CSI5165 COMBINATORIAL ALGORITHMS

Prof. Lucia Moura

Fall 2003

INTRODUCTION TO COMBINATORIAL ALGORTIHMS

Introduction to Combinatorial Algorithms

What are :

- Combinatorial Structures?
- Combinatorial Algorithms?
- Combinatorial Problems?

Combinatorial Structures

Combinatorial structures are *collections* of k-subsets/K-tuple/permutations from a parent set (finite).

Examples:

- **Undirected Graphs:**
  Collections of 2-subsets (edges) of a parent set (vertices).

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

- **Directed Graphs:**
  Collections of 2-tuples (directed edges) of a parent set (vertices).

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

- **Hypergraphs or Set Systems:**
  Similar to graphs, but (hyper) edges may be sets with more than two elements.

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

Building blocks: finite sets, finite lists (tuples)

- Finite Sets

$X = \{1, 2, 3, 5\}$

– undordered structure, no repeats
$\{1, 2, 3, 5\} = \{2, 1, 5, 3\} = \{2, 1, 1, 5, 3\}$
– cardinality (size) = number of elements $|X| = 4$.

A <u>k-subset</u> of a finite set X is a set $S \subseteq X$, $|S| = k$.
For example: $\{1, 2\}$ is a 2-subset of X.

- Finite Lists (or Tuples)

$L = [1, 5, 2, 1, 3]$

– ordered structure, repeats allowed
$[1, 5, 2, 1, 3] \neq [1, 1, 2, 3, 5] \neq [1, 2, 3, 5]$
– length = number of items, length of L is 5.

An <u>n-tuple</u> is a list of length n.
A <u>permutation</u> of an n-set X is a list of length n such that every element of X occurs exactly once.

$$X = \{1, 2, 3\}, \quad \pi_1 = [2, 1, 3] \quad \pi_2 = [3, 1, 2]$$

Graphs

Definition. A *graph* is a pair (V, E) where:
V is a finite set (of vertices).
E is a finite set of 2-subsets (called edges) of V.

Example: G_1 : $V = \{0, 1, 2, 3, 4, 5, 6, 7\}$
$E = \{\{0, 4\}, \{0, 1\}, \{0, 2\}, \{2, 3\}, \{2, 6\},$
$\{3, 7\}, \{4, 5\}, \{4, 6\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$

Complete graphs: graphs with all possile edges.
Examples:

Substructures of a graph:

1. A hamiltonian circuit (hamiltonian cycle) is a closed path that passes through each vertex once.

   The following list describes a hamiltonian cycle in G_1:
   $[0, 1, 5, 4, 6, 7, 3, 2]$ (different lists may describe the same cycle).

Traveling Salesman Problem: given a weight/cost function $w : E \to R$ on the edges of G, find a smallest weight hamiltonian cycle in G.

2. A clique in a graph $G = V, E)$ is a subset $C \subseteq V$ such that $\{x, y\} \in E$ for all $x, y \in C$.
(Or equivalently: the subgraph induced by C is complete).

Example:

G_2 : Some cliques of G_2:

Maximum cliques of G_2:

Famous problems involving cliques:

- Maximum clique problem: find a maximum clique in a graph.
- All cliques problem: find all cliques in a graph without repetition.

Set systems/Hypergraphs

DEFINITION. A *set system (or hypergraph)* is a pair $(X, \mathcal{B})$ where:
X is a finite set (of points).
$\mathcal{B}$ is a finite set of subsets of X (blocks).

Examples:

- Graph: A graph is a set system with every block with cardinality 2.
- Partition of a finite set:

  A partition is a set system $(X, \mathcal{B})$ such that
  $B_1 \cap B_2 = \emptyset$ for all $B_1, B_2 \in \mathcal{B}, B_1 \neq B_2$, and

  $$\cup_{B \in \mathcal{B}} B = X.$$

- Steiner triple system (a type of combinatorial designs):

  $\mathcal{B}$ is a set of 3-subsets of X such that for each $x, y \in X, x \neq y$, there exists eactly one block $B \in \mathcal{B}$ with $\{x, y\} \subseteq B$.

  Example:

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

  $\mathcal{B} =$
  $\{\{0, 1, 2\}, \{0, 3, 4\}, \{0, 5, 6\}, \{1, 3, 5\}, \{1, 4, 6\}, \{2, 3, 6\}, \{2, 4, 5\}\}$

Combinatorial Algorithms

Algorithms for investigating combinatorial structures. Three types:

- **Generation**
  Construct all combinatorial structures of a particular type.
  - Generate all subsets/permutations/partitions of a set.
  - Generate all cliques of a graph.
  - Generate all maximum cliques of a graph.
  - Generate all Steiner triple systems of a finite set.
- **Enumeration**
  Compute the number of different structures of a particular type.
  - Compute the number of subsets/permutat./partitions of a set.
  - Compute the number of cliques of a graph.
  - Compute the number of maximum cliques of a graph.
  - Compute the number of Steiner triple systems of a finite set.
- **Search**
  Find at least one example of a combinatorial structures of a particular type (if one exists).
  Optimization problems can be seen as a type of search problem.
  - Find a Steiner triple system on a finite set. (fesibility)
  - Find a maximum clique of a graph. (optimization)
  - Find a hamiltonian cycle in a graph. (feasibility)
  - Find a smallest weight hamiltonian cycle in a graph. (optimization)

Hardness of Search and Optimization

Many search and optimization problems are NP-hard, which means that

unless $P = NP$ (an important unsolved complexity question)

no polytomial-time algorithm to solve the problem would exist.
Approaches for dealing with NP-hard problems:

- Exhaustive Search
  - exponential-time algorithms.
  - solves the problem exactly

  (Backtracking and Branch-and-Bound)
- Heuristic Search
  - algorithms that explore a search space to find a feasible solution that is "close to" optimal, within a time limit
  - approximates a solution to the problem

  (Hill-climbing, Simulated annealing, Tabu-Search, Genetic Algorithms)
- Approximation Algorithms
  - polynomial time algorithm
  - we have a provable guarantee that the solution found is "close to" optimal.

  (not covered in this course)

Types of Search Problems

1) **Decision Problem**:
A yes/no problem

Problem 1:
Clique (decision)
Instance: graph $G = (V, E)$,
target size k

Question:
Does there exist a clique C of G with $|C| = k$?

2) **Search Problem**:
Find the guy.

Problem 2:
Clique (search)
Instance: graph $G = (V, E)$,
target size k

Find:
a clique C of G with $|C| = k$, if one exists.

3) **Optimal Value**:
Find the largest target size.

Problem 3:
Clique (optimal value)
Instance: graph $G = (V, E)$,

Find:
the maximum value of $|C|$, where C is a clique

4) **Optimization**:
Find an optimal guy.

Problem 4:
Clique (optimization)
Instance: graph $G = (V, E)$,

Find:
a clique C such that $|C|$ is maximize (max. clique)

Plan for the Course

1. **Generating elementary combinatorial objects**

   Sequential generation (successor), rank, unrank.
   Algorithms for subsets, k-subsets, permutations.
   Reference: textbook chapter 2. [2 weeks]

2. **Exhaustive Generation and Exhaustive Search**

   Backtracking algorithms
   (exhaustive generation, exhaustive search, optimization)

   Brach-and-bound
   (exhaustive search, optimization)
   Reference: textbook chapter 4. [3 weeks]

3. **Heuristic Search**

   Hill-climbing, Simulated annealing, Tabu-Search, Genetic Algs.
   Applications of these techniques to various problems.

   Reference: textbook chapter 5. [3 weeks]

4. **Computing Isomorphism and Isomorph-free Exhaustive Generation**

   Graph isomorphism, isomorphism of other structures.
   Computing invariants.
   Computing certificates.
   Isomorph-free exhaustive generation.
   Example: Generate all trees on n vertices, without isomorphic copies.

   Reference: textbook chapter 7, papers. [3 weeks]

GENERATING ELEMENTARY COMBINATORIAL OBJECTS

Combinatorial Generation

We are going to look at combinatorial generation of:

- Subsets
- k-subsets
- Permutations

To do a sequential generation, we need to impose some order on the set of objects we are generating.

Let $\mathcal{S}$ be a finite set and $N = |\mathcal{S}|$.
A rank function is a bijection

$$\text{RANK}: \mathcal{S} \to \{0, 1, \ldots, N-1\}.$$

It has another bijection associated with it

$$\text{UNRANK}: \{0, 1, \ldots, N-1\} \to \mathcal{S}.$$

A rank function defines an ordering on $\mathcal{S}$.
Many types of ordering are possible; we will discuss two types: **lexicographical** ordering and **minimal change** ordering.

Once an ordering is chosen, we can talk about the following types of algorithms:

- Successor: given an object, return its successor.
- Rank: given an object $S \in \mathcal{S}$, return $\text{RANK}(\text{S})$
- Unrank: given a rank $i \in \{0, 1, \ldots, N-1\}$, return $\text{UNRANK}(n)$, its corresponding object.

1. Generating Subsets (of an n-set)

1.1. Generating Subsets: Lexicographical Ordering

Represent a set by its **characteristic vector**:

subset X of {1,2,3}	characteristic vector
{1,2}	[1,1,0]
{3}	[0,0,1]

The **characteristic vector** of a subset $T \subseteq X$ is a vector $\mathcal{X}(T) = [x_{n-1}, x_{n-2}, \ldots, x_1, x_0]$ where

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

Example:

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

Note that the order is lexicographical on $\mathcal{X}(T)$ and not on T.
Note that $\mathcal{X}(T)$ corresponds to the binary representation of rank!

Ranking

More efficient implementation:

SubsetLexRank (n, T)
```
r ← 0;
for i ← 1 to n do
    r ← 2 * r;
    if (i ∈ T) then r ← r + 1;
return r;
```

Books'version:

```
if (i ∈ T) then
  r ← r + 2^(n−i)
```

This is like a conversion from the binary representation to the number.

Unranking

SubsetLexUnrank (n, r)
```
T ← ∅;
for i ← n downto 1 do
    if (r mod 2 = 1) then T ← T ∪ {i};
    r ← ⌊r/2⌋;
return T;
```

This is like a conversion from number to its binary representation.

Successor

The following algorithm is adapted for circular ranking, that is, the successor of the largest ranked object is the object of rank 0.

SubsetLexSuccessor (n, T)

```
    i ← 0;
    while (i ≤ n − 1) and (n − i ∈ T) do
        T ← T \ {n − i};
        i ← i + 1;
    if (i ≤ n − 1) then T ← T ∪ {n − i};
    return T;
```

This algorithm works like an increment on a binary number.

Examples:

1. SubsetLexSuccessor$(3, \{2, 3\}) = \{1\}$.

$\{2, 3\}$	$[\overline{0}, \underline{1, 1}]$
$\{1\}$	$[1, 0, 0]$

2. SubsetLexSuccessor$(4, \{1, 4\}) = \{1, 3\}$.

$\{1, 4\}$	$[1, 0, \overline{0}, \underline{1}]$
$\{1, 3\}$	$[1, 0, 1, 0]$

1.2. Generating Subsets: Minimal Change Ordering

In minimal change ordering, successive sets are as similar as possible.

The **hamming distance** between two vectors is defined as the number of bits in which the two vectors differ.

Example: $dist(\underline{0}00\underline{1}010, \underline{1}00\underline{0}010) = 2$.

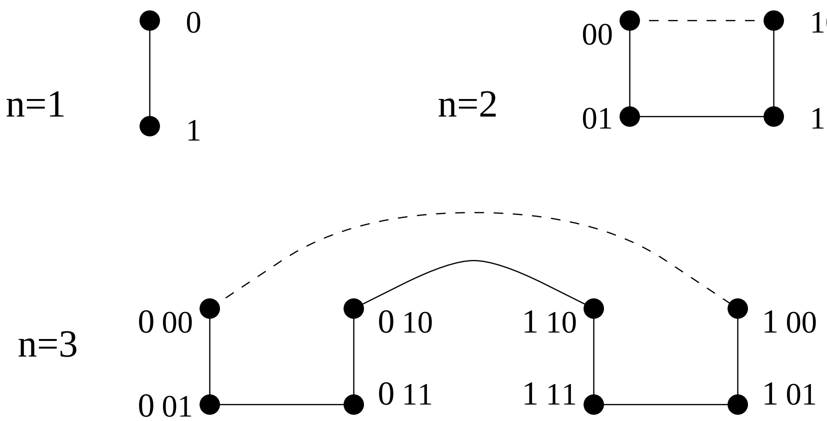
When we apply to the subsets corresponding to the binary vectors, it is equivalent to:
$dist(T_1, T_2) = |T_1 \triangle T_2| = |(T_1 \setminus T_2) \cup (T_2 \setminus T_1)|$.

A **Gray Code** is a sequence of vectors with successive vectors having hamming distance exactly 1.

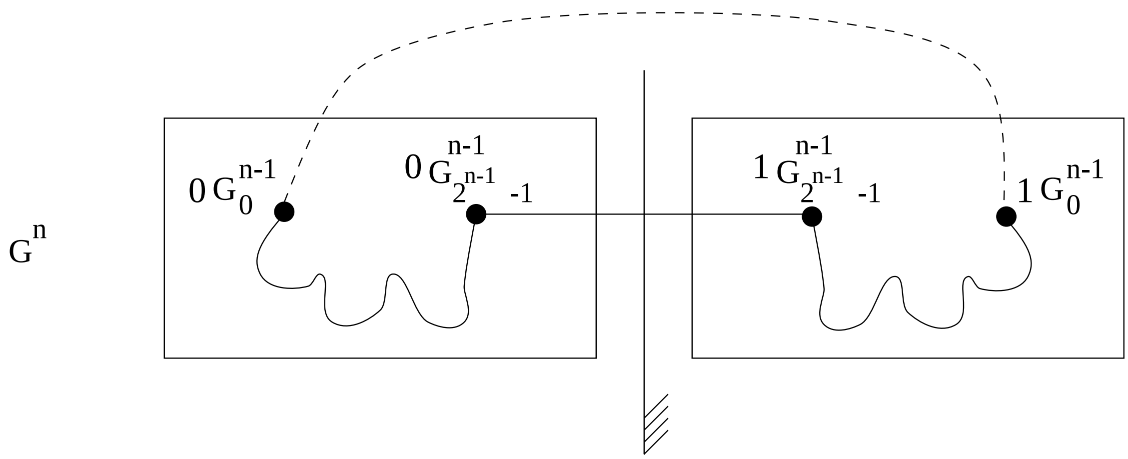
Example: $[00, 01, 11, 10]$.

We will now see a construction for one possible type of Gray Codes...

Construction for Binary Reflected Gray Codes



In general, build G_n as follows:



More formally, we define G^n inductively as follows:

$$\begin{aligned} G^1 &= [0, 1] \\ G^n &= [0G_0^{n-1}, \cdots, 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, \cdots 1G_0^{n-1}] \end{aligned}$$

Theorem 2.1. For any $n \geq 1$, G^n is a gray code.
Exercise: prove this theorem by induction on n.

Successor

Examples:

$$G_3 = [000, 001, 011, 010, 110, 111, 101, 100]$$
$$G_4 = [0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000].$$

Rules for calculating successor:

- If vector has even weight (even number of 1's): flip last bit.
- If vector has odd weight (odd number of 1's): from right to left, flip bit after the first 1.

GRAYCODESUCCESSOR (n, T)
    if (T is even) then
    $U \leftarrow T \triangle \{n\}$;
    else
        $j \leftarrow n$; *(flip last bit)*
        while ($j \notin T$) and ($j > 0$) do $j \leftarrow j - 1$;
        if ($j = 1$) then $U \leftarrow \emptyset$; *(I changed for circular order)*
                else $U \leftarrow T \triangle \{j - 1\}$;
    return U;

Ranking and Unranking

	r	0	1	2	3	4	5	6	7
$b_3b_2b_1b_0$	bin.rep. r	000	001	010	011	100	101	110	111
$a_2a_1a_0$	G^3_r	000	001	011	010	110	111	101	100

Set $b_3 = 0$ in the example above.

We need to relate $(b_nb_{n-1}\dots b_0)$ and $(a_{n-1}a_{n-2},\dots a_0)$.

Lemma 1.
Let $P(n)$: "For $0 \le r \le 2^n - 1, a_j \equiv b_j + b_{j+1} \pmod 2$, for all $0 \le j \le n-1$". Then, $P(n)$ holds for all $n \ge 1$.

Proof: We will prove $P(n)$ by induction on n.
Basis: $P(1)$ holds, since $a_0 = b_0$ and $b_1 = 0$.

Induction step: Assume $P(n-1)$ holds. We will prove $P(n)$ holds.
Case 1. $r \le 2^{n-1} - 1$ **(first half of G_n).**
Note that $b_{n-1} = 0 = a_{n-1}$ and $b_n = 0$, which implies

$$a_{n-1} = 0 = b_{n-1} + b_n. \tag{1}$$

By induction,

$$a_j \equiv b_j + b_{j+1} \pmod 2, \text{ for all} 0 \le j \le n-2. \tag{2}$$

Equations (1) and (2) imply $P(n)$.

Case 2. $2^n \leq r \leq 2^n - 1$ **(second half of G_n).**
Note that $b_{n-1} = 1 = a_{n-1}$ and $b_n = 0$, which implies

$$a_{n-1} \equiv 1 \equiv b_{n-1} + b_n \pmod 2. \tag{3}$$

Now, $G^n_r = 1G^{n-1}_{2^n-1-r} = 1a_{n-2}a_{n-3}\ldots a_1a_0$. The binary representation of $2^n - 1 - r$ is
$0(1-b_{n-2})(1-b_{n-3})\ldots(1-b_1)(1-b_0)$.
By induction hypothesis we know that, for all $0 \leq j \leq n-2$,

$$a_j \equiv (1-b_j) + (1-b_{j+1}) \pmod 2 \tag{4}$$
$$\equiv b_j + b_{j+1} \pmod 2 \tag{5}$$

Equations (3) and (5) imply $P(n)$.

Lemma 2.
Let $n \geq 1$, $0 \leq r \leq 2^n - 1$. Then,

$$b_j \equiv \sum_{i=j}^{n-1} a_i \pmod 2, \quad \text{for all } 0 \leq j \leq n-1.$$

Proof:

$$\begin{aligned}\sum_{i=j}^{n-1} a_i &\equiv \sum_{i=j}^{n-1} b_i + b_{i+1} \pmod 2 \quad \text{[By Lemma 1]}\\ &\equiv b_j + 2b_{j+1} + \ldots + 2b_{n-1} + b_n \pmod 2\\ &\equiv b_j + b_n \pmod 2\\ &\equiv b_j \pmod 2 \qquad \text{[Since } b_n = 0\text{]}.\end{aligned}$$

Let $n \geq 1$, $0 \leq r \leq 2^n - 1$.
We haved proved the following properties hold, for all $0 \leq j \leq n-1$,

$$b_j \equiv \sum_{i=j}^{n-1} a_i \pmod 2.$$

$$a_j \equiv b_j + b_{j+1} \pmod 2,$$

The first property is used for ranking:

```
GrayCodeRank (n, T)
    r ← 0; b ← 0;
    for i ← n − 1 downto 0 do
        if ((n − i) ∈ T) then        ( if a_i = 1)
            b ← 1 − b;               ( b_i = \overline{b_{i+1}})
        r ← 2r + b;
    return r;
```

The second property is used for unranking:

```
GrayCodeUnrank (n, r)
    T ← ∅; b' ← r mod 2; r' ← ⌊r/2⌋;
    for i ← 0 to n − 1 do
        b ← r' mod 2
        if (b ≠ b') then T ← T ∪ {n − i};
        b' ← b; r' ← ⌊r'/2⌋;
    return T;
```

2. Generating k-subsets (of an n-set)

2.1. Generating k-subsets: Lexicographical Ordering

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	$\{3,4,5\}$	$[3,4,5]$

Successor

IDEA: $n = 10$, SUCCESSOR($\{\ldots,\underline{5},8,9,10\}$)=$\{\ldots,\underline{6},7,8,9\}$

KSUBSETLEXSUCCESSOR ($\vec{T}, k, n$)
- $\vec{U} \leftarrow \vec{T}$; $i \leftarrow k$;
- while ($i \geq 0$) and ($t_i = n - k + i$) do $i \leftarrow i - 1$;
- if ($i = 0$) then $\vec{U} = [1, 2, \ldots, k]$;
- else for $j \leftarrow i$ to k do
  - $u_j \leftarrow (t_i + 1) + j - i$;
- return $\vec{U}$;

Ranking

How many subsets preceed $\vec{T} = [t_1, t_2, \ldots, t_k]$?

all sets $[X, \ldots]$ with $1 \leq X \leq t_1 - 1$
$(\Sigma_{j=1}^{t_1-1} \binom{n-j}{k-1})$

all sets $[t_1, X, \ldots]$ with $t_1 + 1 \leq X \leq t_2 - 1$
$(\Sigma_{j=t_1+1}^{t_2-1} \binom{n-j}{k-2})$
$\vdots$
all sets $[t_1, \ldots, t_{k-1}, X, \ldots]$ with $t_{k-1} + 1 \leq X \leq t_k - 1$
$(\Sigma_{j=t_{k-1}+1}^{t_k-1} \binom{n-j}{k-(k-1)})$

Thus,

$$rank(T) = \sum_{i=1}^{k} \sum_{j=t_{i-1}+1}^{t_i-1} \binom{n-j}{k-i}.$$

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

```
r ← 0;
t_0 ← 0;
for i ← 1 to k do
    for j ← t_{i-1} + 1 to t_i − 1 do
        r ← r + C(n−j, k−i);
return r;
```

Unranking

$t_1 = x \iff \Sigma_{j=1}^{x-1} \binom{n-j}{k-1} \leq r < \Sigma_{j=1}^{x} \binom{n-j}{k-1}$
$t_2 = x \iff \Sigma_{j=t_1+1}^{x-1} \binom{n-j}{k-2} \leq r - \Sigma_{j=1}^{t_1-1} \binom{n-j}{k-1} < \Sigma_{j=t_1+1}^{x} \binom{n-j}{k-1}$
etc.

kSubsetLexUnrank (r, k, n)

```
x ← 1;
for i ← 1 to k do
    while (r ≥ C(n−x, k−i)) do
        r ← r − C(n−x, k−i);
        x ← x + 1;
    t_i ← x;
    x ← x + 1;
return T⃗ ;
```