K O L D H

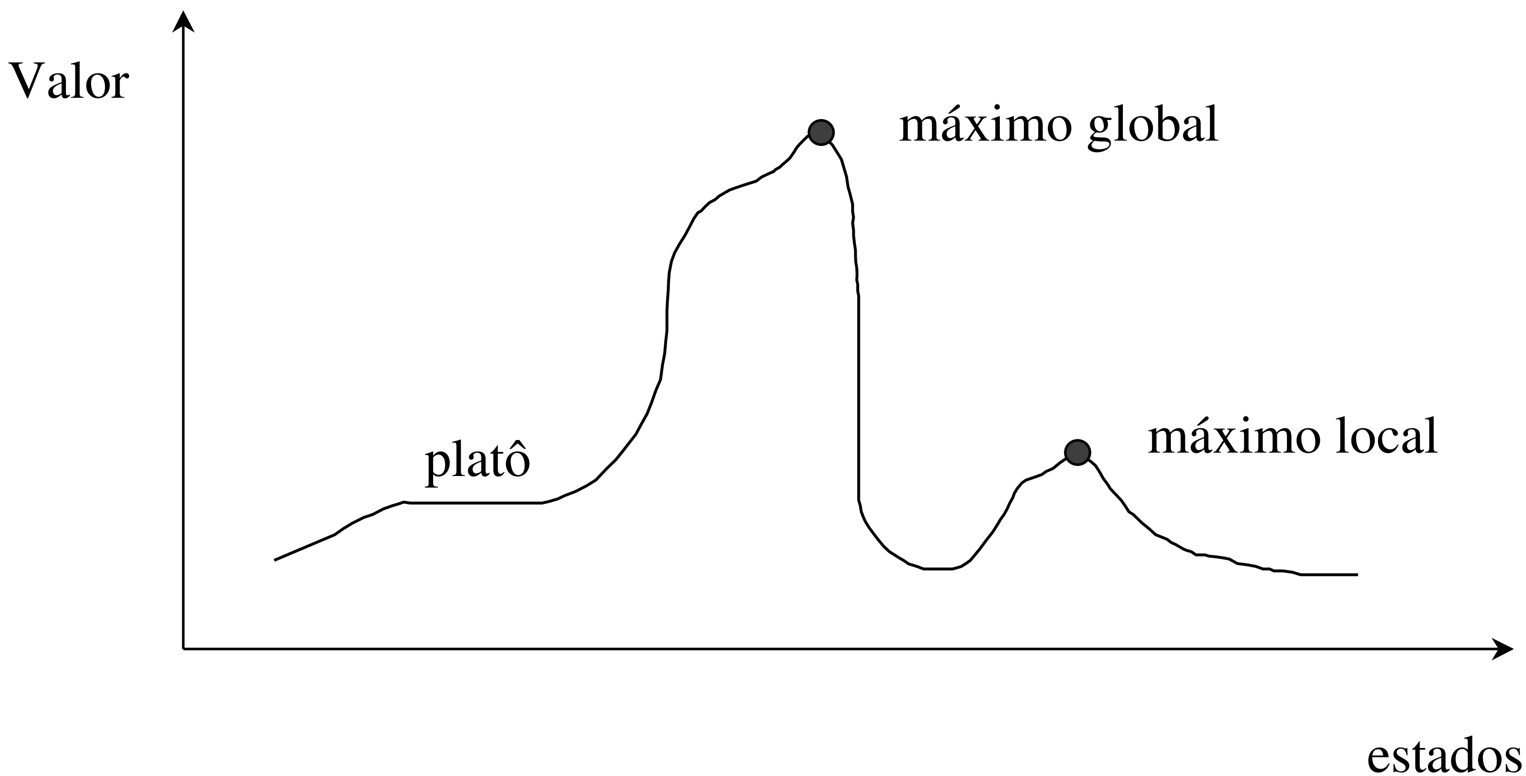
Busca Local

- … ou algoritmo de melhora iterativa
- recomendado quando o estado do mundo contém toda a informação necessária para a solução (exemplo: n-rainhas)
- busca:
  - começa com uma configuração completa
  - modifica o estado para melhorar sua qualidade ou satisfazer uma propriedade (estado meta, em planejamento)

Busca Local

- não mantém a árvore de busca. Guarda somente um estado na memória (em planejamento, guarda também um caminho)
- a busca caminha sobre uma superfície topológica de estados procurando o pico mais alto (ou mais baixo), onde estão as soluções ótimas
- não faz *backtracking*
- só olha para a vizinhança do estado corrente
- Algoritmos mais conhecidos:
  - ***Hill climbing***
  - ***Simulated annealing***

Hill-climbing: topologia do espaço de busca



Hill-climbing para planejamento

- Nó: armazena o estado do mundo (completo)
- planejamento progressivo (I $\rightarrow$ G)
- expande nós sucessores com *h(s)* mínimo
- a busca caminha sobre uma superfície topológica de estados procurando o pico mais BAIXO, onde estão as soluções ótimas
- para, quando encontrar h(s) = 0
- armazena o caminho percorrido pelos nós selecionados

Hill-climbing para planejamento

Mínimo local: não consegue se afastar de uma solução não ótima ou de um estado que não é solução.

```
      /
   /\ /
  /  \/ mínimo local
 /
```

Platô: região do espaço de busca em que a função heurística é constante (busca aleatória)

```
                  /
  _______________/
 /     platô
/
```

Hill-climbing

Função Hill-Climbing (*problema*, h)) **devolve** *um estado solução*

plano ← Cria-Plano /* plano vazio */

estado-corrente ← Faça-nó (Estado-Inicial[*problema*])

enquanto (h(estado-corrente) ≠ 0) **faça**

estados-sucessors ← EXPANDA(*estado-corrente*, Ações[problema])

próximo-estado ← estado-sucessor com h(s) ≤ h(*estado-corrente)*

plano ← Atualiza(plano, próximo-estado, ação)

estado-corrente ← *próximo-estado*

fim

Hill-climbing reforçado

Função Hill-Climbing (*problema*, h)) **devolve** *um estado solução*

plano ← Cria-Plano /* plano vazio */

estado-corrente ← Faça-nó (Estado-Inicial[*problema*])

enquanto (h(*estado-corrente)* ≠ 0) **faça**

busca em largura apartir do estado-corrente por um estado-sucessor *s'*

em que h(s) < h(*estado-corrente)*

se não encontrou s' devolve falha

próximo-estado ← *s'*

plano ← Atualiza(plano, próximo-estado, ação)

estado-corrente ← *próximo-estado*

fim enquanto

devolve plano

Projeto

- implementar o algoritmo HSP com 2 estratégias de busca **hill-climbing reforçado** e **A*** (1 escolha por grupo)
- todos devem implementar os dois tipos de função heurítica do HSP: $\boldsymbol{h_{aditiva}}$ e $\boldsymbol{h_{max}}$
- HSP faz uma busca do tipo *hill-climbing* do estado inicial I a um estado meta s tal que que $\mathrm{G} \subseteq \mathrm{s}$
- **DESAFIO**: implementar a função heurística!

Projeto

- análise do planejador em diferentes domínios de planejamento
  - análise no tempo
  - análise no comprimento do plano solução
- seleção de três domínios de teste
  - obrigatórios: Mundo dos Blocos e Satélite
  - livre: um domínio a escolha de cada grupo que pode ser um dos usados nas competições de planejamento ou um novo domínio modelado pelos elementos do grupo

Heurística do planejador HSP

- heurística $h(s)$ é uma estimativa do número de passos necessários para resolver o problema relaxado, ou seja, num domínio em que as listas de eliminação das ações são desprezadas.
- s: estado, p: proposição
- $g(p;s)$: custo estimado para atingir p apartir de s
- $g(Prec(op;s))$: custo estimado para atingir as precondições da ação op de s

Heurística do planejador HSP

*Função heurística **aditiva***

Custo estimado para atingir p apartir de s

$$g(p;s) = \begin{cases} 0 & se\ p \in s \\ \min_{op \in O(p)} [1 + g(Prec(op); s)] & caso\ contrário \end{cases}$$

Para conjuntos de átomos C:

$$g(C;s) = \sum_{r \in C} g(r;s)$$

Heurística resultante:

$$h_{add}(s) = \sum_{p \in G} g(p;s)$$

Heurística para o HSP

Função heurística ***max***

- Custo estimado para atingir p apartir de s

$$g(p;s) = \begin{cases} 0 & \text{se } p \in s \\ \min_{op \in O(p)} [1 + g(Prec(op); \mathrm{s})] & \text{caso contrário} \end{cases}$$

Para conjuntos de átomos C (:

$$g(C;s) = \max_{r \in C} g(r;s)$$

Heurística resultante

$$h_{max}(s) = \max_{p \in G} g(p;s)$$

Heurísticas para o HSP

heurística aditiva:

- informativa mas não-admissível
- supõe metas independentes

heurística max:

- admissível mas não informatitva
- supõe metas dependentes
- supõe que o plano de comprimento máximo pode atingir um conjunto de sub-metas de uma vez, por exemplo, a lista de precondições de uma ação

Algoritmo para calcular h(G;s)

- polinomial no número de átomos e ações
- versão do algoritmo Bellman-Ford para encontrar o caminho mais curto em grafos

Algoritmo para calcular h(G;s)

- inicializa custos $g(p;s)$

  $g(p;s)=0$ se $p \in s$ e $g(p;s)= \propto$ caso contrário

- faça ***planejamento progressiva*** e quando p for adicionado pelo operador op atualize $g(p;s)$

  $g\ (p;s) = min\ [g(p;s), (1 + g(Prec(op); \mathrm{s})]$

- condição de parada: quando $g(p;s)$ não mudar mais de valor
- duas opções para o cálculo do custo de um conjunto de átomos
  - custo aditivo: $g(C; s) = \sum_{p \in \mathrm{C}} g(r; s)$
  - custo max: $g(C; s) = \max_{\mathrm{p} \in \mathrm{C}} g(r; s)$