Aula 9 - Algoritmo de Huffman

Luís Felipe

UFF

26 de Setembro de 2023

Luís Felipe
26/09/23

Compactação de dados

- Muitas vezes queremos armazenar um arquivo de grande porte e temos restrições de memória.
- Ou talvez queiramos transmitir um arquivo grande via rede e encontramos limitações.
- A codificação de um arquivo é uma saída para este tipo de problema.
  - Um código é criado e utilizado por algoritmos de codificação e decodificação
  - Se o arquivo codificado for menor que o original, a versão codificada implica em menos gasto de memória, por exemplo.
- Este problema é conhecido como o problema da compactação de dados

Luís Felipe

26/09/23

Ideia simples

- Suponham que queiramos compactar um texto alfabético, como por exemplo:

AAABCCDEEEEEEEFFFGCKKKKLLJO

- Um algoritmo simples para executar essa tarefa considera a frequência com que cada caracter aparece no arquivo e justapõe o algarismo que representa essa frequência e o respectivo caracter

3A1B2C1D7E3F1G1C4K2L1J1O

- Podemos admitir que sempre que um algarismo não é anteposto a um caracter, então a ocorrência do caracter é única.

3AB2CD7E3FGC4K2LJO

- Mas e se o arquivo contiver números? Isso traria ambiguidade a essa codificação.
    - Neste caso, um caracter alternativo pode ser justaposto ao algarismo para informar que se trata de um algarismo do texto e não uma frequência.

Luís Felipe

26/09/23

Algoritmo de Huffman

- Método para codificação de textos ótimo sob alguns critérios.
- O texto é composto por símbolos $S = \{s_1, s_2, \ldots, s_n\}$.
- Cada símbolo s_i ocorre com frequência $f_i, 1 \leq i \leq n$.
- O objetivo é atribuir um código a cada símbolo com a restrição de que nenhum código seja prefixo de outro.
- Ou seja, os códigos que procuramos são dados por uma árvore binária de prefixo.

Luís Felipe
26/09/23

Árvore binária de prefixo

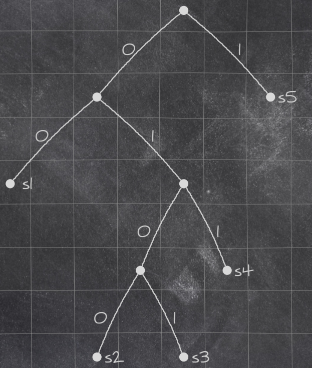
- Árvore estritamente binária
- Dado um nó interno, a aresta que conduz a seu filho esquerdo é rotulada com 0 e a que conduz ao seu filho direito é rotulada com 1.
- Cada símbolo está associado a uma folha, evitando que um código seja prefixo de outro.
- Os códigos são as sequências binárias formadas no percurso da raiz até a folha correspondente ao símbolo.

Luís Felipe
26/09/23

Árvore binária de prefixo

Exemplo:

símbolo	código
s1	00
s2	0100
s3	0101
s4	011
s5	1



Qual seria o código correspondente ao texto:

$s_4\ s_2\ s_5\ s_5\ s_1\ s_2\ s_3\ s_2\ s_4\ s_3\ s_2$?

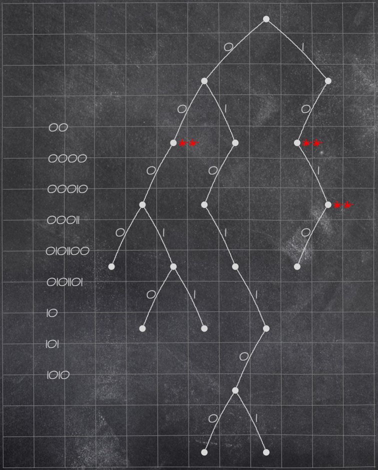
011 0100 1 1 00 0100 0101 0100 011 0101 0100

Luís Felipe
26/09/23

OBS.

Por que a árvore tem que ser de prefixo?

- Para não haver ambiguidade na decodificação.

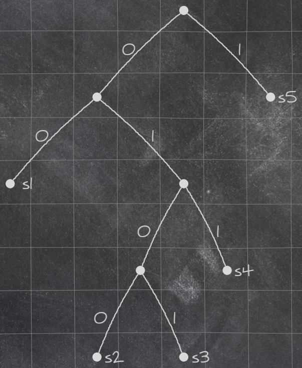


Luís Felipe
26/09/23

Árvore Binária de prefixo

Exemplo:

símbolo	código
s1	00
s2	0100
s3	0101
s4	011
s5	1



Qual seria o código correspondente ao texto:

$s_4\ s_2\ s_5\ s_5\ s_1\ s_2\ s_3\ s_2\ s_4\ s_3\ s_2$?

011 0100 1 1 00 0100 0101 0100 011 0101 0100

Luís Felipe
26/09/23

Algum problema?

- Qual o problema dessa última ávore de prefixo?
- Não estamos nos preocupando com o tamanho da sequência binária de codificação
- Um ponto importante é que s_2 é o maior código e é o símbolo que mais aparece na sequência
- Uma boa estratégia seria, portanto, construir uma árvore de codificação onde os símbolos mais frequentes correspondem aos menores códigos.
- O comprimento da sequência binária da codificação é chamado custo de T e denotado por $c(T)$.
- Assim, queremos construir uma árvore T tal que $c(T)$ seja mínimo.
- Uma árvore de prefixo que satisfaça tal condição é chamada de Árvore de Huffman.
- O custo da árvore do exemplo anterior é 34.

Luís Felipe

26/09/23

Custo da árvore de prefixo

- Considere que o símbolo s_i aparece em um texto com frequência $f_i, 1 \leq i \leq n$
- Se ℓ_i é o comprimento da codificação de s_i, note que s_i contribui com $\ell_i f_i$ para o custo de T.
- $c(T) = \sum_{i=1}^{n} \ell_i f_i$

Luís Felipe

26/09/23

Construção da árvore de Huffman

- Construção gulosa
- Das folhas para a raiz
  - ou seja, os códigos são determinados de trás para frente
- De modo geral, obtemos subcódigos para subconjuntos de símbolos
- Os subcódigos são subárvores
- O passo geral consiste da fusão de duas subárvores em uma.
- O processo termina quando temos uma única subárvore.

Luís Felipe
26/09/23

Construção da árvore de Huffman

- Seja T' uma subárvore binária de prefixo de T
- A frequência de T', $f(T')$ é a soma das frequências dos símbolos s_i que são folhas de T'.
- A construção inicia-se com n subárvores de um nó, que são apenas os símbolos s_i.
  - Frequência de cada subárvore é a frequência do símbolo s_i.
- A fusão, denotada por $\oplus$, ocorre entre duas subárvores T' e T'' de menor frequência.
- Esta operação cria um novo nó cujos filhos esquerdo e direito são as raízes de T' e T'', respectivamente.
- A frequência $f(T' \oplus T'')$ da nova subárvore é $f(T') + f(T'')$.
- O algoritmo termina quando temos uma única subárvore ($n - 1$ execuções da operação $\oplus$).

Luís Felipe
26/09/23

Algoritmo de Huffman

- Dadas as frequências $f_1, f_2, \ldots, f_n$
- Construa uma lista de prioridades com as frequências, de modo que menor frequência seja o de maior prioridade
- Cada elemento da lista corresponde a uma subárvore de um único nó
- A cada iteração, a lista é rearrumada de acordo com as prioridades
- No fim, a árvore de Huffman é a correspondente à que restou na lista de prioridades.
- Lembram da lista de prioridade por Heap?!

Luís Felipe
26/09/23

Heap

Heap é uma lista linear composta de elementos $s_1, \cdots, s_n$, satisfazendo: $s_i \geq s_{\lfloor \frac{i}{2} \rfloor}, i = 1, \cdots, n$

Cada elemento representa sua prioridade

Podemos associar cada heap a uma árvore binária satisfazendo as restrições apresentadas.

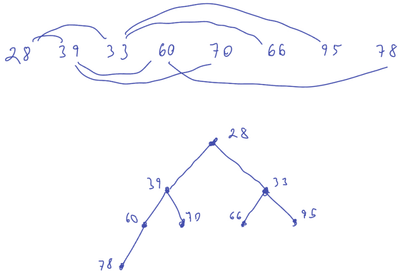
A prioridade $s_i \geq s_{\lfloor \frac{i}{2} \rfloor}, i = 1, \cdots, n$ da heap é equivalente a dizer que cada nó de T possui rótulo menor ou igual aos rótulos de seus filhos.

Toda Heap é associada a uma árvore binária completa, assim, altura é $O(\log n)$.

Luís Felipe

26/09/23

Exemplo



Acessar Maior Prioridade = Menor Frequência = Raiz da Heap

↳ Complexidade: O(1)

Luís Felipe
26/09/23

Operações

Inserção:

- Insira o novo elemento na última posição da lista (i.e. folha irmã da folha no último nível da árvore)
- Compare com a prioridade do pai e troque os elementos caso seja necessário. Faça essa subida enquanto for possível.

  Complexidade: Altura da árvore, ou seja, $O(\log n)$.

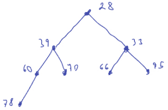
Remoção:

- Troque o elemento desejado x com o último elemento da lista y e remova x (i.e. x estará na folha do último nível mais a direita).
- Acerte a árvore resultante para manter propriedade da heap (i.e. como y foi para a posição do x, deve comparar sua prioridade com seus filhos, fazendo a troca com o filho de menor prioridade)

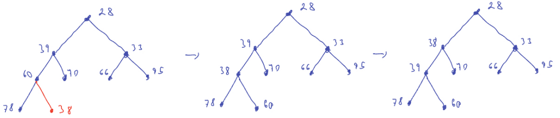
  Complexidade: Altura da árvore, ou seja, $O(\log n)$.

Luís Felipe

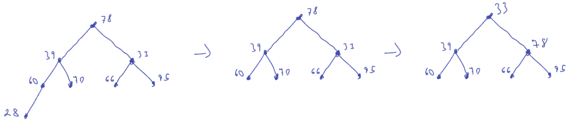
26/09/23

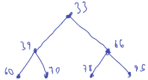


Inserção (38):



Remoção (28):





Luís Felipe
26/09/23

Algoritmo de Huffman

$minimo(F, T)$: Retira elemento de maior prioridade (menor frequência) da lista F e rearruma a lista F. Cada elemento é associado a uma árvore T.

$inserir(T, f, F)$: inclui elemento T de prioridade $f = f(T)$ na lista F

Para $i = 1, 2, \cdots, n-1$ faça:

$minimo(F, T')$
$minimo(F, T'')$
$T = T' \oplus T''$
$f(T) = f(T') + f(T'')$
$inserir(T, f, F)$

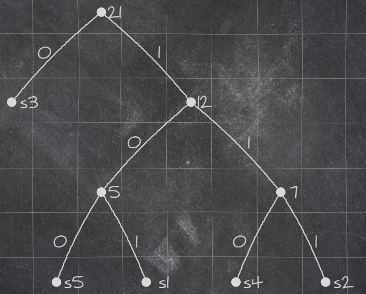
- Cada operação de *minimo* ou *inserir* na lista de prioridade pode ser efetuada em $O(\log n)$, herdada da heap.
- A operação $\oplus$ é constante. Como é executada $n-1$ vezes, a complexidade é $O(n \log n)$.

Luís Felipe

26/09/23

Exemplo

símbolo	frequência	código
s1	3	101
s2	4	111
s3	9	0
s4	3	110
s5	2	100



Luís Felipe
26/09/23

Custo mínimo??? Será?

Lema 1. Sejam s_i símbolos com frequências $f_i, 1 \leq i \leq n, n > 1$, tais que f_1 e f_2 são as duas menores frequências. Então existe uma árvore de Huffman para esses símbolos, em que os nós de s_1 e s_2 são irmãos no último nível da árvore.

Prova: Seja T um árvore de Huffman. Suponha que s_1 não é folha do último nível de T.

Sejam $s_k, k \neq 2$ uma folha do último nível de T e T' a árvore obtida a partir de T pela troca de s_k com s_1. Se $f_1 < f_k$, então $c(T') < c(T)$. Absurdo. Logo $f_1 \leq f_k$.

Como f_1 é a menor frequência, $f_1 = f_k$ e, assim, $c(T') = c(T)$ e T' é também árvore de Huffman, com s_1 no último nível.

Seja s_j irmão de s_1 em T'. Se $j \neq 2$, então troque s_2 com s_j, obtendo a árvore T''. Pelo mesmo motivo anterior, $f_2 \leq f_j$, donde conclui-se que $c(T') = c(T'')$.

Logo, T'' é árvore de Huffman em que s_1 e s_2 são irmãos e estão no último nível. □

Luís Felipe

26/09/23

Custo mínimo??? Será?

Teorema 2. Se T é saída do algoritmo de Huffman com frequências $f_1, f_2, \ldots, f_n, n > 1$, então T é mínima.

Prova: Suponha, sem perda de generalidade, que $f_1 \leq f_2 \leq \ldots f_n$. Seja T_m uma árvore mínima para essas frequências.

Vamos provar por indução sobre n que $c(T) = c(T_m)$. Pelo Lema 1, s_1 e s_2 são irmãos no último nível de T.

Base da Indução: $n = 2$. O resultado é trivial.

HI: Suponha que toda árvore de Huffman com até $n - 1$ frequências seja mínima.

Luís Felipe 24/09/13

Prova (Continuação):

Passo Indutivo: Na construção da árvore de Huffman, no primeiro passo fazemos a fusão de T_1 com T_2, árvores com apenas os nós s_1 e s_2, respectivamente. T_1 e T_2 são eliminadas e substituídas por uma árvore de custo $f_1 + f_2$.

Ou seja, após o primeiro passo, restam $n-1$ frequências no total e $n-2$ frequências de f_3 até f_n: $f_1 + f_2, f_3, \ldots, f_n$. Seja T' a árvore construída pelo algoritmo para essas $n-2$ frequências. Pela HI, T' é mínima, e $c(T) = c(T') + f_1 + f_2$.

Por outro lado, seja T'' a árvore obtida a partir de T_m eliminando-se as folhas irmãs correspondentes às frequências f_1, f_2 e associando-se ao pai delas um novo símbolo com frequência $f_1 + f_2$.

T'' é uma árvore binária de prefixo correspondente às $n-2$ frequências de f_3 até f_n em $f_1 + f_2, f_3, \ldots, f_n$. Além disso, $c(T_m) = c(T'') + f_1 + f_2$.

Observe que $c(T') \leq c(T'')$, pois T' é mínima. Logo, $c(T) = c(T_m)$ e, portanto, T é árvore mínima. $\square$