题目描述

有N个快递站点用字符串标识，某些站点之间有道路连接。
每个站点有一些包裹要运输，每个站点间的包裹不重复，路上有检查站会导致部分货物无法通行，计算哪些货物无法正常投递?

输入描述

- 第一行输入M N，M个包裹N个道路信息.
- 0<=M,N<=100,
- 检查站禁止通行的包裹如果有多个以空格分开

输出描述

- 输出不能送达的包裹，如：package2 package4,
- 如果所有包裹都可以送达则输出：none,
- 输出结果按照升序排列。

用例

输入	4 2 package1 A C package2 A C package3 B C package4 A C A B package1 A C package2
输出	package2
说明	无

题目解析

这题。。。。我能说我读都费劲嘛。。。

这题意思我只能猜一猜，因为我无法肯定自己的理解是对的。

用例第一行

- 输入的4代表有4个包裹，分别是package1、package2、package3、package4
- 输入的2代表：有2个：两站点之间无法通行的包裹，比如A B package1 代表A、B站点之间无法通行包裹package1。

用例第2~5行（对应第一行输入的4），对应4个包裹，以及它们要从哪个站点发往哪个站点。比如package1 A C，表示package1要从A发往C站点。

用例第6~7行（对应第二行输入的2），表示两站点之间无法通行的包裹。

因此，按照上面理解：

A-B无法通行package1，但是package1要从A发往C，因此不受影响。

A-C无法通行package2，而package2刚好要从A发往C，因此受到影响，无法发送。

因此最后，只有package2无法发送。

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

const lines = [];
let m, n;
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

const lines = [];
let m, n;
let want, cant;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [m, n] = lines[0].split(" ").map(Number);
  }

  if (m !== undefined && lines.length === m + 1) {
    want = lines.slice(1).map((line) => line.split(" "));
  }

  if (n !== undefined && lines.length === m + n + 1) {
    cant = lines.slice(m + 1).map((line) => line.split(" "));

    console.log(getResult(want, cant));
    lines.length = 0;
  }
});

/**
 * @param {*} want 二维数组，元素是 [包裹，起始站点，目的站点]
 * @param {*} cant 二维数组，元素是 [起始站点，目的站点，...无法通行的货物列表]
 */
function getResult(want, cant) {
  const wantObj = {};
  const cantObj = {};

  for (let arr of want) {
    const [package, src, dst] = arr;
    const path = `${src}-${dst}`;
    wantObj[path] ? wantObj[path].push(package) : (wantObj[path] = [package]);
  }

  for (let arr of cant) {
    const src = arr[0];
    const dst = arr[1];
    const path = `${src}-${dst}`;

    const packages = arr.slice(2);
    cantObj[path] ? cantObj[path].push(...packages) : (cantObj[path] = [...packages]);
  }

  let ans = [];
  for (let path in wantObj) {
    const wantPKG = new Set(wantObj[path]);
    const cantPKG = new Set(cantObj[path]);

    wantPKG.forEach((pkg) => {
      if (cantPKG.has(pkg)) {
        ans.push(pkg);
      }
    });
  }

  if (!ans.length) return "none";

  return ans.sort((a, b) => Number(a.slice(7)) - Number(b.slice(7))).join(" ");
}
```

Java算法源码

```java
import java.util.*;

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

    Integer[] tmp =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);
    int m = tmp[0];
    int n = tmp[1];

    String[][] want = new String[m][];
    for (int i = 0; i < m; i++) {
      want[i] = sc.nextLine().split(" ");
    }

    String[][] cant = new String[n][];
    for (int i = 0; i < n; i++) {
```

Java算法源码

```java
import java.util.*;

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

    Integer[] tmp =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);
    int m = tmp[0];
    int n = tmp[1];

    String[][] want = new String[m][];
    for (int i = 0; i < m; i++) {
      want[i] = sc.nextLine().split(" ");
    }

    String[][] cant = new String[n][];
    for (int i = 0; i < n; i++) {
      cant[i] = sc.nextLine().split(" ");
    }

    System.out.println(getResult(want, cant));
  }

  /**
   * @param want 二维数组，元素是 [包裹，起始站点，目的站点]
   * @param cant 二维数组，元素是 [起始站点，目的站点，...无法通行的包裹列表]
   * @return 无法通行的包裹
   */
  public static String getResult(String[][] want, String[][] cant) {
    HashMap<String, HashSet<String>> wantMap = new HashMap<>();
    HashMap<String, HashSet<String>> cantMap = new HashMap<>();

    for (String[] arr : want) {
      String pkg = arr[0];
      String path = arr[1] + "-" + arr[2];
      wantMap.putIfAbsent(path, new HashSet<>());
      wantMap.get(path).add(pkg);
    }

    for (String[] arr : cant) {
      String path = arr[0] + "-" + arr[1];
      String[] pkgs = Arrays.copyOfRange(arr, 2, arr.length);
      cantMap.putIfAbsent(path, new HashSet<>());
      cantMap.get(path).addAll(Arrays.asList(pkgs));
    }

    ArrayList<String> ans = new ArrayList<>();

    for (String path : wantMap.keySet()) {
      HashSet<String> wantPKG = wantMap.get(path);
      HashSet<String> cantPKG = cantMap.get(path);

      if (cantPKG == null) continue;

      for (String pkg : wantPKG) {
        if (cantPKG.contains(pkg)) {
          ans.add(pkg);
        }
      }
    }

    if (ans.size() == 0) return "none";

    ans.sort((a, b) -> Integer.parseInt(a.substring(7)) - Integer.parseInt(b.substring(7)));

    StringJoiner sj = new StringJoiner(" ");
    for (String an : ans) {
      sj.add(an);
    }
    return sj.toString();
  }
}
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())

want = []
for i in range(m):
    want.append(input().split())

cant = []
for i in range(n):
    cant.append(input().split())


# 算法入口
def getResult(want, cant):
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())

want = []
for i in range(m):
    want.append(input().split())

cant = []
for i in range(n):
    cant.append(input().split())


# 算法入口
def getResult(want, cant):
    """
    :param want: 二维数组，元素是 [包裹， 起始站点， 目的站点]
    :param cant: 二维数组，元素是 [起始站点， 目的站点，...无法通行的货物列表]
    :return: 不能送达的包裹
    """
    wantDict = {}
    cantDict = {}

    for arr in want:
        package, src, dst = arr
        path = f"{src}-{dst}"
        if wantDict.get(path) is None:
            wantDict[path] = [package]
        else:
            wantDict[path].append(package)

    for arr in cant:
        src = arr[0]
        dst = arr[1]
        path = f"{src}-{dst}"
        packages = arr[2:]
        if cantDict.get(path) is None:
            cantDict[path] = packages[::]
        else:
            cantDict[path].extend(packages)

    ans = []
    for path in wantDict.keys():
        if cantDict.get(path) is None:
            continue

        wantPKG = set(wantDict[path])
        cantPKG = set(cantDict[path])

        for pkg in wantPKG:
            if pkg in cantPKG:
                ans.append(pkg)

    if len(ans) > 0:
        ans.sort(key=lambda x: int(x[7:]))
        return " ".join(ans)
    else:
        return "none"


print(getResult(want, cant))
```