→ **Minimum Difference B/w any 2 nodes** :-
(in a BST)

```
void inOrder(TreeNode* Root, TreeNode* &Prev){
    if (root == Null) return;
    inOrder(root->left, prev);
    if (Prev != Null){
            ans = min(ans, root->val - prev->val)}

    *Prev = root;
    inOrder(root->right, prev)
```

→ **Subset Generator**

[4, 7, 7] → [4], [7], [4, 7], [4, 7, 7]

```
void Backtrack(nums, idx, curr vector<int>){
    if (curr.size() >= 2) ans.pushback(curr);
    unordered_set<int> st;
    for(int i=idx; i<n; i++){
        if ((curr.empty() || curr.back() <= nums[i])
                    &&
                st.find(nums[i]) == st.end())
        { curr.pushback(nums[i]);
          Backtrack(nums, i+1, curr);
          curr.popback();
          st.insert(nums[i]); }
```

Palindrome Partitioning

"a a b" → [["a","a","b"], ["aa","b"]]

✓ Backtrack (s, 0, curr, result)

```
void Backtrack (str, idx, curr, result){
    if (idx == n) result.pushback(curr) return.
    for (i= idx; i< n; i++)
        { if (isPal(str, idx, i))
            { curr.pushback(str.substr(idx, i-idx+1))
              backtrack(s, i+1, curr, result);
              curr.popback(); }
        }
}
```

↓

at least 1 valid will be generated;

"a b c d" → i = idx = 0 ispal(str, 0, 0) ✓ ["a"

"a b c d" → ["a", "b"

["a", "b", "c", "d"]

Distribution of Cookies

return minimum unfairness.

cookies = [8, 15, 10] distribute among K = 2 children.

check all possibility ...

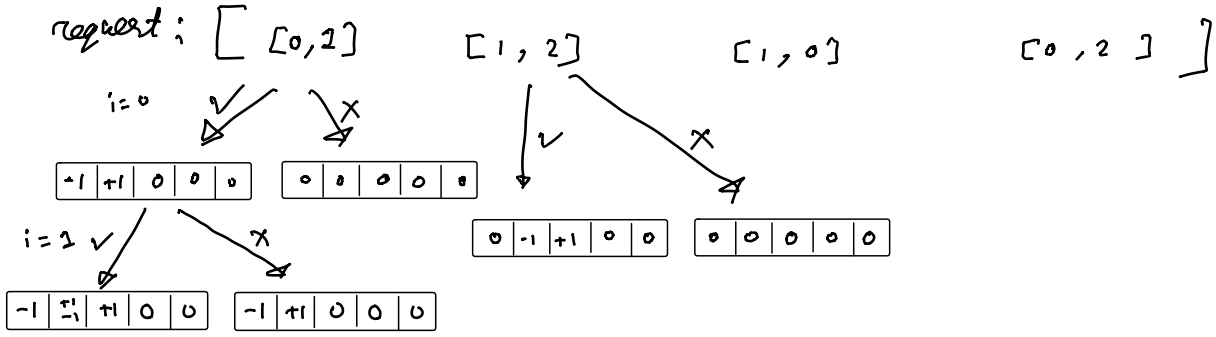
Maximum number of Achievable Transfer Requests

#LeetCode/Hard

input :- n = no. of Building = 5

request = [[0,1],[1,2],[1,0],[0,2]]

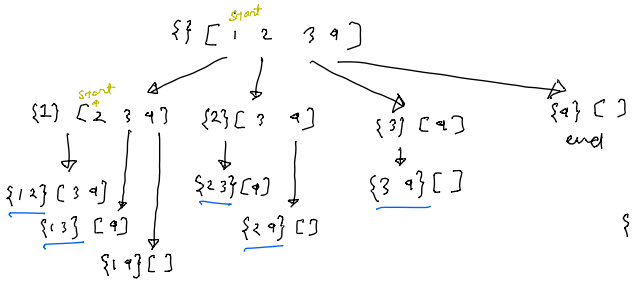
output : max number of requests can be completed.



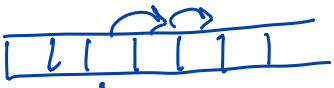
* check all possible combinations and return the maximum number of possible valid transfer.

```
void Backtracking(start, n, k, temp)
  if (k == 0) result.push_back(temp) return
  for (i = start; i < n; i++)
  { temp.push_back(i)
    Backtrack(i+1, n, k-1, temp)
    temp.popback() }
```

[1, 2, 3, 4] K = 2

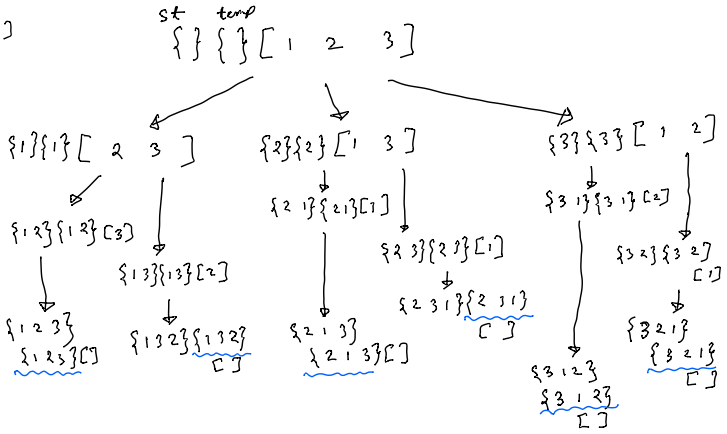


combination



idx start from here

```
unordered set <int> st;
void Backtrack(nums, temp){
  if (temp.size == nums.size) result.push_back(temp) return
  for (i=0; i<n; i++){
    if (st.find(nums[i]) == st.end)
    { temp.push_back(nums[i])
      st.insert(nums[i])
      Backtracking(temp, nums);
      temp.pop_back()
      st.erase(nums[i]); }
```

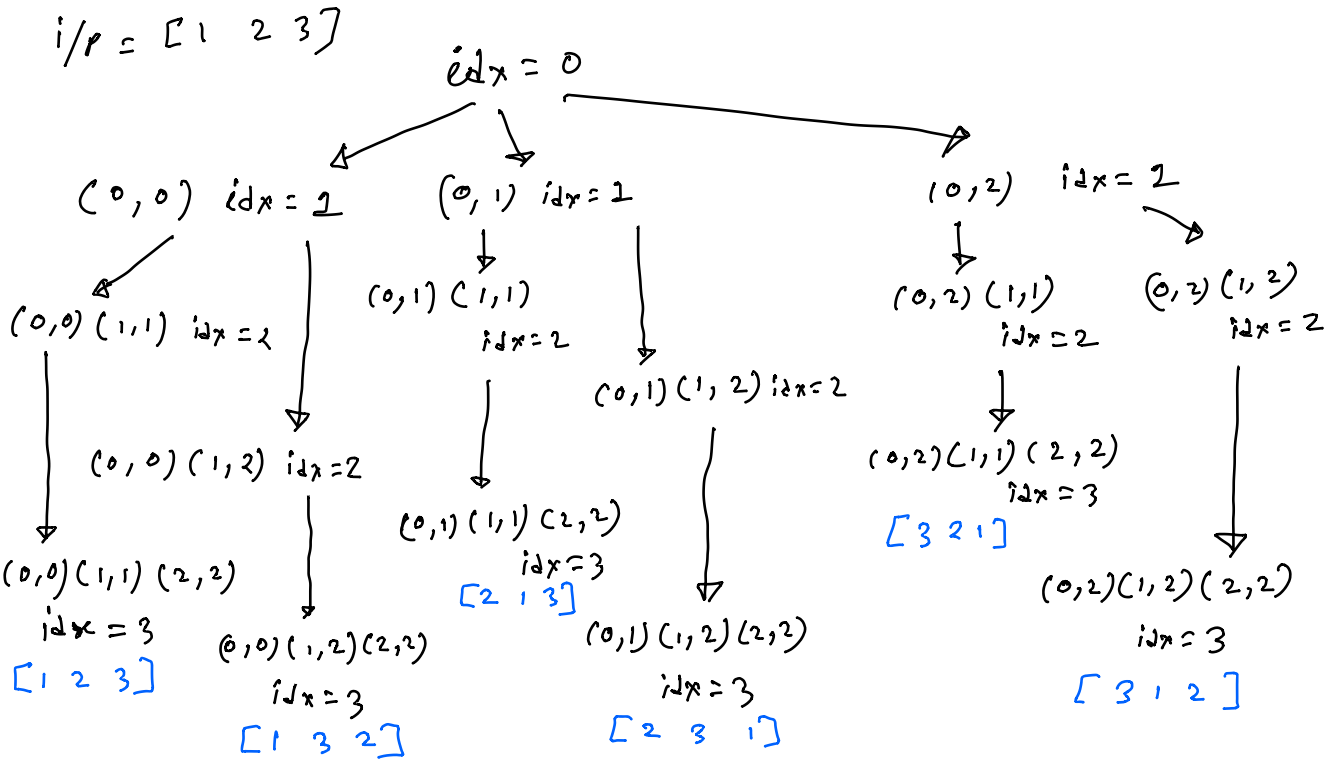


Permutation;

Start from beginning but don't include those which are present in set.

Permutations using Swap :-

```
void Backtrack(idx, temp){
    if (idx == n) result.push_back(temp) return
    for (i = idx; i < n; i++){
        Swap(temp[i], temp[idx]);
        Backtrack(idx+1, nums);
        Swap(temp[i], temp[idx]);
```



Mobile Number Diler :-

input :- string : "2 3" 2 = abc, 3 = def

output = ["ad", "ae", "af", "bd" ...]

```
Void Backtrack( idx , s , temp) {
    if (idx >= s.length()) result .pushback (temp) return
    char ch = S [idx]
    String chars = map [ch]
    for( int i = 0 ; i < chars. length() ; i++){
        temp. pushback ( str [i])
        Backtrack ( idx + 1 , S , temp)
        temp. popback ( )
    }

Backtrack ( 0 , S , string " ")
```