**31. 下一个排列**

实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须原地修改，只允许使用额外常数空间。

以下是一些例子，输入位于左侧列，其相应输出位于右侧列。

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

[9,1,2,3] -> [9,1,3,2]

|  |
| --- |
| class Solution {  public:  void nextPermutation(vector<int>& nums)  {  int i = nums.size() - 1;  while (i > 0 && nums[i - 1] >= nums[i]) {  --i;  }  if (i == 0) {  // 7 6 5 4 3 2 完全逆序，从头来过  reverse(nums.begin(), nums.end());  }  else {  // 不完全逆序，还可以继续调整，举例：8,9,7,2,6,4,3,1  int j = nums.size() - 1;  while (j > i && nums[i - 1] >= nums[j]) --j;  //i - 1为第一次不为逆序的数，找到后面到第一个大于i - 1的数  // 进行交换类似进位 nums[i], nums[j] 6 3  swap(nums[i - 1], nums[j]);  sort(nums.begin() + i, nums.end());  }  }  };  想要创建比当前更大的排列。因此，我们需要将数字 a[i-1]a[i−1] 替换为位于其右侧区域的数字中比它更大的数字，例如 a[j]a[j] |

**result = []**

**def backtrack(路径, 选择列表):**

**if 满足结束条件:**

**result.add(路径)**

**return**

**for 选择 in 选择列表:**

**做选择**

**backtrack(路径, 选择列表)**

**撤销选择**

46. 全排列

给定一个 没有重复 数字的序列，返回其所有可能的全排列。

输入: [1,2,3]

输出:

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

|  |
| --- |
| class Solution {  public:  vector<vector<int>> permute(vector<int>& nums) {  permute(nums, 0);  return permute\_;  }  void permute(vector<int>& nums, int pos) {  if (pos == nums.size()) {  permute\_.push\_back(nums);  return;  }  for (size\_t i = pos; i < nums.size(); i++) {  swap(nums[pos], nums[i]);  permute(nums, pos + 1);  swap(nums[pos], nums[i]);  }  }  private:  vector<vector<int>> permute\_;  };  输出所有排列数。递归中，最快的方法是直接交换，实际上执行的是选择操作，选择了一个数据后，把它交换到前面；可以保证下一次选择不会包含着一数字，并且数字被选择的概率都是相等的。 |

**47. 全排列 II**

给定一个可包含重复数字的序列，返回所有不重复的全排列。

输入: [1,1,2]

输出:

[

[1,1,2],

[1,2,1],

[2,1,1]

]

|  |
| --- |
| class Solution {  vector<vector<int>> ans;  public:  vector<vector<int>> permuteUnique(vector<int>& nums) {  sort(nums.begin(), nums.end());  perm(nums, 0, nums.size() - 1);  return ans;  }  void perm(vector<int> nums, int left, int right) {  if (left == right)  ans.push\_back(nums);  else {  for (int i = left; i <= right; i++) {  if (i != left && nums[left] == nums[i]) continue; # 去重  swap(nums[left], nums[i]);  perm(nums, left + 1, right);  }  }  }  }; |

**60. 第k个排列**

给出集合 [1,2,3,…,n]，其所有元素共有 n! 种排列。

按大小顺序列出所有排列情况，并一一标记，当 n = 3 时, 所有排列如下：

"123"

"132"

"213"

"231"

"312"

"321"

给定 n 和 k，返回第 k 个排列。

给定 n 的范围是 [1, 9]。

给定 k 的范围是[1, n!]。

|  |
| --- |
| class Solution {  public:  string getPermutation(int n, int k) {  string src;  src.resize(n);  iota(src.begin(), src.end(), '1');  int i = 1;  do {  if (i == k) {  break;  }  i++;  } while (next\_permutation(src.begin(), src.end()));  return src;  }  }; |

|  |
| --- |
| class Solution {  String ans ="";  int cnt = 1;  public String getPermutation(int n, int k) {  int[] array = new int[n];  int[] visit = new int[n];  for(int i=0;i<n;i++){  array[i] = i+1;  }  dfs(array,visit,"",k);  return ans;  }  public void dfs(int[] array,int[] visit,String tmp,int k){  if(ans!=""){return;}  if(tmp.length()==array.length){  if(cnt==k){  ans = tmp;  }else{  cnt++;  }  return ;  }  for(int i=0;i<array.length;i++){  if(visit[i]==1)continue;  visit[i] = 1;  tmp = tmp + array[i];  dfs(array,visit,tmp,k);  visit[i] = 0;  tmp = tmp.substring(0,tmp.length()-1);  }  }  } |

求第k个排列数。数学解法，可以找一下规律。比如对于1234的排列数，一共有24种。我们从左到右依次决定排列数是哪些。首先第一个数有4种可选的，一共有24种，那么每种就是6个，我们用 k / 6，看它落在哪个区间，就取哪个数字。

**996. 正方形数组的数目**

给定一个非负整数数组 A，如果该数组每对相邻元素之和是一个完全平方数，则称这一数组为正方形数组。

返回 A 的正方形排列的数目。两个排列 A1 和 A2 不同的充要条件是存在某个索引 i，使得 A1[i] != A2[i]。

输入：[1,17,8]

输出：2

解释：[1,8,17] 和 [17,8,1] 都是有效的排列。

输入：[2,2,2]

输出：1

|  |
| --- |
| class Solution {  public:  int numSquarefulPerms(vector<int>& A)  {  if (A.size() < 1) {  return 0;  }  Permute(A, 0);  return perms\_.size();  }  void Permute(vector<int>& nums, int pos)  {  if (pos == nums.size() && CheckQuare(nums, pos)) {  perms\_.insert(nums);  return;  }  if (!CheckQuare(nums, pos)) {  return;  }  for (size\_t i = pos; i < nums.size(); i++) {  if (pos != i && nums[pos] == nums[i]) {  continue;  }  swap(nums[pos], nums[i]);  Permute(nums, pos + 1);  swap(nums[pos], nums[i]);  }  }  bool CheckQuare(vector<int>& nums, int pos)  {  if (pos < 2) {  return true;  }  int sum = nums[pos - 2] + nums[pos - 1];  int sqroot = static\_cast<int>(sqrt(sum));  if (sum == (sqroot \* sqroot)) {  return true;  } else {  return false;  }  return true;  }  private:  set<vector<int>> perms\_;  }; |

**46. 全排列**

给定一个 没有重复 数字的序列，返回其所有可能的全排列。

输入: [1,2,3]

输出:

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

|  |
| --- |
| class Solution {  public:  vector<vector<int>> permute(vector<int>& nums) {  permute(nums, 0);  return permute\_;  }  void permute(vector<int>& nums, int pos) {  if (pos == nums.size()) {  permute\_.push\_back(nums);  return;  }  for (size\_t i = pos; i < nums.size(); i++) {  swap(nums[pos], nums[i]);  permute(nums, **pos + 1**);  swap(nums[pos], nums[i]);  }  }  private:  vector<vector<int>> permute\_;  }; |

**47. 全排列 II**

给定一个可包含重复数字的序列，返回所有不重复的全排列。

输入: [1,1,2]

输出:

[

[1,1,2],

[1,2,1],

[2,1,1]

]

|  |
| --- |
| class Solution {  vector<vector<int>> ans;  public:  vector<vector<int>> permuteUnique(vector<int>& nums) {  sort(nums.begin(), nums.end());  perm(nums, 0);  return ans;  }  void perm(vector<int> nums, int pos) {  if (pos == nums.size() - 1)  ans.push\_back(nums);  else {  for (int i = pos; i < nums.size(); i++) {  **if (i != pos && nums[pos] == nums[i]) {**  **continue; //# 去重**  **}**  swap(nums[pos], nums[i]);  perm(nums, pos + 1);  }  }  }  }; |

**60. 第k个排列**

给出集合 [1,2,3,…,n]，其所有元素共有 n! 种排列。

按大小顺序列出所有排列情况，并一一标记，当 n = 3 时, 所有排列如下：

"123"

"132"

"213"

"231"

"312"

"321"

给定 n 和 k，返回第 k 个排列。

给定 n 的范围是 [1, 9]。

给定 k 的范围是[1, n!]。

|  |
| --- |
| class Solution {  public:  string getPermutation(int n, int k) {  string src;  src.resize(n);  iota(src.begin(), src.end(), '1');  int i = 1;  do {  if (i == k) {  break;  }  i++;  } while (next\_permutation(src.begin(), src.end()));  return src;  }  }; |

|  |
| --- |
| class Solution {  String ans ="";  int cnt = 1;  public String getPermutation(int n, int k) {  int[] array = new int[n];  int[] visit = new int[n];  for(int i=0;i<n;i++){  array[i] = i+1;  }  dfs(array,visit,"",k);  return ans;  }  public void dfs(int[] array,int[] visit,String tmp,int k){  if(ans!=""){return;}  if(tmp.length()==array.length){  if(cnt==k){  ans = tmp;  }else{  cnt++;  }  return ;  }  for(int i=0;i<array.length;i++){  if(visit[i]==1)continue;  visit[i] = 1;  tmp = tmp + array[i];  dfs(array,visit,tmp,k);  visit[i] = 0;  tmp = tmp.substring(0,tmp.length()-1);  }  }  } |

**996. 正方形数组的数目**

给定一个非负整数数组 A，如果该数组每对相邻元素之和是一个完全平方数，则称这一数组为正方形数组。

返回 A 的正方形排列的数目。两个排列 A1 和 A2 不同的充要条件是存在某个索引 i，使得 A1[i] != A2[i]。

输入：[1,17,8]

输出：2

解释：[1,8,17] 和 [17,8,1] 都是有效的排列。

输入：[2,2,2]

输出：1

|  |
| --- |
| class Solution {  public:  int numSquarefulPerms(vector<int>& A)  {  if (A.size() < 1) {  return 0;  }  Permute(A, 0);  return perms\_.size();  }  void Permute(vector<int>& nums, int pos)  {  if (pos == nums.size() && CheckQuare(nums, pos)) {  perms\_.insert(nums);  return;  }  if (!CheckQuare(nums, pos)) {  return;  }  for (size\_t i = pos; i < nums.size(); i++) {  if (pos != i && nums[pos] == nums[i]) {  continue;  }  swap(nums[pos], nums[i]);  Permute(nums, pos + 1);  swap(nums[pos], nums[i]);  }  }  bool CheckQuare(vector<int>& nums, int pos)  {  if (pos < 2) {  return true;  }  int sum = nums[pos - 2] + nums[pos - 1];  int sqroot = static\_cast<int>(sqrt(sum));  if (sum == (sqroot \* sqroot)) {  return true;  } else {  return false;  }  return true;  }  private:  set<vector<int>> perms\_;  }; |

**39. 组合总和**

给定一个无重复元素的数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的数字可以无限制重复被选取。

所有数字（包括 target）都是正整数。

解集不能包含重复的组合。

输入：candidates = [2,3,6,7], target = 7,

所求解集为：

[7],

[2,2,3]

输入：candidates = [2,3,5], target = 8,

所求解集为：

[2,2,2,2],

[2,3,3],

[3,5]

|  |
| --- |
| class Solution {  public:  void backTrack(vector<int> &candidates, vector<vector<int>> &ans, int start, vector<int> &temp, int target)  {  if(target < 0) return; // 已经爆了  if(target==0) // 刚好合适  {  ans.push\_back(temp);  return ;  }  for(int i=start;i<candidates.size();i++)  {  **if(candidates[i]>target)// 后面全是大于target的，直接跳出**  **break;**  temp.push\_back(candidates[i]);  backTrack(candidates, ans, i, temp, target-candidates[i]);  temp.pop\_back();  }  }  vector<vector<int>> combinationSum(vector<int>& candidates, int target)  {  sort(candidates.begin(), candidates.end());  vector<vector<int>> ans;  vector<int> temp;  backTrack(candidates, ans, 0, temp, target);  return ans;  }  }; |

**40. 组合总和 II**

给定一个数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的每个数字在每个组合中只能使用一次。

所有数字（包括目标数）都是正整数。

解集不能包含重复的组合。

输入: candidates = [10,1,2,7,6,1,5], target = 8,

所求解集为:

[1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6]

输入: candidates = [2,5,2,1,2], target = 5,

所求解集为:

[1,2,2],

[5]

|  |
| --- |
| class Solution {  public:  vector<vector<int>> combinationSum2(vector<int>& candidates, int target)  {  vector<int> path;  sort(candidates.begin(), candidates.end());  Dfs(candidates, target, 0, path);  return res;  }  void Dfs(vector<int>& candidates, int target, int pos, vector<int>& path) {  if (target == 0) {  res.push\_back(path);  return;  }  for (size\_t i = pos; i < candidates.size(); i++) {  if (candidates[i] > target) {  continue;  }  if(i > pos && candidates[i]==candidates[i-1]){ // [1, 1, 7] 第2个1没必要重复计算  continue;  }  path.push\_back(candidates[i]);  Dfs(candidates, target - candidates[i], i + 1, path);  path.pop\_back();  }  }  private:  vector<vector<int>> res;  }; |

**78. 子集**

给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

输入: nums = [1,2,3]

输出:

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]

|  |
| --- |
| class Solution {  public:  vector<vector<int>> subsets(vector<int>& nums) {  vector<vector<int>> res;  vector<int> item;  //res.push\_back(item);  Generate(res, nums, 0, item);  return res;  }  void Generate(vector<vector<int>>& result, const vector<int>& nums, int index, vector<int>& item) {  result.push\_back(item);  for (auto i = index; i < nums.size(); ++i) {  item.push\_back(nums[i]);  Generate(result, nums, i + 1, item);  item.pop\_back();  }  }  }; |

**90. 子集 II**

给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

输入: [1,2,2]

输出:

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

|  |
| --- |
| class Solution {  public:  vector<vector<int>> subsetsWithDup(vector<int>& nums) {  vector<vector<int>> res;  vector<int> item;  sort(nums.begin(), nums.end());  Generate(res, nums, 0, item);  return res;  }  void Generate(vector<vector<int>>& result, const vector<int>& nums, int index, vector<int>& item) {  result.push\_back(item);  for (auto i = index; i < nums.size(); ++i) {  if (i > index && nums[i] == nums[i - 1]) {  continue;  }  item.push\_back(nums[i]);  Generate(result, nums, i + 1, item);  item.pop\_back();  }  }  }; |