Universidade Federal de São Carlos - UFSCar
Departamento de Computação - DC
CEP 13565-905, Rod. Washington Luiz, s/n, São Carlos, SP

Algoritmos Gulosos - Parte 1

Prof. Dr. Alan Demétrius Baria Valejo

CCO-00.2.01 - Projeto e Análise de Algoritmos (*Design And Analysis Of Algorithms*)
1001525 - Projeto e Análise de Algoritmos - Turma A

Roteiro

- Introdução
- Vantagens e limitações
- Algoritmos famosos

Algoritmos gulosos

- Algoritmos gulosos
  - Do inglês Greedy -> Ganancioso
  - Mais conhecido como Guloso
  - Uma das principais técnicas para se projetar algoritmos eficientes para diferentes problemas difíceis

Algoritmos gulosos

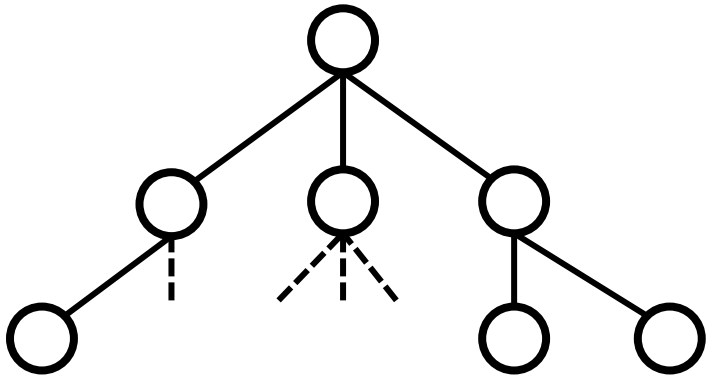
- Algoritmos gulosos
  - Do inglês Greedy -> Ganancioso
  - Mais conhecido como Guloso
  - Uma das principais técnicas para se projetar algoritmos eficientes para diferentes problemas difíceis

Algoritmos gulosos

- Algoritmos gulosos
  - Do inglês Greedy -> Ganancioso
  - Mais conhecido como Guloso
  - Uma das principais técnicas para se projetar algoritmos eficientes para diferentes problemas difíceis

- Algoritmo iterativo
- Dominado por um laço
  - Percorrer a entrada ou o espaço de busca
- Faz escolhas com base no que parece melhor no momento atual

Grafo ou espaço de busca



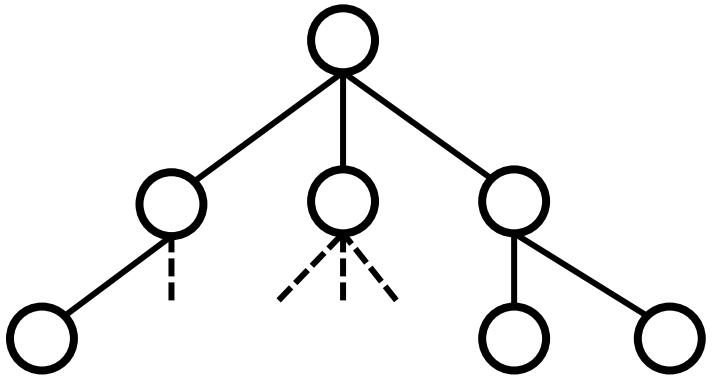
```
moedas = [100, 50, 25, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

- Algoritmo iterativo
- Dominado por um laço
  - Percorrer a entrada ou o espaço de busca
- Faz escolhas com base no que parece melhor no momento atual

Grafo ou espaço de busca



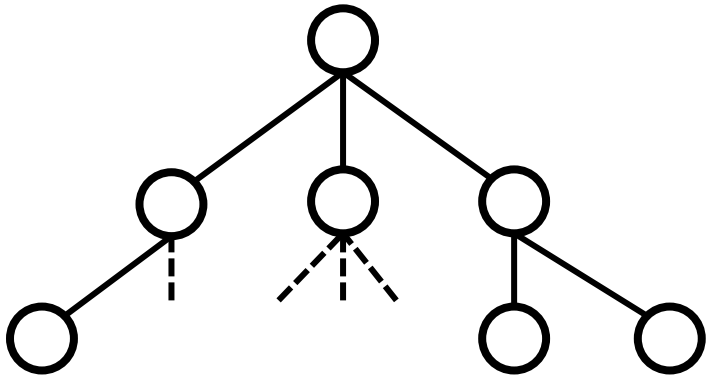
```
moedas = [100, 50, 25, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

- Algoritmo iterativo
- Dominado por um laço
  - Percorrer a entrada ou o espaço de busca
- Faz escolhas com base no que parece melhor no momento atual

Grafo ou espaço de busca



```
moedas = [100, 50, 25, 5, 1]
total = 0
troco = 130

for i in range(len(moedas)):
    num_moedas = troco // moedas[i]
    troco -= num_moedas * moedas[i]
    total += num_moedas

print(total)
```

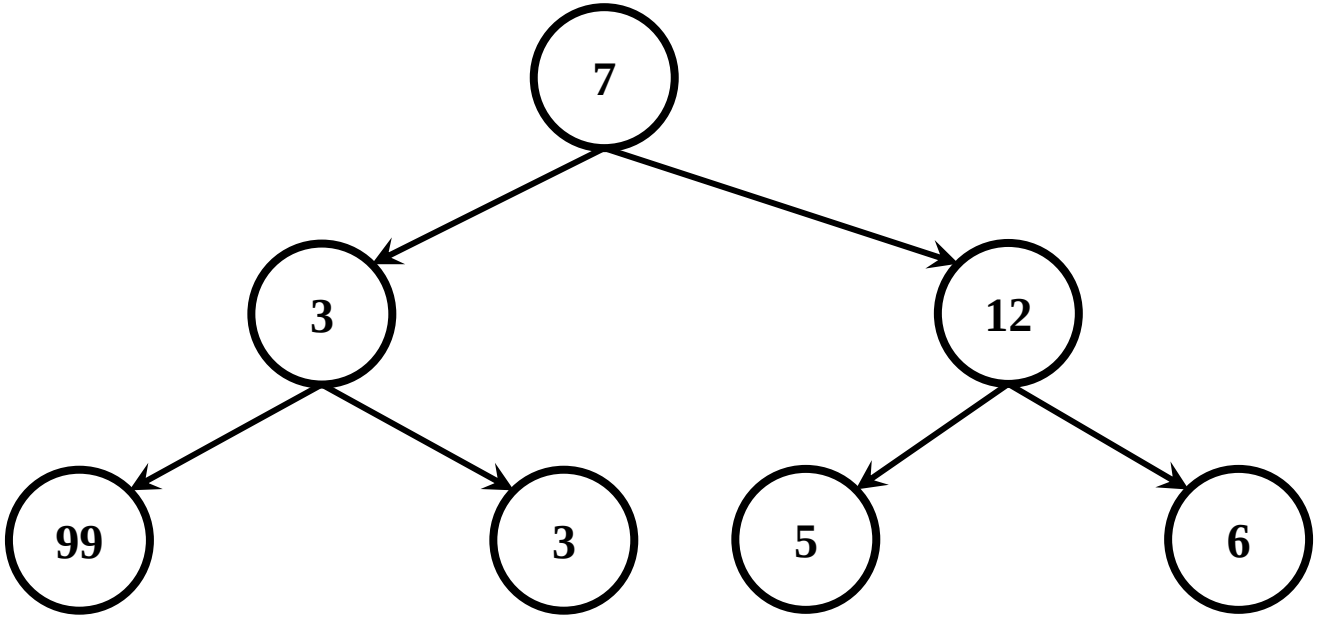
Tudo acaba em Pizza

Vamos para a praia?

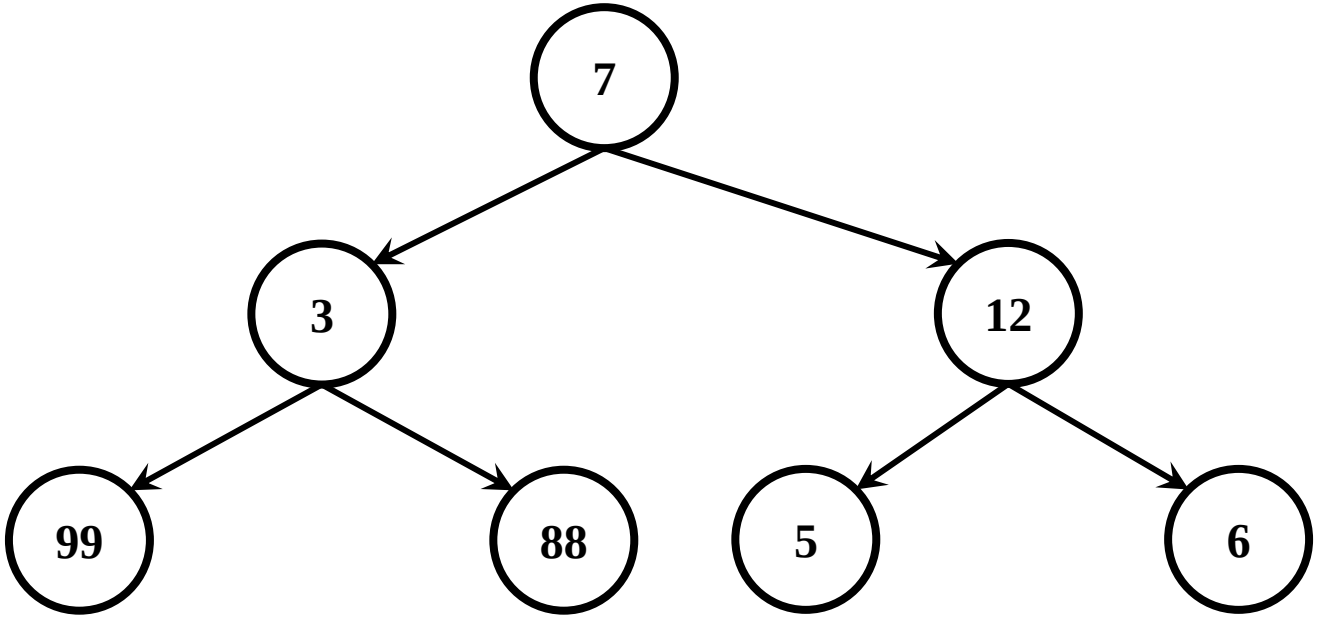
Vamos para a praia?

1. Decisão local
2. Jamais se arrepende
3. Nem sempre obtém o melhor resultado

Vamos para a praia?

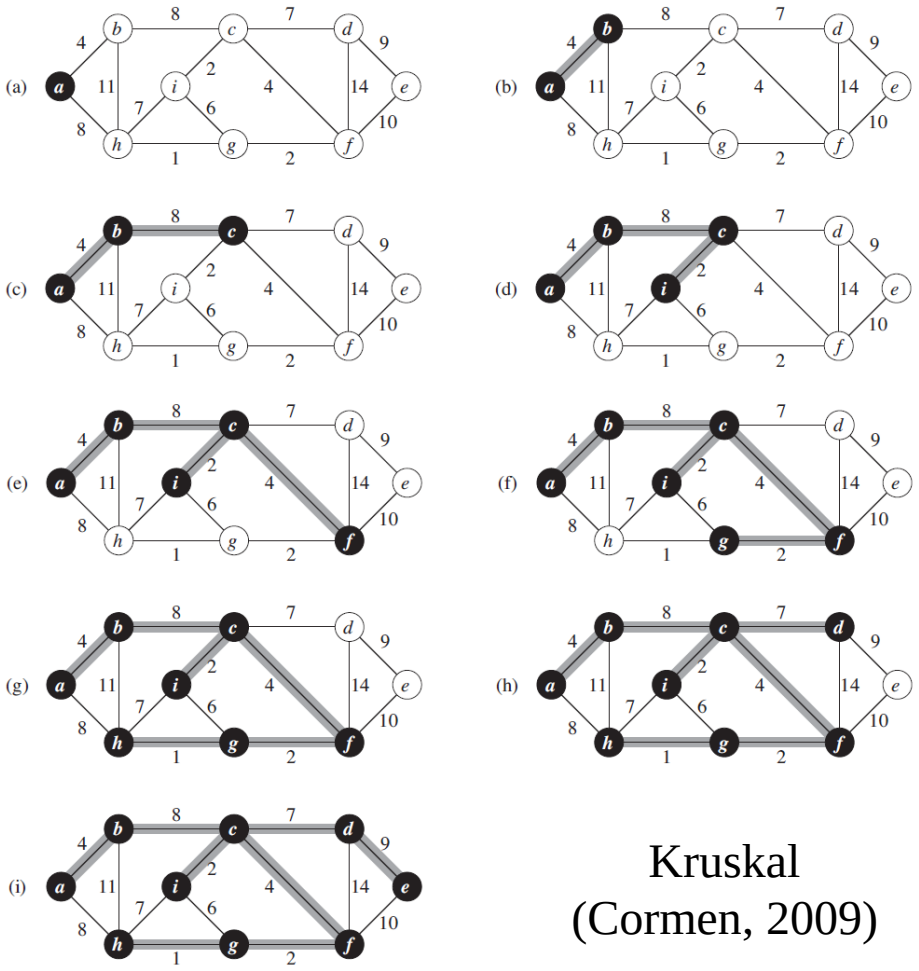


Vamos para a praia?



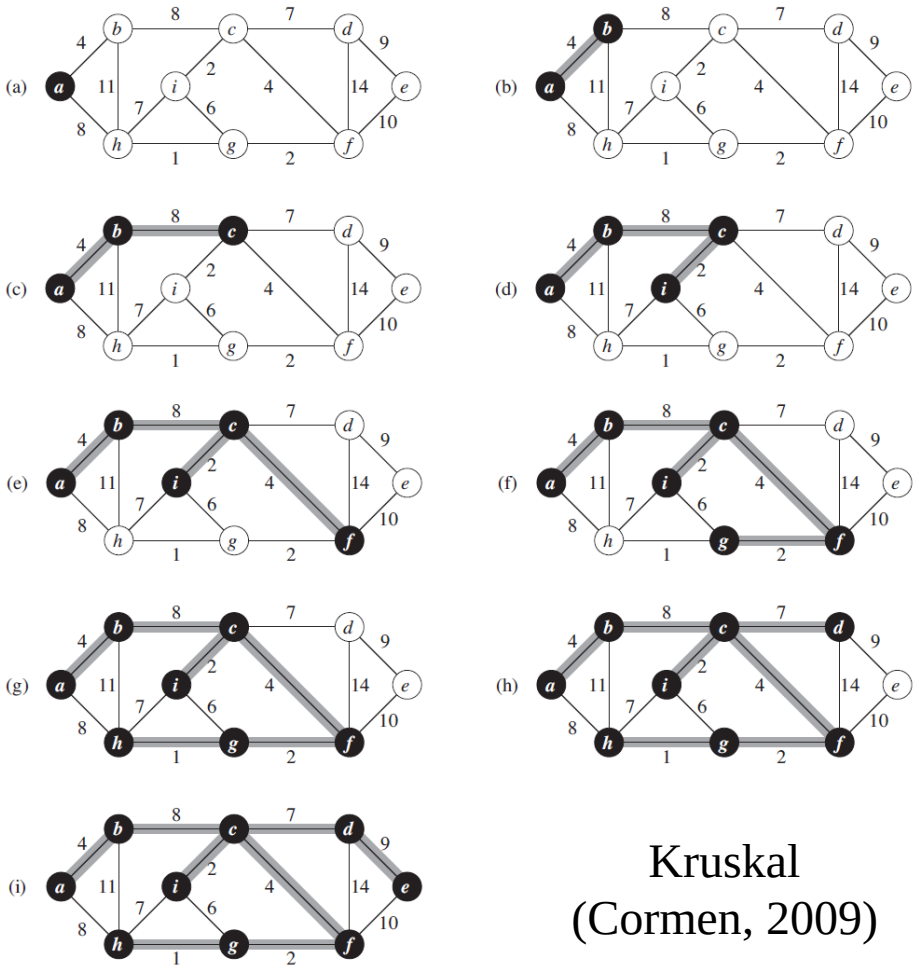
> “*Busque, busque e busque até encontrar o seu caminho ... E jamais olhe pra trás, nem mesmo pra pegar impulso*”
> **Clóvis de Barros Filho**

- Para alguns problemas, existe soluções gulosas bastante eficientes e que não garantem a melhor solução
- Existe algoritmos que fornecem garantias que a solução gerada é correta para qualquer entrada de dados



Kruskal
(Cormen, 2009)

- Para alguns problemas, existe soluções gulosas bastante eficientes e que não garantem a melhor solução
- Existe algoritmos que fornecem garantias que a solução gerada é **correta para qualquer entrada de dados**



Kruskal
(Cormen, 2009)

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

A matemática não será o foco desse bloco

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

- Geralmente é fácil projetar algoritmos gulosos para um problema
- Fácil analisar o tempo de execução desses algoritmos
- Não é trivial projetar um algoritmo guloso correto, que sempre devolve a melhor solução
- Não é fácil provar tal corretude
  - Muitos algoritmos gulosos não possuem prova de corretude
  - As provas de corretude geralmente são:
    - Por indução no número de iterações
      - Mostrando que a escolha gulosa está correta a cada passo
    - Usando um argumento de troca entre soluções:
      - Que pode ser por contradição, para mostrar que algo diferente da solução gulosa é pior
      - Ou que iterativamente vai transformando uma solução qualquer em uma solução gulosa sem piorar seu custo

Algoritmos gulosos
Principais características

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

Algoritmos gulosos
Principais características

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

Algoritmos gulosos
Principais características

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

Algoritmos gulosos
Principais características

- Jamais se arrepende de uma decisão, as escolhas realizadas são definitivas;
- Não leva em consideração as consequências de suas decisões;
- Podem fazer cálculos repetitivos;
- Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- Quanto mais informações, maior a chance de produzir uma solução melhor.

Algoritmos gulosos
Vantagens

- Simples e de fácil implementação;
- Algoritmos de rápida execução;
- Podem fornecer a melhor solução (estado ideal).

Algoritmos gulosos
Vantagens

- Simples e de fácil implementação;
- Algoritmos de rápida execução;
- Podem fornecer a melhor solução (estado ideal).

Algoritmos gulosos
Vantagens

- Simples e de fácil implementação;
- Algoritmos de rápida execução;
- Podem fornecer a melhor solução (estado ideal).

Algoritmos gulosos
Vantagens

- Simples e de fácil implementação;
- Algoritmos de rápida execução;
- Podem fornecer a melhor solução (estado ideal).

Limitações

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

Limitações

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

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

Limitações

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

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

- Prim’s
- Kruskal’s
- Dijkstra’s
- Código de Huffman
- Mochila fracionária
- Seleção de atividades
- Algoritmos para *job scheduling*
- ...

Próxima aula

- Algoritmos Gulosos Parte 2

Obrigado

Dúvidas

Email: alanvalejo@ufscar.br

Acessar o fórum no Moodle