t0626

华为od 机试 2025B卷 - 乘坐保密电梯 (C++ & Python & JAVA & JS

乘坐保密电梯

2025B卷目录点击查看： 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `2025B卷 100分题型`

题目描述

有一座保密大楼，你从0楼到达指定楼层m，必须这样的规则乘坐电梯：
给定一个数字序列，每次根据序列中的数字n，上升n层或者下降n层，前后两次的方向必须相反，规定首次的方向向上，自行组织序列的顺序按规定操作到达指定楼层。
求解到达楼层的序列组合，如果不能到达楼层，给出小于该楼层的最近序列组合。

输入描述

第一行：期望的楼层，取值范围[1,50]; 序列总个数，取值范围[1,23]
第二行：序列，每个值取值范围[1,50]

输出描述

能够达到楼层或者小于该楼层最近的序列

备注

- 操作电梯时不限定楼层范围。
- 必须对序列中的每个项进行操作，不能只使用一部分。

用例1

输入

输出

说明

> 1 2 6，6 2 1均为可行解，按先处理大值的原则结果为6 2 1

题解

思路： `递归回溯`

- 首先确定需要选取上升的数量,规定第一个为上升并且交叉上升和下降，那么选取上升的数量为 `(n + 1) /2` ,向上取整。
- 将输入数组降序排序。
- 使用 `递归回溯` 枚举所有选取 `exceptCount` 上升数，记录其中出现 `上升高度 <= m, 并且最接近m` 的方案。
- 将每个候选方案对应的数组构建出来，选取其中字典序最大的数组就是结果。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<set>
using namespace std;

bool cmp(int x, int y) {
    return x > y;
}
// 整体序列和
int totalSum;
vector<int> res;
int minDiff;
int m,n;

// 递归回溯枚举 选取exceptCount的数
void DFS(vector<int>& ans, int index , int currentSum, int count, int exceptCount, int& visited) {
    if (count > exceptCount) {
        return;
    }
    if (exceptCount == count) {
        // 净增楼层
        int diff = currentSum - (totalSum - currentSum);
       // 上升超过m的不考虑
       if (diff > m) {
         return;
       }

       // 找最小差距
       diff = abs(diff - m);
       if (diff < minDiff) {
           res.clear();
           minDiff = diff;
           res.push_back(visited);
       } else if (diff == minDiff) {
         res.push_back(visited);
       }
       return;
```

```
        }

        for (int i = index; i < n; i++) {
            int num = ans[i];
            int nextCurrentSum = currentSum + num;
            // 剪枝 最终会超过m层
            if (nextCurrentSum - (totalSum - nextCurrentSum) > m) {
                continue;
            }
            // 递归回溯
            visited |= 1 << (n - i -1);
            DFS(ans, i + 1, nextCurrentSum, count + 1, exceptCount, visited);
            visited &= ~(1 << (n - i - 1));

        }
    }

    // 交叉构建结果集
    vector<int> buildArr(vector<int>& ans, int num) {
        vector<bool> visted(n, false);
        for (int i = 0; i < n; i++) {
            visted[i] = 1 & (num >> (n-1-i));
        }
        vector<int> s(n);
        int pos = 0;
        for (int i = 0; i < n; i++) {
            if (visted[i]) {
                s[pos] = ans[i];
                pos += 2;
            }
        }
        pos = 1;
        for (int i = 0; i < n; i++) {
            if (!visted[i]) {
                s[pos] = ans[i];
                pos += 2;
            }
        }
        return s;
    }

    // 在所有mask生成的数组中，找出字典序最大的
    vector<int> findBestArray(vector<int>& ans) {
        vector<int> best;

        for (const int& mask : res) {
            vector<int> curr = buildArr(ans, mask);
            if (best.empty() || curr > best) {
                best = curr;
            }
        }
        return best;
}

    int main() {
        cin >> m >> n;
        vector<int> ans(n);
        int sum = 0;
        for (int i = 0; i < n; i++) {
            cin >> ans[i];
            sum += ans[i];
        }
        //
        totalSum = sum;
        minDiff = sum + m;
        // 从大到小排序
        sort(ans.begin(), ans.end(), cmp);
        // 向上取整 向上的数量
        int exceptUpCount = (n + 1) / 2;

        int status = 0;
        DFS(ans, 0, 0, 0, exceptUpCount, status);

        vector<int>anwser =  findBestArray(ans);
        for (int i = 0; i < n; i++) {
            cout << anwser[i];
            if (i != n-1) {
                cout << " ";
            }
        }
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static int totalSum;
    static List<Integer> res = new ArrayList<>();
    static int minDiff;
    static int m, n;

    // 比较函数，从大到小排序
    static boolean cmp(int x, int y) {
        return x > y;
    }

    // 递归回溯枚举 选取exceptCount的数
    static void DFS(List<Integer> ans, int index, int currentSum, int count, int exceptCount, int visited) {
        if (count > exceptCount) {
            return;
        }
        if (exceptCount == count) {
            // 净增楼层
            int diff = currentSum - (totalSum - currentSum);
            // 上升超过m的不考虑
            if (diff > m) {
                return;
            }
            // 找最小差距
            diff = Math.abs(diff - m);
            if (diff < minDiff) {
                res.clear();
                minDiff = diff;
                res.add(visited);
            } else if (diff == minDiff) {
                res.add(visited);
            }
            return;
        }

        for (int i = index; i < n; i++) {
            int num = ans.get(i);
            int nextCurrentSum = currentSum + num;
            // 剪枝 最终会超过m层
            if (nextCurrentSum - (totalSum - nextCurrentSum) > m) {
                continue;
            }
```

```
                // 递归回溯
                visited |= 1 << (n - i - 1);
                DFS(ans, i + 1, nextCurrentSum, count + 1, exceptCount, visited);
                visited &= ~(1 << (n - i - 1));
            }
        }

    // 交叉构建结果集
    static List<Integer> buildArr(List<Integer> ans, int num) {
        boolean[] visited = new boolean[n];
        for (int i = 0; i < n; i++) {
            visited[i] = (1 & (num >> (n - 1 - i))) == 1;
        }
        List<Integer> s = new ArrayList<>(Collections.nCopies(n, 0));
        int pos = 0;
        for (int i = 0; i < n; i++) {
            if (visited[i]) {
                s.set(pos, ans.get(i));
                pos += 2;
            }
        }
        pos = 1;
        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                s.set(pos, ans.get(i));
                pos += 2;
            }
        }
        return s;
    }

    // 在所有mask生成的数组中，找出字典序最大的
    static List<Integer> findBestArray(List<Integer> ans) {
        List<Integer> best = new ArrayList<>();
        for (int mask : res) {
            List<Integer> curr = buildArr(ans, mask);
            if (best.isEmpty() || compare(curr, best) > 0) {
                best = curr;
            }
        }
        return best;
    }

    // 手动比较两个列表的字典序
    static int compare(List<Integer> a, List<Integer> b) {
        for (int i = 0; i < a.size(); i++) {
            if (!a.get(i).equals(b.get(i))) {
```

```java
                return a.get(i) - b.get(i);
            }
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        m = sc.nextInt();
        n = sc.nextInt();
        List<Integer> ans = new ArrayList<>();
        totalSum = 0;
        for (int i = 0; i < n; i++) {
            int x = sc.nextInt();
            ans.add(x);
            totalSum += x;
        }
        minDiff = totalSum + m;
        // 从大到小排序
        ans.sort((a, b) -> b - a);

        int exceptUpCount = (n + 1) / 2;
        int status = 0;
        DFS(ans, 0, 0, 0, exceptUpCount, status);

        List<Integer> answer = findBestArray(ans);
        for (int i = 0; i < answer.size(); i++) {
            System.out.print(answer.get(i));
            if (i != answer.size() - 1) {
                System.out.print(" ");
            }
        }
    }
}
```

Python

Plain Text

```
totalSum = 0

res = []

minDiff = 0
m, n = 0, 0

def DFS(ans, index, currentSum, count, exceptCount, visited):
    global minDiff, res, totalSum, m, n
    if count > exceptCount:
        return
    if count == exceptCount:

        diff = currentSum - (totalSum - currentSum)

        if diff > m:
            return

        diff = abs(diff - m)
        if diff < minDiff:
            res.clear()
            minDiff = diff
            res.append(visited)
        elif diff == minDiff:
            res.append(visited)
        return

    for i in range(index, n):
        num = ans[i]
        nextCurrentSum = currentSum + num

        if nextCurrentSum - (totalSum - nextCurrentSum) > m:
            continue

        visited |= 1 << (n - i - 1)
        DFS(ans, i + 1, nextCurrentSum, count + 1, exceptCount, visited)
        visited &= ~(1 << (n - i - 1))


def buildArr(ans, mask):
    visited = [(mask >> (n - 1 - i)) & 1 for i in range(n)]
    s = [0] * n
    pos = 0
    for i in range(n):
        if visited[i]:
```

```python
                s[pos] = ans[i]
                pos += 2
    pos = 1
    for i in range(n):
        if not visited[i]:
            s[pos] = ans[i]
            pos += 2
    return s


def findBestArray(ans):
    best = []
    for mask in res:
        curr = buildArr(ans, mask)
        if not best or curr > best:
            best = curr
    return best

if __name__ == "__main__":
    m, n = map(int, input().split())
    ans = list(map(int, input().split()))
    totalSum = sum(ans)
    minDiff = totalSum + m

    ans.sort(reverse=True)

    exceptUpCount = (n + 1) // 2
    status = 0
    DFS(ans, 0, 0, 0, exceptUpCount, status)
    answer = findBestArray(ans)
    print(*answer)
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let input = [];
rl.on('line', function (line) {
    input.push(...line.trim().split(' ').map(Number));
}).on('close', function () {
    let idx = 0;
    const m = input[idx++];
    const n = input[idx++];
    const ans = [];
    for (let i = 0; i < n; i++) {
        ans.push(input[idx++]);
    }
    solve(m, n, ans);
});

let totalSum;
let res = [];
let minDiff;
let m_, n_;

function solve(m, n, ans) {
    m_ = m;
    n_ = n;
    totalSum = ans.reduce((a, b) => a + b, 0);
    minDiff = totalSum + m;

    ans.sort((a, b) => b - a);

    let exceptUpCount = Math.floor((n + 1) / 2);
    let status = 0;
    DFS(ans, 0, 0, 0, exceptUpCount, status);

    const answer = findBestArray(ans);
    console.log(answer.join(' '));
}


function DFS(ans, index, currentSum, count, exceptCount, visited) {
    if (count > exceptCount) {
        return;
    }
    if (count === exceptCount) {

        let diff = currentSum - (totalSum - currentSum);

        if (diff > m_) {
            return;
        }

        diff = Math.abs(diff - m_);
        if (diff < minDiff) {
            res = [];
            minDiff = diff;
            res.push(visited);
        } else if (diff === minDiff) {
            res.push(visited);
        }
        return;
    }
    for (let i = index; i < n_; i++) {
        let num = ans[i];
        let nextCurrentSum = currentSum + num;

        if (nextCurrentSum - (totalSum - nextCurrentSum) > m_) {
            continue;
        }

        visited |= 1 << (n_ - i - 1);
        DFS(ans, i + 1, nextCurrentSum, count + 1, exceptCount, visited);
        visited &= ~(1 << (n_ - i - 1));
    }
}


function buildArr(ans, mask) {
    let visited = Array.from({length: n_}, (_, i) => (mask >> (n_ - 1 - i)) & 1);
    let s = Array(n_).fill(0);
    let pos = 0;
    for (let i = 0; i < n_; i++) {
        if (visited[i]) {
            s[pos] = ans[i];
            pos += 2;
        }
    }
    pos = 1;
    for (let i = 0; i < n_; i++) {
```

```
        if (!visited[i]) {
            s[pos] = ans[i];
            pos += 2;
        }
    }
    return s;
}


function findBestArray(ans) {
    let best = [];
    for (let mask of res) {
        let curr = buildArr(ans, mask);
        if (best.length === 0 || compare(curr, best) > 0) {
            best = curr;
        }
    }
    return best;
}

function compare(a, b) {
    for (let i = 0; i < a.length; i++) {
        if (a[i] !== b[i]) {
            return a[i] - b[i];
        }
    }
    return 0;
}
```

Go

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

var (
  totalSum int
  res      []int
  minDiff  int
  m, n     int
)

func main() {
  fmt.Scan(&m, &n)
  ans := make([]int, n)
  sum := 0
  for i := 0; i < n; i++ {
    fmt.Scan(&ans[i])
    sum += ans[i]
  }
  totalSum = sum
  minDiff = sum + m

  sort.Slice(ans, func(i, j int) bool {
    return ans[i] > ans[j]
  })

  exceptUpCount := (n + 1) / 2
  status := 0
  DFS(ans, 0, 0, 0, exceptUpCount, status)
  answer := findBestArray(ans)
  for i := 0; i < n; i++ {
    fmt.Print(answer[i])
    if i != n-1 {
      fmt.Print(" ")
    }
  }
  fmt.Println()
}


func DFS(ans []int, index, currentSum, count, exceptCount, visited int) {
  if count > exceptCount {
    return
  }
  if count == exceptCount {

    diff := currentSum - (totalSum - currentSum)

    if diff > m {
      return
    }

    if d := abs(diff - m); d < minDiff {
      res = []int{}
      minDiff = d
      res = append(res, visited)
    } else if d == minDiff {
      res = append(res, visited)
    }
    return
  }

  for i := index; i < n; i++ {
    num := ans[i]
    nextCurrentSum := currentSum + num

    if nextCurrentSum-(totalSum-nextCurrentSum) > m {
      continue
    }

    visited |= 1 << (n - i - 1)
    DFS(ans, i+1, nextCurrentSum, count+1, exceptCount, visited)
    visited &= ^(1 << (n - i - 1))
  }
}


func buildArr(ans []int, mask int) []int {
  visited := make([]bool, n)
  for i := 0; i < n; i++ {
    visited[i] = (mask>>(n-1-i))&1 == 1
  }
  s := make([]int, n)
  pos := 0
  for i := 0; i < n; i++ {
    if visited[i] {
      s[pos] = ans[i]
      pos += 2
    }
  }
```

```go
  pos = 1
  for i := 0; i < n; i++ {
    if !visited[i] {
      s[pos] = ans[i]
      pos += 2
    }
  }
  return s
}


func findBestArray(ans []int) []int {
  var best []int
  for _, mask := range res {
    curr := buildArr(ans, mask)
    if best == nil || compare(curr, best) > 0 {
      best = curr
    }
  }
  return best
}

func compare(a, b []int) int {
  for i := 0; i < len(a); i++ {
    if a[i] != b[i] {
      return a[i] - b[i]
    }
  }
  return 0
}

func abs(x int) int {
  if x < 0 {
    return -x
  }
  return x
}
```

> 来自: 华为od 机试 2025B卷 – 乘坐保密电梯 (C++ & Python & JAVA & JS & GO)_2025华为od机试题-CSDN博客

华为OD机试 2025 B卷 - 伐木工 (C++ & Python & JAVA & JS & G

伐木工

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

一根X米长的树木，伐木工切割成不同长度的木材后进行交易，交易价格为每根木头长度的乘积。规定切割后的每根木头长度都为正整数；也可以不切割，直接拿整根树木进行交易。请问伐木工如何尽量少的切割，才能使收益最大化?

输入描述

木材的长度（X ≤ 50）

输出描述

输出最优收益时的各个树木长度，以空格分隔，按升序排列

用例1

输入

输出

说明

> 一根2米长的树木，伐木工不切割，为2 * 1，收益最大为2
> 一根4米长的树木，伐木工不需要切割为2 * 2，省去切割成本，直接整根树木交易，为4 * 1，收益最大为4
> 一根5米长的树木，伐木工切割为2 * 3，收益最大为6
> 一根10米长的树木，伐木工可以切割方式一：3，4，4，也可以切割为方式二：3，2，2，3，但方式二伐木工多切割一次，增加切割成本却买了一样的价格，因此并不是最优收益。

题解

思路： `动态规划 + DFS`

1. 题目要求最大价值前提下，最小切割率的切割方式。
2. 定义 `dp[]` 数组存储内容格式为 `{最大价值， 最小切割数}` ，其中 `dp[i]` 表示含义为x的长度的木板能够获得最大收益并且最小切割数。为了得出切割方案，定义 `path[]` 数组，其中 `path[i]` 表示能获得最大收益且最小切割数的切割长度。
3. 基本逻辑如下：
   - 对于长度为j的木板，初始设置 `dp[j] = {j, 0}, path[j] = 0` .
   - 枚举切割值 `i` （1 <= x < j） ， 如果 `dp[j - i][0] * dp[i] >= dp[j][0]` (能获得更大收益)或者 `dp[j-i][0] * dp[i] == dp[j][0] and dp[j-i][1] + dp[i][1] + 1 < dp[j][1]` 则更新 `dp[j][0] = dp[i][0] * dp[j - i][0], dp[j][1] = dp[i][1] + dp[j - i][1]+1,  path[j] =i `
4. 基于3的规律，枚举j为[1,x]。就能得到长度为x的最大收益以及每一步的切割方案。利用 `path[]` 方程递归就能得到具体的切割路径。
5. 将通过 `DFS` 得到的切割路径，按照升序排序。输出切割路径。

C++

Plain Text

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

// 回溯切割路径
void backtrack(int n, const vector<int>& path, vector<int>& res) {
    if (path[n] == 0) {
        res.push_back(n);  // 不切割，整根加入
    } else {
        backtrack(path[n], path, res);
        backtrack(n - path[n], path, res);
    }
}

int main() {
    int X;
    cin >> X;

    // dp[i] = pair<最大乘积, 最小切割次数>
    vector<pair<int, int>> dp(X + 1, {0, 0});
    vector<int> path(X + 1, 0); // path[i] 表示长度为 i 的第一次切割位置

    for (int i = 1; i <= X; ++i) {
        int maxVal = i;
        int minCuts = 0;
        int cutPos = 0;

        for (int j = 1; j < i; ++j) {
            int leftVal = dp[j].first;
            int leftCuts = dp[j].second;
            int rightVal = dp[i - j].first;
            int rightCuts = dp[i - j].second;

            int prod = leftVal * rightVal;
            int cuts = leftCuts + rightCuts + 1;

            // 优先选最大收益，收益相同时选切割更少的方案
            if (prod > maxVal || (prod == maxVal && cuts < minCuts)) {
                maxVal = prod;
                minCuts = cuts;
                cutPos = j;
            }
        }

        dp[i] = {maxVal, minCuts};
        path[i] = cutPos;
    }

    vector<int> res;
    backtrack(X, path, res);
    sort(res.begin(), res.end());

    for (int i = 0; i < res.size(); ++i) {
        cout << res[i];
        if (i < res.size() - 1)
            cout << " ";
    }
    cout << endl;

    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int X = sc.nextInt();

        // dp[i] = [最大收益, 最小切割次数]
        int[][] dp = new int[X + 1][2];
        int[] path = new int[X + 1]; // path[i] 表示长度为 i 时的第一刀切的位置（0 表示不切）

        for (int i = 1; i <= X; i++) {
            int maxVal = i;      // 不切割的收益
            int minCuts = 0;
            int cutPos = 0;

            for (int j = 1; j < i; j++) {
                int leftVal = dp[j][0], leftCuts = dp[j][1];
                int rightVal = dp[i - j][0], rightCuts = dp[i - j][1];

                int prod = leftVal * rightVal;
                int cuts = leftCuts + rightCuts + 1;

                // 优先收益大，其次切割次数少
                if (prod > maxVal || (prod == maxVal && cuts < minCuts)) {
                    maxVal = prod;
                    minCuts = cuts;
                    cutPos = j;
                }
            }

            dp[i][0] = maxVal;
            dp[i][1] = minCuts;
            path[i] = cutPos;
        }

        // 回溯切割路径
        List<Integer> res = new ArrayList<>();
        backtrack(X, path, res);
        Collections.sort(res);

        // 输出
        for (int i = 0; i < res.size(); i++) {
            System.out.print(res.get(i));
```

```java
            if (i != res.size() - 1)
                System.out.print(" ");
        }
        System.out.println();
    }

    // 回溯函数
    private static void backtrack(int n, int[] path, List<Integer> res) {
        if (path[n] == 0) {
            res.add(n);
        } else {
            backtrack(path[n], path, res);
            backtrack(n - path[n], path, res);
        }
    }
}
```

Python

Plain Text

```
def solve(x):
    dp = [(0, 0)] * (x + 1)
    path = [0] * (x + 1)

    for i in range(1, x + 1):
        maxVal = i
        minCut = 0
        cutPos = 0

        for j in range(1, i):
            leftVal, leftCut = dp[j]
            rightVal, rightCut = dp[i - j]
            prod = leftVal * rightVal
            cuts = leftCut + rightCut + 1


            if prod > maxVal or (prod == maxVal and cuts < minCut):
                maxVal = prod
                minCut = cuts
                cutPos = j

        dp[i] = (maxVal, minCut)
        path[i] = cutPos


    res = []
    def backtrack(n):
        if path[n] == 0:
            res.append(n)
        else:
            backtrack(path[n])
            backtrack(n - path[n])

    backtrack(x)
    print(" ".join(map(str, sorted(res))))


x = int(input())
solve(x)
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let input = [];
rl.on('line', line => {
    input.push(line.trim());
    if (input.length === 1) {
        solve(parseInt(input[0]));
        rl.close();
    }
});

function solve(X) {
    const dp = Array.from({ length: X + 1 }, () => [0, 0]);
    const path = Array(X + 1).fill(0);

    for (let i = 1; i <= X; i++) {
        let maxVal = i;
        let minCuts = 0;
        let cutPos = 0;

        for (let j = 1; j < i; j++) {
            const [leftVal, leftCuts] = dp[j];
            const [rightVal, rightCuts] = dp[i - j];
            const prod = leftVal * rightVal;
            const cuts = leftCuts + rightCuts + 1;

            if (prod > maxVal || (prod === maxVal && cuts < minCuts)) {
                maxVal = prod;
                minCuts = cuts;
                cutPos = j;
            }
        }

        dp[i] = [maxVal, minCuts];
        path[i] = cutPos;
    }


    const res = [];
    function backtrack(n) {
        if (path[n] === 0) {
            res.push(n);
        } else {
            backtrack(path[n]);
            backtrack(n - path[n]);
        }
    }

    backtrack(X);
    res.sort((a, b) => a - b);
    console.log(res.join(' '));
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
)

func backtrack(n int, path []int, res *[]int) {
  if path[n] == 0 {
    *res = append(*res, n)
  } else {
    backtrack(path[n], path, res)
    backtrack(n - path[n], path, res)
  }
}

func main() {
  reader := bufio.NewReader(os.Stdin)
  line, _ := reader.ReadString('\n')
  X, _ := strconv.Atoi(line[:len(line)-1])

  type State struct {
    value int
    cuts  int
  }

  dp := make([]State, X+1)
  path := make([]int, X+1)

  for i := 1; i <= X; i++ {
    maxVal := i
    minCuts := 0
    cutPos := 0

    for j := 1; j < i; j++ {
      left := dp[j]
      right := dp[i-j]
      prod := left.value * right.value
      cuts := left.cuts + right.cuts + 1

      if prod > maxVal || (prod == maxVal && cuts < minCuts) {
        maxVal = prod
```

```
            minCuts = cuts
            cutPos = j
          }
        }

        dp[i] = State{maxVal, minCuts}
        path[i] = cutPos
      }

      var res []int
      backtrack(X, path, &res)
      sort.Ints(res)

      for i, v := range res {
        fmt.Print(v)
        if i != len(res)-1 {
          fmt.Print(" ")
        }
      }
      fmt.Println()
    }
```

来自: 华为OD机试 2025 B卷 – 伐木工 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025 B卷 - 查找接口成功率最优时间段 (C++ & Python & JAVA

查找接口成功率最优时间段

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

服务之间交换的接口成功率作为服务调用关键质量特性，某个时间段内的接口失败率使用一个数组表示，数组中每个元素都是单位时间内失败率数值，数组中的数值为0~100的整数， 给定一个数值(minAverageLost)表示某个时间段内平均失败率容忍值，即平均失败率小于等于minAverageLost，找出数组中最长时间段，如果未找到则直接返回NULL。

输入描述

输入有两行内容，第一行{minAverageLost}，第二行为{数组}，数组元素通过空格(" ")分隔，minAverageLost及数组中元素取值范围为0~100的整数，数组元素的个数不会超过100个。

输出描述

找出平均值小于等于minAverageLost的最长时间段，输出数组下标对，格式{beginIndex}-{endIndx}(下标从0开始)，如果同时存在多个最长时间段，则输出多个下标对且下标对之间使用空格(" ")拼接，多个下标对按下标从小到大排序。

示例1

输入

输出

说明

> 输入解释：minAverageLost=1，数组[0, 1, 2, 3, 4]前3个元素的平均值为1，因此数组第一个至第三个数组下标，即0-2

示例2

输入

输出

说明

> 输入解释：minAverageLost=2，数组[0, 0, 100, 2, 2, 99, 0, 2]通过计算小于等于2的最长时间段为：数组下标为0-1即[0, 0]，数组下标为3-4即[2, 2]，数组下标为6-7即[0, 2]，这三个部分都满足平均值小于等于2的要求，因此输出0-1 3-4 6-7

题解

思路： `前缀和` 基础算法运用

1. 预计算前缀和数组，使用 `前缀和` 数组减少求一个区间和的重复运算。
2. 双for循环从前往后确定两个区间的 `起点和终点` ，判断这个区间的是否满足minAverageLost，并且是否为当前遍历过的最长区间？是的话加入结果候选集。
3. 输出最终最长区间的结果的满足条件的区间起点和终点。如果不存在满足条件的区间，输出NULL。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<map>
using namespace std;

int main() {
    int minAverageLost;

    cin >> minAverageLost;
    map<int, vector<pair<int, int>>> mp;
    vector<int> ans(100, 0);
    int len = 0;
    int tmp;

    while (cin >> tmp) {
        ans[len++] = tmp;
    }
    int currentLen = 0;
    vector<int> prefix(len+ 1, 0);

    for (int i = 1; i <= len; i++) {
        prefix[i] = prefix[i-1] + ans[i-1];
    }

    for (int i = 0; i <= len; i++) {
        for (int j = i + 1 ; j <= len; j++) {
            if (j - i < currentLen) {
                continue;
            }
            int sum = prefix[j] - prefix[i];
            if (sum  * 1.0 / (j-i) <= minAverageLost){
                currentLen = max(currentLen, j - i);
                mp[currentLen].push_back({i, j-1});
            }
        }
    }

    if (currentLen == 0) {
        cout<< "NULL";
        return 0;
    }

    vector<pair<int, int>> maxLenV = mp[currentLen];
    for (int i = 0; i < maxLenV.size(); i++) {
        if (i !=0) {
            cout << " ";
        }
        pair<int, int> p = maxLenV[i];
        cout<< p.first << "-"<<p.second;
    }
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int minAverageLost = scanner.nextInt();
        List<Integer> ans = new ArrayList<>();

        while (scanner.hasNextInt()) {
            ans.add(scanner.nextInt());
        }
        int len = ans.size();

        if (len == 0) {
            System.out.println("NULL");
            return;
        }


        int[] prefix = new int[len + 1];
        for (int i = 1; i <= len; i++) {
            prefix[i] = prefix[i - 1] + ans.get(i - 1);
        }

        int currentLen = 0;
        Map<Integer, List<int[]>> mp = new HashMap<>();


        for (int i = 0; i <= len; i++) {
            for (int j = i + 1; j <= len; j++) {
                if (j - i < currentLen) {
                    continue;
                }
                int sum = prefix[j] - prefix[i];
                if ((double) sum / (j - i) <= minAverageLost) {
                    currentLen = Math.max(currentLen, j - i);
                    mp.putIfAbsent(currentLen, new ArrayList<>());
                    mp.get(currentLen).add(new int[]{i, j - 1});
                }
            }
        }

        if (currentLen == 0) {
            System.out.println("NULL");
```

```java
                return;
            }

            List<int[]> maxLenV = mp.get(currentLen);
            for (int i = 0; i < maxLenV.size(); i++) {
                if (i != 0) System.out.print(" ");
                System.out.print(maxLenV.get(i)[0] + "-" + maxLenV.get(i)[1]);
            }
        }
    }
```

Python

Plain Text

```
import sys

def main():
    data = list(map(int, sys.stdin.read().split()))

    min_average_lost = data[0]
    ans = data[1:]
    len_ans = len(ans)

    if len_ans == 0:
        print("NULL")
        return


    prefix = [0] * (len_ans + 1)
    for i in range(1, len_ans + 1):
        prefix[i] = prefix[i - 1] + ans[i - 1]

    current_len = 0
    mp = {}


    for i in range(len_ans + 1):
        for j in range(i + 1, len_ans + 1):
            if j - i < current_len:
                continue
            total = prefix[j] - prefix[i]
            if total / (j - i) <= min_average_lost:
                current_len = max(current_len, j - i)
                mp.setdefault(current_len, []).append((i, j - 1))

    if current_len == 0:
        print("NULL")
        return

    result = ["{}-{}".format(start, end) for start, end in mp[current_len]]
    print(" ".join(result))

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputData = [];

rl.on("line", (line) => {
    inputData.push(line.trim());
    if (inputData.length === 2) rl.close();
});

rl.on("close", () => {
    if (inputData.length < 2) {
        console.log("NULL");
        return;
    }

    const minAverageLost = parseInt(inputData[0]);
    const ans = inputData[1].split(/\s+/).map(Number);
    const len = ans.length;

    if (len === 0) {
        console.log("NULL");
        return;
    }


    const prefix = new Array(len + 1).fill(0);
    for (let i = 1; i <= len; i++) {
        prefix[i] = prefix[i - 1] + ans[i - 1];
    }

    let currentLen = 0;
    const mp = new Map();


    for (let i = 0; i <= len; i++) {
        for (let j = i + 1; j <= len; j++) {
            if (j - i < currentLen) continue;
            const sum = prefix[j] - prefix[i];
            if (sum / (j - i) <= minAverageLost) {
                currentLen = Math.max(currentLen, j - i);
                if (!mp.has(currentLen)) mp.set(currentLen, []);
                mp.get(currentLen).push([i, j - 1]);
            }
        }
    }

    if (currentLen === 0) {
        console.log("NULL");
        return;
    }

    const maxLenV = mp.get(currentLen);
    console.log(maxLenV.map(([start, end]) => `${start}-${end}`).join(" "));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

func main() {
  scanner := bufio.NewScanner(os.Stdin)


  scanner.Scan()
  minAverageLost, _ := strconv.Atoi(scanner.Text())


  if !scanner.Scan() {
    fmt.Println("NULL")
    return
  }


  numsStr := strings.Fields(scanner.Text())
  var ans []int
  for _, num := range numsStr {
    val, _ := strconv.Atoi(num)
    ans = append(ans, val)
  }

  lenAns := len(ans)
  if lenAns == 0 {
    fmt.Println("NULL")
    return
  }


  prefix := make([]int, lenAns+1)
  for i := 1; i <= lenAns; i++ {
    prefix[i] = prefix[i-1] + ans[i-1]
  }

  currentLen := 0
  mp := make(map[int][][]int)
```

```go


  for i := 0; i <= lenAns; i++ {
    for j := i + 1; j <= lenAns; j++ {
      if j-i < currentLen {
        continue
      }
      sum := prefix[j] - prefix[i]
      if float64(sum)/float64(j-i) <= float64(minAverageLost) {
        currentLen = max(currentLen, j-i)
        mp[currentLen] = append(mp[currentLen], []int{i, j - 1})
      }
    }
  }

  if currentLen == 0 {
    fmt.Println("NULL")
    return
  }


  var result []string
  for _, pair := range mp[currentLen] {
    result = append(result, fmt.Sprintf("%d-%d", pair[0], pair[1]))
  }
  fmt.Println(strings.Join(result, " "))
}


func max(a, b int) int {
  if a > b {
    return a
  }
  return b
}
```

> 来自: 华为OD机试 2025 B卷 – 查找接口成功率最优时间段 (C++ & Python & JAVA & JS & GO)–CSDN博客

华为OD机试 2025 B卷 - 抢7游戏 (C++ & Python & JAVA & JS &

抢7游戏

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

A、B两个人玩抢7游戏，游戏规则为：
A先报一个起始数字 X（10 ≤ 起始数字 ≤ 10000），B报下一个数字 Y （X – Y < 3），A再报一个数字 Z（Y – Z < 3），以此类推，直到其中一个抢到7，抢到7即为胜者；
在B赢得比赛的情况下，一共有多少种组合?

输入描述

起始数字 M。`1 ≤ M ≤ 10000`

输出描述

B能赢得比赛的组合次数

用例1

输入

输出

题解

思路：`动态规划`

1. 其实这道题的思路一道经典题 `爬楼梯方案数` 非常相似。定义 `dpA[]` 其中 `dpA[i]` 表示A含数字i的方案数，定义 `dpB[]` 其中 `dpB[j]` 表示B喊B的方案数。
2. 由题意可以 `dpA[i]` 的值由dpB[i+1]和dpB[i+2]的决定。 `dpB[j]` 类似。所以可以直接得出状态转移方程
   - `dpB[i] = dpA[i + 1] + dpA[i + 2]`

    - `dpA[i] = dpB[i + 1] + dpB[i + 2]`
3. 明白2的状态方程之后，从m枚举到7就能直到，B能赢得比赛的组合数。对应结果为 `dpB[7]` .

> 额外注意，m的值[1,10000]，进行状态转移过程会出现非常大的数，经典的 `大数相加` 问题。对于没有内置大数的编程语言，需要手动定义函数实现(字符串相加)。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
using namespace std;


// 实现两个大整数的加法
string addBigNumbers(const string& num1, const string& num2) {
    string res;
    // 进位
    int carry = 0;

    int i = num1.size() - 1;
    int j = num2.size() - 1;

    // 从低位到高位逐位相加
    while (i >= 0 || j >= 0 || carry) {
        int digit1 = i >= 0 ? (num1[i--] - '0') : 0;
        int digit2 = j >= 0 ? (num2[j--] - '0') : 0;
        int sum = digit1 + digit2 + carry;
        res += (sum % 10 + '0');
        carry = sum / 10;
    }

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

int main() {
    int m;
    cin >> m;
    // i 由 i+1 i+2 累加得来，所以定义长度为m + 2
    vector<string> dpA(m + 2, "0");
    dpA[m] = "1";
    vector<string> dpB(m + 2, "0");
    for (int i = m - 1; i >= 7; i--) {
        // B得出的方案数 由 A叫 i + 1  + 叫 i+2 的方案数得来
        dpB[i] =  addBigNumbers(dpA[i + 1],dpA[i + 2]);
        // A得出的方案数 由 B叫 i + 1  + 叫 i+2 的方案数得来
        dpA[i] = addBigNumbers(dpB[i + 1],dpB[i + 2]);
    }
    cout << dpB[7];
```

```
        return 0;
    }
```

JAVA

Plain Text

```
import java.math.BigInteger;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt();

        // i 由 i+1 i+2 累加得来，所以定义长度为 m + 2
        BigInteger[] dpA = new BigInteger[m + 2];
        BigInteger[] dpB = new BigInteger[m + 2];

        for (int i = 0; i < m + 2; i++) {
            dpA[i] = BigInteger.ZERO;
            dpB[i] = BigInteger.ZERO;
        }

        dpA[m] = BigInteger.ONE;

        for (int i = m - 1; i >= 7; i--) {
            // B得出的方案数 由 A叫 i + 1  + 叫 i+2 的方案数得来
            dpB[i] = dpA[i + 1].add(dpA[i + 2]);
            // A得出的方案数 由 B叫 i + 1  + 叫 i+2 的方案数得来
            dpA[i] = dpB[i + 1].add(dpB[i + 2]);
        }

        System.out.println(dpB[7]);
    }
}
```

Python

Plain Text

```
def main():
    m = int(input())


    dpA = [0] * (m + 2)
    dpB = [0] * (m + 2)

    dpA[m] = 1


    for i in range(m - 1, 6, -1):

        dpB[i] = dpA[i + 1] + dpA[i + 2]

        dpA[i] = dpB[i + 1] + dpB[i + 2]

    print(dpB[7])

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on('line', function (line) {
    inputLines.push(line);
}).on('close', function () {
    let m = parseInt(inputLines[0]);


    const dpA = Array(m + 2).fill(0n);
    const dpB = Array(m + 2).fill(0n);

    dpA[m] = 1n;

    for (let i = m - 1; i >= 7; i--) {

        dpB[i] = dpA[i + 1] + dpA[i + 2];

        dpA[i] = dpB[i + 1] + dpB[i + 2];
    }

    console.log(dpB[7].toString());
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math/big"
)

func main() {
  var m int
  fmt.Scan(&m)


  dpA := make([]*big.Int, m+2)
  dpB := make([]*big.Int, m+2)

  for i := 0; i < m+2; i++ {
    dpA[i] = big.NewInt(0)
    dpB[i] = big.NewInt(0)
  }

  dpA[m] = big.NewInt(1)

  for i := m - 1; i >= 7; i-- {

    dpB[i] = new(big.Int).Add(dpA[i+1], dpA[i+2])

    dpA[i] = new(big.Int).Add(dpB[i+1], dpB[i+2])
  }

  fmt.Println(dpB[7].String())
}
```

> 来自：华为OD机试 2025 B卷 – 抢7游戏（C++ & Python & JAVA & JS & GO)-CSDN博客

抢7游戏

华为OD机试真题目录点击查看：华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

A、B两个人玩抢7游戏，游戏规则为：
A先报一个起始数字 X（10 ≤ 起始数字 ≤ 10000），B报下一个数字 Y （X – Y < 3），A再报一个数字 Z（Y – Z < 3），以此类推，直到其中一个抢到7，抢到7即为胜者；
在B赢得比赛的情况下，一共有多少种组合?

输入描述

起始数字 M。`1 ≤ M ≤ 10000`

输出描述

B能赢得比赛的组合次数

用例1

输入

输出

题解

思路：`动态规划`

1. 其实这道题的思路一道经典题 `爬楼梯方案数` 非常相似。定义 `dpA[]` 其中 `dpA[i]` 表示A含数字i的方案数，定义 `dpB[]` 其中 `dpB[j]` 表示B喊B的方案数。
2. 由题意可以 `dpA[i]` 的值由dpB[i+1]和dpB[i+2]的决定。 `dpB[j]` 类似。所以可以直接得出状态转移方程
   - `dpB[i] = dpA[i + 1] + dpA[i + 2]`
   - ` dpA[i] = dpB[i + 1] + dpB[i + 2]`
3. 明白2的状态方程之后，从m枚举到7就能直到，B能赢得比赛的组合数。对应结果为 `dpB[7]` .

> 额外注意，m的值[1,10000]，进行状态转移过程会出现非常大的数，经典的 `大数相加` 问题。对于没有内置大数的编程语言，需要手动定义函数实现(字符串相加)。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
using namespace std;


// 实现两个大整数的加法
string addBigNumbers(const string& num1, const string& num2) {
    string res;
    // 进位
    int carry = 0;

    int i = num1.size() - 1;
    int j = num2.size() - 1;

    // 从低位到高位逐位相加
    while (i >= 0 || j >= 0 || carry) {
        int digit1 = i >= 0 ? (num1[i--] - '0') : 0;
        int digit2 = j >= 0 ? (num2[j--] - '0') : 0;
        int sum = digit1 + digit2 + carry;
        res += (sum % 10 + '0');
        carry = sum / 10;
    }

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

int main() {
    int m;
    cin >> m;
    // i 由 i+1 i+2 累加得来，所以定义长度为m + 2
    vector<string> dpA(m + 2, "0");
    dpA[m] = "1";
    vector<string> dpB(m + 2, "0");
    for (int i = m - 1; i >= 7; i--) {
        // B得出的方案数 由 A叫 i + 1  + 叫 i+2 的方案数得来
        dpB[i] =  addBigNumbers(dpA[i + 1],dpA[i + 2]);
        // A得出的方案数 由 B叫 i + 1  + 叫 i+2 的方案数得来
        dpA[i] = addBigNumbers(dpB[i + 1],dpB[i + 2]);
    }
    cout << dpB[7];
    return 0;
}
```

JAVA

```
import java.math.BigInteger;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt();

        // i 由 i+1 i+2 累加得来，所以定义长度为 m + 2
        BigInteger[] dpA = new BigInteger[m + 2];
        BigInteger[] dpB = new BigInteger[m + 2];

        for (int i = 0; i < m + 2; i++) {
            dpA[i] = BigInteger.ZERO;
            dpB[i] = BigInteger.ZERO;
        }

        dpA[m] = BigInteger.ONE;

        for (int i = m - 1; i >= 7; i--) {
            // B得出的方案数 由 A叫 i + 1  + 叫 i+2 的方案数得来
            dpB[i] = dpA[i + 1].add(dpA[i + 2]);
            // A得出的方案数 由 B叫 i + 1  + 叫 i+2 的方案数得来
            dpA[i] = dpB[i + 1].add(dpB[i + 2]);
        }

        System.out.println(dpB[7]);
    }
}
```

Python

Plain Text

```
def main():
    m = int(input())


    dpA = [0] * (m + 2)
    dpB = [0] * (m + 2)

    dpA[m] = 1


    for i in range(m - 1, 6, -1):

        dpB[i] = dpA[i + 1] + dpA[i + 2]

        dpA[i] = dpB[i + 1] + dpB[i + 2]

    print(dpB[7])

if __name__ == "__main__":
    main()
```

JavaScript

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on('line', function (line) {
    inputLines.push(line);
}).on('close', function () {
    let m = parseInt(inputLines[0]);


    const dpA = Array(m + 2).fill(0n);
    const dpB = Array(m + 2).fill(0n);

    dpA[m] = 1n;

    for (let i = m - 1; i >= 7; i--) {

        dpB[i] = dpA[i + 1] + dpA[i + 2];

        dpA[i] = dpB[i + 1] + dpB[i + 2];
    }

    console.log(dpB[7].toString());
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math/big"
)

func main() {
  var m int
  fmt.Scan(&m)


  dpA := make([]*big.Int, m+2)
  dpB := make([]*big.Int, m+2)

  for i := 0; i < m+2; i++ {
    dpA[i] = big.NewInt(0)
    dpB[i] = big.NewInt(0)
  }

  dpA[m] = big.NewInt(1)

  for i := m - 1; i >= 7; i-- {

    dpB[i] = new(big.Int).Add(dpA[i+1], dpA[i+2])

    dpA[i] = new(big.Int).Add(dpB[i+1], dpB[i+2])
  }

  fmt.Println(dpB[7].String())
}
```

来自: 华为OD机试 2025 B卷 – 抢7游戏 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD 机试 2025 B卷 - 数组二叉树 (C++ & Python & JAVA & JS

数组二叉树

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

二叉树也可以用数组来存储，给定一个数组，树的根节点的值存储在下标1，对于存储在下标N的节点，它的左子节点和右子节点分别存储在下标2*N和2*N+1，并且我们用值-1代表一个节点为空。
给定一个数组存储的二叉树，试求**从根节点到最小的叶子节点的路径**，路径由节点的值组成。

输入描述

输入一行为数组的内容，数组的每个元素都是正整数，元素间用空格分隔。
注意第一个元素即为根节点的值，即数组的第N个元素对应下标N，下标0在树的表示中没有使用，所以我们省略了。
输入的树最多为7层。

输出描述

输出从根节点到最小叶子节点的路径上，各个节点的值，由空格分隔，用例保证最小叶子节点只有一个。

用例1

输入

输出

说明

> 最小叶子节点的路径为3 7 2。

示例二

输入

```
5 9 8 -1 -1 7 -1 -1 -1 -1 -1 6
```

输出

说明

```
最小叶子节点的路径为5 8 7 6，注意数组仅存储至最后一个非空节点，故不包含节点“7”右子节点的-1。
```

题解

思路： `逻辑分析题 + 模拟`

1. 题目要求 `从根节点到最小叶子节点的路径` ，这题如果从根节点出发去寻找路径需要考虑的情况蛮多了，但是如果你先找到最小叶子节点，再往上递推到根节点就会非常容易。 `转变思路`
2. 接收输入。从后往前遍历找到最小叶子节点位置 `pos` 。
3. 循环迭代，获取从最小叶子节点到到根节点的路径。在数组中对于 `possition` 的父节点位置为 `(posistion - 1) /2` .
4. 经过3处理之后，输出得到的路径即可。

C++

Plain Text

```
#include <cstdint>
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include <cmath>
#include <climits>
using namespace std;

// 计算完全二叉树的层数
int compute_height(int N) {
    return ceil(log2(N + 1));
}

bool judege(vector<int> ans, int pos) {
    int n = ans.size();
    return (pos >= n || ans[pos] == -1);
}

int main() {
    vector<int> ans;
    int tmp;
    while (cin >> tmp) {
        ans.push_back(tmp);
    }

    vector<int> res;
    int n = ans.size();
    // 最小叶子节点值位置
    int pos = -1;
    // 最小叶子节点值位置
    int minValue = INT_MAX;

    for (int i = 0; i < n; i++) {
        int leftIndex = 2 * (i+1) -1;
        int rightIndex = 2 * (i+1);
        if (ans[i] == -1) {
            continue;
        }
        // 判断是否是叶子节点
        if (judege(ans, leftIndex) &&judege(ans, rightIndex)) {
              if (ans[i] < minValue) {
                 pos = i;
```

```
                    minValue = ans[i];
                }
            }
        }
        // 添加叶子节点到根节点的值
        while (pos != 0) {
            res.push_back(ans[pos]);
            pos = (pos-1) / 2;
        }
        res.push_back(ans[0]);

        for (int i = res.size()-1; i >= 0; i--) {
            cout << res[i];
            if (i != 0) {
                cout << " ";
            }
        }
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 计算完全二叉树的层数
    public static int computeHeight(int N) {
        return (int) Math.ceil(Math.log(N + 1) / Math.log(2));
    }

    public static boolean judge(List<Integer> ans, int pos) {
        return pos >= ans.size() || ans.get(pos) == -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Integer> ans = new ArrayList<>();

        // 读取输入
        while (sc.hasNextInt()) {
            ans.add(sc.nextInt());
        }

        List<Integer> res = new ArrayList<>();
        int n = ans.size();

        int pos = -1;
        int minValue = Integer.MAX_VALUE;

        // 查找最小的叶子节点值位置
        for (int i = 0; i < n; i++) {
            int leftIndex = 2 * (i + 1) - 1;
            int rightIndex = 2 * (i + 1);
            if (ans.get(i) == -1) {
                continue;
            }

            // 判断是否是叶子节点
            if (judge(ans, leftIndex) && judge(ans, rightIndex)) {
                if (ans.get(i) < minValue) {
                    pos = i;
                    minValue = ans.get(i);
                }
            }
        }

        // 从叶子节点到根节点
        while (pos != 0) {
            res.add(ans.get(pos));
            pos = (pos - 1) / 2;
        }
        res.add(ans.get(0));

        // 输出结果
        for (int i = res.size() - 1; i >= 0; i--) {
            System.out.print(res.get(i));
            if (i != 0) {
                System.out.print(" ");
            }
        }

        sc.close();
    }
}
```

Python

Plain Text

```
import math


def compute_height(N):
    return math.ceil(math.log2(N + 1))


def judge(ans, pos):
    return pos >= len(ans) or ans[pos] == -1

def main():
    ans = []


    try:

        inputs = input().split()
        for num in inputs:
            ans.append(int(num))
    except EOFError:
        pass

    res = []
    n = len(ans)


    pos = -1
    min_value = float('inf')


    for i in range(n):
        left_index = 2 * (i + 1) - 1
        right_index = 2 * (i + 1)
        if ans[i] == -1:
            continue


        if judge(ans, left_index) and judge(ans, right_index):
            if ans[i] < min_value:
                pos = i
                min_value = ans[i]


    while pos != 0:
        res.append(ans[pos])
```

```python
            pos = (pos - 1) // 2
        res.append(ans[0])


    print(" ".join(map(str, res[::-1])))

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
function computeHeight(N) {
    return Math.ceil(Math.log2(N + 1));
}


function judge(ans, pos) {
    return pos >= ans.length || ans[pos] === -1;
}

function main() {

    let input = '';
    let stdin = process.stdin;
    stdin.setEncoding('utf-8');
    stdin.on('data', function (data) {
        input += data;
    });

    stdin.on('end', function () {

        const ans = input.trim().split(/\s+/).map(Number);

        let res = [];
        const n = ans.length;


        let pos = -1;
        let minValue = Infinity;


        for (let i = 0; i < n; i++) {
            let leftIndex = 2 * (i + 1) - 1;
            let rightIndex = 2 * (i + 1);

            if (ans[i] === -1) {
                continue;
            }


            if (judge(ans, leftIndex) && judge(ans, rightIndex)) {
                if (ans[i] < minValue) {
                    pos = i;
                    minValue = ans[i];
                }
            }
        }


        while (pos !== 0) {
            res.push(ans[pos]);
            pos = Math.floor((pos - 1) / 2);
        }
        res.push(ans[0]);


        console.log(res.reverse().join(' '));
    });
}

main();
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "math"
  "os"
  "strconv"
  "strings"
)




func judge(ans []int, pos int) bool {
  return pos >= len(ans) || ans[pos] == -1
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)

  scanner.Scan()
  input := scanner.Text()
  inputs := strings.Fields(input)


  var ans []int
  for _, val := range inputs {
    num, err := strconv.Atoi(val)
    if err != nil {
      fmt.Println("输入无效")
      return
    }
    ans = append(ans, num)
  }

  var res []int
  pos := -1
  minValue := math.MaxInt


  for i := 0; i < len(ans); i++ {
    leftIndex := 2*(i+1) - 1
    rightIndex := 2*(i+1)
    if ans[i] == -1 {
      continue
    }

    if judge(ans, leftIndex) && judge(ans, rightIndex) {
      if ans[i] < minValue {
        pos = i
        minValue = ans[i]
      }
    }
  }


  for pos != 0 {
    res = append(res, ans[pos])
    pos = (pos - 1) / 2
  }
  res = append(res, ans[0])


  for i := len(res) - 1; i >= 0; i-- {
    fmt.Print(res[i])
    if i != 0 {
      fmt.Print(" ")
    }
  }
}
```

来自: 华为OD 机试 2025 B卷 – 数组二叉树 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD 机试 2025 B卷 - 路灯照明问题 (C++ & Python & JAVA & JS

路灯照明问题

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

在一条笔直的公路上安装了N个路灯，从位置0开始安装，路灯之间间距固定为100米。
每个路灯都有自己的照明半径，请计算第一个路灯和最后一个路灯之间，无法照明的区间的长度和。

输入描述

第一行为一个数N，表示路灯个数，1<=N<=100000
第二行为N个空格分隔的数，表示路灯的照明半径，1<=照明半径<=100000*100

输出描述

第一个路灯和最后一个路灯之间，无法照明的区间的长度和.

用例1

输入

输出

说明

> 路灯1覆盖0-50，路灯2覆盖50-100，路灯1和路灯2之间(0米-100米)无未覆盖的区间。

用例2

输入

输出

说明

> [170,180],[220,230]，两个未覆盖的区间，总里程为20

题解

思路： `区间合并`

1. 通过路灯数量，可以得出第一个路灯和最后一个路灯的总距离。
2. 通过每一个路灯的照明半径 `r` 可以得出，这个路灯的覆盖区间. 例如现在枚举的路灯为i，照明长度为 [i * 100 - r, i * 100 + r]。
3. 通过第二步可以得出 `n` 的区间，接下来进行区间合并就行. `注意递归合并区间` 。
4. 合并区间之后，使用总距离 - `每个合并 覆盖区间长度` 就是结果。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<stack>
using namespace std;

int main() {
    int n ;
    cin >> n;
    vector<int> ans(n);
    stack<pair<int,int>> stk;
    for (int i = 0; i < n; i++) {
        cin >> ans[i];
        int left = (i * 100) - ans[i];
        int right = (i*100) + ans[i];
        // 递归区间合并
        while (!stk.empty() && stk.top().second >= left) {
            pair<int,int> tmp = stk.top();
            stk.pop();
            left = min(tmp.first, left);
            right = max(tmp.second, right);
        }
        stk.push({left, right});
    }
    int res = 0;
    // 计算多个区间中空白照明长度
    while (stk.size() != 1) {
        pair<int,int> top = stk.top();
        stk.pop();
        res += top.first - stk.top().second;
    }

    cout << res;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取整数 n
        int n = scanner.nextInt();
        int[] ans = new int[n];
        Stack<int[]> stk = new Stack<>();

        // 读取 n 个数并计算区间合并
        for (int i = 0; i < n; i++) {
            ans[i] = scanner.nextInt();
            int left = (i * 100) - ans[i];
            int right = (i * 100) + ans[i];

            // 递归区间合并
            while (!stk.isEmpty() && stk.peek()[1] >= left) {
                int[] tmp = stk.pop();
                left = Math.min(tmp[0], left);
                right = Math.max(tmp[1], right);
            }
            stk.push(new int[]{left, right});
        }

        // 计算多个区间中空白照明长度
        int res = 0;
        while (stk.size() > 1) {
            int[] top = stk.pop();
            res += top[0] - stk.peek()[1];
        }

        // 输出结果
        System.out.println(res);
    }
}
```

Python

Plain Text

```
import sys

def main():

    n = int(sys.stdin.readline().strip())
    ans = list(map(int, sys.stdin.readline().strip().split()))
    stk = []


    for i in range(n):
        left = (i * 100) - ans[i]
        right = (i * 100) + ans[i]


        while stk and stk[-1][1] >= left:
            tmp = stk.pop()
            left = min(tmp[0], left)
            right = max(tmp[1], right)
        stk.append((left, right))


    res = 0
    while len(stk) > 1:
        top = stk.pop()
        res += top[0] - stk[-1][1]


    print(res)

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on("line", (line) => {
    inputLines.push(line);
}).on("close", () => {

    let n = parseInt(inputLines[0]);
    let ans = inputLines[1].split(" ").map(Number);
    let stk = [];


    for (let i = 0; i < n; i++) {
        let left = (i * 100) - ans[i];
        let right = (i * 100) + ans[i];


        while (stk.length > 0 && stk[stk.length - 1][1] >= left) {
            let tmp = stk.pop();
            left = Math.min(tmp[0], left);
            right = Math.max(tmp[1], right);
        }
        stk.push([left, right]);
    }


    let res = 0;
    while (stk.length > 1) {
        let top = stk.pop();
        res += top[0] - stk[stk.length - 1][1];
    }


    console.log(res);
});
```

Go

Plain Text

```
package main

import (
  "fmt"
)

func main() {
  var n int
  fmt.Scan(&n)

  ans := make([]int, n)
  for i := 0; i < n; i++ {
    fmt.Scan(&ans[i])
  }

  type Interval struct {
    left  int
    right int
  }

  stack := []Interval{}
  for i := 0; i < n; i++ {
    left := i*100 - ans[i]
    right := i*100 + ans[i]


    for len(stack) > 0 && stack[len(stack)-1].right >= left {
      top := stack[len(stack)-1]
      stack = stack[:len(stack)-1]
      left = min(top.left, left)
      right = max(top.right, right)
    }
    stack = append(stack, Interval{left, right})
  }


  res := 0
  for i := 1; i < len(stack); i++ {
    res += stack[i].left - stack[i-1].right
  }

  fmt.Println(res)
}

func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}

func max(a, b int) int {
  if a > b {
    return a
  }
  return b
}
```

> 来自: 华为OD 机试 2025 B卷 – 路灯照明问题 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025 B卷 - 最佳投资方式 (C++ & Python & JAVA & JS

最佳投资方式 / 虚拟理财游戏

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

在一款虚拟游戏中生活，你必须进行投资以增强在虚拟游戏中的资产以免被淘汰出局。
现有一家Bank，它提供有若干理财产品 m 个，风险及投资回报不同，你有 N（元）进行投资，能接收的总风险值为X。
你要在可接受范围内选择最优的投资方式获得最大回报。
备注：

- 在虚拟游戏中，每项投资风险值相加为总风险值；
- 在虚拟游戏中，最多只能投资2个理财产品；
- 在虚拟游戏中，最小单位为整数，不能拆分为小数；
- 投资额*回报率=投资回报

输入描述

第一行：

- 产品数（取值范围[1,20]）
- 总投资额（整数，取值范围[1, 10000]）
- 可接受的总风险（整数，取值范围[1,200]）

第二行：产品投资回报率序列，输入为整数，取值范围[1,60]
第三行：产品风险值序列，输入为整数，取值范围[1, 100]
第四行：最大投资额度序列，输入为整数，取值范围[1, 10000]

输出描述

每个产品的投资额序列

示例1

输入

```
5 100 10
10 20 30 40 50
3 4 5 6 10
20 30 20 40 30
```

输出

说明

> 投资第二项30个单位，第四项40个单位，总的投资风险为两项相加为4+6=10

题解

思路： `模拟`

1. 只需要考虑两种投资方式
    a. 只投资一种产品。
    b. 组合投资，两两组合产品进行投资，优先把金额投资到回报率高的产品上。
2. 结果为上述两种情况中出现的回报额最大的组合情况。尝试 `一种` 或者 `两种组合` 需要考虑以下因素。
    - 投资产品风险和是否大于 `X` . 超过则说明组合不合法。
    - 两种组合情况下，优先将资金投入到回报率大的商品。
3. 通过循环匹配不同组合方案，记录其中能得到最大投资回报的方案就是结果，按题目要求格式输出即可。

C++

Plain Text

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include<iterator>

using namespace std;

vector<int> readIntArray() {
    string line;
    getline(cin, line);
    istringstream iss(line);
    return vector<int>(istream_iterator<int>(iss), {});
}

int main() {
    vector<int> input = readIntArray();
    int m = input[0], N = input[1], X = input[2];
    vector<int> returns = readIntArray();
    vector<int> risks = readIntArray();
    vector<int> maxInvestments = readIntArray();

    int maxReturn = 0;
    vector<int> bestInvestments(m, 0);

    for (int i = 0; i < m; i++) {
        if (risks[i] > X) continue;

        int investI = min(N, maxInvestments[i]);
        int retI = investI * returns[i];
        if (retI > maxReturn) {
            maxReturn = retI;
            fill(bestInvestments.begin(), bestInvestments.end(), 0);
            bestInvestments[i] = investI;
        }

        for (int j = i + 1; j < m; j++) {
            if (risks[i] + risks[j] > X) continue;

            int investI, investJ;

            if (returns[i] > returns[j]) {
                investI = min(N, maxInvestments[i]);
                investJ = min(N - investI, maxInvestments[j]);
            } else {
                investJ = min(N, maxInvestments[j]);
                investI = min(N - investJ, maxInvestments[i]);
            }

            int retPair = investI * returns[i] + investJ * returns[j];
            if (retPair > maxReturn) {
                maxReturn = retPair;
                fill(bestInvestments.begin(), bestInvestments.end(), 0);
                bestInvestments[i] = investI;
                bestInvestments[j] = investJ;
            }
        }
    }

    for (int investment : bestInvestments) {
        cout << investment << " ";
    }
    cout << endl;

    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {

    private static List<Integer> readIntArray(Scanner scanner) {
        String[] tokens = scanner.nextLine().split(" ");
        List<Integer> numbers = new ArrayList<>();
        for (String token : tokens) {
            numbers.add(Integer.parseInt(token));
        }
        return numbers;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        List<Integer> input = readIntArray(scanner);
        int m = input.get(0), N = input.get(1), X = input.get(2);


        List<Integer> returns = readIntArray(scanner);
        List<Integer> risks = readIntArray(scanner);
        List<Integer> maxInvestments = readIntArray(scanner);

        int maxReturn = 0;
        int[] bestInvestments = new int[m];

        for (int i = 0; i < m; i++) {
            if (risks.get(i) > X) continue;


            int investI = Math.min(N, maxInvestments.get(i));
            int retI = investI * returns.get(i);
            if (retI > maxReturn) {
                maxReturn = retI;
                Arrays.fill(bestInvestments, 0);
                bestInvestments[i] = investI;
            }


            for (int j = i + 1; j < m; j++) {
                if (risks.get(i) + risks.get(j) > X) continue;

                int investJ;
                if (returns.get(i) > returns.get(j)) {
```

```java
                    investI = Math.min(N, maxInvestments.get(i));
                    investJ = Math.min(N - investI, maxInvestments.get(j));
                } else {
                    investJ = Math.min(N, maxInvestments.get(j));
                    investI = Math.min(N - investJ, maxInvestments.get(i));
                }

                int retPair = investI * returns.get(i) + investJ * returns.get(j);
                if (retPair > maxReturn) {
                    maxReturn = retPair;
                    Arrays.fill(bestInvestments, 0);
                    bestInvestments[i] = investI;
                    bestInvestments[j] = investJ;
                }
            }
        }


        for (int investment : bestInvestments) {
            System.out.print(investment + " ");
        }
        System.out.println();
    }
}
```

Python

Plain Text

```
import sys

def read_int_array():
    """ 从标准输入读取一行并转换为整数列表 """
    return list(map(int, sys.stdin.readline().split()))

def main():

    m, N, X = read_int_array()
    returns = read_int_array()
    risks = read_int_array()
    max_investments = read_int_array()

    max_return = 0
    best_investments = [0] * m

    for i in range(m):
        if risks[i] > X:
            continue


        invest_i = min(N, max_investments[i])
        ret_i = invest_i * returns[i]
        if ret_i > max_return:
            max_return = ret_i
            best_investments = [0] * m
            best_investments[i] = invest_i


        for j in range(i + 1, m):
            if risks[i] + risks[j] > X:
                continue

            if returns[i] > returns[j]:
                invest_i = min(N, max_investments[i])
                invest_j = min(N - invest_i, max_investments[j])
            else:
                invest_j = min(N, max_investments[j])
                invest_i = min(N - invest_j, max_investments[i])

            ret_pair = invest_i * returns[i] + invest_j * returns[j]
            if ret_pair > max_return:
                max_return = ret_pair
                best_investments = [0] * m
                best_investments[i] = invest_i
```

```python
                best_investments[j] = invest_j

    print(" ".join(map(str, best_investments)))

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on('line', (line) => {
    inputLines.push(line.trim());
}).on('close', () => {
    const readIntArray = (index) => inputLines[index].split(" ").map(Number);


    const [m, N, X] = readIntArray(0);
    const returns = readIntArray(1);
    const risks = readIntArray(2);
    const maxInvestments = readIntArray(3);

    let maxReturn = 0;
    let bestInvestments = new Array(m).fill(0);

    for (let i = 0; i < m; i++) {
        if (risks[i] > X) continue;

        let investI = Math.min(N, maxInvestments[i]);
        let retI = investI * returns[i];
        if (retI > maxReturn) {
            maxReturn = retI;
            bestInvestments.fill(0);
            bestInvestments[i] = investI;
        }

        for (let j = i + 1; j < m; j++) {
            if (risks[i] + risks[j] > X) continue;

            let investI, investJ;
            if (returns[i] > returns[j]) {
                investI = Math.min(N, maxInvestments[i]);
                investJ = Math.min(N - investI, maxInvestments[j]);
            } else {
                investJ = Math.min(N, maxInvestments[j]);
                investI = Math.min(N - investJ, maxInvestments[i]);
            }

            let retPair = investI * returns[i] + investJ * returns[j];
            if (retPair > maxReturn) {
                maxReturn = retPair;
                bestInvestments.fill(0);
                bestInvestments[i] = investI;
                bestInvestments[j] = investJ;
            }
        }
    }

    console.log(bestInvestments.join(" "));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)


func readIntArray(scanner *bufio.Scanner) []int {
  scanner.Scan()
  fields := strings.Fields(scanner.Text())
  numbers := make([]int, len(fields))
  for i, field := range fields {
    numbers[i], _ = strconv.Atoi(field)
  }
  return numbers
}

func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)


  input := readIntArray(scanner)
  m, N, X := input[0], input[1], input[2]


  returns := readIntArray(scanner)
  risks := readIntArray(scanner)
  maxInvestments := readIntArray(scanner)

  maxReturn := 0
  bestInvestments := make([]int, m)


  for i := 0; i < m; i++ {
    if risks[i] > X {
      continue
    }


    investI := min(N, maxInvestments[i])
    retI := investI * returns[i]
    if retI > maxReturn {
      maxReturn = retI
      for k := range bestInvestments {
        bestInvestments[k] = 0
      }
      bestInvestments[i] = investI
    }


    for j := i + 1; j < m; j++ {
      if risks[i]+risks[j] > X {
        continue
      }

      var investI, investJ int

      if returns[i] > returns[j] {
        investI = min(N, maxInvestments[i])
        investJ = min(N-investI, maxInvestments[j])
      } else {
        investJ = min(N, maxInvestments[j])
        investI = min(N-investJ, maxInvestments[i])
      }

      retPair := investI*returns[i] + investJ*returns[j]
      if retPair > maxReturn {
        maxReturn = retPair
        for k := range bestInvestments {
          bestInvestments[k] = 0
        }
        bestInvestments[i] = investI
        bestInvestments[j] = investJ
      }
    }
  }


  for i, val := range bestInvestments {
    if i > 0 {
      fmt.Print(" ")
    }
```

```
        fmt.Print(val)
    }
    fmt.Println()
}
```

> 来自: 华为OD机试 2025 B卷 – 最佳投资方式 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD 机考 2025B卷 - 计算网络信号 / 信号强度 (C++ & Python & JAV

计算网络信号 / 信号强度

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> `华为OD机试2025B卷 100分题型`

题目描述

网络信号经过传递会逐层衰减，且遇到阻隔物无法直接穿透，在此情况下需要计算某个位置的网络信号值。

注意:网络信号可以绕过阻隔物。

- array[m][n] 的二维数组代表网格地图，
- array[i][j] = 0代表i行j列是空旷位置;
- array[i][j] = x(x为正整数)代表i行j列是信号源，信号强度是x;
- array[i][j] = -1代表i行j列是阻隔物。
- 信号源只有1个，阻隔物可能有0个或多个
- 网络信号衰减是上下左右相邻的网格衰减1

现要求输出对应位置的网络信号值

输入描述

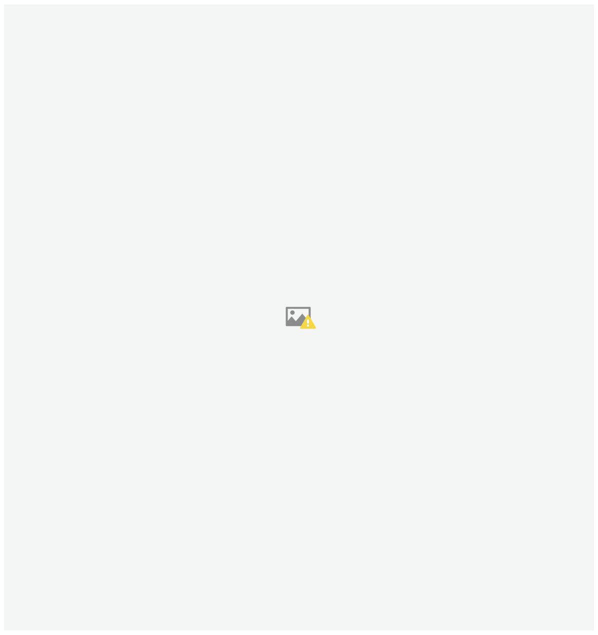
输入为三行，

- 第一行为 m 、n ，代表输入是一个 m × n 的数组。
- 第二行是一串 m × n 个用空格分隔的整数。每连续 n 个数代表一行，再往后 n 个代表下一行，以此类推。对应的值代表对应的网格是空旷位置，还是信号源，还是阻隔物。
- 第三行是 i 、 j，代表需要计算array[i][j]的网络信号值。

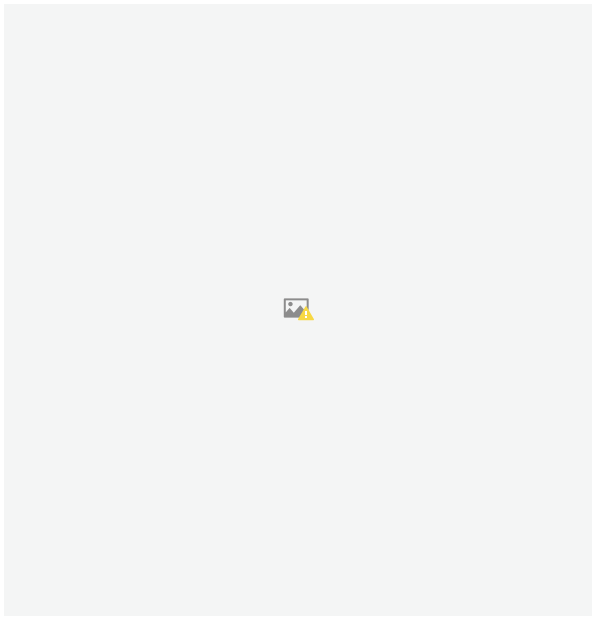
注意：此处 i 和 j 均从 0 开始，即第一行 i 为 0。

```
6 5
0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0
1 4
```

代表如下地图:



需要输出第1行第4列的网络信号值，值为2。



输出描述

输出对应位置的网络信号值，如果网络信号未覆盖到，也输出0。
一个网格如果可以途径不同的传播衰减路径传达，取较大的值作为其信号值。

示例1

输入

```Plain Text
6 5
0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0
1 4
```

输出

示例2

输入

```Plain Text
6 5
0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1
```

输出

题解

思路： `BFS`

1. 一道经典的多源 `BFS` 模板题。
2. 使用 `队列` 模拟进行 `BFS` 扩散。初始将所有值大于 `0` 的位置加入到队列中。接下来循环迭代将队列中的元素进行四周扩散，更新四周的信号值。具体逻辑参照下面代码
3. 输出指定位置的信号强度。

C++

Plain Text

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

int main() {
    int rows, cols;
    cin >> rows >> cols;

    vector<int> grid(rows * cols);
    queue<pair<int, int>> bfsQueue;


    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> grid[i * cols + j];
            if (grid[i * cols + j] > 0) {
                bfsQueue.emplace(i, j);
            }
        }
    }


    constexpr pair<int, int> directions[] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};


    while (!bfsQueue.empty()) {
        auto [x, y] = bfsQueue.front();
        bfsQueue.pop();

        int currentSignal = grid[x * cols + y];
        if (currentSignal == 1) continue;

        for (auto [dx, dy] : directions) {
            int newX = x + dx, newY = y + dy;
            if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && grid[newX * cols + newY] == 0) {
                grid[newX * cols + newY] = currentSignal - 1;
                bfsQueue.emplace(newX, newY);
            }
        }
    }

    int targetX, targetY;
    cin >> targetX >> targetY;
    cout << grid[targetX * cols + targetY] << endl;

    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int rows = scanner.nextInt();
        int cols = scanner.nextInt();

        int[] grid = new int[rows * cols];
        Queue<int[]> bfsQueue = new LinkedList<>();


        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int index = i * cols + j;
                grid[index] = scanner.nextInt();
                if (grid[index] > 0) {
                    bfsQueue.offer(new int[]{i, j});
                }
            }
        }


        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};


        while (!bfsQueue.isEmpty()) {
            int[] pos = bfsQueue.poll();
            int x = pos[0], y = pos[1];
            int currentSignal = grid[x * cols + y];

            if (currentSignal == 1) continue;

            for (int[] dir : directions) {
                int newX = x + dir[0], newY = y + dir[1];
                if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && grid[newX * cols + newY] == 0) {
                    grid[newX * cols + newY] = currentSignal - 1;
                    bfsQueue.offer(new int[]{newX, newY});
                }
            }
        }

        int targetX = scanner.nextInt();
        int targetY = scanner.nextInt();
        System.out.println(grid[targetX * cols + targetY]);
    }
}
```

Python

Plain Text

```
import sys
from collections import deque


rows, cols = map(int, sys.stdin.readline().split())


data = list(map(int, sys.stdin.readline().split()))
grid = data[:]
bfs_queue = deque()

for i in range(rows * cols):
    if grid[i] > 0:
        bfs_queue.append((i // cols, i % cols))


directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]


while bfs_queue:
    x, y = bfs_queue.popleft()
    current_signal = grid[x * cols + y]

    if current_signal == 1:
        continue

    for dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < rows and 0 <= new_y < cols and grid[new_x * cols + new_y] == 0:
            grid[new_x * cols + new_y] = current_signal - 1
            bfs_queue.append((new_x, new_y))


target_x, target_y = map(int, sys.stdin.readline().split())
print(grid[target_x * cols + target_y])
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on("line", (line) => {
    inputLines.push(line);
}).on("close", () => {
    let [rows, cols] = inputLines[0].split(" ").map(Number);
    let grid = inputLines[1].split(" ").map(Number);
    let bfsQueue = [];

    for (let i = 0; i < grid.length; i++) {
        if (grid[i] > 0) {
            bfsQueue.push([Math.floor(i / cols), i % cols]);
        }
    }


    const directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];


    while (bfsQueue.length) {
        let [x, y] = bfsQueue.shift();
        let currentSignal = grid[x * cols + y];

        if (currentSignal === 1) continue;

        for (let [dx, dy] of directions) {
            let newX = x + dx, newY = y + dy;
            if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && grid[newX * cols + newY] === 0) {
                grid[newX * cols + newY] = currentSignal - 1;
                bfsQueue.push([newX, newY]);
            }
        }
    }

    let [targetX, targetY] = inputLines[2].split(" ").map(Number);
    console.log(grid[targetX * cols + targetY]);
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  dimensions := strings.Fields(scanner.Text())
  rows, _ := strconv.Atoi(dimensions[0])
  cols, _ := strconv.Atoi(dimensions[1])

  scanner.Scan()
  data := strings.Fields(scanner.Text())
  grid := make([]int, rows*cols)
  var bfsQueue [][2]int

  for i := 0; i < rows*cols; i++ {
    grid[i], _ = strconv.Atoi(data[i])
    if grid[i] > 0 {
      bfsQueue = append(bfsQueue, [2]int{i / cols, i % cols})
    }
  }


  directions := [][2]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}


  for len(bfsQueue) > 0 {
    x, y := bfsQueue[0][0], bfsQueue[0][1]
    bfsQueue = bfsQueue[1:]

    if grid[x*cols+y] == 1 {
      continue
    }

    for _, d := range directions {
      newX, newY := x+d[0], y+d[1]
      if newX >= 0 && newX < rows && newY >= 0 && newY < cols && grid[newX*cols+newY] == 0 {
```

```
                grid[newX*cols+newY] = grid[x*cols+y] - 1
                bfsQueue = append(bfsQueue, [2]int{newX, newY})
            }
        }
    }

    scanner.Scan()
    target := strings.Fields(scanner.Text())
    targetX, _ := strconv.Atoi(target[0])
    targetY, _ := strconv.Atoi(target[1])
    fmt.Println(grid[targetX*cols+targetY])
}
```

> 来自: 华为OD 机考 2025B卷 – 计算网络信号 / 信号强度 (C++ & Python & JAVA & JS & GO)–CSDN博客