Algoritmos Combinatórios: Introdução

Lucia Moura
lucia@site.uottawa.ca

UFSC, Fevereiro, 2010

Introdução a Algoritmos Combinatórios

O que são:

- Estruturas Combinatórias?
- Algoritmos Combinatórios?
- Problemas Combinatórios?

Objetos Combinatórios Básicos I

- **Conjuntos finitos e seus subconjuntos**

  Exemplo: $X = \{1, 2, 3, 5\}$
  - estrutura não ordenada, sem repetições
    $\{1, 2, 3, 5\} = \{2, 1, 5, 3\} = \{2, 1, 1, 5, 3\}$
  - cardinalidade (tamanho) de um conjunto é o número de elementos do conjunto
    $|X| = 4$.
  - um k-conjunto e' um conjunto de cardinalidade k.
    X é um 4-conjunto.
  - Um k-subconjunto de um conjunto finito X é um k-conjunto S com $S \subseteq X$.
    Exemplo: $\{2, 5\}$ é um 2-subconjunto de X.

Objetos Combinatórios Básicos II

- **Lista finita ou tupla finita**
  $L = [1, 5, 2, 1, 3]$
  - estrutura ordenada, repetições permitidas
    $[1, 5, 2, 1, 3] \neq [1, 1, 2, 3, 5] \neq [1, 2, 3, 5]$
  - comprimento = número of itens, o comprimento de L é 5.

  Um n-tupla e' uma lista/tupla de comprimento n.

- **Permutações**
  Uma permutação de um n-conjunto X é um lista de comprimento n tal que todo elemento de X aparece exatamente uma vez.

$$X = \{1, 2, 3\}$$

duas permutações de X: $\pi_1 = [2, 1, 3]$ $\pi_2 = [3, 1, 2]$

Estruturas Combinatórias

Estruturas combinatórias são *coleções* de k-subconjuntos/k-tuplas/permutações de um conjunto (finito) base.

- **Grafos não direcionados:**
  Coleções de 2-subconjuntos (arestas) de um conjunto base (vértices).

$$V = \{1, 2, 3, 4\} \quad A = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\}$$

- **Grafos direcionados:**
  Coleções de 2-tuplas (arestas direcionadas) de um conjunto base (vértices).

$$V = \{1, 2, 3, 4\} \quad A = \{(2, 1), (3, 1), (1, 4), (3, 4)\}$$

- **Hipergrafos ou Sistemas de Conjuntos:**
  Similar a grafos, mas hiper-arestas são conjuntos com arestas não necessariamente com 2 elementos.

$$V = \{1, 2, 3, 4\} \quad E = \{\{1, 3\}, \{1, 2, 4\}, \{3, 4\}\}$$

Grafos

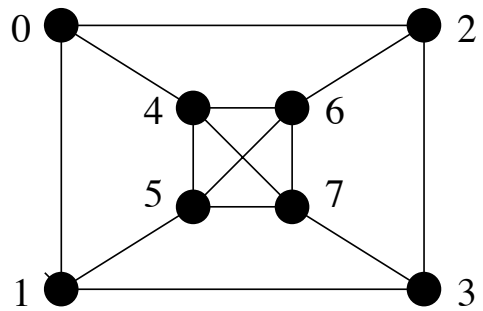
Definição

Um grafo é um par (V, A) onde:
V é um conjunto finito (seus elementos são chamados vértices).
A é um conjunto de 2-subconjuntos (chamados arestas) de V.

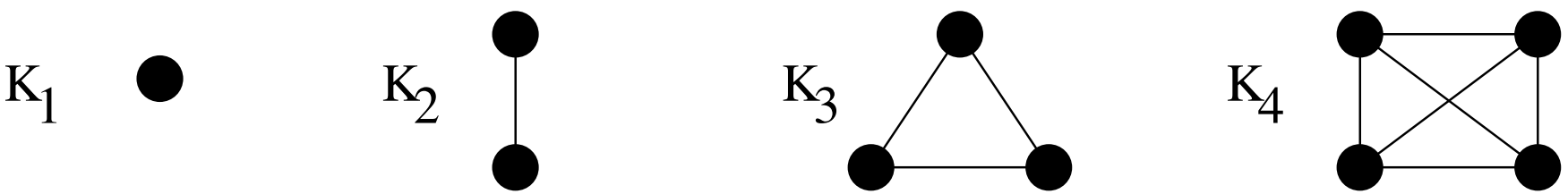
$G_1 = (V, A)$

$V = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$A = \{\{0,4\},\{0,1\},\{0,2\},\{2,3\},\{2,6\},$
$\{1,3\},\{1,5\},\{3,7\},\{4,5\},\{4,6\},$
$\{4,7\},\{5,6\},\{5,7\},\{6,7\}\}$



Grafos completos são grafos com todas as arestas possíveis.
Denotamos o grafo completo com n vértices por K_n.

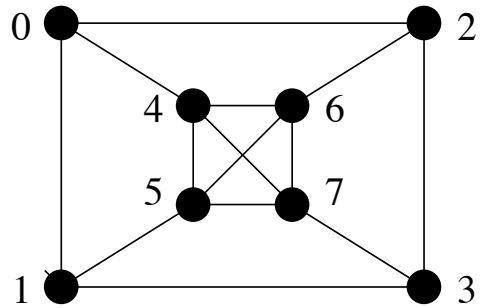


Subestruturas de um grafo: circuito hamiltoniano

Definição

Um circuito hamiltoniano é um caminho fechado que passa por cada vértice exatamente uma vez.

A lista $[0, 1, 5, 4, 6, 7, 3, 2]$ descreve um circuito hamiltoniano no grafo:
(note que várias listas podem descrever o mesmo circuito hamiltoniano.)



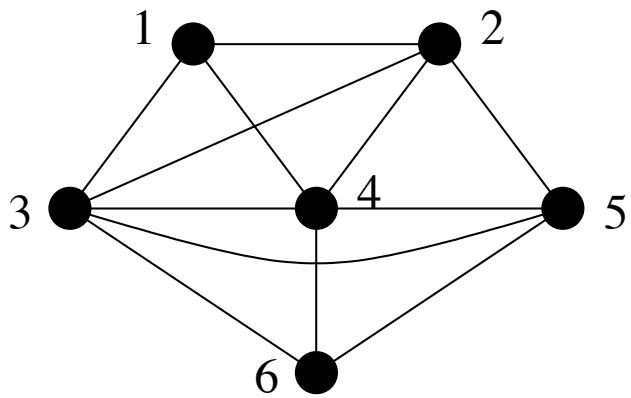
Problema (Problema do caixeiro viajante)

Dados pesos/custos nas arestas de $G = (V, A)$, isto é, uma função $w : A \to R$, encontre o circuito hamiltonianos de menor peso em G.

Subestruturas de um grafo: cliques

Definição

Um clique de um grafo $G = (V, A)$ é um subconjunto $C \subseteq V$ tal que $\{x, y\} \in A$, for all $x, y \in C$ with $x \neq y$.
(Ou equivalentemente: o subgrafo induzido por C é completo).



- Alguns cliques: $\{1, 2, 3\}$, $\{2, 4, 5\}$, $\{4, 6\}$, $\{1\}$, $\emptyset$
- Cliques máximos (maiores): $\{1, 2, 3, 4\}$, $\{3, 4, 5, 6\}$, $\{2, 3, 4, 5\}$

Problemas famosos envolvendo cliques

Problema (Problema do clique máximo)

Encontre um clique de cardinalidade máxima num grafo.

Problema (Problema de todos os cliques)

Encontre todos os cliques de um grafo sem repetições.

Sistemas de conjuntos/Hipergrafos

Definição

Um sistema de conjuntos (ou hipergrafo) é um par $(X, \mathcal{B})$ onde:
X é um conjunto finito (elementos chamados pontos/vértices).
$\mathcal{B}$ é um conjunto de subconjuntos de X (blocos/hiperarestas).

Sistemas de conjuntos/Hipergrafos

Definição

Um sistema de conjuntos (ou hipergrafo) é um par $(X, \mathcal{B})$ onde:
X é um conjunto finito (elementos chamados pontos/vértices).
$\mathcal{B}$ é um conjunto de subconjuntos de X (blocos/hiperarestas).

- **Grafo:** Um grafo é um sistema de conjuntos com cada bloco tendo cardinalidade 2

Sistemas de conjuntos/Hipergrafos

Definição

Um sistema de conjuntos (ou hipergrafo) é um par $(X, \mathcal{B})$ onde:
X é um conjunto finito (elementos chamados pontos/vértices).
$\mathcal{B}$ é um conjunto de subconjuntos de X (blocos/hiperarestas).

- **Grafo:** Um grafo é um sistema de conjuntos com cada bloco tendo cardinalidade 2
- **Partição de um conjunto finito:**
  $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\mathcal{B} = \{\{1, 2, 4\}, \{3, 9\}, \{5, 6, 7, 8\}\}$

Sistemas de conjuntos/Hipergrafos

Definição

Um sistema de conjuntos (ou hipergrafo) é um par $(X, \mathcal{B})$ onde:
X é um conjunto finito (elementos chamados pontos/vértices).
$\mathcal{B}$ é um conjunto de subconjuntos de X (blocos/hiperarestas).

- **Grafo:** Um grafo é um sistema de conjuntos com cada bloco tendo cardinalidade 2
- **Partição de um conjunto finito:**
  $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\mathcal{B} = \{\{1, 2, 4\}, \{3, 9\}, \{5, 6, 7, 8\}\}$
- **Sistema de triplas de Steiner:**
  $\mathcal{B}$ é um conjunto de 3-subconjuntos de X tal que para cada $\{x, y\} \subset X, x \neq y$, existe um único bloco $B \in \mathcal{B}$ com $\{x, y\} \subseteq B$.
  $X = \{0, 1, 2, 3, 4, 5, 6\}$
  $\mathcal{B} = \{\{0, 1, 2\}, \{0, 3, 4\}, \{0, 5, 6\}, \{1, 3, 5\}, \{1, 4, 6\}, \{2, 3, 6\}, \{2, 4, 5\}\}$

Algoritmos Combinatórios

Algoritmos combinatórios são algoritmos para investigar estruturas combinatórias.

- **Geração**
  Construa todas as estruturas combinatórias de um certo tipo.
- **Enumeração**
  Compute o número de todas as distintas estruturas de um certo tipo.
- **Busca**
  Encontre pelo menos um exemplo de uma estrutura combinatória de um certo tipo (se tal estrutura existir).
  Problemas de otimização podem ser vistos como um tipo especial de problema de busca (a busca de um objeto ótimo)

- **Geração**
  Construir todas as estruturas combinatórias de um certo tipo.
  - Gere todos os subconjuntos/permutações/partições de um conjunto.
  - Gere todos os cliques de um grafo.
  - Gere todos os cliques máximos de um grafo.
  - Gere todos os sistemas de tripla de Steiner de um conjunto finito.

- **Geração**
  Construir todas as estruturas combinatórias de um certo tipo.
  - Gere todos os subconjuntos/permutações/partições de um conjunto.
  - Gere todos os cliques de um grafo.
  - Gere todos os cliques máximos de um grafo.
  - Gere todos os sistemas de tripla de Steiner de um conjunto finito.

- **Enumeração**
  Compute o número de todas as distintas estruturas de um certo tipo.
  - Compute o número de todos os subconjuntos/permutações/partições de um conjunto.
  - Compute o número de todos os cliques de um grafo.
  - Compute o número de todos os cliques máximos de um grafo.
  - Compute o número de todos os sistemas de tripla de Steiner de um conjunto finito.

- **Busca**
  Encontre pelo menos um exemplo de uma estrutura combinatória de um certo tipo (se tal estrutura existir).
  Problemas de otimização podem ser vistos como um tipo especial de problema de busca (a busca de um objeto ótimo)
  - Encontre um sistema de triplas de Steiner de um conjunto finito, se existir. (viabilidade)
  - Encontre um clique de tamanho máximo. (otimização)
  - Encontre, se existir, um circuito hamiltoniano num grafo. (viabilidade)
  - Encontre circuito hamiltoniano de menor peso/custo num grafo [problema do caixeiro viajante] (otimização).

Complexidade de algoritmos e complexidade de problemas

- A partir dos anos 70 se estabeleceu a noção de que algoritmos eficientes são os que tem um tempo assintótico que é uma função **polinomial** do tamanho da entrada.
- A análise de algoritmos visa estudar o comportamento assintótico dos algoritmos.
- A teoria de complexidade computacional visa estudar a dificuldade relativa de diversos problemas.
- Os problemas para os quais existe um algoritmo polinomial para sua solução fazem parte da classe computacional de problemas **P**.
  Exemplo: A ordenação de um vetor de n elementos pode ser feita em tempo $O(n \log n)$. Ou seja, o problema de ordenação pertence a classe **P**.
- Para muitos problemas de busca e otimização combinatória, se desconhece um algoritmo polinomial para sua solução, mas se conhece um algoritmo polinomial para a **verificação** de uma solução. Fazem parte da classe **NP**, e são os problemas mais difíceis em **NP**.

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.
- **P** = classe de problemas de decisão que podem ser **resolvidos** em tempo polinomial. (ex.: Caminho mais curto em um grafo $\in$ **P**)

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.
- **P** = classe de problemas de decisão que podem ser **resolvidos** em tempo polinomial. (ex.: Caminho mais curto em um grafo $\in$ **P**)
- **NP** = classe de problemas de decisão que podem ser **verificados** em tempo polinomial. Note que **P** $\subseteq$ **NP**.
  (ex.: Existência de um caminho hamiltoniano em um grafo $\in$ **NP**)

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.
- **P** = classe de problemas de decisão que podem ser **resolvidos** em tempo polinomial. (ex.: Caminho mais curto em um grafo $\in$ **P**)
- **NP** = classe de problemas de decisão que podem ser **verificados** em tempo polinomial. Note que **P** $\subseteq$ **NP**.
  (ex.: Existência de um caminho hamiltoniano em um grafo $\in$ **NP**)
- Problemas **NP-completos** são aqueles em **NP** que são pelo menos "tão difíceis quanto" qualquer problema em **NP** (com respeito a reducibilidade polinomial).

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.
- **P** = classe de problemas de decisão que podem ser **resolvidos** em tempo polinomial. (ex.: Caminho mais curto em um grafo $\in$ **P**)
- **NP** = classe de problemas de decisão que podem ser **verificados** em tempo polinomial. Note que **P** $\subseteq$ **NP**.
  (ex.: Existência de um caminho hamiltoniano em um grafo $\in$ **NP**)
- Problemas **NP-completos** são aqueles em **NP** que são pelo menos "tão difíceis quanto" qualquer problema em **NP** (com respeito a reducibilidade polinomial).
- Um dos problemas matemáticos e computacionais mais importantes é a pergunta **P=NP?**.

Dificuldade/complexidade de problemas de busca e otimização combinatória

- Muitos problemas de busca e otimização são **NP-completos**.
- **P** = classe de problemas de decisão que podem ser **resolvidos** em tempo polinomial. (ex.: Caminho mais curto em um grafo $\in$ **P**)
- **NP** = classe de problemas de decisão que podem ser **verificados** em tempo polinomial. Note que **P** $\subseteq$ **NP**.
  (ex.: Existência de um caminho hamiltoniano em um grafo $\in$ **NP**)
- Problemas **NP-completos** são aqueles em **NP** que são pelo menos "tão difíceis quanto" qualquer problema em **NP** (com respeito a reducibilidade polinomial).
- Um dos problemas matemáticos e computacionais mais importantes é a pergunta **P=NP?**.
- Muitos pensam que **P**$\neq$**NP**, o que implicaria a inexistência de algoritmos polinomiais p/ todos os problemas **NP-completos**.

Algoritmos Combinatórios lidam com problemas NP-completos

- **Busca exaustiva**
  - algoritmos de tempo exponencial;
  - resolvem o problema exatamente.

  Métodos: Backtracking, Branch-and-Bound.
- **Busca heurística** (tópico não será coberto neste curso)
  - algoritmos que exploram o espaço de possíveis soluções tentando encontrar uma solução ótima do problema;
  - aproximam uma solução ao problema, sem garantia de valor objetivo próximo ao ótimo.

  Métodos: Algoritmos de busca local, Hill-climbing, Simulated annealing, Busca Tabu, Algoritmos Genéticos, etc.
- **Algoritmos de aproximação** (tópico não será coberto neste curso)
  - algoritmos de tempo polinomial;
  - neste caso, temos uma garantia de que a solução encontrada tem um valor objetivo próximo do ótimo.

Geração de Objetos Básicos

Isto inclui geração de objetos combinatórios básicos, como:

- n-tuplas de um alfabeto finito,
- subconjuntos de um conjunto finito,
- k-subconjuntos de um conjunto finito,
- permutações, etc.

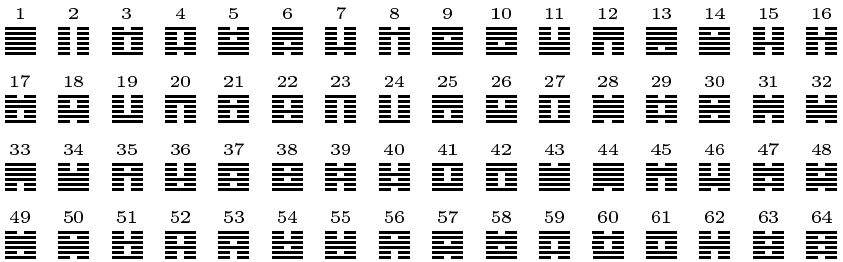
Para fazer uma geração sequencial, precisamos impor uma ordem no conjunto de objetos a serem gerados.

As ordens mais comuns são ordem lexicográfica e ordem de mudança mínima.

Geração Combinatória: um assunto antigo

Extraido de: D. Knuth, *History of Combinatorial Generation*, in pre-fascicle 4B , The Art of Computer Programming Vol 4.

Lists of binary n-tuples can be traced back thousands of years to ancient China, India, and Greece. The most notable source—because it still is a best-selling book in modern translations—is the Chinese *I Ching* or *Yijing*, whose name means "the Bible of Changes." This book, which is one of the five classics of Confucian wisdom, consists essentially of $2^6 = 64$ chapters; and each chapter is symbolized by a hexagram formed from six lines, each of which is either -- ("yin") or — ("yang"). For example, hexagram 1 is pure yang, ䷀; hexagram 2 is pure yin, ䷁; and hexagram 64 intermixes yin and yang, with yang on top: ䷿. Here is the complete list:



(1)

This arrangement of the 64 possibilities is called King Wen's ordering, because the basic text of the *I Ching* has traditionally been ascribed to King Wen (c. 1100 B.C.), the legendary progenitor of the Chou dynasty. Ancient texts are, however, notoriously difficult to date reliably, and modern historians have found no solid evidence that anyone actually compiled such a list of hexagrams before the third century B.C.

Geração de k-subconjuntos (de um n-conjunto): Ordem Lexicográfica

Exemplo: $k = 3$, $n = 5$.

rank	T	$\vec{T}$
0	$\{1, 2, 3\}$	$[1, 2, 3]$
1	$\{1, 2, 4\}$	$[1, 2, 4]$
2	$\{1, 2, 5\}$	$[1, 2, 5]$
3	$\{1, 3, 4\}$	$[1, 3, 4]$
4	$\{1, 3, 5\}$	$[1, 3, 5]$
5	$\{1, 4, 5\}$	$[1, 4, 5]$
6	$\{2, 3, 4\}$	$[2, 3, 4]$
7	$\{2, 3, 5\}$	$[2, 3, 5]$
8	$\{2, 4, 5\}$	$[2, 4, 5]$
9	$\{3, 4, 5\}$	$[3, 4, 5]$

Algoritmos:
sucessor
rank
unrank

Gerando todos os k-subconjuntos via algoritmo de sucessor

Exemplo/idéia: $n = 10$, SUCESSOR($\{\ldots, \underline{4}, 8, 9, 10\}$)=$\{\ldots, \underline{5}, 6, 7, 8\}$

kSUBCONJUNTOLEXSUCESSOR $(\vec{T}, k, n)$

$\vec{U} \leftarrow \vec{T}$; $i \leftarrow k$;
while $(i \geq 1)$ and $(t_i = n - k + i)$ do $i \leftarrow i - 1$;
if $(i = 0)$ then $\vec{U}$ =“indefinido”;
else for $j \leftarrow i$ to k do
$u_j \leftarrow (t_i + 1) + j - i$;
return $\vec{U}$;

Main (k, n) /* Gerar todos os k-subconjuntos de $\{1, 2, \ldots, n\}$ */

$\vec{T} \leftarrow [1, 2, \ldots, k]$
repeat
output $\vec{T}$
$\vec{T} \leftarrow$ kSUBCONJUNTOLEXSUCESSOR $(\vec{T}, k, n)$
until $\vec{T}$ =“indefinido”

Geração de Permutações: Ordem Lexicográfica

Uma permutação é uma bijeção $\Pi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$.

Nós a representamos como uma lista : $\Pi = [\Pi[1], \Pi[2], \ldots, \Pi[n]]$.

Ordem Lexicográfica: $n = 3$

rank	permutação
0	$[1, 2, 3]$
1	$[1, 3, 2]$
2	$[2, 1, 3]$
3	$[2, 3, 1]$
4	$[3, 1, 2]$
5	$[3, 2, 1]$

Sucessor para permutações em ordem lexicográfica

Exemplo: $\Pi = [3, 5, \mathbf{4}, \underline{7, \overline{6}, 2, 1}]$

Seja $i =$ índice anterior ao sufixo decrescente $= 3$.
Seja $j=$ índice do sucessor de $\pi[i]$ in $\{\Pi[i+1], \ldots, \Pi[n]\}$
$\pi[i] = 4$, sucessor de $\pi[i] = 4$ em $\{7, 6, 2, 1\}$ é 6, $\pi[5] = 6$, então $j = 5$.

Troque $\Pi[i]$ e $\Pi[j]$, e inverta a ordem de $\{\Pi[i+1], \ldots, \Pi[n]\}$.

$$\textsc{Sucessor}(\Pi) = [3, 5, \mathbf{6}, \underline{1, 2, 4, 7}]$$

Note que: $i = \max\{l : \Pi[l] < \Pi[l+1]\}$

$j = \max\{l : \Pi[l] > \Pi[i]\}$.

No algoritmo seguinte, criamos uma sentinela: $\Pi[0] = 0$.

PermLexSucessor(n, Π)

$\Pi[0] \leftarrow 0$;

$i \leftarrow n-1$;

while $(\Pi[i] > \Pi[i+1])$ do $i \leftarrow i-1$;

if $(i = 0)$ then return "indefinido"

$j \leftarrow n$;

while $(\Pi[j] < \Pi[i])$ do $j \leftarrow j-1$;

$t \leftarrow \Pi[j]$; $\Pi[j] \leftarrow \Pi[i]$; $\Pi[i] \leftarrow t$; (troque $\Pi[i]$ e $\Pi[j]$)

// Inversão de $\Pi[i+1], \ldots, \Pi[n]$ no local:

for $h \leftarrow i+1$ to $\lfloor \frac{n-i}{2} \rfloor$ do

$t \leftarrow \Pi[h]$; $\Pi[h] \leftarrow \Pi[n+i+1-h]$;

$\Pi[n+i+1-h] \leftarrow t$;

return Π;

Main (n) /* Gerar todos as permutações de $\{1, 2, \ldots, n\}$ */
    $\Pi \leftarrow [1, 2, \ldots, n]$
    repeat
        output Π
        $\Pi \leftarrow$ PermLexSucessor(n, Π)
    until $\Pi =$ "indefinido"

Geração de k-subconjuntos: Ordem de Mudança Mínima

A menor diferença possível entre dois k-subconjuntos ocorre quando eles contém exatamente $k-1$ elementos em comum.

Ordem da Porta Giratória

Baseia-se na identidade (triângulo) de Pascal: $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.

Defina a sequência de k-subconjuntos $A^{n,k}$ baseada em $A^{n-1,k}$ e o reverso de $A^{n-1,k-1}$, da seguinte maneira:

$$\begin{aligned} A^{n,k} &= \left[A_0^{n-1,k}, \ldots, A_{\binom{n-1}{k}-1}^{n-1,k}, |A_{\binom{n-1}{k-1}-1}^{n-1,k-1} \cup \{n\}, \ldots, A_0^{n-1,k-1} \cup \{n\}\right], \\ &\quad for\ 1 \leq k \leq n-1 \\ A^{n,0} &= [\emptyset] \\ A^{n,n} &= [\{1, 2, \ldots, n\}] \end{aligned}$$

Exemplo: Construção de $A^{5,3}$ usando $A^{4,3}$ e $A^{4,2}$

$$A^{4,3} = [\{1,2,3\},\{1,3,4\},\{2,3,4\},\{1,2,4\}]$$
$$A^{4,2} = [\{1,2\},\{2,3\},\{1,3\},\{3,4\},\{2,4\},\{1,4\}]$$

$$A^{5,3} = [\{1,2,3\},\{1,3,4\},\{2,3,4\},\{1,2,4\},|$$
$$|\{1,4,\mathbf{5}\},\{2,4,\mathbf{5}\},\{3,4,\mathbf{5}\},\{1,3,\mathbf{5}\},\{2,3,\mathbf{5}\},\{1,2,\mathbf{5}\}]$$

Para verificar que a ordem da porta giratória é uma ordem de mudança mínima, prove:

1. $A^{n,k}_{\binom{n}{k}-1} = \{1, 2, \ldots, k-1, n\}$.
2. $A^{n,k}_0 = \{1, 2, \ldots, k\}$.
3. Para todo n, k, $1 \leq k \leq n$, $A^{n,k}$ é uma ordem de mudança mínima de $\mathcal{S}^n_k$.

Omitimos os algoritmos para gerar k-subconjuntos seguindo a ordem da porta giratória, já que requereria mais tempo.

Geração de permutações: Ordem de Mudança Mínima

Ordem de Mudança Mínima para permutações: duas permutações contíguas devem diferir por uma transposição de elementos adjacentes.
O algoritmo de Trotter-Johnson usa a seguinte ordem:

$$
\begin{aligned}
T^1 &= [[1]] \\
T^2 &= [[1,\mathbf{2}],[\mathbf{2},1]] \\
T^3 &= [[1,2,\mathbf{3}],[1,\mathbf{3},2],[\mathbf{3},1,2][\mathbf{3},2,1],[2,\mathbf{3},1],[2,1,\mathbf{3}]]
\end{aligned}
$$

Como construir T^3 usando T^2:

	1		2	**3**
	1	**3**	2	
3	1		2	
3	2		1	
	2	**3**	1	
	2		1	**3**

Exercício: construa T^4 usando T^3.

Omitimos os algoritmos para gerar permutações seguindo a ordem deTrotter-Johnson, já que requereria mais tempo.