CI1238 - Otimização

Aula 08 - Enumeração

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

21/03/2023

Algoritmos Combinatórios

- Lidam com estruturas combinatórias: conjuntos (discretos), partições, permutações, grafos, etc
- Tópico da aula de hoje: algoritmos de enumeração
- Atentar para vocabulário:
  - **Contagem**: Determinar quantas estruturas de um certo tipo existem

    Exemplo: Dado conjunto S de tamanho n, **quantos** subconjuntos $S' \subseteq S$ de tamanho k existem? **resp:** $\binom{n}{k}$.

    No livro: *enumeration*.
  - Enumeração: obter cada uma das estruturas combinatórias em questão.

    Exemplo: Dado conjunto S de tamanho n, gerar cada um dos subconjuntos $S' \subseteq S$ tal que $|S'| = k$.

    No livro: *generation*.

    - Algoritmos de Busca/Otimização podem usar enumeração.
    - Algoritmos de Enumeração também são interessantes por si mesmos

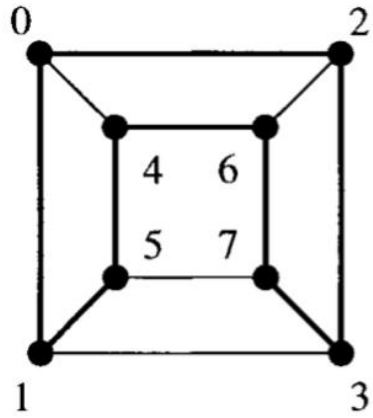
Definições auxiliares: Conjuntos, listas e permutações

Notação e exemplos:

- Conjuntos: $A = \{1, 5, 9, 7\}$
- Notação de Lista: $A = [1, 5, 9, 7, 1]$ (existe ordem e possibilidade de repetição)
  Usamos a notação $A[1] = 1$, $A[2] = 5$, $A[3] = 9$, ... (semelhante a vetor)
- Uma permutação π de um conjunto S de tamanho n é
  - uma lista com n elementos de S sem repetição

Definições auxiliares: Grafos

Considere o grafo abaixo:



Tanto um circuito hamiltoniano quanto um caminho hamiltoniano em um grafo é uma permutação de $V(G)$.

- Caminho hamiltoniano: $[0, 1, 5, 4, 6, 2, 3, 7]$
- Circuito hamiltoniano: $[0, 1, 3, 2, 6, 7, 5, 4]$

  Obs: se existe um circuito hamiltoniano, então existe um caminho hamiltoniano

  (a volta não não necessariamente é verdadeira)

Definições auxiliares: Grafos

Grafo cubo com n vértices (denotado G_n):

- $V(G_n) =$ "strings de n bits"
- Ligue vértices que diferem por um bit

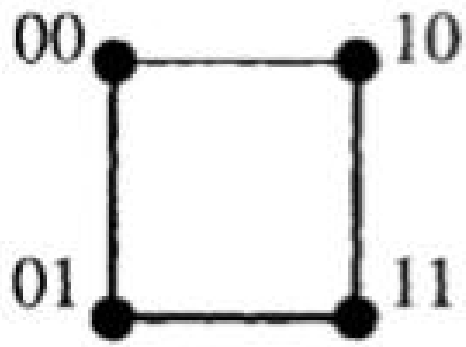
- Grafo G_1 abaixo:

Definições auxiliares: Grafos

Grafo cubo com n vértices (denotado G_n):

- $V(G_n) =$ "strings de n bits"
- Ligue vértices que diferem por um bit

- Grafo G_2 abaixo:

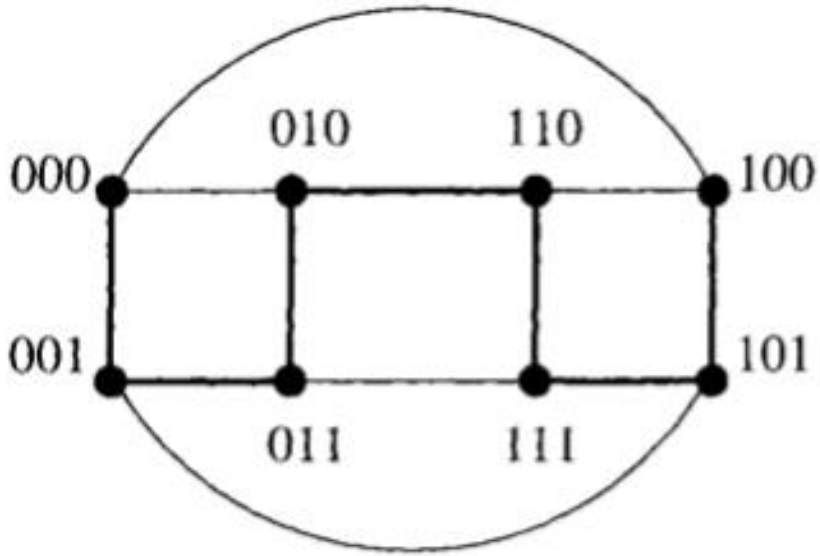


Definições auxiliares: Grafos

Grafo cubo com n vértices (denotado G_n):

- $V(G_n) =$ "strings de n bits"
- Ligue vértices que diferem por um bit

- Grafo G_3 abaixo:

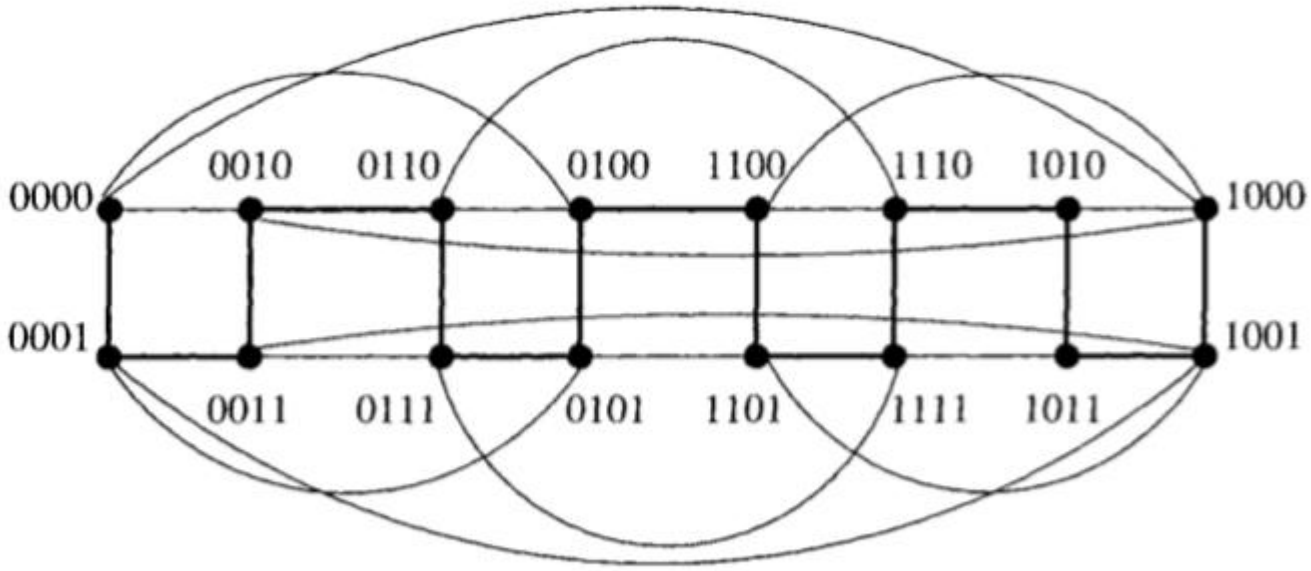


Definições auxiliares: Grafos

Grafo cubo com n vértices (denotado G_n):

- $V(G_n) =$ "strings de n bits"
- Ligue vértices que diferem por um bit

- Grafo G_4 abaixo:



Algoritmos Combinatórios: Enumeração

Seja A um conjunto de objetos combinatórios que queremos enumerar. Seja $|A| = N$.

- Para enumerar objetos, vamos estabelecer uma ordem destes.
- Vamos estabelecer uma função ***rank***()

A função *rank*() é uma função tal que:

- $rank : A \rightarrow \{0, ..., N-1\}$

Uma vez definida a função acima, temos também:

- $unrank : \{0, ..., N-1\} \rightarrow A$
- $successor : A \rightarrow A$

Algoritmos Combinatórios: Enumeração

Do slide anterior:

- $rank : A \to \{0, ..., N-1\}$

Para *unrank*(), a ideia é que $\forall a \in A$ e $\forall i \in \{0, ..., N-1\}$, temos:

$$rank(a) = i \Leftrightarrow unrank(i) = a$$

Para *sucessor*(), queremos:

$$sucessor(a) = b \Leftrightarrow rank(b) = rank(a) + 1$$

Podemos definir *sucessor*() em termos de *rank*() e *unrank*():

$$\text{sucessor(s)} = \begin{cases} unrank(rank(a)+1), & \text{se } rank(a) < N-1, \\ indefinida, & \text{se } rank(a) = N-1. \end{cases}$$

Note: aqui definimos primeiro *rank*() e *unrank*() e depois *sucessor*()

- Em alguns contextos é conveniente fazer o inverso: definimos primeiro *sucessor*() e depois definimos *rank*() e *unrank*() a partir disso.

Enumeração: Ranking usando Ordem Lexicográfica

Objetos que vamos enumerar: subconjuntos do conjunto $S = \{1, ..., n\}$

- Conjunto de objetos que queremos enumerar $A = \mathcal{P}(S)$
- Ordem usada: lexicográfica
- Mais para frente veremos outras ordens
  (mais precisamente, um exemplo de ordem de mudança mínima)

Dado $T \in \mathcal{P}(\mathcal{S})$, o vetor característico de T é $\chi(T) = [x_{n-1}, ..., x_0]$, onde

$$x_i = \begin{cases} 1 & \text{se } n - i \in T, \\ 0 & \text{se } n - i \notin T. \end{cases}$$

Exemplo: Seja $S = \{1, 2, 3\}$

- Se $T = \{2, 3\}$, então $\chi(T) =$???
  $i = 2$: $3 - 2 = 1 \notin T \therefore x_2 = 0$
  $i = 1$: $3 - 1 = 2 \in T \therefore x_1 = 1$
  $i = 0$: $3 - 1 = 2 \in T \therefore x_0 = 1$
  Com isso $\chi(T) = 011$
- Outro exemplo: Se $T = \{1\}$, então $\chi(T) = 100$

Ideia: $rank(T)$ é inteiro cuja representação binária é $\chi(T)$.

$$rank(T) = \sum_{i=0}^{n-1} x_i 2^i$$

Algoritmos Combinatórios: Enumeração

Considere $n = 3$ e $S = \{1, 2, 3\}$

T	$\chi(T) = [x_2, x_1, x_0]$	rank(T)
$\emptyset$	$[0, 0, 0]$	0
$\{3\}$	$[0, 0, 1]$	1
$\{2\}$	$[0, 1, 0]$	2
$\{2, 3\}$	$[0, 1, 1]$	3
$\{1\}$	$[1, 0, 0]$	4
$\{1, 3\}$	$[1, 0, 1]$	5
$\{1, 2\}$	$[1, 1, 0]$	6
$\{1, 2, 3\}$	$[1, 1, 1]$	7

Algoritmos Combinatórios: Enumeração

Algoritmos *rank*() e *unrank*()

Algorithm 2.1: SUBSETLEXRANK (n, T)

$r \leftarrow 0$
for $i \leftarrow 1$ **to** n
  do { **if** $i \in T$
    then $r \leftarrow r + 2^{n-i}$
return (r)

Algorithm 2.2: SUBSETLEXUNRANK (n, r)

$T \leftarrow \emptyset$
for $i \leftarrow n$ **downto** 1
  do { **if** $r \bmod 2 = 1$
    then $T \leftarrow T \cup \{i\}$
    $r \leftarrow \lfloor \frac{r}{2} \rfloor$
return (T)

Algoritmos Combinatórios: Enumeração

Algorithm 2.1: SUBSETLEXRANK (n, T)

$r \leftarrow 0$
for $i \leftarrow 1$ **to** n
  do { **if** $i \in T$
    then $r \leftarrow r + 2^{n-i}$
return (r)

Exemplo: $n = 8$ e $T = \{1, 3, 4, 6\}$:

$$\begin{aligned} \mathsf{rank}(T) &= 2^7 + 2^5 + 2^4 + 2^2 \\ &= 128 + 32 + 16 + 4 \\ &= 180. \end{aligned}$$

Algoritmos Combinatórios: Enumeração

Algorithm 2.2: SUBSETLEXUNRANK (n, r)

$T \leftarrow \emptyset$
for $i \leftarrow n$ **downto** 1
do {
  if $r \bmod 2 = 1$
  then $T \leftarrow T \cup \{i\}$
  $r \leftarrow \lfloor \frac{r}{2} \rfloor$
}
return (T)

Exemplo: $n = 8$ e $r = 180$:

i	r	$r \bmod 2$	T
8	180	0	$\emptyset$
7	90	0	$\emptyset$
6	45	1	$\{6\}$
5	22	0	$\{6\}$
4	11	1	$\{4,6\}$
3	5	1	$\{3,4,6\}$
2	2	0	$\{3,4,6\}$
1	1	1	$\{1,3,4,6\}$.

Algoritmos Combinatórios: Enumeração

Relembrando: agora que temos uma função *unrank*(), para enumerar todos os conjuntos em ordem lexicográrica fazemos:

for $i = 0$ *to* $N - 1$ **do**
  | $unrank(i)$;
end

Ok, mas se tivermos um conjunto S' diferente de $S = \{1, 2, ..., n\}$?

- Obtenha uma bijeção $S \leftrightarrow S'$ (qualquer uma serve)
- Isso induz uma bijeção $\phi : \mathcal{P}(S) \leftrightarrow \mathcal{P}(S')$
- Agora seja $rank_S$ e $unrank_S$ as funções para elementos de $\mathcal{P}(S)$
- Então $\forall X \in \mathcal{P}(S')$

$$rank(X) = rank_S(\phi(X))$$

- Dado $r \in \mathbb{N}$

$$unrank(r) = \phi^{-1}(unrank_S(r))$$

Enumeração: Ranking usando Códigos de Gray

- Ordem lexicográfica é simples
- Entretanto, não é conveniente 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 (de fato, "tão diferentes quanto possível", pois são complementares)
- Ideia: Quantificar quão semelhantes são dois subconjuntos de S
- Na enumeração, fazer que subconjuntos consecutivos sejam semelhantes

Enumeração: Ranking usando Códigos de Gray

Dados $T_1, T_2 \subseteq S$,

Diferença simétrica: a diferença simétrica de T_1 e T_2 é

$$T_1 \Delta T_2 = (T_1 \setminus T_2) \cup (T_2 \setminus T_1)$$

Distância: a distância entre T_1 e T_2 é

$$dist(T_1, T_2) = |T_1 \Delta T_2|$$

- Observe que $dist(T_1, T_2)$ é a quantidades de componentes distintas que os vetores $\chi(T_1)$ e $\chi(T_2)$ possuem.
- Em outros contextos essa é a *Distância de Hamming* entre $\chi(T_1)$ e $\chi(T_2)$.
- Ideia: Gerar os subconjuntos $T_1, T_2, ..., T_{2^n}$ de S de forma que

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

- Ordenação por códigos de Gray: Um tipo de ordenação de mundança mínima.

Enumeração: Ranking usando Códigos de Gray

Para $n = 3$, considere a seguinte ordenação de mudança mínima:

$$\emptyset, \{3\}, \{2,3\}, \{2\}, \{1,2\}, \{1,2,3\}, \{1,3\}, \{1\}$$

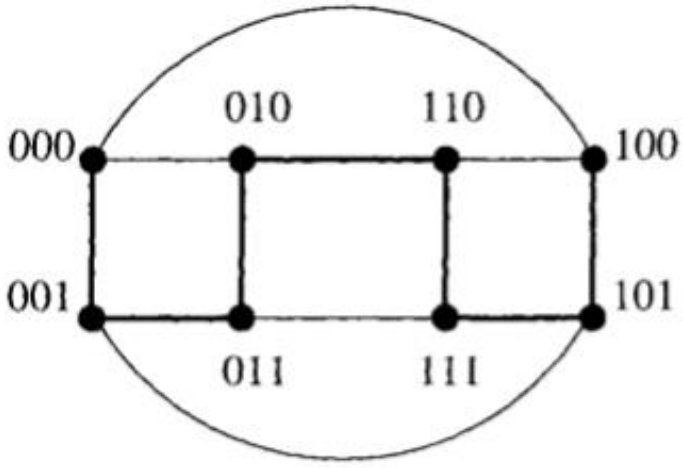
Considere agora as strings dos vetores característicos dos conjuntos:

$$000,\ 001,\ 011,\ 010,\ 110,\ 111,\ 101,\ 100$$

- A sequência de strings acima é um exemplo de Código de Gray.
- Veremos a seguir uma definição recursiva para sequências como essa
- Mas antes, observe a relação disso com hipercubos

Relação de Códigos de Gray com Hipercubos

Considere o grafo Q_3:



O Código de Gray do slide anterior é um caminho hamiltoniano neste grafo:

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

- Código de Gray para strings de tamanho n: caminho hamiltoniano no grafo G_n
- Este caso particular de Código de Gray: Código de Gray Binário Refletito

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:

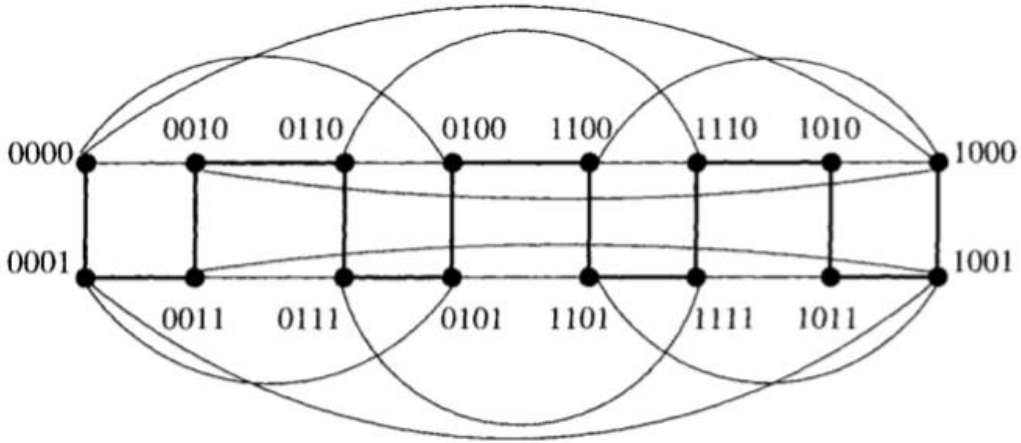
- Base: $G^1 = [0, 1]$
- Caso geral:
  - Dado G^{n-1}, definimos G^n como:
  - $[0G_0^{n-1}, ..., G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, ..., 1G_0^{n-1}]$

Exemplos:

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

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

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



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:
  - $[0G_0^{n-1}, ..., G_{2^{n-1}-1}^{n-1}, G_{2^{n-1}-1}^{n-1}, ..., G_0^{n-1}]$

Dando a definição recursiva de maneira explícita:

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

Para Códigos de Gray vamos:

- Primeiro definir a função *sucessor*
- Depois definir a função *rank* e *unrank*.

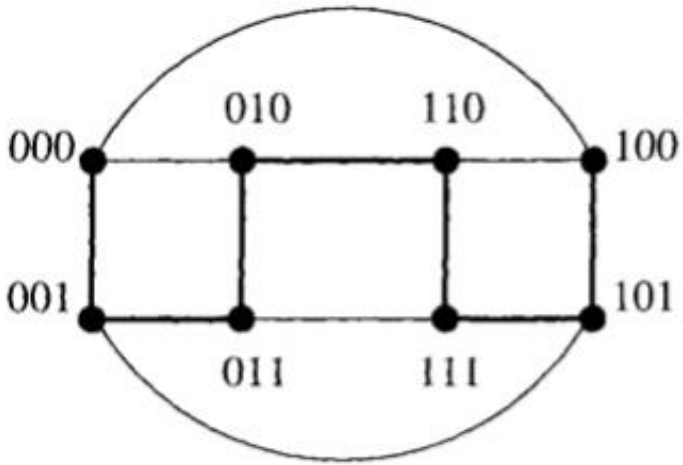
Códigos de Gray

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

- O Peso de Hamming do vetor A é dado por $w(A) = \sum_{i=0}^{n-1} a_i$
  - Note: Isso é quantidade de bits 1 do vetor
  - Note também que $w(A) = |T|$
- Note: Qualquer que seja o algoritmo *sucessor*(), o sucessor de A deve ser um vetor A' produzido a partir de um *bit flip* em A.
- Ideia do Algoritmo *sucessor*():
  - 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
- O último vetor da sequência é $[1, 0, ..., 0]$ (que corresponde ao conjunto $\{1\}$).

Olhando *sucessor*() no Hipercubo

Grafo Q_3:



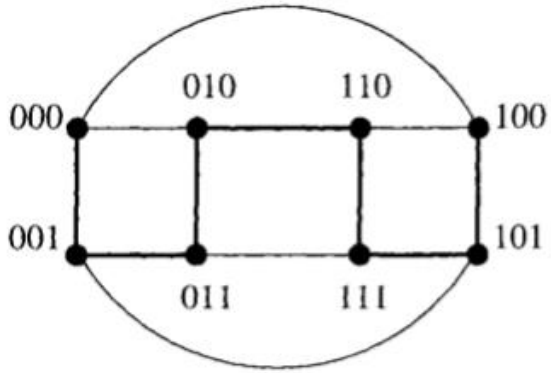
O caminho hamiltoniano induzido pelo Código de Gray é:

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

- Note: Quando o caminho P anda na vertical (subindo ou descendo), saímos de uma string com $w(A)$ par com destino à uma string obtida ao fazermos o flip de a_0 em A (e.g., o trecho de 000 para 001, ou o trecho de 011 para 010, etc)
- Quando caminho anda na horizontal (note: na horizontal P sempre segue a direita), saímos de uma string com $w(A)$ ímpar e o destino é obtido ao fazermos o flip a esquerda do bit 1 menos significativo

Olhando *sucessor*() no Hipercubo

Grafo Q_3:

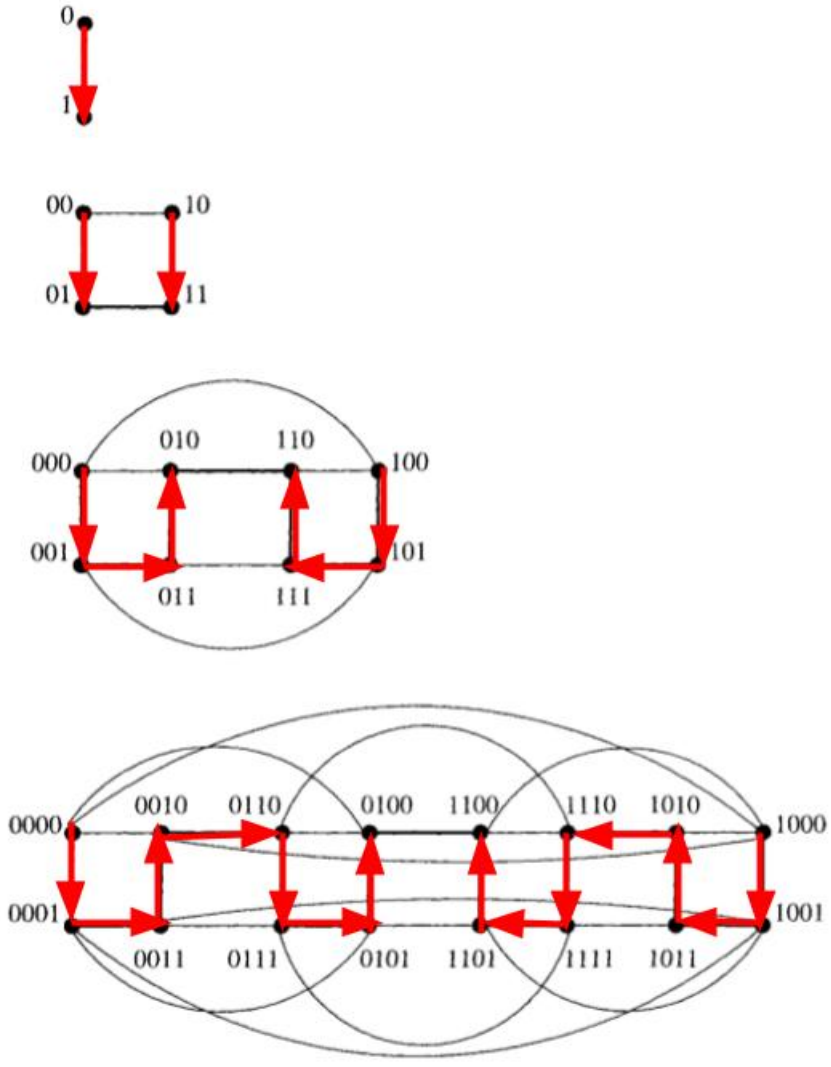


Agora o caminho de trás para frente: $P^{-1} = [100, 101, 111, 110, 010, 011, 001, 000]$

- Na vertical (subindo ou descendo) temos a mesma coisa: saímos de uma string com $w(A)$ par e o destino é obtido ao fazermos o flip de a_0 em A
- Na horizontal (agora andamos para a esquerda): saímos de string com $w(A)$ ímpar com destino a string com $w(A)$ par obtida ao fazermos o flip a esquerda do bit 1 menos significativo
- Como obtemos o grafo Q_4 a partir de Q_3?
  – Obtemos uma cópia refletida do Q_3;
  – Adicionamos bit mais significativo 0 na cópia original e 1 na cópia refletida.
- A ideia: Obter novo caminho hamiltoniano com as propriedades do slide anterior

Olhando *sucessor*() no Hipercubo

Grafos $Q_1, ..., Q_4$:



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

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)

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

Prova: Opcional (por indução no tamanho n das strings)