CI1238 - Otimização

Aula 10 - Algoritmos de Enumeração (Parte 2)

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

2026 / Primeiro Semestre

Enumeração por ordem de mudança mínima

- Aula anterior: Ordem lexicográfica
- Algoritmo simples, mas inconveniente para geração incremental dos subconjuntos.
- Por exemplo, considere $n = 3$:
  - subconjunto com rank 3: $\{2, 3\}$
  - subconjunto com rank 4: $\{1\}$
  - Estes dois conjuntos são bem diferentes.

    (note: são "tão diferentes quanto possível", pois são complementares)
- Ideia: Quantificar quão semelhantes são dois subconjuntos de S
- Na enumeração, fazer que subconjuntos consecutivos sejam semelhantes.
- Um tipo de ordenação conhecida como ordenação de mundança mínima.
  – ordem de mudança mínima também pode ser usada em ordenação de subconjuntos de tamanho fixo, partições, permutações, etc.
- Em particular, veremos ordenação por códigos de Gray.

  (existem diversas variações desta ordenação – veremos uma delas)

Definições auxiliares: Grafos Cubo

Um *n-cubo*, denotado $Q_n = (V_n, E_n)$, é um grafo com

- V_n = "conjunto de strings de n bits"
- E_n = se duas strings diferem por um bit, coloque aresta

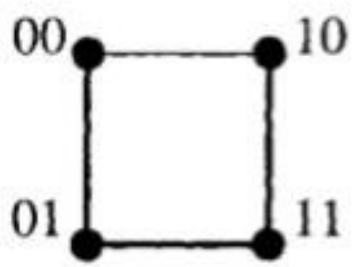
- Grafo Q_1 abaixo:

Definições auxiliares: Grafos

Um *n-cubo*, denotado $Q_n = (V_n, E_n)$, é um grafo com

- V_n = "conjunto de strings de n bits"
- E_n = se duas strings diferem por um bit, coloque aresta

- Grafo Q_2 abaixo:

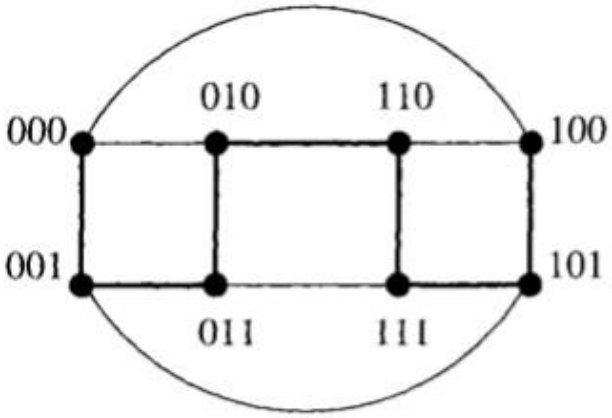


Definições auxiliares: Grafos

Um *n-cubo*, denotado $Q_n = (V_n, E_n)$, é um grafo com

- V_n = "conjunto de strings de n bits"
- E_n = se duas strings diferem por um bit, coloque aresta

- Grafo Q_3 abaixo:

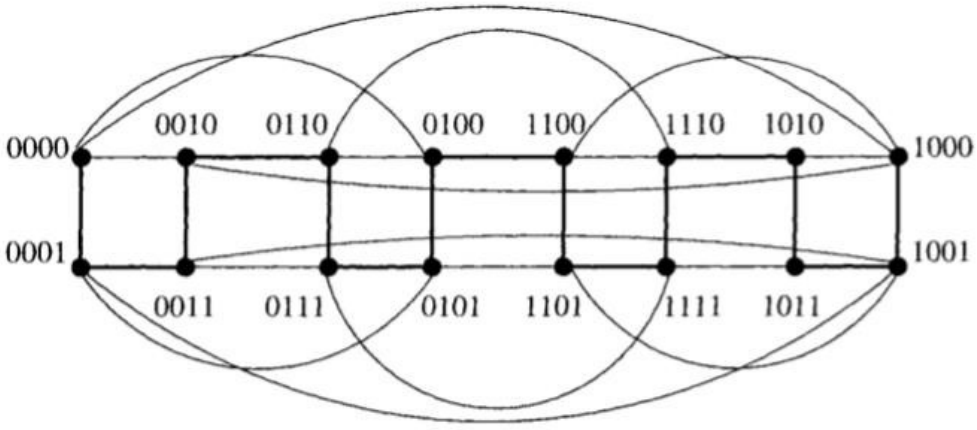


Definições auxiliares: Grafos

Um *n-cubo*, denotado $Q_n = (V_n, E_n)$, é um grafo com

- V_n = "conjunto de strings de n bits"
- E_n = se duas strings diferem por um bit, coloque aresta

- Grafo Q_4 abaixo:



Importante: Um caminho hamiltoniano em Q_n define uma ordenação de mudança mínima dos subconjuntos de $\{1, 2, ..., n\}$. – mais detalhes nos próximos slides

Definições auxiliares: Distância de Hamming

Dados $R, T \subseteq S$,

Diferença simétrica: a diferença simétrica de R e T é

$$R \Delta T = (R \setminus T) \cup (T \setminus R)$$

Distância: a distância entre os conjuntos R e T é denotada e definida abaixo:

$$dist(R, T) = |R \Delta T|$$

- Observe que $dist(R, T)$ é a quantidades de bits distintos que as strings binárias χ_R e χ_T possuem entre si.

  A medida $dist(R, T)$ é chamada de *Distância de Hamming* entre χ_R e χ_T.

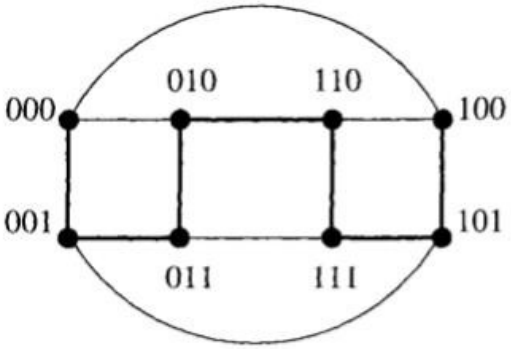
- Nosso objetivo: Gerar os subconjuntos $R_1, R_2, ..., R_{2^n}$ de S de forma que

  $$dist(R_i, R_{i+1}) = 1 \quad \text{(menor distância possível)}$$

  - Sabemos que um caminho hamiltoniano em Q_n define tal sequência
    $\Rightarrow$ não vamos explicitamente construir o grafo e encontrar o caminho
    $\Rightarrow$ mas tais ideias ajudam o entendimento do algoritmo de enumeração

Relação de Códigos de Gray com Hipercubos

Considere o grafo Q_3:



Considere o seguinte caminho hamiltoniano no grafo Q_3:

$$[000, 001, 011, 010, 110, 111, 101, 100]$$

- Esta sequência de strings conhecida como um tipo de *Código de Gray*
- Código de Gray para strings de tamanho n: caminho hamiltoniano no grafo Q_n
- Obs: Aqui vamos nos ater à um caso particular de Código de Gray, chamado de Código de Gray Binário Refletito

Definição formal de Códigos de Gray

O Código de Gray para strings de n bits é a lista de strings $G^n = [G_0^n, ..., G_{2^n-1}^n]$, sendo

- Base: $G^1 = [0, 1]$
- Caso geral:
  - Dado G^{n-1}, definimos G^n como:
  - $[0G_0^{n-1}, ..., 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, ..., 1G_0^{n-1}]$

Exemplos:

Para $n = 1$: $G^1 = [0, 1]$

Para $n = 2$: $G^2 = [00, 01, 11, 10]$

Definição formal de Códigos de Gray

O Código de Gray para strings de n bits é a lista de strings $G^n = [G_0^n, ..., G_{2^n-1}^n]$, sendo

- Base: $G^1 = [0, 1]$
- Caso geral:
  - Dado G^{n-1}, definimos G^n como:
  - $[0G_0^{n-1}, ..., 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, ..., 1G_0^{n-1}]$

Exemplos:

Para $n = 1$: $G^1 = [0, 1]$

Para $n = 2$: $G^2 = [00, 01, 11, 10]$

Definição formal de Códigos de Gray

O Código de Gray para strings de n bits é a lista de strings $G^n = [G_0^n, ..., G_{2^n-1}^n]$, sendo

- Base: $G^1 = [0, 1]$
- Caso geral:
  - Dado G^{n-1}, definimos G^n como:
  - $[0G_0^{n-1}, ..., 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, ..., 1G_0^{n-1}]$

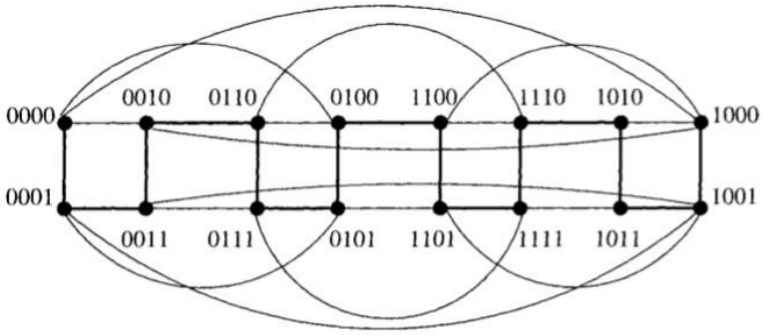
Exemplos:

Para $n = 1$: $G^2 = [0, 1]$

Para $n = 2$: $G^2 = [00, 01, 11, 10]$

Para $n = 3$: $G^3 = [000, 001, 011, 010, 110, 111, 101, 100]$

Para $n = 4$: $G^4 = [0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,$
$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000]$



Definição formal de Códigos de Gray

O Código de Gray para strings de n bits é a lista $G^n = [G_0^n, ..., G_{2^n-1}^n]$ definida:

- Base: $G^1 = [0, 1]$
- Caso geral:
  - Dado G^{n-1}, definimos G^n como:
  - $[0G_0^{n-1}, ..., 0G_{2^{n-1}-1}^{n-1}, G_{2^{n-1}-1}^{n-1}, ..., G_0^{n-1}]$

Agora vamos apresentar uma definição recursiva para a i-ésima string de G^n :

$$G_i^n = \begin{cases} 0G_i^{n-1} & \text{se } 0 \leq i \leq 2^{n-1} - 1, \\ 1G_{2^n-1-i}^{n-1} & \text{se } 2^{n-1} \leq i \leq 2^n - 1. \end{cases}$$

- **Exercício:** Apresente um algoritmo recursivo para a função *unrank* baseado na definição acima.
- Para apresentarmos nosso algoritmo de enumeração, que será iterativo, nosso objetivo nos slides a seguir é definir a função *sucessor* para Códigos de Gray
- Livro texto: apresentada também *rank* e *unrank* (não vamos abordar aqui).

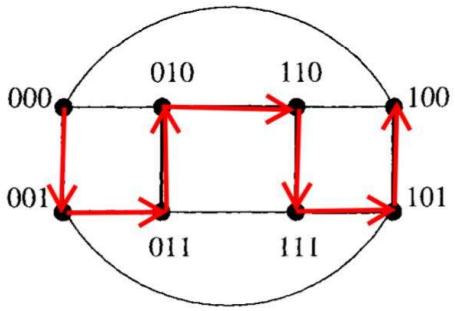
Algoritmo iterativo para gerar Código de Gray

Seja $T \in \mathcal{P}(\mathcal{S})$ e considere seu vetor característico $\chi_T = [x_{n-1}, ..., x_1, x_0]$.

- O Peso de Hamming de χ_T é dado por $w(\chi_T) = \sum_{i=0}^{n-1} x_i$
  - Note: Isso é quantidade de bits 1 de χ_T
  - Note também que $w(\chi_T) = |T|$
- Fato: Qualquer que seja o algoritmo *sucessor*(), obrigatoriamente o sucessor de T deve ser um conjunto T' tal que
  - $\chi_{T'}$ seja obtido de χ_T por meio de um *bit flip*.
  - A questão chave é saber em qual bit é feito o flip:
- *sucessor*(χ_T):
  - Se χ_T tem quantidade par de bits 1, faça um bit flip em x_0
  - Se χ_T tem quantidade ímpar de bits 1, faça bit flip no bit à esquerda do bit 1 menos significativo
- A última string da sequência é 10...0 (que corresponde ao conjunto $\{1\}$).

Olhando *sucessor*() no Hipercubo

Considere o exemplo do grafo Q_3 abaixo

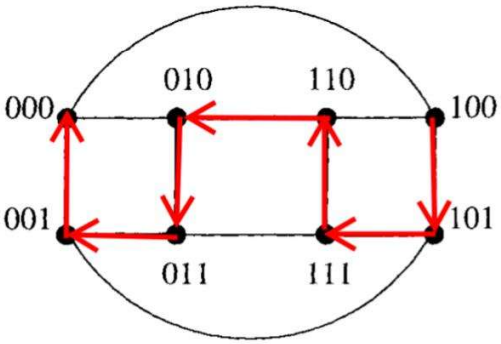


com o caminho $P = [000, 001, 011, 010, 110, 111, 101, 100]$

- Quando o caminho P anda na vertical (subindo ou descendo):
  - saímos de uma string $x = [x_n...x_0]$ com número par de 1's
  - com destino à uma string obtida ao fazermos o flip de x_0 (e.g., aresta de 000 para 001 ou aresta de 011 para 010)
- Quando caminho anda na horizontal (i.e., para a direita),
  - saímos de uma string com com número ímpar de 1's
  - destino: string obtida por flip à esquerda do bit 1 menos significativo

Olhando *sucessor*() no Hipercubo

Agora olhando o caminho reverso:

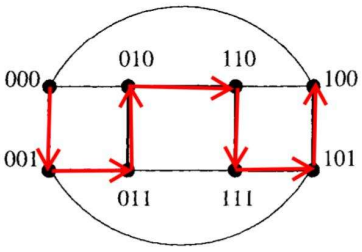


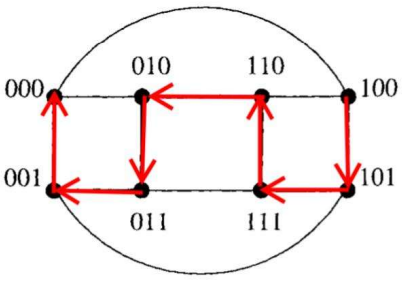
com o caminho $P^{-1} = [100, 101, 111, 110, 010, 011, 001, 000]$

- Quando o caminho P^{-1} anda na vertical (subindo ou descendo):
  - saímos de uma string $x = [x_n...x_0]$ com número ímpar de 1's
  - com destino à uma string obtida ao fazermos o flip de x_0 (e.g., aresta de 000 para 001 ou aresta de 011 para 010)
- Quando caminho anda na horizontal (agora para a esquerda),
  - saímos de uma string com com número **par** de 1's **(única diferença em relação à P)**
  - destino: string obtida por flip à esquerda do bit 1 menos significativo

Olhando *sucessor*() no Hipercubo

Obtendo Q_4 a partir de duas cópias do Q_3:





- Como obtemos o grafo Q_4 a partir de Q_3?
  – Juntamos o Q_3 com uma cópia refletida do Q_3;
  – Adicionamos bit mais significativo 0 na cópia original;
  – Adicionamos bit mais significativo 1 na cópia refletida
  (note, a única diferença de P^{-1} em relação à P deixa de existir)
- Nesta construção o caminho hamiltoniano composto em Q_4 tem as mesmas propriedades dos caminhos de Q_3

Códigos de Gray: Função *sucessor*()

De alguns slides atrás, a ideia do Algoritmo *sucessor*() é:

- Se A tem quantidade par de bits 1, flip ocorre em a_0
- Se A tem quantidade ímpar de bits 1, flip ocorre no bit a esquerda do bit 1 menos significativo

Teorema: Dado T, o algoritmo acima computa corretamente o sucessor de χ_T na ordenação por *Código de Gray Binário Refletido*.

Prova: Opcional (por indução no tamanho n das strings seguindo a ideia da construção dos caminhos hamiltonianos)

Algoritmo de Enumeração

sucessor(χ_T)

- Se χ_T tem quantidade par de bits 1, faça um bit flip em x_0
- Se χ_T tem quantidade ímpar de bits 1, faça bit flip no bit à esquerda do bit 1 menos significativo

Em termos de conjuntos, temos:

Algorithm 2.3: GRAYCODESUCCESSOR (n, T)

if $|T|$ is even
  then $U \leftarrow T\Delta\{n\}$
  else
    $j \leftarrow n$
    while $j \notin T$ **and** $j > 0$
      do $j \leftarrow j - 1$
    if $j = 1$
      then return ("undefined")
    $U \leftarrow T\Delta\{j - 1\}$
    return (U)

Algoritmo de enumeração:

$T = \varnothing$
while $T \neq \{1\}$ **do**
  Print(T);
  T = *sucessor*(T);
end