```
// ////////////////////////////////////  Quick Sort: Lintcode 5  //////////////////////////////////////////
class Solution {
private:
int partition(vector<int>& nums, int low, int high) {
   int l = low;
   int r = high + 1;
   while (l < r) {
      while (l < high && nums[low] > nums[++l]);
      while (r > low && nums[low] < nums[--r]);
      if (l >= r) {
         break;
      }
      swap(nums[l], nums[r]);
   }
   swap(nums[r], nums[low]);
   return r;
}

void quickSort(vector<int>& nums, int l, int r) {
   if (l <= r) {
      int p = partition(nums, l, r);
      quickSort(nums, l, p - 1);
      quickSort(nums, p + 1, r);
   } else {
      return;
   }
}

public:
   int kthLargestElement(int n, vector<int> &nums) {
      // write your code here
      quickSort(nums, 0, nums.size() - 1);
      return nums[nums.size() - n];
   }
};


/////////////////////////////////////  Quick Sort List: Lintcode 98  ///////////////////////////////////////////
class Solution {
private:
ListNode* partition(ListNode* low, ListNode* high) {
   ListNode* p = low;
   ListNode* q = low->next;
   int key = low->val;
   while (q != high) {
      if (q->val < key) {
         p = p->next;
         swap(p->val, q->val);
      }
      q = q->next;
   }
```

```
    swap(low->val, p->val);
    return p;
}

void quickSortList(ListNode* low, ListNode* high) {
    if (low != high) {
        ListNode* p = partition(low, high);
        quickSortList(low, p);
        quickSortList(p->next, high);
    }
}

public:
    /**
     * @param head: The head of linked list.
     * @return: You should return the head of the sorted linked list, using constant space complexity.
     */
    ListNode * sortList(ListNode * head) {
        // write your code here
        ListNode* low = head;
        ListNode* high = nullptr;   // one of the new features
        quickSortList(low, high);
        return head;
    }
};

////////////////////////////////////  Top K: Lintcode 5  //////////////////////////////////////////
/////  priority_queue
int kthLargestElement(int n, vector<int> &nums) {
        // write your code here
        //int k = nums.size() - n;
        priority_queue<int, vector<int>> pq;
        for (auto num : nums) {
            pq.push(num);
        }

        int res;
        for (int i = 0; i < n; ++i) {
            res = pq.top();
            pq.pop();
        }

        return res;
    }

    ///// half-quick-sort: O(n)
    int kthLargestElement(int n, vector<int> &nums) {
            // write your code here
            int k = nums.size() - n;
            int low = 0;
            int high = nums.size() - 1;
            while (low <= high) {
                int p = partition(nums, low, high);
```

```
            if (p > k) {
                high = p - 1;
            } else if (p < k) {
                low = p + 1;
            } else {
                break;
            }
        }
        return nums[k];
    }
```

////////////////////////////// Binary Search Template: no lintcode (借用 458) ////////////////////////////

```
int findPosition(vector<int> nums, int target) {
        if (!nums.size()) {
                return -1;
        }
        int l = 0, r = nums.size() – 1;
        while (l + 1 < r) {
                int m = l + (r - l) / 2;
                if (nums[m] == target) { return m; }
                else if (nums[m] < target) { l = m; }
                else { r = m; }
        }
        if (nums[l] == target) { return l; }
        if (nums[r] == target) { return r; }
        return -1;
}
```

/////////////////////////////// Last Position: lintcode 458 //////////////////////////////////

```
int lastPosition(vector<int> &nums, int target) {
        // write your code here
        int size = nums.size();
        if (!size) return -1;

        int l = 0;
        int r = size - 1;
        while (l + 1 < r) {
            int m = l + (r - l) / 2;
            if (nums[m] > target) {
                r = m;
            }
            else if (nums[m] <= target) {
                l = m;
            }
        }

        if (nums[r] == target) {
            return r;
        }

        if (nums[l] == target) {
            return l;
        }
```

```
        return -1;
    }
/////////////////////////////// first position: lintcode 14  /////////////////////////////////
int binarySearch(vector<int> &nums, int target) {
        // write your code here
        int size = nums.size();
        if(!size) return -1;
        int l = 0, r = size - 1;
        while(l + 1 < r) {
            int m = l + (r - l) / 2;
            if (nums[m] >= target) {
                r = m;
            }
            else {
                l = m;
            }
        }
        if (nums[l] == target) {
            return l;
        }
        if (nums[r] == target) {
            return r;
        }
        return -1;
    }
////////////////////////////// Top k: lintcode 460  ///////////////////////////////////
int getLeftCloest(vector<int> A, int target) {
        int l = 0;
        int r = A.size() - 1;
        while (l + 1 < r) {
            int m = l + (r - l) / 2;
            if (A[m] < target) {
                l = m;
            } else {
                r = m;
            }
        }
        if (A[r] < target) {
            return r;
        }
        if (A[l] < target) {
            return l;
        }
        return -1;
    }

vector<int> kClosestNumbers(vector<int> &A, int target, int k) {
        // write your code here
        vector<int> res;

        int size = A.size();
        if (!size) return res;
        if (k > size) return res;
```

```
    int leftCloest = getLeftCloest(A, target);
    int l = leftCloest;
    int r = l + 1;
    for (int i = 0; i < k; ++i) {
      if (l >= 0 && r < size) {
        if (abs(A[l] - target) <= abs(A[r] - target)) {
          res.push_back(A[l]);
          --l;
        } else {
          res.push_back(A[r]);
          ++r;
        }
      }
      else {
        if (l < 0) {
          res.push_back(A[r]);
          ++r;
        }
        else if (r >= size) {
          res.push_back(A[l]);
          --l;
        }
      }
    }
    return res;
  }

///////////////////////////////// Inorder Tree Traverse: Lintcode 67  ////////////////////////////////
vector<int> inorderTraversal(TreeNode * root) {
    // write your code here
    // if (root) {
    //    inorderTraversal(root->left);
    //    res.push_back(root->val);
    //    inorderTraversal(root->right);
    // }
    stack<TreeNode*> st;
    TreeNode* p = root;
    while (!st.empty() || p) {
      if (p) {
        st.push(p);
        p = p->left;
      } else {
        TreeNode* c = st.top();
        st.pop();
        res.push_back(c->val);
        p = c->right;
      }
    }
    return res;
  }

/////////////////////////////// Preorder Tree Traverse: Lintcode 66 ////////////////////////////////////
```

```
vector<int> preorderTraversal(TreeNode * root) {
      // write your code here
      // if (root) {
      //     res.push_back(root->val);
      //     preorderTraversal(root->left);
      //     preorderTraversal(root->right);
      // }

      stack<TreeNode*> st;
      if (root) {
         st.push(root);
         while(!st.empty()) {
            TreeNode* p = st.top();
            st.pop();
            res.push_back(p->val);
            if (p->right) {
               st.push(p->right);
            }
            if (p->left) {
               st.push(p->left);
            }
         }
      }

      return res;
   }

//////////////////////////////// Postorder Tree Traverse: Lintcode 68 //////////////////////////////////////
class Solution {
   vector<int> res;
public:
   /**
    * @param root: A Tree
    * @return: Postorder in ArrayList which contains node values.
    */
   vector<int> postorderTraversal(TreeNode * root) {
      // write your code here
      // if (root) {
      //     postorderTraversal(root->left);
      //     postorderTraversal(root->right);
      //     res.push_back(root->val);
      // }

      stack<TreeNode*> st;
      if (root) {
         st.push(root);
         while(!st.empty()) {
            TreeNode* p = st.top();
            st.pop();
            res.push_back(p->val);
            if (p->left) {
               st.push(p->left);
            }
```

```
            if (p->right) {
               st.push(p->right);
            }

         }
      }
      reverse(res.begin(), res.end());

      return res;
   }
};

///////////////////////////////// Level Order: Lintcode 69 ////////////////////////////////////////
vector<vector<int>> levelOrder(TreeNode * root) {
      // write your code here
      if (!root) return {};
      queue<TreeNode*> q;
      q.push(root);

      vector<vector<int>> res;

      while (!q.empty()) {
         int n = q.size();
         vector<int> cur;
         while(n--) {
            TreeNode* node = q.front();
            q.pop();
            cur.push_back(node->val);
            if (node->left) {
               q.push(node->left);
            }
            if (node->right) {
               q.push(node->right);
            }
         }
         res.push_back(cur);
      }
      return res;
   }

//////////////////////////////// zigzag Lintcode: 71  ///////////////////////////////////////////////
vector<vector<int>> zigzagLevelOrder(TreeNode * root) {
      // write your code here
      if (!root) return {};
      deque<TreeNode*> dq;
      dq.push_back(root);
      vector<vector<int>> res;

      bool isOdd = true;
      while(!dq.empty()) {
         int n = dq.size();
         vector<int> cur;
```

```
        while (n--) {
          if(isOdd) {
            TreeNode* node = dq.front();
            dq.pop_front();
            cur.push_back(node->val);
            if(node->left)
              dq.push_back(node->left);
            if(node->right)
              dq.push_back(node->right);

          } else {
            TreeNode* node = dq.back();
            dq.pop_back();
            cur.push_back(node->val);
            if (node->right)
              dq.push_front(node->right);
            if (node->left)
              dq.push_front(node->left);

          }
        }
        res.push_back(cur);
        isOdd = !isOdd;
      }
      return res;
    }
```

//////////////////////////////// Balanced Binary Tree: Lintcode 93 //////////////////////////////

```
class Solution {
private:
    int getDepth(TreeNode* node, int depth) {
      if (!node) return depth - 1;
      return max(getDepth(node->left, depth + 1),
              getDepth(node->right, depth + 1));
    }

public:
    /**
     * @param root: The root of binary tree.
     * @return: True if this Binary tree is Balanced, or false.
     */
    bool isBalanced(TreeNode * root) {
      // write your code here
      if (!root) return true;
      int leftDepth = getDepth(root->left, 1);
      int rightDepth = getDepth(root->right, 1);
      return abs(leftDepth - rightDepth) <= 1 &&
          isBalanced(root->left) &&
          isBalanced(root->right);
    }
};
```

//////////////////////////////// Validated BST: Lintcode 95 //

```
class Solution {
private:
    bool validBST(TreeNode* node, long min, long max) {
        if (!node) return true;
        if (node->val <= min || node->val >= max) {
            return false;
        }

        return validBST(node->left, min, node->val) &&
              validBST(node->right, node->val, max);
    }
public:
    /**
     * @param root: The root of binary tree.
     * @return: True if the binary tree is BST, or false
     */
    bool isValidBST(TreeNode * root) {
        // write your code here
        if(!root) return true;
        return validBST(root, LONG_MIN, LONG_MAX);
    }
};

/////////////////////////////// reverse linked list: lintcode 35  ///////////////////////////////
ListNode * reverse(ListNode * head) {
        // write your code here
        // if (!head || !head->next) return head;
        // ListNode* p = nullptr;
        // while (head) {
        //     ListNode* next = head->next;
        //     head->next = p;
        //     p = head;
        //     head = next;
        // }
        // return p;

        if (!head || !head->next) return head;
        ListNode* newhead = reverse(head->next);
        head->next->next = head;
        head->next = nullptr;
        return newhead;
    }

/////////////////////////////// merge 2 linked list: lintcode 165  ///////////////////////////////
ListNode * mergeTwoLists(ListNode * l1, ListNode * l2) {
        // write your code here
        if (!l1) return l2;
        if (!l2) return l1;
        ListNode* p;
        ListNode* head;
        if (l1->val > l2->val) {
            head = l2;
            p = l2;
```

```
            l2 = l2->next;
        }
        else {
            head = l1;
            p = l1;
            l1 = l1->next;
        }
        while(l1 && l2) {
            if (l1->val > l2->val) {
                p->next = l2;
                l2 = l2->next;
            } else {
                p->next = l1;
                l1 = l1->next;
            }
            p = p->next;
        }

        if (l1) {
            p->next = l1;
        }
        if (l2) {
            p->next = l2;
        }
        return head;
    }

////////////////////////////// Sqrt(x): 141  //////////////////////////////////////
    int mySqrt(long x) {
        // binary search
//        if (x <= 1) return x;
//        long eps = 0.000001;

//        int low = 1;
//        int high = x / 2;
//        while (low <= high) {
//            long mid = low + (high - low) / 2;
//            if (mid * mid == x) {
//                return mid;
//            } else if (mid * mid < x) {
//                if ((mid + 1) * (mid + 1) > x) {
//                    return mid;
//                }
//                low = mid + 1;
//            } else {
//                high = mid - 1;
//            }
//        }
//        return 1;

        // newton:
        long a = x;
        long eps = 0.00001;
```

```
    while (a * a - x > eps) {
      a = 0.5 * (a + x / a);
    }
    return a;
  }
```

////////////////////////// pow(x, n): lintcode 428 ///////////////////////////////////

```
  double myPow(double x, long long n) {
      if(n == 0)
        return 1;
      if(n < 0){
        n = -n;
        x = 1 / x;
      }
      return !(n & 1) ? myPow(x * x, n / 2) : x * myPow(x * x, n / 2);
    }

  // this is for integer & matrix!
  long myPow(int x, int x) {
      int res = 1;
      int tmp = x;
      while (n) {
        if (n & 1) {
          res *= tmp;
        }
        tmp *= tmp;
        n >>= 1;
      }
      return res;
  }
```

////////////////////////////////// Binary Maze ////////////////////////////////////

```
int getShortestPath(vector<vector<int>>& maze, pair<int, int> start, pair<int, int> end){
       // never forget to check corner cases [or ask your interviewer to make sure your inputs are
valid]
       int rows = maze.size();
       if (!rows) return -1;
       int cols = maze[0].size();
       if (!cols) return -1;
       if (start.first < 0 || start.second < 0 || start.first >= rows || start.second >= cols ||
              end.first < 0 || end.second < 0 || end.first >= rows || end.second >= cols) {
              return -1;
       }
       // do real things now.
       // initialization
       queue<pair<int, int>> q;
       q.push(start);
       vector<vector<int>> visited(rows, vector<int>(cols, -1));
       visited[start.first][start.second] = 1;
       vector<pair<int, int>> dir = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };

       while (!q.empty()) {
              pair<int, int> cur = q.front();
              q.pop();

              if (cur.first == end.first && cur.second == end.second) {
                     return visited[cur.first][cur.second];
              }
```

```
            for (auto d : dir) {
                int new_r = cur.first + d.first;
                int new_c = cur.second + d.second;
                if (new_r < 0 || new_r >= rows || new_c < 0 || new_c >= cols ||
                    visited[new_r][new_c] != -1 ||
                    !maze[new_r][new_c]) {
                    continue;
                }
                else {
                    pair<int, int> new_node = { new_r, new_c };
                    visited[new_r][new_c] = visited[cur.first][cur.second] + 1;
                    q.push(new_node);
                }
            }
        }
        return -1;
  }
int main() {
        vector<vector<int>> maze1 =
        {
            { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
            { 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
            { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
            { 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },
            { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
            { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
            { 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
            { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
            { 1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }
        };

        //vector<int> start = { 1, 0 };
        //vector<int> end = { 1, 3 };
        pair<int, int> start = { 0, 0 };
        pair<int, int> end = {8, 1};

        int shortestPath = 0;

        shortestPath = getShortestPath(maze1, start, end);
        std::cout << "shortest path length: " << shortestPath << endl;

        char e;
        std::cin >> e;
        return 0;
  }
```

////////////////////////////////// Normal Maze ////////////////////////////////////

// Your task!

////////////////////////////////// Number Island I: Lincode 433 ////////////////////////////////////

```
class Solution {
private:
   void dfs(vector<vector<bool>>& grid, int r, int c) {
      if (r < 0 || r >= grid.size() || c < 0 || c >= grid[0].size() ||
         grid[r][c] == false) {
            return;
      }

      grid[r][c] = false;
      dfs(grid, r + 1, c);
      dfs(grid, r - 1, c);
```

```
        dfs(grid, r, c + 1);
        dfs(grid, r, c - 1);
        return;
    }

public:
    /**
     * @param grid: a boolean 2D matrix
     * @return: an integer
     */
    int numIslands(vector<vector<bool>> &grid) {
        // write your code here
        int rows = grid.size();
        if (!rows) return 0;
        int cols = grid[0].size();
        if (!cols) return 0;
        int num = 0;

        for (int r = 0; r < rows; ++r) {
            for (int c = 0; c < cols; ++c) {
                if (grid[r][c]) {
                    dfs(grid, r, c);
                    ++num;
                }
            }
        }
        return num;
    }
};

////////////////////////////////// Number Island II: Lincode 434 /////////////////////////////////////
vector<int> numIslands2(int n, int m, vector<Point> &operators) {
        // write your code here
        vector<int> res;
        if (!(n * m)) return res;
        if (!operators.size()) return res;
        vector<int> roots(n * m, -1);
        vector<pair<int, int>> dir = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
        unordered_set<int> us;
        int num = 0;

        for (auto p : operators) {
            int r = p.x;
            int c = p.y;

            int id = r * m + c;
            roots[id] = id;
            if (us.find(id) != us.end()) {
                res.push_back(num);
                continue;
            }
            us.insert(id);
            ++num;
```

```
        for (auto d : dir) {
          int nr = r + d.first;
          int nc = c + d.second;
          int nid = nr * m + nc;
          if (nr >= 0 && nr < n && nc >= 0 && nc < m && roots[nid] != -1) {
            while (id != roots[id]) {
              roots[id] = roots[roots[id]];
              id = roots[id];
            }
            while(nid != roots[nid]) {
              roots[nid] = roots[roots[nid]];
              nid = roots[nid];
            }
            if (roots[nid] != roots[id]) {
              roots[nid] = roots[id];
              --num;
            }
          }
        }
        res.push_back(num);
      }
      return res;
    }
```

```
// convolution-series
// 2d median filter - no stride & zero padding
vector<vector<int>> medianFilter(vector<vector<int>>& img, int kw, int kh) {
       int rows = img.size();
       int cols = img[0].size();

       vector<vector<int>> res(rows, vector<int>(cols));

       for (int r = 0; r < rows; ++r) {
              for (int c = 0; c < cols; ++c) {
                     int left = max(0, int(c - kw / 2.0 + 1));
                     int right = min(cols - 1, c + kw / 2);
                     int upper = max(0, int(r - kh / 2.0 + 1));
                     int lower = min(rows - 1, r + kh / 2);

                     int n = (right - left + 1) * (lower - upper + 1);

                     /// task 1: this is for kth-largest element
                     priority_queue<int> pq;
                     for (int h = upper; h <= lower; ++h) {
                            for (int w = left; w <= right; ++w) {
                                   // now we are duling with how to find out the median value
                                   // if odd: [n / 2]
                                   // if even: ([n / 2] + [(n - 1) / 2]) / 2
                                   pq.push(img[h][w]);
                            }
                     }
                     if !((n & 1)) {      // if it's even
                            int val1;
                            for (int i = 0; i <= (n - 1) / 2; ++i) {
                                   val1 = pq.top();
                                   pq.pop();
                            }
                            int val2 = pq.top();
```

```
                                        res[r][c] = (val1 + val2) / 2;
                                }
                                else { // if it's odd
                                        int val;
                                        for (int i = 0; i <= n / 2; ++i) {
                                                val = pq.top();
                                                pq.pop();
                                        }
                                        res[r][c] = val;
                                }
                                // time complexity of this is O(nlogk) , k is kth
                                /// end for kth-largest element

/// task 2: if it's for the maximum within a window
// the best normal way is to traverse the window directly and the time complexity in total would be
// O(rows * cols * kw * kh)
//
// the better way to find out the maximum is try to use deque in 2d (seperately in c-direction and r-
direction)
// time complexity would be O(rows * cols * kw) + O(rows * cols * kh) = O(rows * cols * (kw + kh))
/// end for fidning maximum with in a window
                        }
                }
                return res;
}
// 3d average pooling - with zero padding & stride
vector<vector<vector<int>>> pooling3D(vector<vector<vector<int>>>& img, int kw, int kh, int stride) {
                int rows = img.size();
                int cols = img[0].size();
                int channels = img[0][0].size();
                int res_rows = rows / stride;
                int res_cols = cols / stride;
                int res_channels = channels;
                vector<vector<vector<int>>>
                        res(res_rows, vector<vector<int>>(res_cols, vector<int>(res_channels)));

                for (int cl = 0; cl < channels; ++cl) {
                        int r_num = 0;
                        for (int r = 0; r < rows; r += stride) {
                                int c_num = 0;
                                for (int c = 0; c < cols; c += stride) {
                                        int left = max(0, int(c - kw / 2.0 + 1));
                                        int right = min(cols - 1, c + kw / 2);
                                        int upper = max(0, int(r - kh / 2.0 + 1));
                                        int lower = min(rows - 1, r + kh / 2);

                                        int area = (right - left + 1) * (lower - upper + 1);
                                        int sum = 0;

                                        /// this is for average pooling in 3d
                                        for (int h = upper; h <= lower; ++h) {
                                                for (int w = left; w <= right; ++w) {
                                                        sum += img[h][w][cl];
                                                }
                                        }
                                        res[r_num][c_num][cl] = sum / area;
                                        /// end for average pooling
// time complexity of this method is O(W * H * C * kw * kh)
//
// better result for this:
// best time complmexity: O(W * H * C);
// Because we are using sum with an area. this is actually leetcode/lintcode problem: "Range Sum in 2D
- Immutable"
```

```
// by using techniques provided in that problem, which is called Integeral Image, we can compress time complexity apperently.
                        ++c_num;
                    }
                    ++r_num;
                }
            }
            return res;
}
```

//////////////////////////////////// Sliding Window Maximum: Lintcode 362 //

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    vector<int> res;
    deque<int> index;
    int size = nums.size();
    if (k > size) return res;

    for (int i = 0; i < k; ++i) {
      while (!index.empty() && nums[i] >= nums[index.back()]) {
        index.pop_back();
      }
      index.push_back(i);
    }
    for (int i = k; i < size; ++i) {
      res.push_back(nums[index.front()]);
      while (!index.empty() && nums[i] >= nums[index.back()]) {
        index.pop_back();
      }
      while (!index.empty() && index.front() <= i - k) {
        index.pop_front();
      }

      index.push_back(i);
    }
    res.push_back(nums[index.front()]);
    return res;
}
```

//////////////////////////////////// Range Sum Query: Leetcode 303 //

```
class NumArray {
private:
  int len;
  vector<int> cumSum;
public:
  int len;
  vector<int> cumSum;
  NumArray(vector<int> nums) {
    len = nums.size();
    cumSum = vector<int>(len + 1, 0);
    if (len) {
      for (int i = 1; i <= len; ++i) {
        cumSum[i] = cumSum[i - 1] + nums[i - 1];
      }
    }
  }
```

```
    int sumRange(int i, int j) {
        return (cumSum[j + 1] - cumSum[i]);
    }
};

////////////////////////////////// Range Sum Query 2D: Leetcode 304 /////////////////////////////////////
class NumMatrix {
private:
    vector<vector<int>> integralMatrix;

public:
    NumMatrix(vector<vector<int>> matrix) {
        if (!matrix.size() || !matrix[0].size()) return;
        int rows = matrix.size();
        int cols = matrix[0].size();
        integralMatrix.resize(rows + 1, vector<int>(cols + 1, 0));

        for (int r = 1; r < rows + 1; ++r) {
            for (int c = 1; c < cols + 1; ++c) {
                integralMatrix[r][c] =
                        -integralMatrix[r - 1][c - 1] + integralMatrix[r - 1][c] + integralMatrix[r][c - 1] +  matrix[r - 1][c - 1];
            }
        }

        return;
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        return integralMatrix[row2 + 1][col2 + 1] + integralMatrix[row1][col1] -
              integralMatrix[row2 + 1][col1] - integralMatrix[row1][col2 + 1];
    }
};

////////////////////////////////// Longest Common Subsequence: Lintcode 77 /////////////////////////////////////
int longestCommonSubsequence(string &A, string &B) {
        // write your code here
        if (!(A.size() * B.size())) return 0;
        int rows = A.size();
        int cols = B.size();
        vector<vector<int>> dp(rows + 1, vector<int>(cols + 1, 0));

        for (int r = 1; r <= rows; ++r) {
            for (int c = 1; c <= cols; ++c) {
                if (A[r - 1] == B[c - 1]) {
                    dp[r][c] = dp[r - 1][c - 1] + 1;
                } else {
                    dp[r][c] = max(dp[r - 1][c], dp[r][c - 1]);
                }
            }
        }
        return dp[rows][cols];
    }
```

////////////////////////////////// 硬币翻转/红绿灯问题 ////////////////////////////////////

```
float getProb(vector<float> p, int k) {
	int n = p.size();
	vector<vector<float>> dp(n + 1, vector<float>(k + 1, 0.0));
	dp[0][0] = 1.0;
	for (int i = 1; i <= n; ++i) {
		dp[i][0] = (1.0 - p[i - 1]) * dp[i - 1][0];   // 反面朝上概率
	}
	for (int i = 1; i <= n + 1; ++i) {
		for (int j = 1; j <= i && j <= k; ++j) {
			dp[i][j] = dp[i - 1][j - 1] * p[i - 1] + // 本次是正面
				dp[i - 1][j] * (1.0 - p[i - 1]);         // 本次是反面的概率
		}
	}
	return dp[n][k];
}
```

////////////////////////////////// word search II: Lintcode 132 ////////////////////////////////////

```
class Solution {
 private:
     struct TrieNode {
         vector<TrieNode*> child;
         string word;
         TrieNode() : word(""), child(vector<TrieNode*>(26, nullptr)) {}
     };

 TrieNode* buildTrie(vector<string>& words) {
     TrieNode* root = new TrieNode();
     for (string w : words) {
         TrieNode* curr = root;
         for (char c : w) {
             int i = c - 'a';
             if (curr->child[i] == NULL) curr->child[i] = new TrieNode();
             curr = curr->child[i];
         }
         curr->word = w;
     }
     return root;
 }

     void dfs(vector<vector<char>>& board, int i, int j, TrieNode* curr, vector<string>& out) {
         if (i < 0 || j < 0 || i >= board.size() || j >= board[0].size()) return;
         char c = board[i][j];
         if(c == '#' || curr->child[c - 'a'] == NULL) return;
         curr = curr->child[c - 'a'];
         if (curr->word != "") {
             out.push_back(curr->word);
             curr->word = "";
         }
         board[i][j] = '#';
         dfs(board, i - 1, j, curr, out);
         dfs(board, i, j - 1, curr, out);
         dfs(board, i + 1, j, curr, out);
         dfs(board, i, j + 1, curr, out);
         board[i][j] = c;
     }

 public:
     vector<string> findWords(vector<vector<char>>& board, vector<string>& words) {
         vector<string> out;
         TrieNode* root = buildTrie(words);
         for(int i = 0; i < board.size(); ++i)
```

```
            for(int j = 0; j < board[0].size(); ++j)
                dfs(board, i, j, root, out);
        return out;
    }
};
```