Aula 8 - Algoritmos Gulosos

Luís Felipe

UFF

21 de Setembro de 2023

Luís Felipe

21/09/23

A estratégia gulosa

- Em um jogo de xadrez, jogar pensando apenas no que é bom no momento da jogada, pode ser desastroso.
- Mas em vários outros jogos, como o jogo de palavras cruzadas, é possível se sair bem fazendo o que parece melhor no momento.
- A estratégia gulosa é aquela na qual a solução é construída passo a passo e o próximo passo é sempre o mais vantajoso no momento. Pode ser comparado a pessoas:
  - míopes, ou
  - que não pensam no futuro.
- Um algoritmo guloso nunca se arrepende de um passo anterior.
- Para vários problemas, esta pode não ser a estratégia ótima, mas para vários outros um algoritmo guloso pode ser uma abordagem atraente e ótima.

Luís Felipe

21/09/23

O problema do troco

- Moedas estão em falta atualmente devido a crescente onda de cofrinhos nos domicílios brasileiros.
- Um problema muito comum para os comerciantes é o troco, sobretudo quando envolve moedas.
- Assim, os comerciantes têm buscado dar o troco com o menor número de moedas, dentre as que eles têm disponíveis.
- Como vocês poderiam ajudar esses comerciantes? Qual poderia ser uma estratégia eficiente para que o troco seja dado sempre com o menor número de moedas?
- Poderíamos pensar em uma estratégia gulosa onde escolhemos primeiro as moedas de maior valor.
- Seria uma boa estratégia?

Luís Felipe

21/09/23

Exemplos

1. moedas $\{9, 6, 5, 1\}$
   - troco 7 — gulosamente uma de 6 e uma de 1 tem como melhorar?

     NÃO!!
   - troco 15 — gulosamente uma de 9 e uma de 6 tem como melhorar?

     NÃO!!
   - troco 11 — gulosamente uma de 9 e duas de 1 tem como melhorar?

     SIM!!

     Observe que sem usar a estratégia gulosa, podemos dar um troco de 11 com apenas duas moedas: uma de 6 e uma de 5.

Luís Felipe

21/09/23

Escalonamento de tarefas

- Imaginem que vocês estejam participando de um congresso
- Como vocês estão ávidos por conhecimento (ou estão sendo observados pelos orientadores), querem assistir o máximo de palestras possível
- Em congressos, não é de bom tom entrar no meio de uma palestra e nem sair antes que uma palestra acabe
- Assim, dada a programação do evento que consta dos horários iniciais e finais de cada palestra, qual seria a melhor estratégia gulosa dentre as descritas abaixo para maximizar o número de palestras assistidas?
  - Assistir as que começam mais cedo
  - Assistir as que têm menor duração
  - Assistir as que têm menos conflitos
  - Assistir as que terminam mais cedo

Luís Felipe

21/09/23

Contra-exemplos

- Começam mais cedo:

- Mesma duração:

- Menos conflitos:

Luís Felipe

21/09/23

Guloso sim, poderoso também

Estratégia: Assistir as que terminam mais cedo

Justificativa: Sejam $s(x)$ e $f(x)$ os horários que começam e terminam o evento x, resp. Seja A uma solução fornecida pela estratégia gulosa e seja $\mathcal{O}$ uma solução ótima.

Considere $i_1, i_2, \ldots, i_k$ as escolhas de A e $j_1, j_2, \ldots, j_m$ as escolhas em $\mathcal{O}$.

Vamos mostrar que para todo $r \leq k$ temos que $f(i_r) \leq f(j_r)$.

Por Indução sobre a quantidade de intervalos k. Suponha que a afirmação seja verdadeira para $r-1$ intervalos, i.e., $f(i_{r-1}) \leq f(j_{r-1})$. Como os intervalos em $\mathcal{O}$ são compatíveis, temos que $f(j_{r-1}) \leq s(j_r)$, temos, por transitividade, que $f(i_{r-1}) \leq s(j_r)$.

Assim, no momento em que o algoritmo guloso seleciona i_r, j_r está disponível. Logo, $f(i_r) \leq f(j_r)$. Mas por que isso mostra que $\mathcal{A}$ é ótimo?

Luís Felipe

21/09/23

Guloso sim, ótimo também

Suponha por contradição que A não seja ótima, ou seja, existe um conjunto ótimo $\mathcal{O}$ com mais intervalos que A.

Suponha $|A| = k$. Pela indução, temos $f(i_k) \leq f(j_k)$, mas como $\mathcal{O}$ tem mais intervalos que A, existe j_{k+1} que tem início após o fim de j_k. Ou seja, j_{k+1} estava disponível após a escolha de i_k, mas o algoritmo guloso parou. Absurdo.

Luís Felipe

21/09/23

Árvore geradora Mínima

- Imaginem que temos alguns computadores e precisamos fazer uma rede com esses computadores.
- A conexão entre computadores possui um custo.
- Nosso objetivo é interligá-los com custo mínimo.
- Este problema pode ser modelado utilizando Teoria dos Grafos.
- Cada computador é um vértice, e as arestas entre os vértices indicam possíveis ligaçõ es entre os computadores.
- Que estrutura devemos construir para minimizar o custo?
    - Manter ciclos seria uma boa estratégia?

Luís Felipe

21/09/23

Teoria dos Grafos

- Dado um grafo conexo $G = (V, E)$, uma árvore geradora de G é um subgrafo gerador $H = (V, E')$ de G, i.e., um subgrafo que possui o mesmo conjunto de vértices de G, que é uma árvore, i.e., um grafo conexo e acíclico.
- No problema da árvore geradora mínima (MST), o grafo G é ponderado, ou seja, a cada aresta é atribuído um custo.
- O objetivo é construir uma árvore geradora para G de menor custo total.
- A principal justificativa para a construção de uma árvore geradora é que a remoção de arestas de ciclos não desconecta o grafo.

Luís Felipe

21/09/23

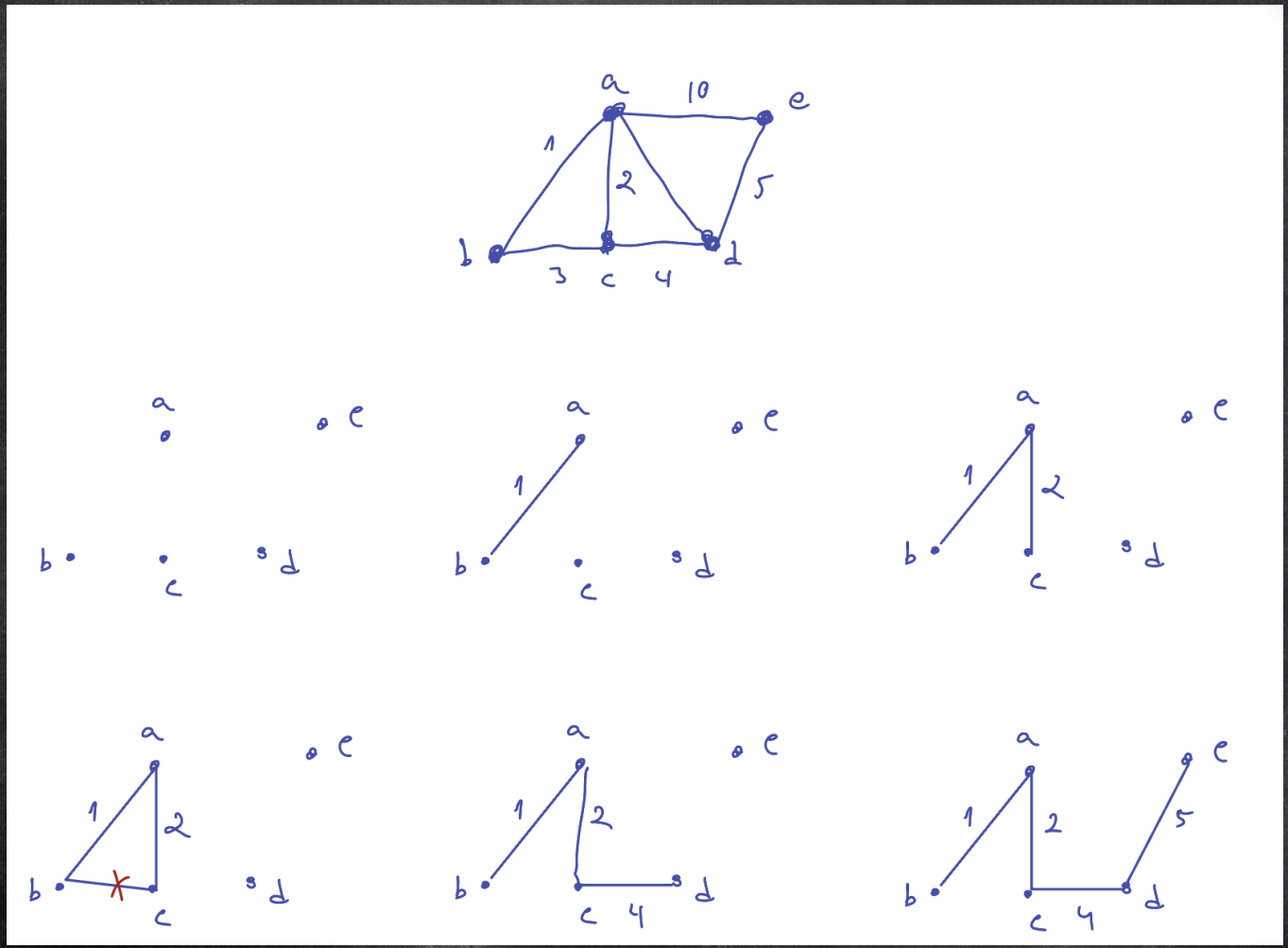
Abordagem Gulosa I - Kruskal

- Começa com o grafo nulo, i.e., sem arestas.
- Adiciona sempre a aresta de menor custo e que não forma ciclo no momento.
- É guloso, pois adicionar a aresta de menor custo é a estratégia mais vantajosa no momento.

Luís Felipe

21/09/23

Exemplo



Luís Felipe

21/09/23

Corretude do Kruskal

- Baseia-se em propriedade da teoria da conectividade
- Um corte de arestas, denotado por $[S, S']$ é um conjunto de arestas que têm um extremo em $S \in V$ e outro em $S' = V \setminus S$.

  Propriedade do corte: Suponha que um conjunto de arestas X faça parte de uma MST de $G = (V, E)$. Para todo $S \in V$ tal que X não possua aresta em $[S, S']$, $X \cup \{e\}$ faz parte de alguma MST, onde e é a aresta de $[S, S']$ de menor custo.
- Tradução: Você sempre pode adicionar a aresta de menor custo de qualquer corte, dado que X não possui arestas no corte.
- Mas por que funciona??

Luís Felipe

21/09/23

Corretude do Kruskal

Ideia da prova:

Seja X o conjunto de arestas de uma MST, T. Se e é uma aresta de X, não há nada a mostrar.

Suponha que $e \notin X$. Vamos construir uma árvore T' diferente de T de custo mínimo e que possua a aresta e.

Adicione e, aresta de custo mínimo em $[S, S']$ a T. Cria-se um ciclo. Existe outra aresta, digamos e', no corte $[S, S']$. Remova e'.

O grafo é conexo e continua com o mesmo número de arestas de T. Logo, o grafo é uma árvore.

O custo total de T' é o custo de T - o custo de e' + custo de e.

Como e tem custo mínimo em $[S, S']$, o custo de e' é no mínimo o custo de e, mas como T é MST, então sabemos que o custo de e é igual ao custo de e'.

Logo, T' é uma MST e e faz parte de T'.

Luís Felipe

21/09/23

Complexidade do Kruskal

- Primeiro ordena as arestas do grafo por peso - $O(m \log m)$, onde m é o número de arestas
- Depois, utilizando uma estrutura de dados adequada, verifica a não formação de ciclos e faz-se a fusão - $O(m \log n)$, onde n é o número de vértices
- Como $\log m = O(\log n^2) = O(2 \log n) = O(\log n)$, temos complexidade $O(m \log n)$

Luís Felipe

21/09/23

Como é essa estrutura de dados?

A cada etapa, o algoritmo escolhe uma aresta para ser adicionada na solução parcial.

Assim, precisamos testar se uma aresta candidata uv é de forma que u e v estejam em componentes conexas distintas.

Em caso positivo, as componentes devem ser unidas.

Estrutura utilizada: Conjuntos Disjuntos

Operações:

- *makeset*(x): cria um conjunto unitário contendo x
- *find*(x): retorna qual conjunto contém x
- *union*(x, y): junta conjuntos contendo x e y.

Com isso, podemos escrever com mais detalhes o algoritmo de Kruskal.

Luís Felipe

21/09/23

Algoritmo de Kruskal

Entrada: Um grafo ponderado conexo $G(V, E)$
Saída: Uma MST definida pelas arestas X.

1. Para cada vértice $u \in V$:
2. $\quad$ *makeset*(u)

3. $X = \{\}$
4. $E =$ conjunto E ordenado pelos pesos
5. Para cada aresta $\{u, v\} \in E$:
6. $\quad$ Se *find*$(u) \neq$ *find*(v):
7. $\quad\quad$ adicione aresta $\{u, v\}$ a X
8. $\quad\quad$ *union*(u, v)

Obs.: Total de operações:
makeset: $|V|$, linha 1;
find: $2|E|$, linhas 5 e 6, uma para cada extremo de aresta;
union: $|V| - 1$, linha 8, uma árvore há $|V| - 1$ arestas.

Luís Felipe

21/09/23

Como efetuar as operações?

Armazenamos um conjunto em uma árvore direcionada:

- A raiz representa o nome do conjunto;
- Cada elemento x possui um marcador $rank(x)$, representando qual altura da árvore com x sendo a raiz.
- Cada nó x possui um ponteiro para seu nó pai na árvore, $\pi(x)$. Se x for raiz, então $\pi(x) = x$.

makeset(x):

$\pi(x) = x$

$rank(x) = 0$

Complexidade: $O(1)$

find(x):

Enquanto $x \neq \pi(x)$:

$x = \pi(x)$

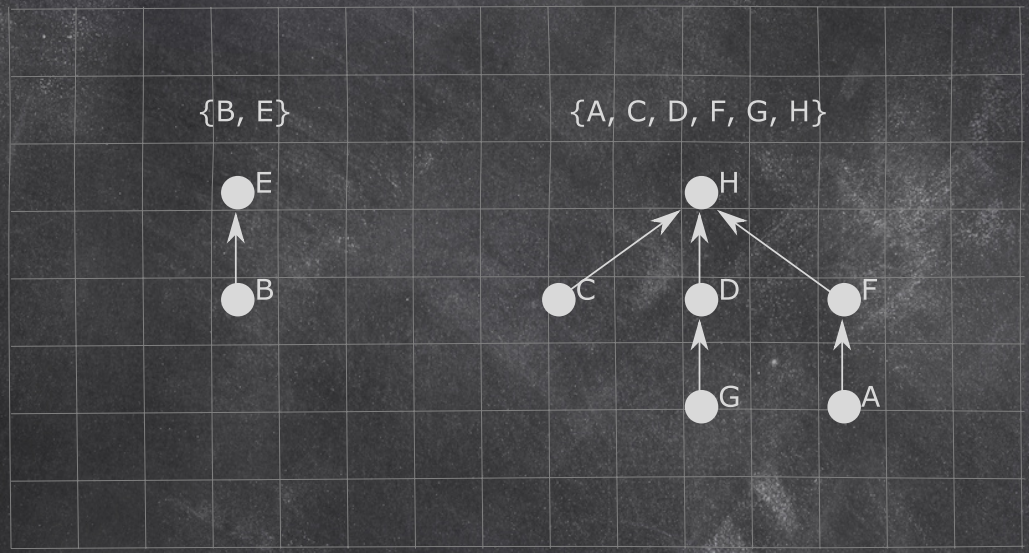
Retorne x

Complexidade: $O(h)$, onde h é a altura da árvore associada a x

Luís Felipe

21/09/23

Árvore



Note que a altura da árvore associada a um conjunto depende da operação de união de conjuntos.

Desejamos construir uma árvore de menor altura possível.

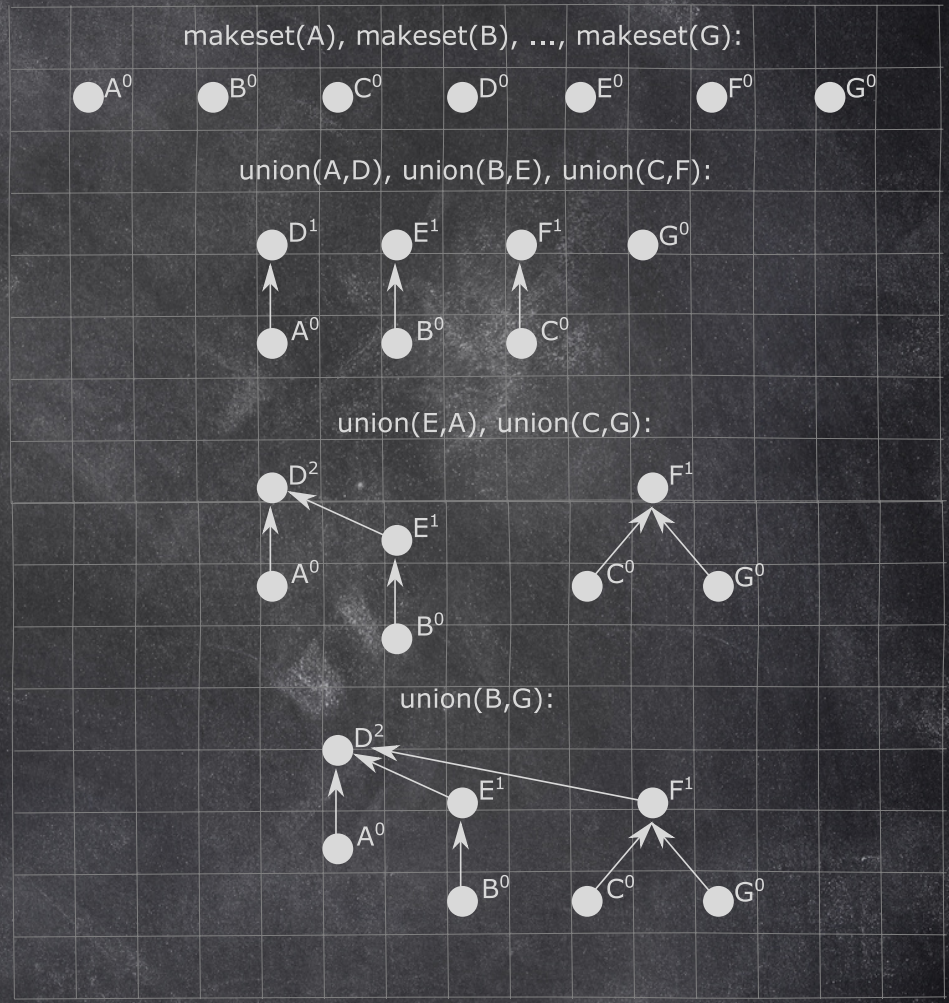
Unimos duas árvores tornando a raiz de uma delas a raiz da nova árvore.

Luís Felipe

21/09/23

União de duas árvores

union(x, y):
    $r_x = find(x)$
    $r_y = find(y)$
    Se $r_x = r_y$:
        Retorne r_x
    Se $rank(r_x) > rank(r_y)$:
        $\pi(r_y) = r_x$
    Senão:
        $\pi(r_x) = r_y$
        Se $rank(r_x) = rank(r_y)$:
            $rank(r_y) = rank(r_y) + 1$



$rank(x)$ está representado pelo índice sobrescrito do nó x.

Luís Felipe

21/09/23

Propriedade central

Note que para saber se dois elementos estão em um mesmo conjunto (que é o mesmo que dizer que dois vértices estão numa mesma componente conexa), devemos verificar se *find* dos dois vértices é igual ou não.

Isso implica percorrermos no pior caso a altura das árvores associadas para ter essa resposta.

- A altura da árvore é exatamente o rank(x) tal que x é a raiz da árvore.

  Por que? Consequência do algoritmo $union(x, y)$

Lema: O *rank* de qualquer árvore é no máximo $O(\log n)$.

Prova: Para que uma árvore passe a ter $rank = k$ pela primeira vez como resultado de uma união, temos pelo menos duas árvores de altura $k - 1$ cada.

Isso faz com que após a união uma subárvore da raiz tenha altura $k - 2$ e a outra com altura $k - 1$. Seja $|T_h|$ número de nós numa árvore de altura h. (continua...)

Luís Felipe

21/09/23

Assim, $|T_k| \geq 1 + |T_{k-1}| + |T_{k-2}|$, onde $|T_0| = 1$, $|T_1| \geq 2$.

Considerando $|F_k|$ o k-ésimo elemento da sequência de Fibonacci, temos: $|T_k| \geq |F_k|$.

Como $\frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^k < 1$, para $k > 0$, então:
$|F_k| = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^k - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^k > \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^k - 1$:

$|T_k| > \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^k - 1$, como a base da potenciação é uma constante, então:

$|T_k| > O(c^k)$.

Aplicando $\log_c$ em ambos os lados e fazendo mudança de base:
$\log_c |T_k| > k \ \ \therefore k < \frac{1}{\log_2 c} \log_2 |T_k| = O(\log |T_k|)$

Ou seja: $k < O(\log n)$

Luís Felipe

21/09/23

Concluindo análise do Kruskal

- Ordenação das arestas: $O(m \log m) = O(m \log n)$
- Para cada aresta verificamos se seus extremos estão numa mesma árvore parcial: $O(\log n)$.
  - Como fazemos isso para todas arestas no pior caso, então: $O(m \log n)$
- Ao todos, temos $O(m \log n) + O(m \log n) = O(m \log n)$

Luís Felipe

21/09/23

Algoritmo de Prim

- Tem o mesmo objetivo do algoritmo de Kruskal.
- Começa considerando apenas um vértice.
- Escolhe sempre a aresta de menor peso incidente àquele vértice que não gere ciclo.
- Diferentemente do algoritmo de Kruskal, que constrói florestas e depois as funde, em cada passo do algoritmo de Prim tem-se uma árvore.
- A corretude deste algoritmo também é justificada pela propriedade do corte.
- A complexidade do Algoritmo de Prim é $O(m \log n)$, se uma estrutura de dados adequada for utilizada.