**1094. 拼车**

假设你是一位顺风车司机，车上最初有 capacity 个空座位可以用来载客。由于道路的限制，车 只能向一个方向行驶（也就是说，不允许掉头或改变方向，你可以将其想象为一个向量）。

这儿有一份乘客行程计划表 trips[][]，其中 trips[i] = [num\_passengers, start\_location, end\_location] 包含了第 i 组乘客的行程信息：

必须接送的乘客数量；乘客的上车地点；以及乘客的下车地点。

这些给出的地点位置是从你的 初始 出发位置向前行驶到这些地点所需的距离（它们一定在你的行驶方向上）。

请你根据给出的行程计划表和车子的座位数，来判断你的车是否可以顺利完成接送所有乘客的任务（当且仅当你可以在所有给定的行程中接送所有乘客时，返回 true，否则请返回 false）。

输入：trips = [[2,1,5],[3,3,7]], capacity = 4

输出：false

输入：trips = [[2,1,5],[3,3,7]], capacity = 5

输出：true

输入：trips = [[2,1,5],[3,5,7]], capacity = 3

输出：true

输入：trips = [[3,2,7],[3,7,9],[8,3,9]], capacity = 11

输出：true

|  |
| --- |
| class Solution {  public:  bool carPooling(vector<vector<int>>& trips, int capacity) {  map<int, int> record;  for (auto& item : trips) {  record[item[1]] += item[0];  record[item[2]] -= item[0];  }  int sum = 0;  for (auto item : record) {  sum += item.second;  if (sum > capacity) {  return false;  }  }  return true;  }  }; |

**1208. 尽可能使字符串相等**

给你两个长度相同的字符串，s 和 t。

将 s 中的第 i 个字符变到 t 中的第 i 个字符需要 |s[i] - t[i]| 的开销（开销可能为 0），也就是两个字符的 ASCII 码值的差的绝对值。

用于变更字符串的最大预算是 maxCost。在转化字符串时，总开销应当小于等于该预算，这也意味着字符串的转化可能是不完全的。

如果你可以将 s 的子字符串转化为它在 t 中对应的子字符串，则返回可以转化的最大长度。

如果 s 中没有子字符串可以转化成 t 中对应的子字符串，则返回 0。

输入：s = "abcd", t = "bcdf", cost = 3

输出：3

解释：s 中的 "abc" 可以变为 "bcd"。开销为 3，所以最大长度为 3。

输入：s = "abcd", t = "cdef", cost = 3

输出：1

解释：s 中的任一字符要想变成 t 中对应的字符，其开销都是 2。因此，最大长度为 1。

输入：s = "abcd", t = "acde", cost = 0

输出：1

解释：你无法作出任何改动，所以最大长度为 1。

|  |
| --- |
| class Solution {  public:  int equalSubstring(string s, string t, int maxCost) {  int sum = 0;  int i = 0, j = 0;  int ans = 0;  while (j < s.length()) {  sum += abs(s[j] - t[j]);  j++;  while (sum > maxCost) {  sum -= abs(s[i] - t[i]);  i++;  }  ans = max(ans, j - i);  }  return ans;  }  }; |

**1263. 推箱子**

「推箱子」是一款风靡全球的益智小游戏，玩家需要将箱子推到仓库中的目标位置。

游戏地图用大小为 n \* m 的网格 grid 表示，其中每个元素可以是墙、地板或者是箱子。

现在你将作为玩家参与游戏，按规则将箱子 'B' 移动到目标位置 'T' ：

玩家用字符 'S' 表示，只要他在地板上，就可以在网格中向上、下、左、右四个方向移动。

地板用字符 '.' 表示，意味着可以自由行走。

墙用字符 '#' 表示，意味着障碍物，不能通行。

箱子仅有一个，用字符 'B' 表示。相应地，网格上有一个目标位置 'T'。

玩家需要站在箱子旁边，然后沿着箱子的方向进行移动，此时箱子会被移动到相邻的地板单元格。记作一次「推动」。

玩家无法越过箱子。

返回将箱子推到目标位置的最小 推动 次数，如果无法做到，请返回 -1。

输入：grid = [["#","#","#","#","#","#"],

["#","T","#","#","#","#"],

["#",".",".","B",".","#"],

["#",".","#","#",".","#"],

["#",".",".",".","S","#"],

["#","#","#","#","#","#"]]

输出：3

解释：我们只需要返回推箱子的次数。

|  |
| --- |
| struct Status {  int px, py; // 人的位置  int bx, by; // 箱子的位置  int cnt; // 箱子的移动次数  Status() {}  Status(int \_px, int \_py, int \_bx, int \_by, int \_cnt) : px(\_px), py(\_py), bx(\_bx), by(\_by), cnt(\_cnt) {}  };  class Solution {  public:  int minPushBox(vector<vector<char>>& grid) {  int m = grid.size();  int n = grid[0].size();  // 记忆初始化  vector<vector<vector<vector<int>>>> visited(m, vector<vector<vector<int>>>(n, vector<vector<int>>(m, vector<int>(n, false))));  // 确定初始状态  Status root;  for (int i = 0; i < m; i++) {  for (int j = 0; j < n; j++) {  if (grid[i][j] == 'S') {  root.px = i;  root.py = j;  } else if (grid[i][j] == 'B') {  root.bx = i;  root.by = j;  }  }  }  root.cnt = 0;  // 利用双向队列, 达到箱子移动次数的层级分割  deque<Status> Q;  visited[root.px][root.py][root.bx][root.by] = true;  Q.push\_back(root);  // 单层 BFS 部分  int dir[5] = {0, 1, 0, -1, 0};  while (!Q.empty()) {  Status cur = Q.front(); Q.pop\_front();  int px = cur.px;  int py = cur.py;  int bx = cur.bx;  int by = cur.by;  int cnt = cur.cnt;  // 尝试四个方向  for (int i = 0; i < 4; i++) {  int new\_px = px + dir[i];  int new\_py = py + dir[i + 1];  // 人的移动不合法  if (new\_px < 0 || new\_px == m || new\_py < 0  || new\_py == n || grid[new\_px][new\_py] == '#')  continue;  // 人走到箱子的位置  if (new\_px == bx && new\_py == by) {  int new\_bx = bx + dir[i];  int new\_by = by + dir[i + 1];  // 箱子移动不合法  if (new\_bx < 0 || new\_bx == m || new\_by < 0  || new\_by == n || grid[new\_bx][new\_by] == '#') continue;  // 达到终态  if (grid[new\_bx][new\_by] == 'T') {  return cnt + 1;  } else {  // 步数增加了，放队列后面  if (visited[new\_px][new\_py][new\_bx][new\_by] == false) {  visited[new\_px][new\_py][new\_bx][new\_by] = true;  // 箱子移动次数增加, 插入后端  Q.emplace\_back(new\_px, new\_py, new\_bx, new\_by, cnt + 1);  }  }  } else {  // 箱子没有移动，插在后端  if (visited[new\_px][new\_py][bx][by] == false) {  visited[new\_px][new\_py][bx][by] = true;  Q.emplace\_front(new\_px, new\_py, bx, by, cnt);  }  }  }  }  // 终态不可达  return -1;  }  }; |

**321. 拼接最大数**

给定长度分别为 m 和 n 的两个数组，其元素由 0-9 构成，表示两个自然数各位上的数字。现在从这两个数组中选出 k (k <= m + n) 个数字拼接成一个新的数，要求从同一个数组中取出的数字保持其在原数组中的相对顺序。

求满足该条件的最大数。结果返回一个表示该最大数的长度为 k 的数组。

说明: 请尽可能地优化你算法的时间和空间复杂度。

输入: nums1 = [3, 4, 6, 5] nums2 = [9, 1, 2, 5, 8, 3] k = 5

输出: [9, 8, 6, 5, 3]

输入: nums1 = [6, 7] nums2 = [6, 0, 4] k = 5

输出: [6, 7, 6, 0, 4]

输入: nums1 = [3, 9] nums2 = [8, 9] k = 3

输出: [9, 8, 9]

|  |
| --- |
| class Solution {  public:  vector<int> maxNumber(vector<int>& nums1, vector<int>& nums2, int k) {  vector<int> res(k, 0);  int n = nums1.size(), m = nums2.size();  // 假设有最大子序列中有s个元素来自nums1，对所有可能的s值遍历  for (int s=max(0, k-m); s<=min(k, n); s++){  vector<int> temp;  int i = 0, j = 0;  // nums1中长度为s的最大子序列  vector<int> temp1 = maxKsequence(nums1, s);  // nums2中长度为k-s的最大子序列  vector<int> temp2 = maxKsequence(nums2, k-s);  // 对两个子序列进行归并  // lexicographical\_compare：比较两个序列的字典序大小  auto iter1 = temp1.begin(), iter2 = temp2.begin();  while (iter1 != temp1.end() || iter2 != temp2.end()){  temp.push\_back(lexicographical\_compare(iter1, temp1.end(), iter2, temp2.end()) ? \*iter2++ : \*iter1++);  }  // 如果归并后的最大子序列大于目前已找到的最大子序列，则更新解  res = lexicographical\_compare(res.begin(), res.end(), temp.begin(), temp.end()) ? temp : res;  }  return res;  }  // 求数组v的长度为k的最大子序列  vector<int> maxKsequence(vector<int> v, int k){  int n = v.size();  if (n <= k)  return v;  vector<int> res;  int pop = n-k;  for (int i=0; i<n; i++){  while(!res.empty() && v[i]>res.back() && pop-->0)  res.pop\_back();  res.push\_back(v[i]);  }  res.resize(k);  return res;  }  }; |

private boolean compare(int[] nums1, int[] nums2){

int n = Math.min(nums1.length,nums2.length);

for(int i=0;i<n;i++){

if(nums1[i]>nums2[i]) return true;

else if(nums1[i]<nums2[i]) return false;

else continue;

}

return nums1.length>nums2.length;

}