算法 OJ 题目与解析

2023 秋 lxy 班

Ning Li

School of Computer Science and Technology
(USTC)

2024 年 1 月 24 日

目录

前言

各题通过率如图 1，由于题目难度较大，推荐找软柿子捏，做完 4 题即可拿满分

Problems List

	#	Title	Total	AC Rate
✓	5-1	小花排队	369	28.73%
	5-2	合并	264	4.55%
✓	5-3	小花游泳	384	16.67%
✓	5-4	小花的鱼缸	427	18.03%
	5-5	小花的试卷堆	89	11.24%
✓	5-6	小花的积木	299	16.39%
	5-7	小花的西瓜树	43	16.28%
✓	5-8	小花的子串	419	9.07%

图 1: 各题通过率

note

本文为我学习 beamer 而做，答案仅供参考

目录

题目描述

小花与她的同学们在体育馆排成长队准备进行核酸检测。每名同学都有自己的绩点，对于任意两名同学（记为小红和小蓝），如果小红的绩点不低于小蓝，且小红小蓝之间没有绩点严格低于小蓝的同学，那么小红就会受到小蓝的羡慕。每名同学可以同时羡慕多名同学，也可以同时被多名同学羡慕。小花想知道每名同学分别是多少名同学的羡慕对象。

输入输出

第一行输入一个 N，表示 N 个同学
第二行输入 N 个绩点 G_i
$(1 \leq N \leq 10^5, 0 \leq G_i \leq 2^{31} - 1)$

测试样例:

Input	Output
5 1 2 3 4 5	0 1 2 3 4
10 1 5 3 2 6 7 2 9 0 4	1 5 4 3 4 5 3 4 0 1

解析

先考虑一边，使用**单调栈**[1]，栈顶是当前的最大值，记为 top，当前访问的值记为 cur

(1) 若 cur>top，则栈里的人都羡慕 cur，给 cur 加上羡慕值，入栈，访问下一个

(2) 若 cur<top，则弹出 top，再与新栈顶比较，回到（1）

另一边逆向来一遍即可。

(1) 如何加羡慕值？

利用“传递性”，若 a 羡慕 top，则 a 一定羡慕 cur，所以只需要加上 top 的羡慕值，再加一。

(2) 为什么能弹出？

因为若 cur<top，根据题意，top 不会再羡慕后面的人了

[1] 单调栈-OI Wiki

代码

Listing 1: 单调栈

```
1 for (int i = 0; i < grades.size(); i++)
2     while (!s.empty())
3         if (grades[s.top()] > grades[i])
4             s.pop();
5         else
6             res[i] += res[s.top()] + 1;
7             break;
8     s.push(i);
```

目录

题目描述

小花在游泳馆游泳，游泳馆中有许多游泳池，游泳池之间有一些单向连通的水管，每根水管的流量都有一定的上限。小花好奇：如果同时打开进水口和出水口，在保证水不溢出的条件下，她能够以多快的速度注水。假设进水口和出水口没有流量上限。

输入输出

第一行输入 4 个整数 N,M,S,T，分别表示池子个数，水管个数，进水池，出水池

第 $2 \sim M+1$ 行，每行三个整数 u,v,c，表示池子 u 到 v 有一个容量为 c 的水管

$1 \leq N \leq 100, 1 \leq M \leq 5000, 0 \leq c \leq 2^{31}-1$

测试样例:

Input	Output
7 14 1 7 1 2 5 1 3 6 1 4 5 2 3 2 2 5 3 3 2 2 3 4 3 3 5 3 3 6 7 4 6 5 5 6 1 6 5 1 5 7 8 6 7 7	14

解析

最大流问题，这里使用 Dinic 算法[2]（复杂度较优）

(1) 构造残存图

- ➢ 正向边表示可扩流量
- ➢ 反向边表示可反悔流量

(2) BFS+DFS 找增广路径

- ❑ BFS 用于节点分层，同时可判断是否存在增广路径
- ❑ DFS 找增广路径，每次只往下一层找

(3) 优化 DFS 过程

- ❍ 当前弧优化，dfs 过的弧已经被榨干，跳过
- ❍ 残枝优化，可扩充流量为 0 的节点已经被榨干，从残存图中删去

note

注意最大流应该用 long long，防溢出

[2]最大流-OI Wiki

代码：BFS

```
 1  bool bfs()
 2      memset(level, 0, sizeof(level));
 3      queue<int> q;
 4      q.push(S);
 5      level[S] = 1;
 6      while (!q.empty())
 7          int u = q.front();
 8          q.pop();
 9          for (int i = 0; i < adj[u].size(); i++)
10              int v = adj[u][i]->end;
11              if (level[v] == 0 && adj[u][i]->weight > 0)
12                  level[v] = level[u] + 1;
13                  q.push(v);
14                  if (v == T)
15                      return true;
16      return false;
```

DFS I

```
 1 LL dfs(int u, LL mf)
 2     if (u == T)
 3         return mf; // 剩余最大流量
 4     LL sum = 0; // 节点可扩充的流量
 5     for (int i = cur[u]; i < adj[u].size(); i++)
 6         cur[u] = i; // 当前弧优化
 7         int v = adj[u][i]->end;
 8         if (level[v] == level[u] + 1 && adj[u][i]->weight >
               0)
 9             LL flow = dfs(v, min(mf, adj[u][i]->weight));
10             adj[u][i]->weight -= flow;
11             adj[u][i]->rev->weight += flow;
12             sum += flow;
13             mf -= flow;
14             if (mf == 0)
15                 break;
16     if (sum == 0) // 残支优化
```

DFS II

```
17          level[u] = 0;
18      return sum;
```

主函数只需要循环调用 bfs 对原图分层，dfs 扩大流即可。

```
1  LL dinic()
2      LL maxflow = 0;
3      while (bfs())
4          memset(cur, 0, sizeof(cur));
5          maxflow += dfs(S, 1e9);
6      return maxflow;
```

目录

题目描述

小花的鱼缸里养了很多不同种类的水生动物，比如大鱼、小鱼、虾米、海绵宝宝、派大星等等，这些物种之间形成了一张食物网。吃与被吃这一关系具有传递性，比如已知大鱼吃小鱼、小鱼吃虾米，那么大鱼也一定可以吃虾米。小花想要知道，在这些物种之中，有哪些物种可以吃其他所有物种，又有哪些物种可以作为其他所有物种的食物。

输入输出

第一行输入两个整数 N,M，分别表示 N 个物种和 M 对相食关系

第 $2 \sim M+1$ 行，每行两个整数 x,y，表示 x 可以吃 y

$1 \leq N \leq 10^4, 0 \leq M \leq 5 \times 10^4$

测试样例:

Input	Output
3 3 1 2 2 1 2 3	2 1
6 4 1 3 3 5 2 4 4 6	0 0

解析

本题属于**强联通分量**[3]的应用，采用 Tarjan 算法求出强联通分量，找入度为 0 和出度为 0 的分量即可。如果有多个，则不存在。

Tarjan 算法：

- dfn[u]: 深度优先搜索遍历时结点 u 被搜索的次序
- low[u]: u 或 u 的子孙通过回边能到达的最小 dfn
- dfn=low 时弹出栈顶到 u 的元素作为一个强连通分量

为什么？

i. 入度为 0 的分量设为 a，因为图联通，对任意分量 b，a 到 b 一定有条路

ii. 假设有两个入度为 0 的连通分量 a 和 b，则 a 不能吃 b，且 b 也不能吃 a，所以不存在

[3]强联通分量-OI Wiki

代码 I

```
1  void tarjan(vector<vector<int>> &g, int u)
2      dfn[u] = low[u] = ++dfn_cnt;
3      s.push(u);
4      in_stack[u] = true;
5      for (int v : g[u])
6          if (!dfn[v])
7              tarjan(g, v);
8              low[u] = min(low[u], low[v]);
9          else if (in_stack[v])
10             low[u] = min(low[u], dfn[v]);
11     if (dfn[u] == low[u])
12         scc_cnt++;
13         while (true)
14             int x = s.top();
15             s.pop();
16             in_stack[x] = false;
17             scc[x] = scc_cnt;
```

代码 II

```
18              scc_size[scc_cnt]++;
19              if (x == u)
20                  break;
```

遍历原图，记录出入度.

```
1  for (int i = 1; i <= n; i++)
2      if (!dfn[i])
3          tarjan(g, i);
4  for (int u = 1; u <= n; u++)
5      for (int v : g[u])
6          if (scc[u] != scc[v])
7              in_degree[scc[v]]++;
8              out_degree[scc[u]]++;
```

目录

题目描述

初始时 N 个积木各成一堆，第 i 块积木的重量为 a_i，给定 M 次操作，格式如下：

(1) 合并：格式为 1 i j，表示将积木 i 所在的堆和积木 j 所在的堆合并

(2) 查询：格式为 2 i，表示查询积木 i 所在的堆的重量之积

(3) 移动：格式为 3 i j，表示将积木 i 从原来的堆移动到积木 j 所在的堆

输入输出

第一行输入两个整数 N,M，分别表示 N 堆积木和 M 次操作

第二行输入 N 个整数，为 $a_1, a_2, ...a_N$

接下来 M 行，每行一个操作，格式如题

$1 \leq N, M \leq 10^6, 1 \leq a_i \leq 10^4$

测试样例:

Input	Output
5 10 4 4 5 3 6 3 1 2 3 2 5 2 4 3 4 5 2 4 3 4 3 2 3 1 4 1 1 5 3 2 1	3 72 15 1440

解析

本题考察**并查集**[4]，要实现查询，合并，移动三个操作。查询和合并都很简单，直接找父亲即可。移动操作可以用副本实现（如参考链接所述），但写这道题时未充分查阅资料，就自己造了轮子（比较暴力）

(1) 如果要移动的不是根，先把孩子交给自己的父亲，再移动

(2) 否则，与自己孩子交换位置，然后同（1）

也可以通过虚拟根节点将两种情况合并。

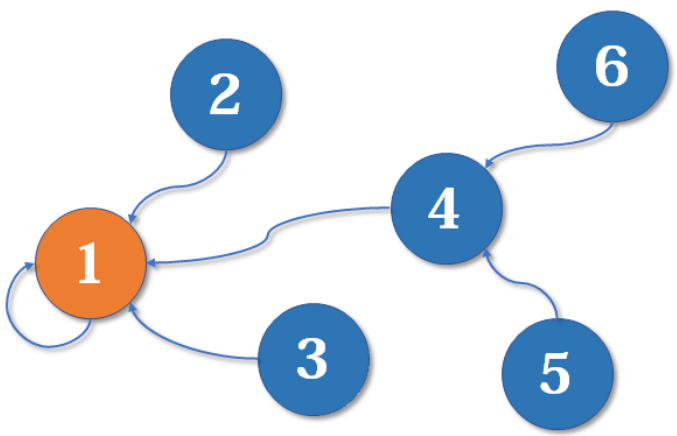

图 2: 并查集图示

[4]并查集-OI Wiki

代码 I

查询

```
1  int Query(int a)
2      int root_a = father[a];
3      while (root_a != father[root_a])
4          root_a = father[root_a];
5      return weight_product[root_a];
```

合并

```
1  void Merge(int a, int b)
2      int root_a = find(a);
3      int root_b = find(b);
4      if (root_a != root_b)
5          father[root_a] = root_b;
6          children[root_b].insert(root_a);
7          weight_product[root_b] = (long long)weight_product[
               root_b] * weight_product[root_a] % MOD;
```

代码 II

移动

```
1  void Move(int a, int b)
2      int root_a = find(a);
3      int root_b = find(b);
4      if (root_a == a && children[a].size() > 0)
5          int c = *children[a].begin();
6          father[c] = c;
7          father[a] = c;
8          children[a].erase(c);
9          children[c].insert(a);
10         for (auto child : children[a])
11             father[child] = c;
12             children[c].insert(child);
13         weight_product[c] = weight_product[root_a];
14         root_a = c;
15     for (auto child : children[a])
16         father[child] = father[a];
```

代码 III

```
17          children[father[a]].insert(child);
18      children[father[a]].erase(a);
19      weight_product[root_a] = (long long)weight_product[
            root_a] * inv(weight[a]) % MOD;
20      father[a] = root_b;
21      children[a].clear();
22      children[root_b].insert(a);
23      weight_product[root_b] = (long long)weight_product[
            root_b] * weight[a] % MOD;
```

目录

题目描述

小花有一个字符串 S，她发现其中包含很多重复的子串，她为每个子串评分，分数定义为：子串的长度 × 子串在字符串出现的次数。她想知道，分数最高的子串的分数是多少?

输入输出

第一行输入一个整数 N，表示字符串 S 的长度

第二行为字符串 S

$1 \leq N, M \leq 5000$

测试样例:

Input	Output
5 bbbba	6
18 cabcabcabcabcabcab	36

解析

本题使用**字符串哈希**[5]，为了减少冲突，需要采用双哈希，当两个哈希值都相等时认为字符串相同。哈希值通过递推计算，设 $dp[i][j]$ 表示 S[i:j] 的哈希值，则有递推式如下：

$$dp[i][j] = (dp[i][j-1] * p + dp[j][j])\%MOD$$

使用哈希表 (<unordered_map>) 记录所有可能的哈希值对作为 key，对应子串长度作为 value，可以做到 O(1) 查询。每查询到一次就加分，分值为子串长度。

hint

按长度找子串，clear 后再找下一个长度。可以很大程度上避免冲突，而且能节省空间。

[5]字符串哈希-OI Wiki

代码

```
for (int len = 1; len <= n; len++)
    for (int i = 0; i + len - 1 < n; i++)
        int j = i + len - 1;
        if (len == 1)
            dp2[i][j] = dp1[i][j] = s[i] - 'a' + 1;
        else
            dp1[i][j] = (dp1[i][j - 1] * 131 + dp1[j][j]) %
                1000000007;
            dp2[i][j] = (dp2[i][j - 1] * 131 + dp2[j][j]) %
                998244353;
        pair<int, int> val = make_pair(dp1[i][j], dp2[i][j
            ]);
        m[val] += len;
        maxScore = max(maxScore, m[val]);
    m.clear(); //避免冲突&节省空间
return maxScore;
```

Thanks!