Brute Force Algorithm

Direct approach in solving problem

Nattee Niparnan

Department of Computer Engineering
Chulalongkorn University

Key Concept

- This is general problem-solving technique
  - Work with very broad class of problems called constraint satisfactory problem (CSP) and its generalization called constraint optimization problem (COP)
  - Most problems can be modelled as CSPs
- Brute Force is a fundamental tools for solving several problems, however, Brute Force is usually inefficient (slow)
  - Work by defining a set of all candidate solutions of the problem instance then enumerating each solution and check if it satisfies the requirement on the problem
  - Enumeration can be done easily by recursive
- Has many improvements and extension (cover later in the class)
  - Backtracking
  - Branch-and-bound

Constraint Satisfaction Problem (CSP)

- The problem must also give the set of possible value of each input variables (maybe implicitly)
  - This is usually very common in any problem
- The problem must give the constraints that we have to satisfy (usually over a set of variables that describes the output)
- Many problem may not be directly described as a CSP, but we can formulate it as one.

Example: Find a value in an array

- Task: Finding a position of a value in an array
- Input: Array A[1..n] and a value k
- Output: an integer i (in the range of 1 to n) such that A[i] = k, or 0 when no such i exists
- Example Instance:
    - A = [1, -3, 5, 2, 3, 1, 5, 7, 9, 11, 4]
    - K = 5

Formulating a problem as a CSP

- Must define a description of a candidate solution
  - Usually, this is the same as an output
- Must define a set of candidate solution
  - Usually, this is given as a range (or set) of possible value of each variable in the output
- Must define constraints,
  - Define in a way that we can check if a candidate solution satisfies the constraints
  - Usually, this mean we can write a code to check it
- There can be multiple way to formulate a problem as a CSP

Example: Find a value in an array

- Task: Finding a position of a value in an array
- Input: Array A[1..n] and a value k
- Output: an integer i in the range 1 to n such that A[i] = k, or 0 when no such i exists

Candidate Solution	Set of candidate solution	Satisfaction condition
A single integer i	$\{0,1,\ldots,N\}$	When i > 0, A[i] = k When i = 0, there must be no k in A

Using Brute Force to solve a problem

- Let S be a set of candidate solutions
- Let T(x) be a function that test if a candidate solution x satisfies all constraints

```
def brute_force(S,T)
  for each x in S
    if T(x)
      return x
```

In practice, we need to write a code that enumerate all candidate solution and test according to the input of the problem

- That’s it
- O(|S| * O(T))

Example: Find a value in an array

- Task: Finding a position of a value in an array
- Input: Array A[1..n] and a value k
- Output: an integer i in the range 1 to n such that A[i] = k, or 0 when no such i exists

```
def find(A,k)
  for i from 1 to A.length
    if A[i] = k
      return i
```

Non unique solutions

- It is possible that there are non unique solutions in the candidate solution set that satisfy the constraints

A = [1, -3, 5, 2, 3, 1, 5, 7, 9, 11, 4]
K = 5

The solution can be either 3 or 7
because A[3] = 5 and A[7] = 5

Example: Find a pair sum equal to K

- Task: Given an array, find two distinct elements in the array such that its summation is equal to k
- Input: A[1..n], k
- Output
  - Two integers, p and q such that A[p] + A[q] = k and p != q
  - Two integers, 0 and 0 when we cannot find such p and q

Candidate Solution	Set of candidate solution	Satisfaction condition
(p,q)	$\{(1,1),(1,2),\dots(1,N),$ $(2,1),(2,2),\dots(2,N),$ $\dots,$ $(N,1),(N,2),\dots,(N,N),(0,0)\}$	p != q When p != 0 and q != 0, A[p] + A[q] = k When p = 0 and q = 0, there is no other member in the candidate solution that satisfy A[p] + A[q] = k

Constraint and Set of candidate solutions

- Set of candidate solutions and constraints are often related
- One problem can be formulated with different constraints and set of candidate solutions
- For example, consider a pair sum equal to K problem

Larger set, need more time to enumerate

Candidate Solution	Set of candidate solution	Satisfaction condition
(p,q)	$\{(1,1),(1,2),\ldots(1,N),$ $(2,1),(2,2),\ldots(2,N),$ $\ldots,$ $(N,1),(N,2),\ldots(N,N),(0,0)\}$	p != q When p != 0 and q != 0, A[p] + A[q] = k When p = 0 and q = 0, there is no other member in the candidate solution that satisfy A[p] + A[q] = k

Candidate Solution	Set of candidate solution	Satisfaction condition
(p,q)	$\{(1,2),(1,3),\ldots(1,N),$ $(2,3),(2,4),\ldots(2,N),$ $\ldots,$ $(N-1,N),(0,0)\}$	When p != 0 and q != 0, A[p] + A[q] = k When p = 0 and q = 0, there is no other member in the candidate solution that satisfy A[p] + A[q] = k

Example: Common Divisor

- Task: Find any common divisor
- Input: Two positive integers A and B
- Output: a positive integer d such that A % d == 0 and B % d == 0

Candidate Solution	Set of candidate solution	Satisfaction condition
d	$\{1, \ldots, \min(A, B)\}$	A % d == 0 and B % d = 0

Constraint Optimization Problem (COP)

- An extension to CSP by including an objective function in the problem
- The goal is not only to find a solution that satisfies all constraints, but the solution must give minimal (or maximal) value of the objective function over all satisfied solution

Example: Greatest Common Divisor

- Task: Find a maximum common divisor
- Input: Two positive integers A and B
- Output: a positive integer d such that A % d == 0 and B % d == 0 that is maximum
- Objective function: f(d) = d
    - (we just need a maximum value of the output)

Candidate Solution	Set of candidate solution	Satisfaction condition
d	$\{1, \ldots, \min(A, B)\}$	A % d == 0 and B % d = 0 d is maximal

Using Brute Force for COP

- Let S be a set of candidate solutions
- Let T(x) be a test function
- Let O(x) be an objective function
- Very similar to CSP
  - But we must enumerate every member of S
    - Or find some way to guarantee that the value of O(x) is optimal

```
def brute_force_otp(S,T,O)
  best = INFINITY
  for each x in S
    if T(x) && O(x) < best
      best = O(x)
      best_answer = x
  return best_answer
```

Example: Maximum Different Value in an Array

- Task: Find two different elements in the array such that their different is maximum
- Input: A[1..n]
- Output: Two integers, p and q such that p != q
- Objective function: f(p,q) = |A[p] – A[q]|

```
def two_diff(A)
  max_diff = 0
  ans = nil
  for i in 1..(n-1)
    for j in (i+1)..n
      diff = abs(A[i]-A[j])
      if diff > max_diff
        max_diff = diff
        ans = [i,j]
  return ans
end
```

Exercise

- Write
  - Definition of a candidate solution
  - A candidate solution set
  - A function to check if a candidate solution is the one that we want

Ex1

- Task: find a perfect number in the range a to b
- Input: two integers a and b
- Output: and integer x that a <= x <= b and x is perfect (sum of its sum of its positive divisor equal to itself, e.g., 6 is a perfect number because 1+2+3 = 6)

Ex2

- Task: find smallest rectangle that contains all points in a grid map
- Input: A 2D array A[1..R][1..C] where A[i][j] is either true or false
  - A is a grid map
  - A[i][j] indicates whether coordinate (i,j) has a point
- Output: (r1,c1) and (r2,c2) such that for every (i,j) that A[i][j] is true, (r1 <= i <= r2) and (c1 <= j <= c2)

Ex3

- Task: Maximum sum in range
- Input: An array A[1..n] and an integer w
- Output: an index b such that sum of A[b] + A[b+1] + … + A[b+w-1] is maximal

Combination and Permutation

Candidate Set based on perm and combi

- Often, the candidate set consists of permutations of a sequence, or a combination of a set
- Permutation of a sequence is an arrangement of a sequence
  - E.g., for a sequence [1,2,3], there are 6 permutations: [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]
- Combination of a set is a selection of members of the set
  - E.g., for a set {a,b,c}, there are 8 combinations of its members, {}, {a}, {b}, {c}, {a,b}, {b,c}, {a,c}, {a,b,c}
- Enumerating all combinations or permutation can be done easily by recursion

Combination Example

- Subset sum problem
- Task: find a subset of a given array such that its sum is K
- Input: An array A, an integer K
- Output: a set $\{i_1,i_2,..,i_m\}$ such that $A[i_1] + A[i_2] +...+A[i_m] = K$

Candidate Solution	Set of candidate solution	Satisfaction condition
$\{i_1,i_2,..,i_m\}$	Power set of {1,2,...,N}	$A[i_1] + A[i_2] +...+A[i_m] = k$

Example Instance

- Ex1:
  - A = [9,4,5], K = 9
  - Solution
    - {1} (A[1] = 9)
    - {2,3} (A[2]+a[3] = 9)
- Ex2:
  - A = [10,40,30,20], k = 60
  - Solution
    - {2,4} (a[2] + a[4] = 60)
    - {1,3,4} (a[1] + a[3] + a[4] = 60)

A[1]	A[2]	A[3]	Candidate solution
✓			
	✓		
✓	✓		
		✓	
✓		✓	
	✓	✓	
✓	✓	✓	

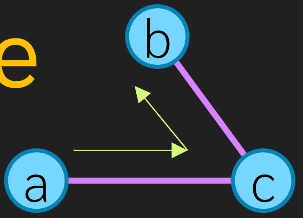
A[1]	A[2]	A[3]	A[4]	Candidate solution
				{ }
✓				{1}
	✓			{2}
✓	✓			{1,2}
		✓		{3}
✓		✓		{1,3}
	✓	✓		{2,3}
✓	✓	✓		{1,2,3}
			✓	{4}
✓			✓	{1,4}
	✓		✓	{2,4}
✓	✓		✓	{1,2,4}
		✓	✓	{3,4}
✓		✓	✓	{1,3,4}
	✓	✓	✓	{2,3,4}
✓	✓	✓	✓	{1,2,3,4}

Permutation Example

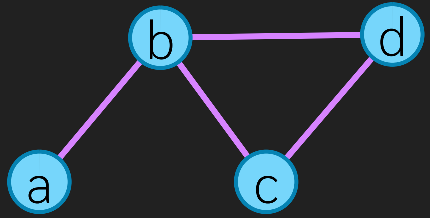
- Task: find a path in a graph
- Input: A graph $G=(V,E)$, two vertices p and q
- Output: A path in the graph start with p and end with q

Candidate Solution	Set of candidate solution	Satisfaction condition
$[v_1,v_2,..,v_k]$	Every permutation of size 1 .. $\|V\|$ of vertices	(v_i,v_{i+1}) is an edge for every i from 1 to k-1 $V_1 = p$ $V_k = q$

Example Instance



- Ex1: V E
  - G = ({a,b,c},{(a,c),(c,b)})
  - p = a, q = b
  - Solution
    - [a,c,b]
- Ex2: V E
  - G = ({a,b,c,d},{(a,b),(b,d),(b,c),(c,d)})
  - p = a, q = d
  - Solution
    - [a,b,c,d]
    - [a,b,d]



Path length	Candidate solution
1	[a], [b], [c],
2	[a,b],[a,c],[b,a],[b,c],[c,a],[c,b]
3	[a,b,c], [a,c,b], [b,a,c], [b,c,a], [c,a,b], [c,b,a]

Path length	Candidate solution
1	[a], [b], [c], [d]
2	[a,b],[a,c],[b,a],[b,c],[c,a],[c,b] [a,d],[b,d],[c,d],[d,a],[d,b],[d,c]
3	[a,b,c], [a,c,b], [b,a,c], [b,c,a], [c,a,b], [c,b,a] [a,b,d], [a,c,d], [a,d,b], [a,d,c], [b,a,d], [b,c,d], [b,d,a], [b,d,c], [c,a,d], [c,b,d], [,c,d,a], [c,d,b]
4	[a,b,c,d],[a,b,c,d],[a,c,b,d],[a,c,d,b],[a,d,b,c],[a,d,c,b], [b,a,c,d],[b,a,d,c],[b,c,a,d],[b,c,d,a],[b,c,a,d],[b,c,d,a], [c,a,b,d],[c,a,d,b],[c,b,a,d],[c,b,d,a],[c,d,a,b],[c,d,b,a], [d,a,b,c],[d,a,c,b],[d,b,a,c],[d,b,c,a],[d,c,a,b],[d,c,b,a]

Generating all combinations

- We have N items, we want to generate all combinations of these items
- Recursive Programming
  - Very similar to the binary counter in the complexity analysis topics
  - At ith step, we decides if the ith item is selected
- combination(len, sol)
  - Array sol (sol[i] == true when we use ith item)
  - Start by call `combination(N,[])`
  - Each candidate solution is enumerated every time we reach the else block

```
def combination(N,sol)
  if sol.length < N
    sol_a = sol + [0]
    combination(N, sol_a)
    sol_b = sol + [1]
    combination(N, sol_b)
  else
    #sol is array of length N
    #sol[i] = 1 when we pick item I
    print sol
    #each candidate solution is here
  end
end
```

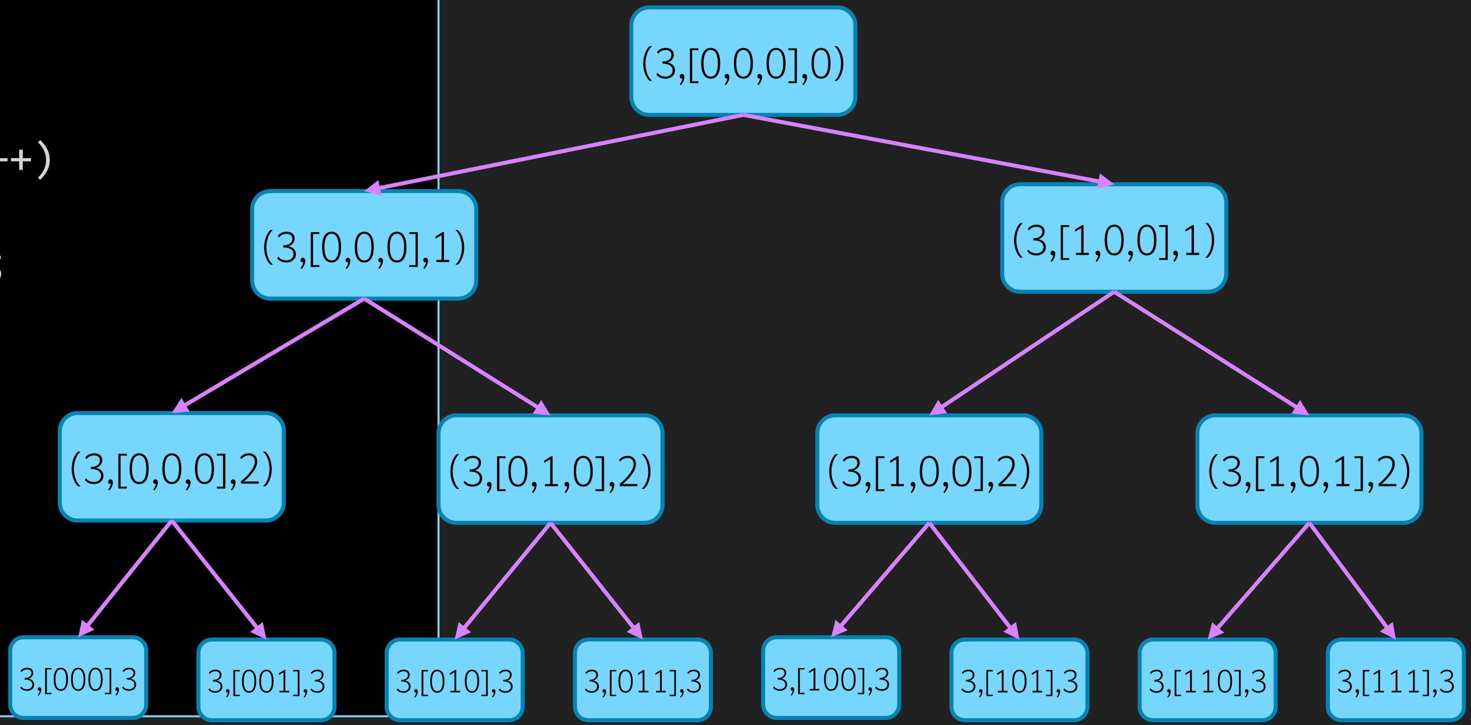
Gen combination (c++)

```cpp
#include <iostream>
#include <vector>
using namespace std;

void combi(int n,vector<int> &sol,int len) {
  if (len < n) {
    sol[len] = 0;
    combi(n,sol,len+1);
    sol[len] = 1;
    combi(n,sol,len+1);
  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << "." << endl;
  }
}

int main() {
  vector<int> sol(3);
  combi(3,sol,0);
}
```

- Slightly different from the pseudo-code
  - Create the array with large enough size
  - len indicates the current actual size
  - Use pass-by-reference to speed up

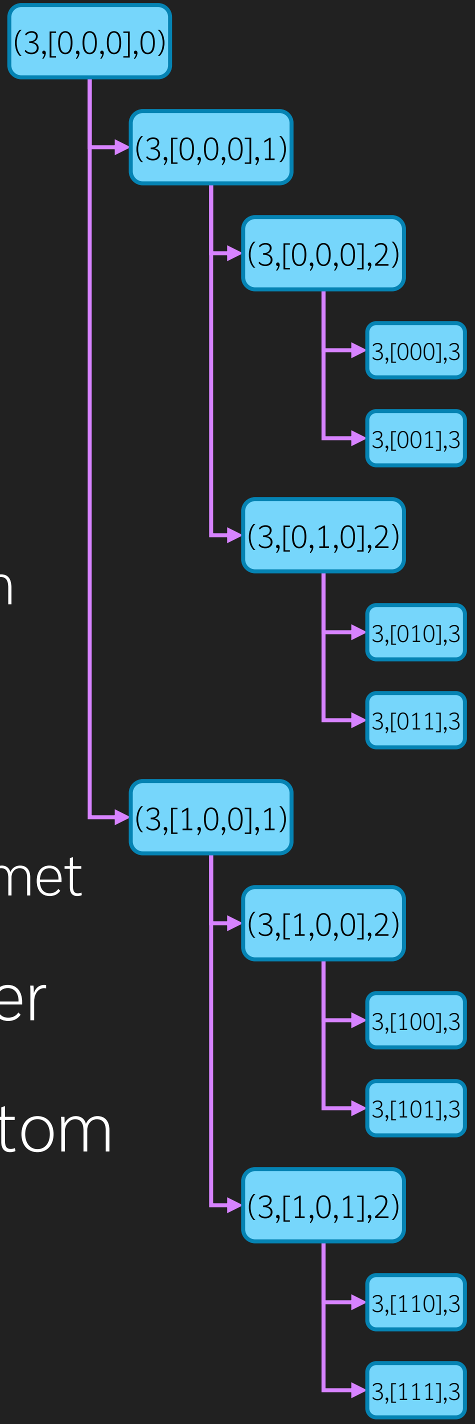


output

```
.
3 .
2 .
2 3 .
1 .
1 3 .
1 2 .
1 2 3 .
```

Recursion Tree

- A tree that display function calling
- Nodes = each function call
  - Put parameters (or related input) in a node, can omit irrelevant one
  - Root node display the first function call
  - Leaf nodes are where terminating condition is met
- Directed edges = associate calling and caller
- Can draw one node per line and top-to-bottom to emphasize order of calling



Exercise

```
void combi(int n,vector<int> &sol,int len) {
  if (len < n) {
    sol[len] = 0;
    combi(n,sol,len+1);         A
    sol[len] = 1;
    combi(n,sol,len+1);         B
  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << "." << endl;
  }
}

int main() {
  vector<int> sol(3);
  combi(3,sol,0);
}
```

- What happen when we swap A and B
  - what is the output
  - Can we draw a recursion tree

Generating all permutations

```
def permutation(N,sol)
  if sol.length < N
    for i in {1..N}
      if there is no i in sol
        sol_new = sol + [i]
        permutation(N,sol_new)
  else
    #sol is array of length N
    #sol[i] = 1 when we pick item i
    print sol
  end
end
```

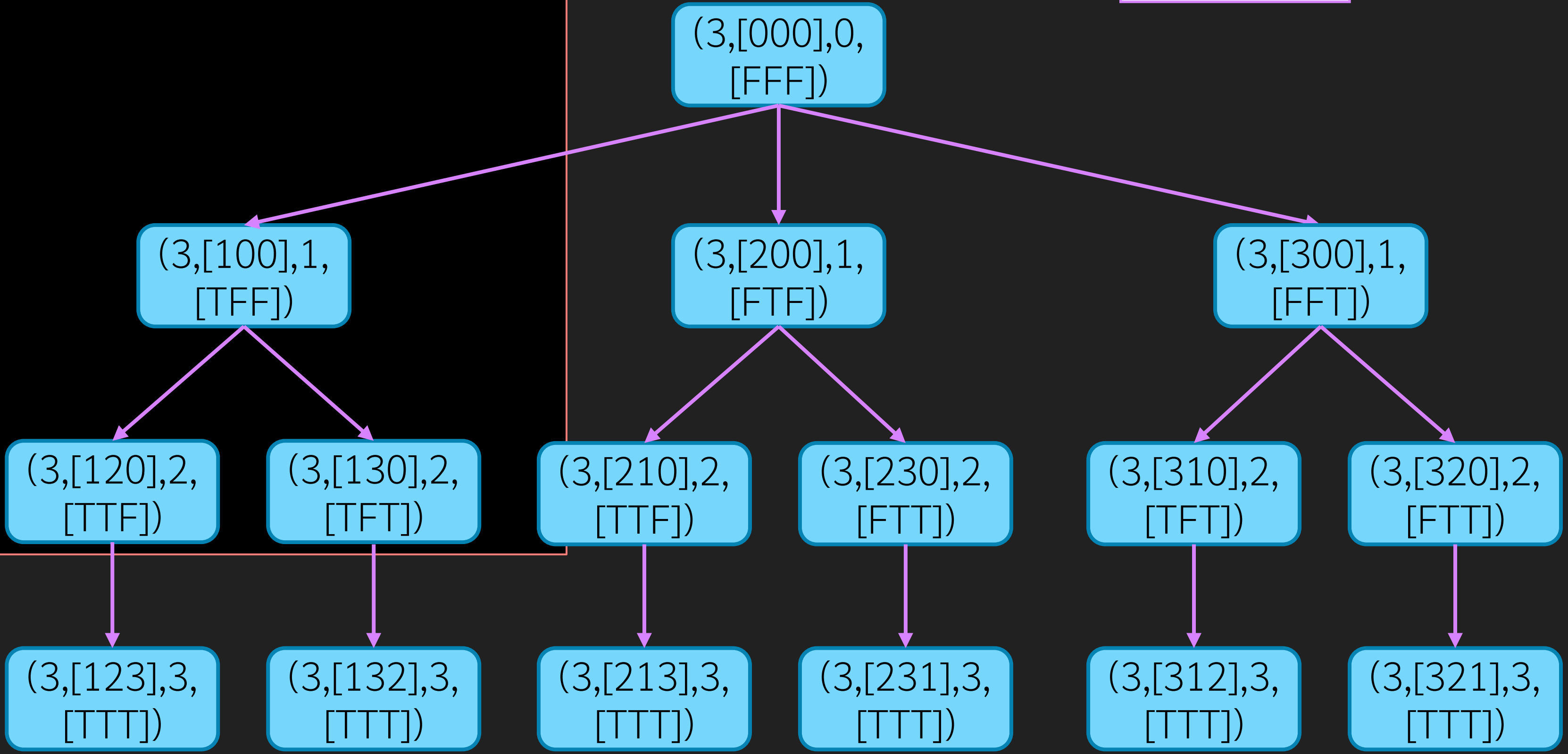
- Also like the combination, except
- At ith step, we decides if the item for the ith position of the answer
  - There are N choices at each step (recursion tree is N-ary tree)
- Do not pick item that is already included
  - If it’s permutation with replacement, we can skip this one

Gen permutation (c++)

```cpp
void perm(int n,vector<int> &sol,int len,vector<bool> &used) {
  if (len < n) {
    for (int i = 1;i<=n;i++) {
      if (used[i] == false) {
        used[i] = true;
        sol[len] = i;
        perm(n,sol,len+1,used);
        used[i] = false;
      }
    }
  } else {
    for (auto &x : sol) cout << x;
    cout << endl;
  }
}
```

output

```
123
132
213
231
312
321
```



- used[i] indicates if ith item is used in the sol
- Pass-by-value

More example

- Permutation of k items from n items

```
void perm(int n,
          vector<int> &sol,
          int len,
          vector<bool> &used) {
  if (len < n) {
    for (int i = 1;i<=n;i++) {
      if (used[i] == false) {
        used[i] = true;
        sol[len] = i;
        perm(n,sol,len+1,used);
        used[i] = false;
      }
    }
  } else {
    for (auto &x : sol) cout << x;
    cout << endl;
  }
}
```

original

```
void perm_kn(int n,
             vector<int> &sol,
             int len,
             vector<bool> &used,int k) {
  if (len < k) {
    for (int i = 1;i<=n;i++) {
      if (used[i] == false) {
        used[i] = true;
        sol[len] = i;
        perm_kn(n,sol,len+1,used,k);
        used[i] = false;
      }
    }
  } else {
    for (auto &x : sol) cout << x;
    cout << endl;
  }
}
```

k items

Output n = 4,k = 3

```
123
124
132
134
142
143
213
214
231
234
241
243
312
314
321
324
341
342
412
413
421
423
431
432
```

More example

- Permutation of k items from n items, with replacement

```
void perm(int n,
          vector<int> &sol,
          int len,
          vector<bool> &used) {
  if (len < n) {
    for (int i = 1;i<=n;i++) {
      if (used[i] == false) {
        used[i] = true;
        sol[len] = i;
        perm(n,sol,len+1,used);
        used[i] = false;
      }
    }
  } else {
    for (auto &x : sol) cout << x;
    cout << endl;
  }
}
```

original

```
void perm_kn_replace(int n,
                     vector<int> &sol,
                     int len,
                     int k) {
  if (len < k) {
    for (int i = 1;i<=n;i++) {


      sol[len] = i;
      perm_kn_replace(n,sol,len+1,k);


    }
  } else {
    for (auto &x : sol) cout << x;
    cout << endl;
  }
}
```

k items, with replacement

Output n = 4,k = 2

```
11
12
13
14
21
22
23
24
31
32
33
34
41
42
43
44
```

More example

- Combination, choose not more than k items from n items

```
void combi(int n,
           vector<int> &sol,
           int len
          ) {
  if (len < n) {
    sol[len] = 0;
    combi(n,sol,len+1);

    sol[len] = 1;
    combi(n,sol,len+1);

  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << "." << endl;
  }
}
```

original

```
void combi_kn(int n,
              vector<int> &sol,
              int len,
              int k,int chosen) {
  if (len < n) {
    sol[len] = 0;
    combi_kn(n,sol,len+1,k,chosen);
    if (chosen < k) {
      sol[len] = 1;
      combi_kn(n,sol,len+1,k,chosen+1);
    }
  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << endl;
  }
}
```

k items

output, n = 4, k = 2

```
.
4 .
3 .
3 4 .
2 .
2 4 .
2 3 .
1 .
1 4 .
1 3 .
1 2 .
```

More example

- Combination, choose exactly k items from n items

```
void combi(int n,
           vector<int> &sol,
           int len
          ) {
  if (len < n) {

    sol[len] = 0;
    combi(n,sol,len+1);


    sol[len] = 1;
    combi(n,sol,len+1);

  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << "." << endl;
  }
}
```

original

```
void combi_exact(int n,
                 vector<int> &sol,
                 int len,
                 int k,int chosen) {
  if (len < n) {
    if (len - chosen < n-k) {
      sol[len] = 0;
      combi_exact(n,sol,len+1,k,chosen);
    }
    if (chosen < k) {
      sol[len] = 1;
      combi_exact(n,sol,len+1,k,chosen+1);
    }
  } else {
    for (int i = 0;i < n;i++)
      if (sol[i] == 1)
        cout << i+1 << " ";
    cout << endl;
  }
}
```

k items

output, n = 4, k = 2

```
3 4 .
2 4 .
2 3 .
1 4 .
1 3 .
1 2 .
```

Sorting problem as CSP

- Task: Sort an array
- Input: An array A[1..n]
- Output: o[1..n], which is an ordering of the items in the array, where A[o[1]] <= A[o[2]] <= A[o[3]] <= … <= A[o[n]]
- Example instance:
  - A = [40,10,30,20]
  - Output = [2,4,3,1]

Candidate Solution	Set of candidate solution	Satisfaction condition
[o_1,o_2,..,o_n]	Ever permutation of {1..N}	A[o[1]] <= A[o[2]] <= … <= A[o[n]]