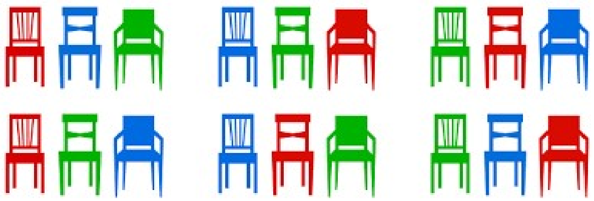
Recursion

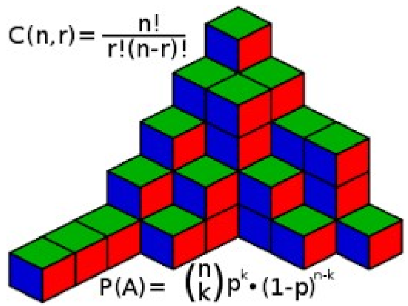
Combinatorics

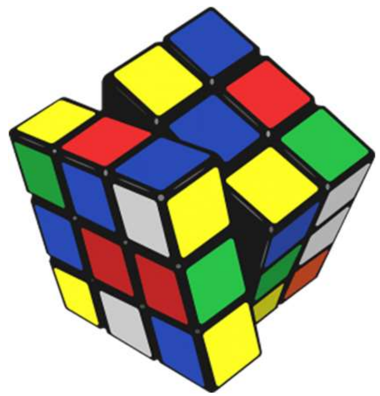
Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Definition

- **Combinatorics** is an area of matheamatics primarily concerned with counting, both as a means and an end in obtaining results, and certain properties of finite structures
- Combinatorial problems arise in many areas of pure mathematics
  - Algebra, probability theory, topology, and geometry as well as in its many application areas







Definition

- Combinatorics
  - **Count** on how many subsets of a given set a property holds
  - **Determines** in how many ways the elements of a same group may be associated according to predefined rules
- Combinatorics is used frequently in computer science to obtain formulas and estimates in the analysis of algorithms
  - In problem-solving we need to **generate** all samples, **not only** to count them

Models

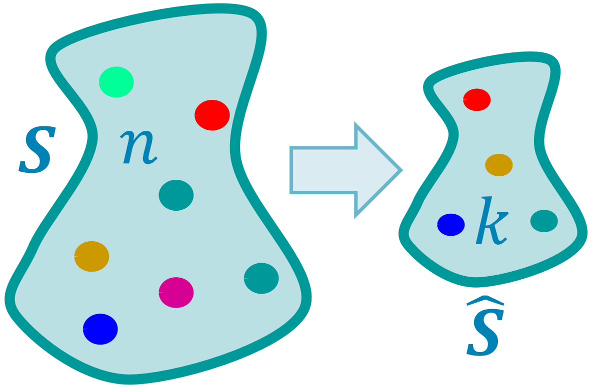
- The search space may modelled as
  - Multiplication principles
  - Arrangements
    - Simple arrangements
    - Arrangements with repetitions
  - Permutations
    - Simple permutations
    - Permutations with repetition
  - Combinations
    - Simple combinations
    - Combinations with repetitions

7 different functions to generate this samples

We are going to analyze an implementation frame/scheme for each one of these models

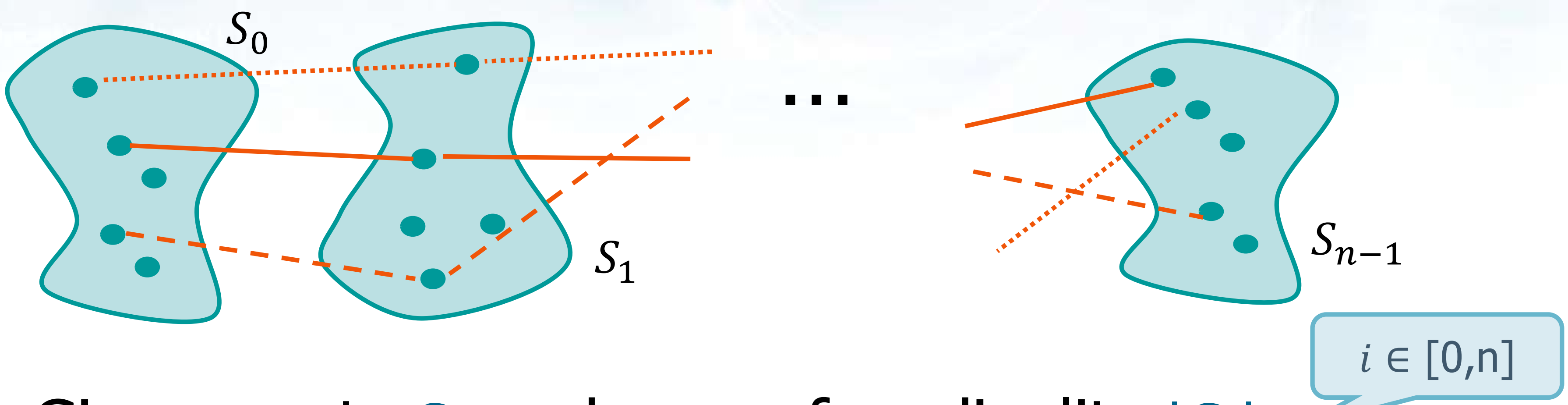
Grouping criteria

- Given a group S of n elements, we can select k objects to generate the set $\hat{S}$ keeping into account



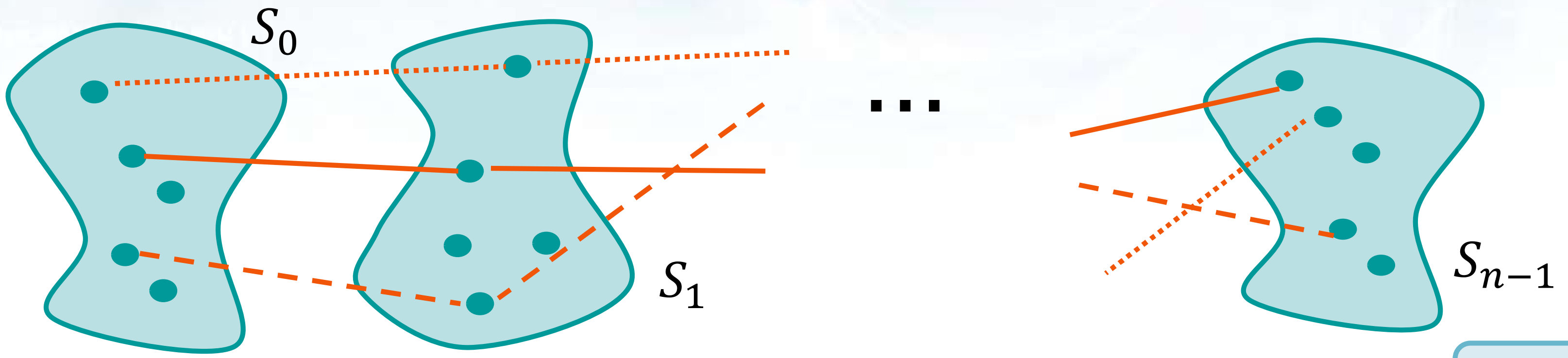
  - Unicity
    - Are all elements in group S distinct?
  - Ordering
    - Are two configurations the same if they have a different ordering of the elements?
  - Repetition
    - May the same object be used several times within the same grouping?

Multiplication principle



- Given n sets S_i each one of cardinality $|S_i|$
  - How many ordered tuples we can extract?
  - How can we generate them?

Multiplication principle



$i \in [0,n]$

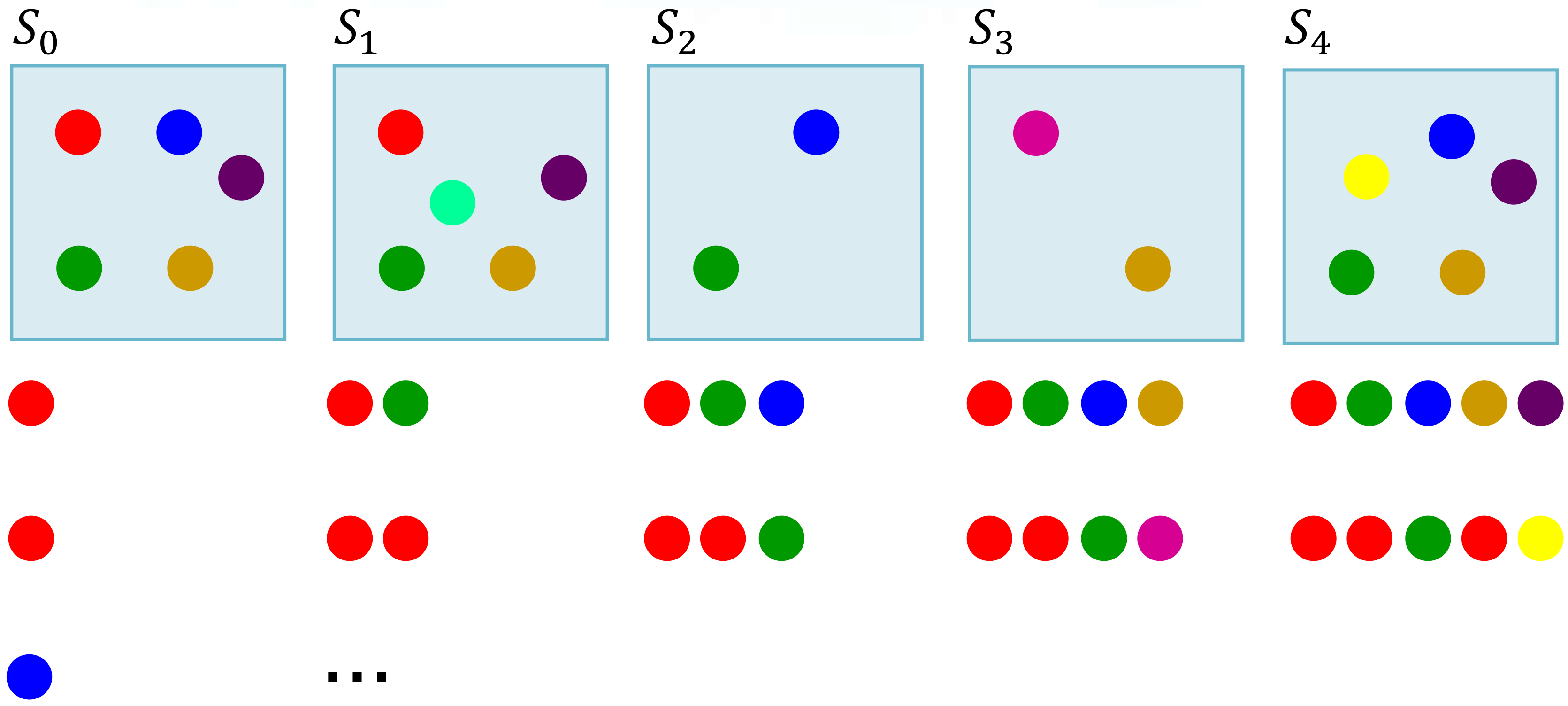
- Given n sets S_i each one of cardinality $|S_i|$
  - How many ordered tuples we can extract?
  - How can we generate them?
- The number of ordered t-uples $(s_0 s_1 \ldots\ s_n)$ with $s_0 \in S_0, s_1 \in S_1, \ldots, s_{n-1} \in S_{n-1}$ is

$$M_{S_1,S_2,\ldots,S_{n-1}} = \prod_{i=0}^{n-1} |S_i|$$

- How do we generate them?

Example

- Select one ball from each set



$$M_{S_1,S_2,\ldots,S_{n-1}} = \prod_{i=0}^{n-1} |S_i|$$

$$M_{S_1,S_2,\ldots,S_{n-1}} = 5 \cdot 5 \cdot 2 \cdot 2 \cdot 5 = 600$$

Example

- In a restaurant a menu is served made of
  - Appetizers, 2 overall
  - First course, 3 overall
  - Second course, 2 overall
- Any customer can choose 1 appetizer, 1 first course, and 1 second course
- Problem
  - How many different menus can the restaurant offer?
  - How are these menu composed?

We want to count the number of solution **and** generate those solutions

Solution

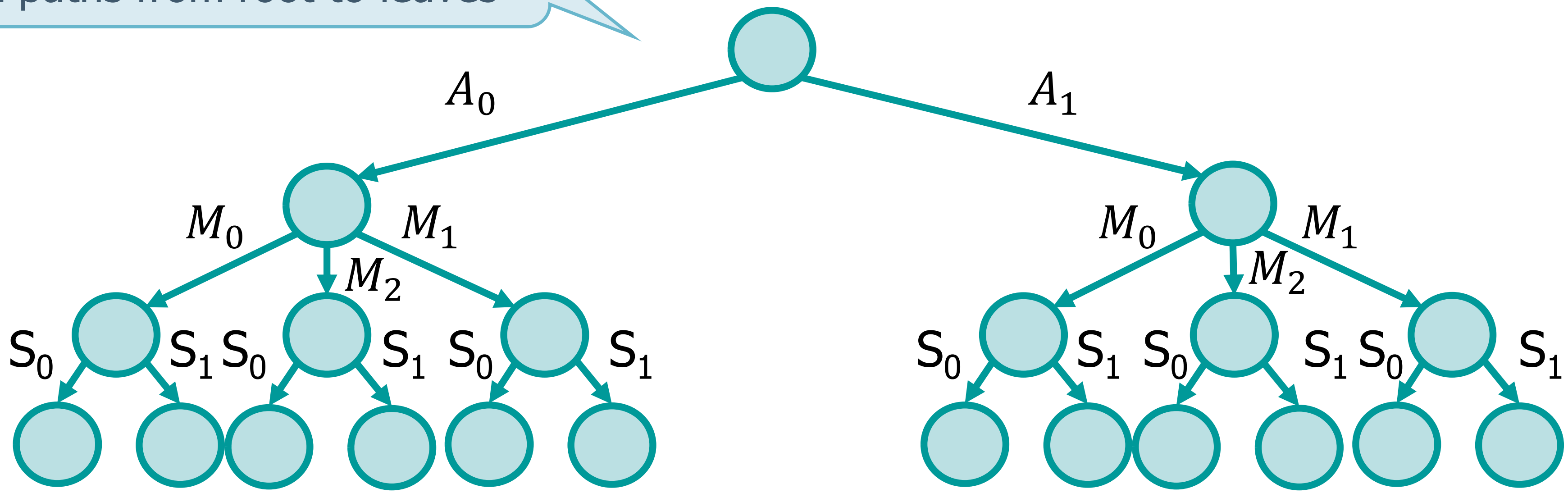
2 appetizers (A_0, A_1),
3 main courses (M_0, M_1, M_2),
2 second courses (S_0, S_1)

$n = 3$
$k = 3$

$$M_{S_1,S_2,\dots,S_{n-1}} = \prod_{i=0}^{n-1} |S_i| = 2 \cdot 3 \cdot 2 = 12$$

$$M_{S_1,S_2,\dots,S_{n-1}} = \{$$
$$(A_0, M_0, S_0), (A_0, M_0, S_1), (A_0, M_1, S_0), (A_0, M_1, S_1),$$
$$(A_0, M_2, S_0), (A_0, M_2, S_1), (A_1, M_0, S_0), (A_1, M_0, S_1),$$
$$(A_1, M_1, S_0), (A_1, M_1, S_1), (A_1, M_2, S_0), (A_1, M_2, S_1)$$
$$\}$$

Tree of degree 3, height 3,
12 paths from root to leaves



Solution

- Choices are made in sequence
  - They are represented by a tree
  - The number of choices
    - Is fixed for a level
    - Varies from level to level
  - Nodes have a number of children that varies according to the level
    - Each one of the children is one of the choices at that level
    - The maximum number of children determines the degree of the tree
  - The tree's height is n (the number of groups)

Solution

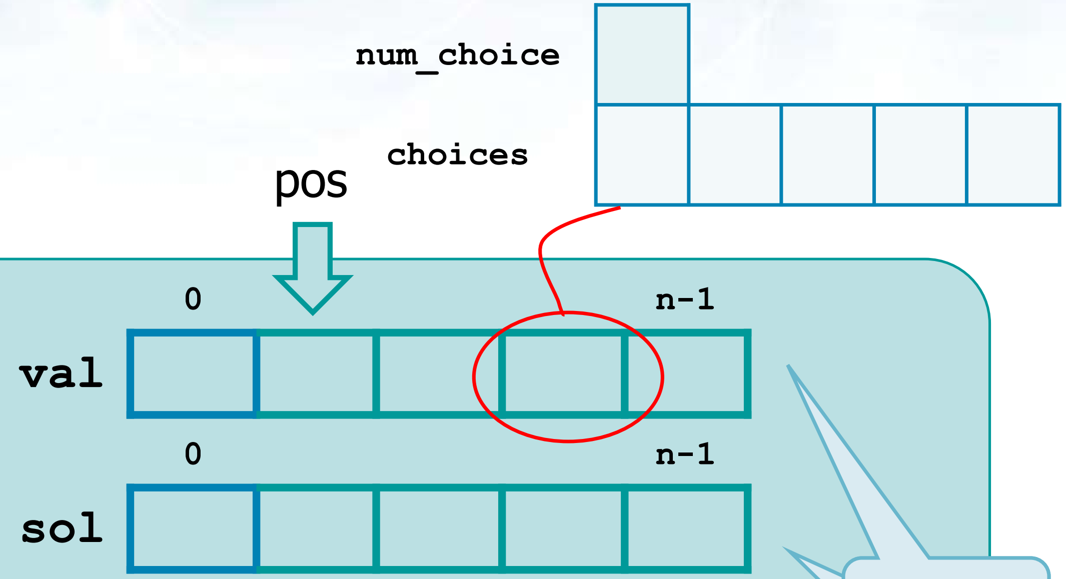
- Given the recursion tree, solutions are the labels of the edges along each path from root to node
  - The goal is to enumerate all solutions, searching their space
    - All solutions are valid
  - Each new recursive call increases the size of the solution
    - The total number of recursive calls along each path is equal to n
  - Termination
    - Size of current solution equals final desired size n

Implementation

- As far as the data-base is concerned
  - There is a 1:1 matching between choices and a (possibly non contiguous) subset of integers
  - Possible choices are stored in array **val** of size n containing structures of type **Level**
    - Each structure contains
      - An integer field **num_choice** for the number of choices at that level
      - An array ***choices** of **num_choice** integers storing the available choices
  - A solution is represented as an array **sol** of n elements that stores the choices at each step

Implementation

The check for NULL is missing



Size n

```
typedef struct val_s {
  int num_choice;
  int *choices;
} val_t;

val = malloc(n*sizeof(val_t));

for (i=0; i<n; i++)
  val[i].choices =
    malloc(val[i].n_choice*sizeof(int));

sol = malloc(n*sizeof(int));
```

Implementation

- As far as the recursive function is concerned
  - At each step index **pos** indicates the size of the partial solution
    - If **pos>=n** a solution has been found
  - The recursive step iterates on possible choices for the current value of **pos**
    - The contents of **sol[pos]** is taken from **val[pos].choices[i]** extending each time the solution's size by 1 and recurs on the pos+1-th choice
  - Variable **count** is the integer return value for the recursive function and counts the number of solutions

pos is the recursion level (level)

Implementation

```
int mult_princ (val_t *val, int *sol,
                int n, int count, int pos) {
  int i;

  if (pos >= n) {
    for (i = 0; i < n; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }
  for (i=0; i<val[pos].num_choice; i++) {
    sol[pos] = val[pos].choices[i];
    count = mult_princ (val,sol,n,count,pos+1);
  }
  return count;
}
```

Termination condition

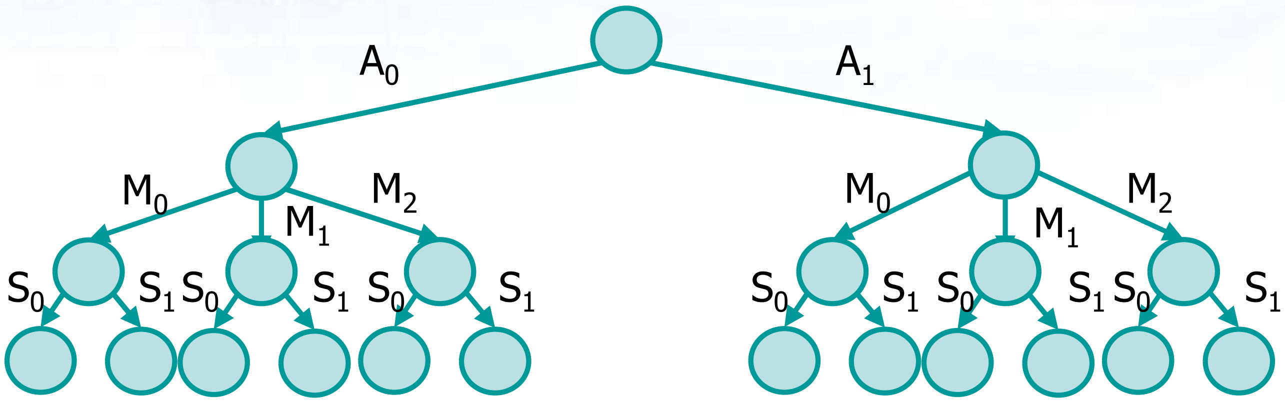
Iteration on n choices

Choose

Recur

Passing pos+1 does not modify pos at this recursion level

Implementation




```
int mult_princ (...) {
  int i;
  if (pos >= n) {
    print ...
    return count+1;
  }
  for (i=0; i<val[pos].num_choice; i++) {
    sol[pos] = val[pos].choices[i];
    count = mult_princ (...);
  }
  return count;
}
```

Simple arrangements

Simple means no repetitions

Distinct means the group is a set

- A simple arrangement $D_{n,k}$ of n distinct objects of class k is an ordered subset composed by k out of n objects $(0 \leq k \leq n)$

Order matters

Class k means size k (set taken k by k)

  - The number of simple arrangements of n objects k by k is

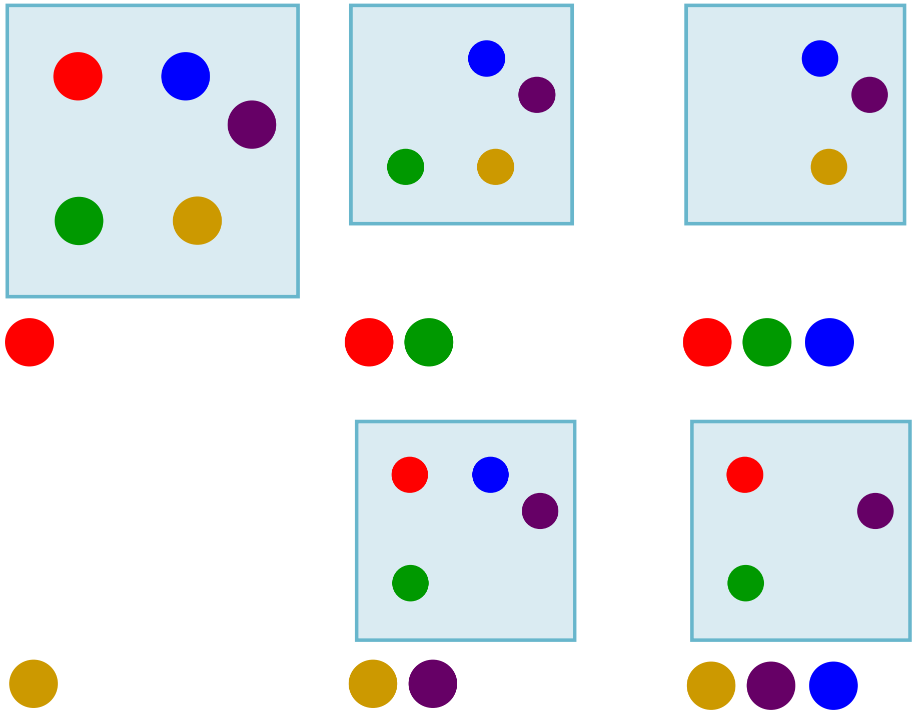
$$D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

Rationale: I select an object out of n, then I select an object out of the $n-1$ remaining, etc.

Two groups differ either because there is at least a different element or because the ordering is different

Example

- Extract 3 balls from the set



...

$$D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

$$D_{5,3} = 5 \cdot (5-1) \cdot (5-2) = \frac{5!}{(5-3)!} = 60$$

Example

Positional representation: order matters

$k = 2$

- How many and which are the numbers on two distinct digits composed with digits $\{ 4, 9, 1, 0 \}$?

No repeated digits

$val = \{ 4, 9, 1, 0 \}$

$n = 4$

$$D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

$$D_{4,2} = \frac{4!}{(4-2)!} = 4 \cdot 3 = 12$$

$$\{ 49, 41, 40, 94, 91, 90, 14, 19, 10, 04, 09, 01 \}$$

Example

Positional representation: order matters

$k = 2$

- How many strings of 2 characters can be formed selecting chars within the group of 5 vowels $\{ A, E, I, O, U \}$?

$val = \{A, E, I, O, U\}$

No repeated digits

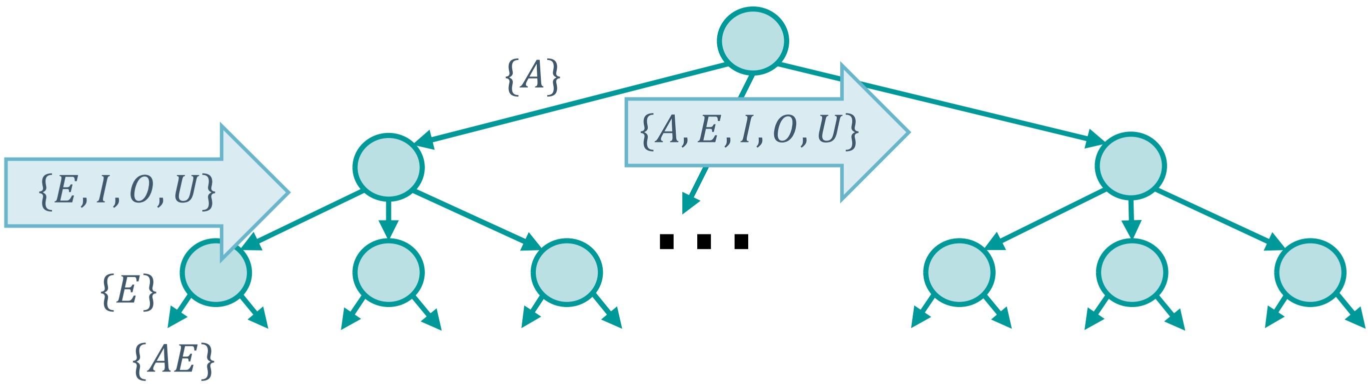
$n = 5$

$$D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

$$D_{5,2} = \frac{5!}{(5-2)!} = 5 \cdot 4 = 20$$

$\{ AE, AI, AO, AU, EA, EI,$
$EO, EU, IA, IE, IO, IU, OA, OE,$
$OI, OU, UA, UE, UI, UO \}$

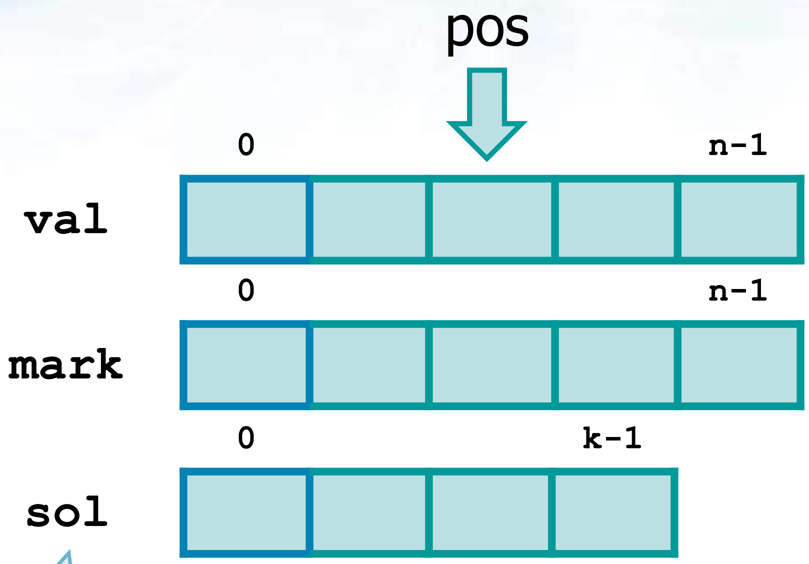
Tree of degree 5, height 2, 20 paths from root to leaves



Implementation

As for the multiplication principle with the same set to which one element is extracted, recursion level after recursion level

Size n



As the set is the same, the array **val** become an array of flags **mark**

Size k

```
val = malloc (n * sizeof(int));
mark = malloc (n * sizeof(int));

sol = malloc (k * sizeof(int));
```

Implementation

- In order not to generate repeated elements
  - An array **mark** records already taken elements
    - **mark[i]=0** implies that i-th element not yet taken, else 1
    - The cardinality of mark equals the number of elements in val (all distinct, being a set)
  - While choosing
    - The i-th element is taken only if **mark[i]==0**, **mark[i]** is assigned with 1
  - While backtracking
    - **mark[i]** is assigned with 0
  - Array **count** records the number of solutions

Implementation

```
int arr (int *val, int *sol, int *mark,
         int n, int k, int count, int pos){
  int i;

  if (pos >= k){
    for (i=0; i<k; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }
  for (i=0; i<n; i++){
    if (mark[i] == 0) {
      mark[i] = 1;
      sol[pos] = val[i];
      count = arr(val,sol,mark,n,k,count,pos+1);
      mark[i] = 0;
    }
  }
  return count;
}
```

Termination condition

Iteration on n choices

Mark and choose

beofre moving down I set the flag to true so that I dont select the same element I am working with, but just after my operations I need to reset the flag to false to use it for the other numbers

Recur

Unmark

Arrangements with repetitions

Repetitions

Set

- An arrangement with repetitions $D'_{n,k}$ of n distinct objects of class k (k by k) is an ordered_subset composed of k out of n objects ($k \geq 0$) each of whom may be taken up to k times
- The number of arrangements with repetitions of n objects taken k by k is

Order matters

$$\boxed{D'_{n,k} = n \cdot n \cdot n \cdot \ldots \cdot n = n^k}$$

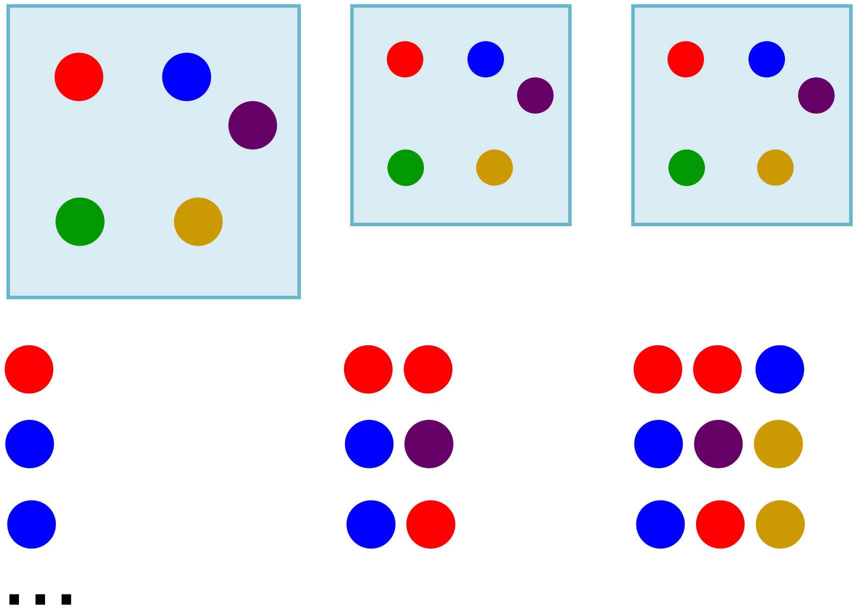
I select an object out of n, then I select an object out of n, etc.

Arrangements with repetitions

- Note that
  - Arrangements with repetitions are
    - Distinct $\Rightarrow$ the group is a set
    - Ordered $\Rightarrow$ order matters
    - As "simple" is not mentioned $\Rightarrow$ in every grouping the same object can occur repeatedly at most k times
      - k may be > n
  - Two groupings differ if one of them
    - Contains at least an object that doesn't occur in the other group or
    - Objects occur in different orders or
    - Objects that occur in one grouping occur also in the other one but are repeated a different number of times

Example

- Select 3 balls from the set (without extraction), i.e., the same ball can be selected more than once



$$D'_{n,k} = n \cdot n \cdot n \cdot \ldots \cdot n = n^k$$
$$D'_{5,3} = 5 \cdot 5 \cdot 5 = 5^3 = 125$$

Example

Positional representation: order matters!

$n = 4$

- How many binary numbers can be created with 4 bits?

$val = \{ 0, 1 \}$,
$k = 2$, repeated digits

$$D'_{n,k} = n \cdot n \cdot n \cdot \ldots \cdot n = n^k$$
$$D'_{2,4} = 2 \cdot 2 \cdot 2 \cdot 2 = 2^4 = 16$$

$$\{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$$

Example

Positional representation: order matters!

$k = 2$

- How many strings of 2 characters can be formed selecting chars with repetitions within the group of 5 vowels $\{A, E, I, O, U\}$?

$val = \{ A, E, I, O, U \}, n = 5$

Repeated digits

$$D'_{n,k} = n \cdot n \cdot n \cdot \ldots \cdot n = n^k$$
$$D'_{5,2} = 5 \cdot 5 = 5^2 = 25$$

$$\{AA, AE, AI, AO, AU, EE, EI, EO, EU, IA, IE, II, IO, IU, OA, OE, OI, OO, OU, UA, UE, UI, UO, UU\}$$

Solution

- Each element can be repeated up to k times
  - There in no bound on $\mathbf{k}$ imposed by $\boldsymbol{n}$
  - For each position we enumerate all possible choices
  - Array **count** stores the number of solutions

As the multiplication principle but extracting from the same set over and over again

As simple arrangements with NO mark array, as all elements can be selected at any level

Implementation

Multiplication: same set
Simple arragements: no mark

```
int arr_rep (int *val, int *sol,
             int n, int k, int count, int pos) {
  int i;

  if (pos >= k) {
    for (i=0; i<k; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }
  for (i=0; i<n; i++) {
    sol[pos] = val[i];
    count = arr_rep(val,sol,n,k,count,pos+1);
  }
  return count;
}
```

Termination condition

Iteration on n choices

Choose

Recur

Simple arrangements

Simple means no repetitions

Distinct means the group is a set

permutations

- A simple arrangement $D_{n,n}$ of n distinct objects of class n (n by n) is a simple permutation P_n
  - A simple permutation is an ordered subset made of n objects

As simple arrangements with $\mathrm{k} == n$ (k does not exist)

Order matters

  - The number of simple permutations of n objects is

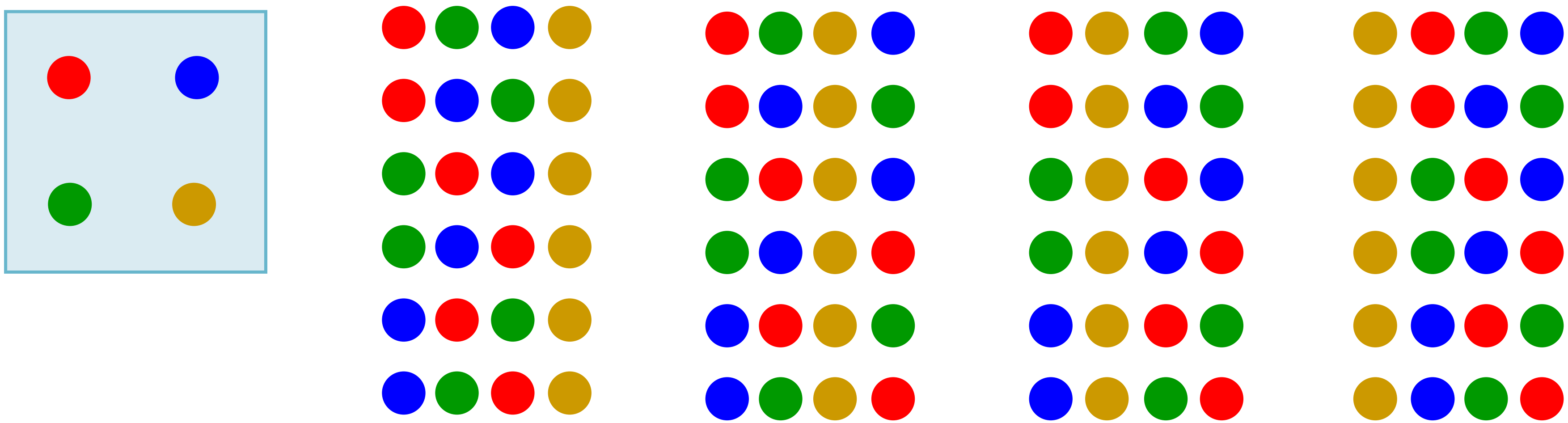
$$P_n = D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-n+1) = n!$$

Rationale: I select an object out of n, then I select an object out of the $n-1$ remaining, etc.

Two groups differ because the elements are the same, but they appear in a different order

Example

- Generate all permutations of the four balls in the box



$$P_n = D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-n+1) = n!$$
$$P_3 = D_{3,3} = 4 \cdot (4-1) \cdot (4-2) \cdot (4-3) = 4 \cdot 3 \cdot 2 \cdot 1 = 4! = 24$$

Example

Positional representation: order matters!

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

- Given a set val of 3 integers, generate all possible numbers containing these 3 digits once

No repetition

$n = 3$

$$P_n = D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-n+1) = n!$$
$$P_3 = 3! = 6$$

$$\{ 123, 132, 213, 312, 231, 321 \}$$

Example

Positional representation: order matters!

$val = \{ O, R, A \}$

- How many and which are the anagrams of the string ORA (string of 3 distinct letters)?

$n = 3$

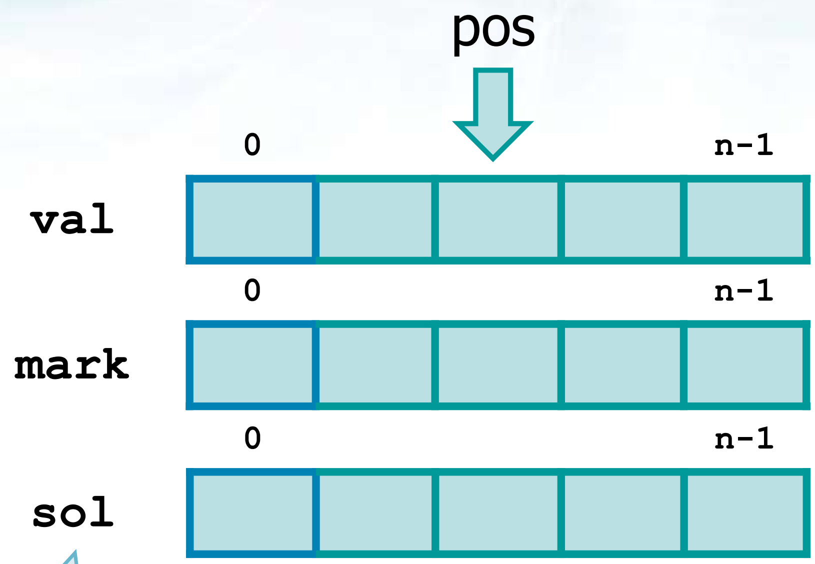
No repetition

$$P_n = D_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-n+1) = n!$$
$$P_3 = 3! = 6$$

$$\{ ORA, OAR, ROA, AOR, RAO, ARO \}$$

Implementation

As simple arrangements
with $k == n$
(we select n elements
out of n)



Size n

Don't forget to
check for NULL

```
val = malloc (n * sizeof(int));
sol = malloc (n * sizeof(int));
mark = malloc (n * sizeof(int));
```

Solution

- In order not to generate repeated elements
  - An array **mark** records already taken elements
    - **mark[i]=0** implies that the i-th element not yet taken, else 1
    - The cardinality of **mark** equals the number of elements in **val** (all distinct, being a set)
  - While choosing
    - The i-th element is taken only if **mark[i]==0**, **mark[i]** is assigned with 1
  - During backtrack
    - **mark[i]** is assigned with 0
  - **Count** stores the number of solutions

Implementation

As simple arrangements with $k == n$

```
int perm (int *val, int *sol, int *mark,
          int n, int count, int pos){
  int i;
  if (pos >= n){
    for (i=0; i<n; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }
  for (i=0; i<n; i++)
    if (mark[i] == 0) {
      mark[i] = 1;
      sol[pos] = val[i];
      count = perm(val,sol,mark,n,count,pos+1);
      mark[i] = 0;
  }
  return count;
}
```

Termination condition

Iteration on n choices

Mark and choose

Unmark

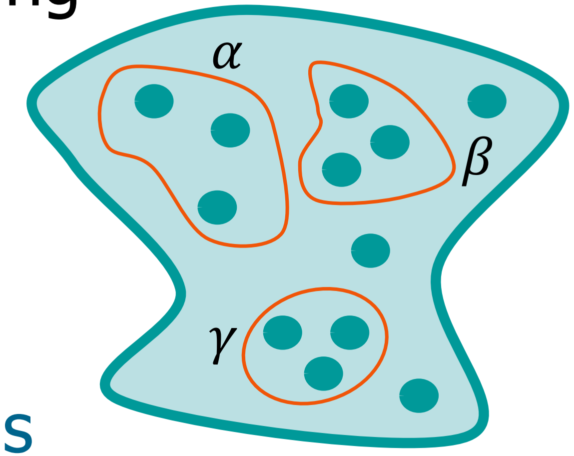
Recur

Permutations with repetitions

Repeated elements
Distinct is not mentioned
The group is a multiset

Permutations:
order matters

- Given a set (multiset) of n objects among which
  - α objects are identical
  - β objects are identical
  - etc.
  - The number of distinct permutations with repeated objects is



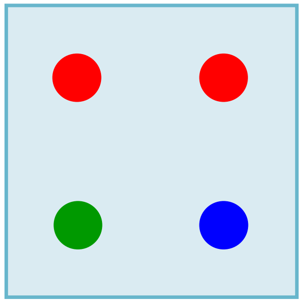
$$P_n^{(\alpha,\beta,\dots)} = \frac{n!}{\alpha!\ \beta!\ \gamma!\ \dots}$$

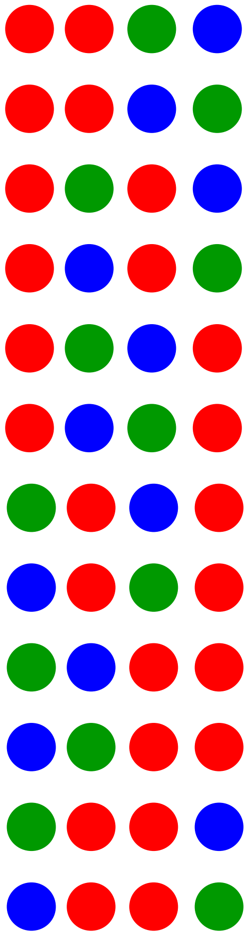
Rationale: From permutation
$P_n = n!$
divided by the permutations of the repeated objects

Two groups differ because the elements are the same but are repeated a different number of times or because the order differs

Example

- Generate all permutations wit repetitions of the four balls in the box





$$P_n^{(\alpha,\beta,\dots)} = \frac{n!}{\alpha!\ \beta!\ \gamma!\ \dots}$$

$$P_4^{(2)} = \frac{n!}{\alpha!} = \frac{4!}{2!} = 12$$

Example

Positional representation: order matters!

- How many and which are the distinct anagrams of string ORO (string of 3 characters, 2 being identical)?

n = 3

α = 2

$$P_n^{(\alpha,\beta,\dots)} = \frac{n!}{\alpha!\ \beta!\ \gamma!\ \dots}$$

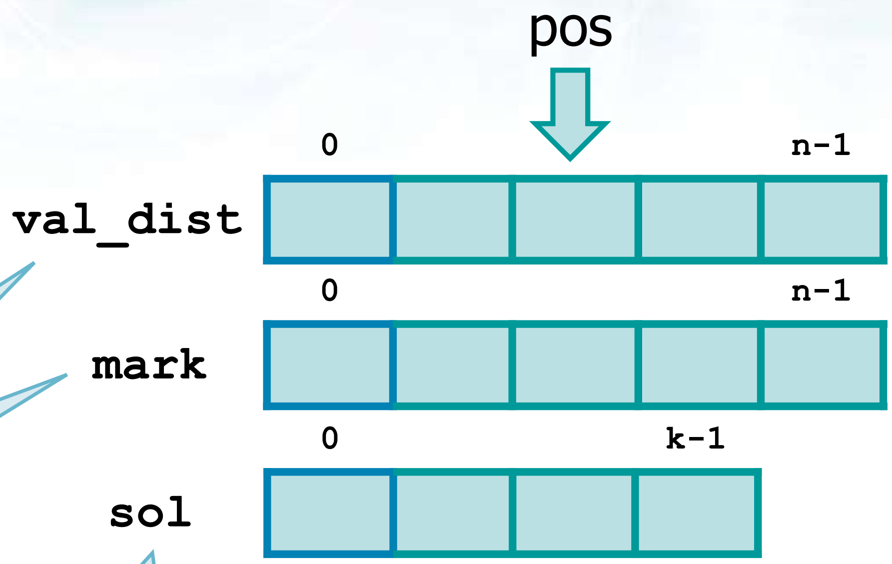
$$P_4^{(2)} = \frac{n!}{\alpha!} = \frac{3!}{2!} = 3$$

$$\{\,OOR, ORO, ROO\}$$

O_1O_2R and O_2O_1R

Implementation

As simple arrangements but **mark** is an array of counters not of flags and there are **val_dist** distinct values



Size n

Size k

Don't forget to check for NULL

```
val_dist = malloc (n_dist*sizeof(int));
mark = malloc (n_dist*sizeof(int));

sol = malloc (k*sizeof(int));
```

Implementation

- As far as the data-base is concerned
  - It is the same as for simple permutations, with these changes
    - **n** is the cardinality of the multiset
    - **n_dist** is the number of distinct elements of the multiset
    - **val** is the set of (n) elements in the multuise4t
    - **val_dist** is the set of (n_dist) distinct elements of the multiset
    - **count** stores the number of solutions
  - Element **val_dist[i]** is taken if **mark[i]> 0**, **mark[i]** is decremented

Implementation

As simple arrangements but **mark** is an array of counters

```
int perm_rep (int *val_dist, int *sol, int *mark,
  int n, int n_dist, int count, int pos) {
  int i;
  if (pos >= n) {
    for (i=0; i<n; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }
  for (i=0; i<n_dist; i++) {
    if (mark[i] > 0) {
      mark[i]--;
      sol[pos] = val_dist[i];
      count = perm_rep (
        val_dist,sol,mark,n,n_dist,count,pos+1);
      mark[i]++;
    }
  }
  return count;
}
```

Termination condition

Iteration on n_dist choices

Occurrence control

Mark and choose

Recur

Unmark

Simple combinations

Simple means no repetitions

Distinct means the group is a set

- A simple combination $C_{n,k}$ of n distinct objects of class k (k by k) is a non ordered subset composed by k of n objects ($0 \leq k \leq n$)

Order does not matters

  - The number of combinations of n elements k by k

$$C_{n,k} = \frac{D_{n,k}}{P_k} = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

For the first time order does not matter !

Number of arrangements of n elements k by k divided by the number of permutations of k elements

Binomial coefficient (n choose k , $k \leq n$)

Two groups differ because there is at least a different element

Example

Order does not matter

$k = 3$

- How many sets of 3 characters can be formed with the 4 characters $\{ A, B, C, D \}$?

$val = \{A, B, C, D\}$ and $n = 4$

$$C_{n,k} = \frac{D_{n,k}}{P_k} = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

$$C_{4,3} = \frac{D_{4,3}}{P_3} = \binom{4}{3} = \frac{4!}{3!\,(4-3)!} = 4$$

$$\{ ABC, ABD, ACD, BCD \}$$

Example

Order does not matter

$k = 4$

- How many sets of 4 digits can be formed with the 5 digits $\{ 7, 2, 0, 4, 1 \}$?

$val = \{7, 2, 0, 4, 1\}$ *and* $n = 5$

$$C_{n,k} = \frac{D_{n,k}}{P_k} = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

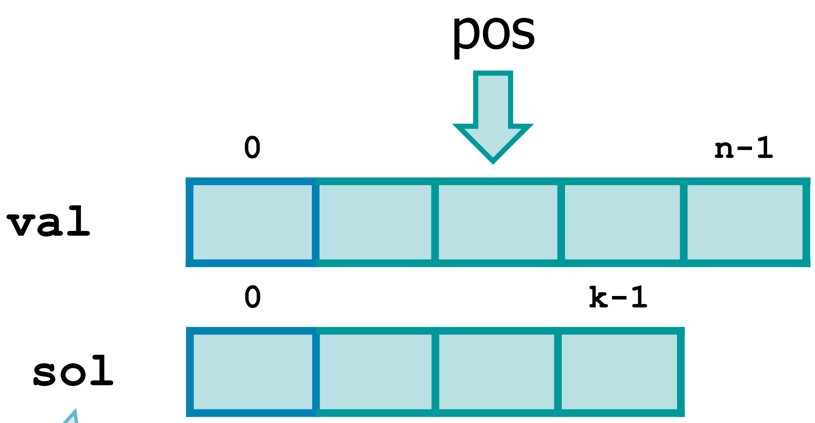
$$C_{5,4} = \frac{D_{5,4}}{P_4} = \binom{5}{4} = \frac{5!}{4!\,(5-4)!} = 5$$

$$\{ 7204, 7201, 7241, 7041, 2041 \}$$

Implementation

As simple arrangements but **mark** does not exist and we begin from **start** at each selection iteration

Size n



Size k

Don't forget to check for NULL

```
val = malloc (n * sizeof(int));
sol = malloc (k * sizeof(int));
```

Implementation

- With respect to simple arrangements it is necessary to "force" one of the possible orderings
  - Index **start** determines from which value of **val** we start to fill-in **sol**
  - Array
    - **val** is visited thanks to index **i** starting from **start**
    - **sol** is assigned starting from index **pos** with possible values of **val** from start onwards
    - Once value **val[i]** is assigned to **sol**, recur with **i+1** and **pos+1**
    - **mark** is not needed
  - Variable count stores the number of solutions

Implementation

As simple arrangements but **start** forces a specific order

```
int comb (int *val, int *sol, int n, int k,
          int start, int count, int pos) {
  int i, j;

  if (pos >= k) {
    for (i=0; i<k; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }

  for (i=start; i<n; i++) {
    sol[pos] = val[i];
    count = comb(val,sol,n,k,i+1,count,pos+1);
  }
  return count;
}
```

Termination condition

Iteration on n choices

sol[pos] filled with possible values of val from start onwards

Recur (next position and next choice)

Combinations with repetition

- In the combinations with repetition, we
  - Suppose there are n elements in a set S
  - Select k elements from this set, given that each element can be selected multiple times
- In other words
  - A combination with repetitions $\boldsymbol{C'_{n,k}}$ of n distinct objects of class k is a non ordered subset made of k of the n objects ($k \geq 0$)

1. The generated set must be distinct

2. Order does not matter

3. Each element can be repeated

4. No upper bound for k, k can be larger than n

Combinations with repetition

- Note that
  - The combinations with repetition are
    - Distinct, i.e., the group is a set
    - Unordered, i.e., order does not matter
    - With repetition, i.e., "simple« is not mentioned and in each group the same object may occur repeatedly
    - Unlimited, i.e., k may be larger than n
  - Two groups differ if
    - One of them contains at least an object that doen't occur in the other one
    - The objects that appear in one group appear also in the other one but are repeated a different number of times

How many $C'_{n,k}$ do we have?

- The number of combinations with repetitions of n objects k by k is

$$C'_{n,k} = \binom{n+\mathrm{k}-1}{k} = \binom{n+\mathrm{k}-1}{n-1} = \frac{(n+k-1)!}{k!\cdot(n-1)!}$$

- Can we prove it?
  - Let's try to use an example and work it out

How many $C'_{n,k}$ do we have?

$n = 5$

- Spec
  - Let us say there are five flavors of icecream
    - banana, chocolate, lemon, strawberry, vanilla
  - We can have three scoops
  - How many variations will there be?

$k = 3$

$C'_{n,k} = ?$

- Proof
  - Let's use letters for the flavors
    - {b, c, l, s, v}
  - Let’s suppose ice cream being in boxes
    - Thus to select {c, c, c} (3 scoops of chocolate), we
      - Move past the first box, then take 3 scoops, then move along 3 more boxes to the end

How many $C'_{n,k}$ do we have?

- Thus, to select {c, c, c} we can write down

→	o	o	o	→	→	→

Move onto the next box

Take a scoop

- In how many different ways can we arrange arrows and circles?

$(n-1)$ arrows
k circles

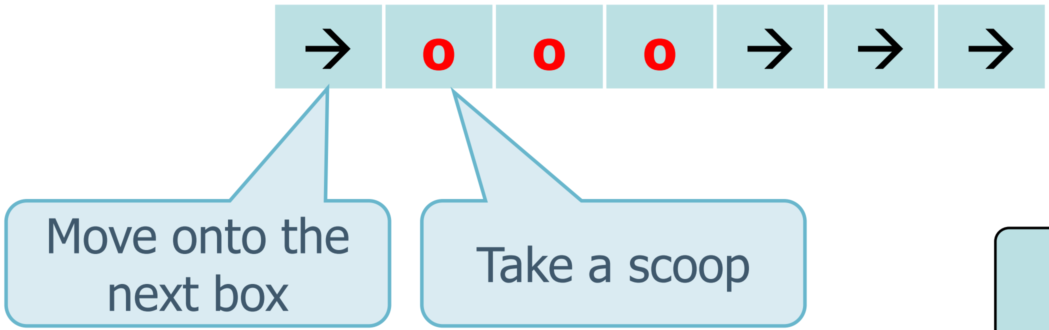
  - We have $((n-1)+k)$ positions $((5-1)+3))$
  - We want to choose k circles out of them
  - Thus, we have a number of possibilities equal to

$$\boxed{\binom{n+\mathrm{k}-1}{k}}$$

Simple combinations but with different numbers

How many $C'_{n,k}$ do we have?

- Interestingly, we can look at the arrows instead of the circles, i.e., we can choose n-1 arrows



$$\binom{n+\mathrm{k}-1}{n-1}$$

➢ Thus

$$C'_{n,k} = \binom{n+\mathrm{k}-1}{k} = \binom{n+\mathrm{k}-1}{n-1} = \frac{(n+k-1)!}{k!\cdot(n-1)!}$$

Example

$n = 6$

- When simultaneously casting two dices, how many compositions of values may appear on two faces?

$k = 2$

$$C'_{n,k} = \binom{n+\mathrm{k}-1}{k} = \binom{n+\mathrm{k}-1}{n-1} = \frac{(n+k-1)!}{k!\cdot(n-1)!}$$

$$C'_{6,2} = \frac{(6+2-1)!}{2!\cdot(6-1)!} = 21$$

$$\{\,11, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26,$$
$$33, 34, 35, 36, 44, 45, 46, 55, 56, 66\,\}$$

Example

$n = 5$

- There are five colored balls in a pool
  - black, white, red, green, yellow
- All balls are of different colors. The selection of a ball can be repeated. In how many ways can we choose four pool balls?

$k = 4$

$$C'_{n,k} = \binom{n + k - 1}{k} = \binom{n + k - 1}{n - 1} = \frac{(n+k-1)!}{k! \cdot (n-1)!}$$

$$C'_{5,4} = \frac{(5+4-1)!}{4! \cdot (5-1)!} = 70$$

$$\{\, bbbb, wwww, etc. \}$$

Example

$n = 8$

- There are eight different ice-cream flavors in the ice-cream shop. One ice-cream flavor can be selected multiple times
- In how many ways can we choose five flavors out of these eight flavors?

$k = 5$

$$C'_{n,k} = \binom{n+\mathrm{k}-1}{k} = \binom{n+\mathrm{k}-1}{n-1} = \frac{(n+k-1)!}{k!\cdot(n-1)!}$$

$$C'_{8,5} = \frac{(8+5-1)!}{5!\cdot(8-1)!} = 792$$

$$\{\, ccccc, bbbbb, etc. \}$$

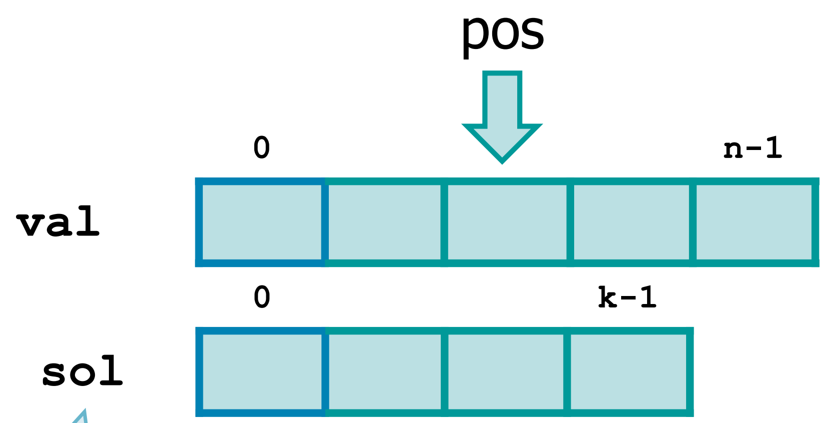
Solution

- Same as simple combinations, but
  - Recursion occurs only for **pos+1** and not for **i+1**
  - Index **start** is incremented each time the for loop on choices
  - **count** records the number of solutions

Implementation

As simple combinations but **i** is not incremented when recurring to re-consider the same object over and over again

Size n



Size k

Don't forget to check for NULL

```
val = malloc(n * sizeof(int));
sol = malloc(k * sizeof(int));
```

Implementation

As simple combinations but we must re-consider the same object

```
int comb_rep (int *val, int *sol, int n, int k,
              int start, int count, int pos) {
  int i, j;

  if (pos >= k) {
    for (i=0; i<k; i++)
      printf("%d ", sol[i]);
    printf("\n");
    return count+1;
  }

  for (i=start; i<n; i++) {
    sol[pos] = val[i];
    count = comb_rep(val,sol,n,k,i,count,pos+1);
  }
  return count;
}
```

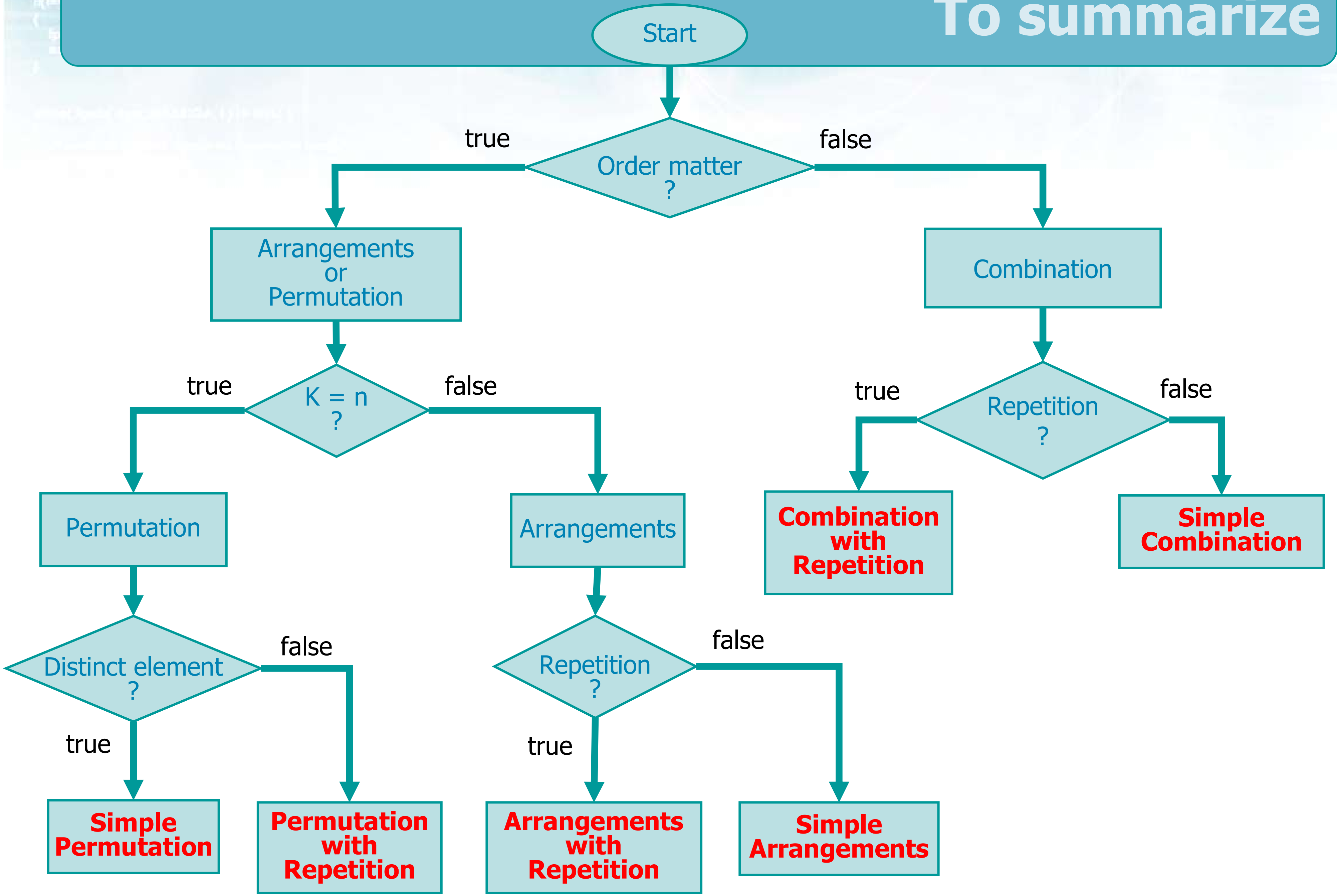
Termination condition

Iteration on n choices

sol[pos] filled with possible values of val from start onwards

Recur (next position)

To summarize



To summarize

```
for (i=0; i<val[pos].num_choice; i++) {
  sol[pos] = val[pos].choices[i];
  count = mult_princ (val,sol,n,count,pos+1);
}
```

Multiplication principle

Simple arrangements

```
for (i=0; i<n; i++){
  if (mark[i] == 0) {
    mark[i] = 1;
    sol[pos] = val[i];
    count = arr(val,sol,mark,n,k,count,pos+1);
    mark[i] = 0;
  }
}
```

Arrangements with repetitions

```
for (i=0; i<n; i++) {
  sol[pos] = val[i];
  count = arr_rep(val,sol,n,k,count,pos+1);
}
```

To summarize

Simple permutations

```
for (i=0; i<n; i++)
  if (mark[i] == 0) {
    mark[i] = 1;
    sol[pos] = val[i];
    count = perm(val,sol,mark,n,count,pos+1);
    mark[i] = 0;
}
```

Permutations with repetitions

```
for (i=0; i<n_dist; i++) {
  if (mark[i] > 0) {
    mark[i]--;
    sol[pos] = val_dist[i];
    count = perm_rep (
      val_dist,sol,mark,n,n_dist,count,pos+1);
    mark[i]++;
  }
}
```

Simple combinations

```
for (i=start; i<n; i++) {
  sol[pos] = val[i];
  count = comb(val,sol,n,k,i+1,count,pos+1);
}
```

Combinations with repetitions

```
for (i=start; i<n; i++) {
  sol[pos] = val[i];
  count = comb_rep(val,sol,n,k,i,count,pos+1);
}
```