

TGU ACM 2026 新生赛题解

本场出题人: fusun, Hailuo, Liang_SYEA, liyuff, shuiyin, sibycoin

技术支持: tiny_fish 的 Hydro OJ 和评测服务器

验题人: fusun, Hailuo, Liang_SYEA, liyuff, Luzheqiu, Mothfeir, shuiyin, sibycoin, tiny_fish, Yuanyin

感谢以上同学的贡献!

预估难度

Easy: A, B, F

Easy-Mid: C, D, G

Mid: I, K

Mid-Hard: E, H, J

Hard: L

A. An ACMer's Life from Zero

Idea by Hailuo

Editorial by Hailuo

思路

签到题, 按要求输出即可。时间复杂度 $O(1)$ 。

Code

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "I want to be an acmer!!" << endl;
7     return 0;
8 }
```

B. Bring Your Partner

Idea by Hailuo

Editorial by Hailuo

思路

按照题目要求, 有奶蛋就选奶蛋, 否则输出字典序最小的。

C++ 的 `string` 类重载了比较运算符，可以使用 `<` 来比较字符串的字典序大小。

注意第一次比较的时候，`res` 字符串是空的，特判一下即可。

时间复杂度 $O(n)$ 。

Code

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      ios::sync_with_stdio(false), cin.tie(0);
7
8      int n;
9      cin >> n;
10
11     string res;
12     bool flag = false;
13
14     for (int i = 1; i <= n; i++)
15     {
16         string s;
17         cin >> s;
18
19         if (s == "naidan")
20             res = s, flag = true;
21
22         if (!flag)
23         {
24             if (res.empty() || s < res)
25                 res = s;
26         }
27     }
28
29     cout << res << endl;
30     return 0;
31 }
```

C. Constructive Problem, Again

Idea by `Liang_SYEA`

Editorial by `Hailuo4ever`

思路

考虑构造一个从 1 到 n 的排列作为矩阵的第一行，为了保证行列上元素均不相同，可以每次对第一行循环移位，移位 n 次即可。

标程代码在实现上是循环左移的。时间复杂度 $O(n^2)$ 。

Code

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  void solve()
5  {
6      int n;
7      cin >> n;
8
9      vector<int> a(n);
10     iota(a.begin(), a.end(), 1);
11
12     for (int i = 0; i < n; i++)
13     {
14         for (int j = 0; j < n; j++)
15             cout << a[j] << " \n"[j == n - 1];
16         rotate(a.begin(), a.begin() + 1, a.end());
17     }
18 }
19
20 int main()
21 {
22     int T = 1;
23     // cin >> T;
24
25     while (T--)
26         solve();
27
28     return 0;
29 }
30
```

D. Do You Want to Start a Band?

Idea by shuiyin

Editorial by shuiyin

思路

考虑将所有成员的演奏值从小到大排序。对于此排序后相邻的两个演奏值，如果它们的差大于 k ，那么这两个值之间就形成了一个无法通过其他排序方式来解决的断点。

对于存在一个这样断点的情况，我们可以在这两个演奏值之间将队伍划分成两个部分来解决。

对于存在多个断点的情况，则无法解决。

时间复杂度 $O(n \log n)$ ，瓶颈在于排序。

Code

```
1  #include<bits/stdc++.h>
```

```

2
3 using namespace std;
4
5 int main() {
6     int n, k;
7     cin >> n >> k;
8     vector<int> a(n + 1);
9     for (int i = 1; i <= n; i++) {
10         int u;
11         cin >> u;
12         a[i] = u;
13     }
14     sort(a.begin() + 1, a.end());
15     int cnt = 0;
16     for (int i = 2; i <= n; i++) {
17         if (abs(a[i] - a[i - 1]) > k) cnt++;
18     }
19
20     if (cnt <= 1) cout << "YES" << '\n';
21     else cout << "NO" << '\n';
22
23     return 0;
24 }

```

E. Evening of the Starlight Festival

Idea by [1iyuff](#)

Editorial by [1iyuff](#)

思路

每个星灯有两个参数，一个是维修时间，一个是截止时间。

首先我们可能会想到一个假的做法，就是按照维修时间长短排序，每次修所需时间最小的，但是由于截止时间的影响，我们并不能在截止时间后修星灯，所以这种贪心是错的。

但是这个错误的做法给了我们一点启发：一个灯只能在截止时间前维修，也就是我们应该**从截止时间方面去思考**，所以我们想到：如果按照截止时间排序，从小到大枚举所有截止时间，记当前枚举的截止时间为 d ，那么就可以对于每个截止时间把所有灯分为两部分：截止时间 $\leq d$ ，和 $> d$ 。

接着考虑，所有截止时间 $\leq d$ 的星灯，它们都在可选择范围内，我们可以贪心选择维修时间最小的那几个，把时间长的淘汰掉。

而这个过程可以通过大根堆来模拟，如果所选星灯维修时间总和 $sum > d$ ，那么就淘汰维修时间最长的那个。

时间复杂度 $O(n \log n)$ 。

Code

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4

```

```

5  int main() {
6      int n;
7      cin >> n;
8
9      vector<pair<int, int>> a(n + 1);
10     for (int i = 1; i <= n; i++) {
11         int t, d;
12         cin >> t >> d;
13         a[i] = {d, t};
14     }
15     sort(a.begin() + 1, a.end());
16
17     priority_queue<int> q;
18     ll sum = 0;
19     for (int i = 1; i <= n; i++) {
20         auto [d, t] = a[i];
21
22         sum += t;
23         q.push(t);
24
25         if(sum > d) {
26             sum -= q.top();
27             q.pop();
28         }
29     }
30
31     cout << q.size() << endl;
32 }

```

F. For Stalin, Let's Sort!

Idea by [Hailuo](#)

Editorial by [Hailuo](#)

思路

直接模拟即可，注意 pre 的初值要设置成负无穷大。时间复杂度 $O(n)$ 。

Code

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int inf = 0x3f3f3f3f;
4
5  void solve()
6  {
7      int n;
8      cin >> n;
9
10     vector<int> a(n + 1);
11     for (int i = 1; i <= n; i++)

```

```

12     cin >> a[i];
13
14     int pre = -inf;
15     for (int i = 1; i <= n; i++)
16     {
17         if (a[i] >= pre)
18             cout << a[i] << ' ', pre = a[i];
19     }
20     cout << '\n';
21 }
22
23 int main()
24 {
25     ios::sync_with_stdio(false), cin.tie(0);
26
27     int T;
28     cin >> T;
29
30     while (T--)
31         solve();
32
33     return 0;
34 }

```

G. Grand Finale of the Witch Trial

Idea by [shuiyin](#)

Editorial by [shuiyin](#)

思路

本题可以通过 找规律 或 记忆化搜索暴力 通过此题

前倾提要: *P态*: 当前状态进行操作的玩家必输

N态: 当前状态进行操作的玩家必胜

对于未确定的状态, 若存在可以通过一步到达 *P态* 的方式, 则当前为 *N态*, 否则为 *P态*。

换句话说, 如果当前状态可以通过一步操作让对手陷入必输的状态, 则当前状态操作的玩家必胜

找规律: 状态 (0, 0) 时操作的玩家必败, 为 *P态*

状态 (1, 0), 状态 (0, 1) 和 状态 (1, 1) 可以一次操作到达 *P态* 的 (0, 0) 因此为 *N态*

状态 (2, 0) 只能到达 *N态*, 所以为 *P态*。状态 (2, 1) 可以一次操作到 *P态* (2, 0) 为 *N态*

同理 (2, 2) 为 *P态*。

可以看出 当 *a* 和 *b* 都为偶数时 为 *P态*, 其余状态为 *N态*

下可以通过递推DP打表证明, 也可以数学归纳证明。

Code

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  void solve(){
6      int a,b;
7      cin>> a >> b;
8      if(!(a & 1) && !(b & 1)){
9          cout<< "mingchaonb" <<'\n';
10     }else{
11         cout<< "yuanshennb" <<'\n';
12     }
13 }
14
15 int main(){
16     int t;
17     cin>> t;
18     while(t--){
19         solve();
20     }
21     return 0;
22 }
```

递推DP: $O(n^2)$ 暴力可解

```
1  void solve() {
2      int a, b;
3      cin >> a >> b;
4
5      vector<vector<int>> dp(a + 1, vector<int>(b + 1));
6      dp[0][0] = 0; //0表示先手必输, 1表示先手必赢
7
8      for (int i = 0; i <= a; i++) {
9          for (int j = 0; j <= b; j++) {
10             if (i == 0 && j == 0) {
11                 continue;
12             }
13
14             if (i > 0 && !dp[i - 1][j]) dp[i][j] = 1;
15             if (j > 0 && !dp[i][j - 1]) dp[i][j] = 1;
16             if (i > 0 && j > 0 && !dp[i - 1][j - 1]) dp[i][j] = 1;
17         }
18     }
19     if (dp[a][b]) {
20         cout << "yuanshennb" << '\n';
21     } else {
22         cout << "mingchaonb" << '\n';
23     }
24 }
```

H. How Hard Can Math Be?

Idea by `fusun_2`

Editorial by `fusun_2`

思路

设 $g = \gcd(a, b)$, 则 $a = gx$, $b = gy$, 且 $\gcd(x, y) = 1$ 。

因为 $a + b = n$, 所以 $g(x + y) = n$ 。

因此 g 一定是 n 的因子。又因为 $a, b > 0$, 所以 $x + y \geq 2$ 。于是 $g = n/(x + y) \leq n/2$ 。

也就是说, g 必须是 n 的真因子, 要让 g 最大, 就要取 n 的最大真因子。而一个数的最大真因子等于 n 除以最小质因子。

所以我们构造 $a = n/p$, $b = n - n/p$ 。此时 $\gcd(a, b) = \gcd(n/p, n - n/p) = n/p$, 达到最大。

综上, 本题简化为找到 n 的最小质因子 p , 输出 n/p 和 $n - n/p$ 即可。时间复杂度 $O(\sqrt{n})$ 。

Code

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  void solve() {
5      int n; cin >> n;
6      int p = n;
7      for (int i = 2; i * i <= n; i++) {
8          if (n % i == 0) {
9              p = i;
10             break;
11         }
12     }
13
14     int a = n / p, b = n - a;
15     cout << a << ' ' << b << '\n';
16 }
17
18 int main() {
19     int T; cin >> T;
20     while (T--) {
21         solve();
22     }
23     return 0;
24 }
```

I. It's Time for hrz to Sleep

idea by `Siby`

Editorial by `Siby`

思路

本题有多种做法，这里给出一种使用 `map` 实现的做法。

可以把 $a_i - a_j = x$ 看作 $a_i - x = a_j$ ，考虑到数据范围为 10^9 ，将每个数字出现的次数使用 `map` 统计，将所有 $a_i - x$ 在 `map` 中出现次数相加即可获得答案。

Code

```
1  #include<iostream>
2  #include<algorithm>
3  #include<cstring>
4  #include<vector>
5  #include<cmath>
6  #include<map>
7  #define int long long
8  #define IOS ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
9  using namespace std;
10 int T;
11 int n;
12 int x;
13 int a[100010];
14 map<int,int>m;
15 signed main()
16 {
17     IOS;
18     cin>>T;
19     while(T--){
20         cin>>n>>x;
21         for(int i=1;i<=n;i++){
22             cin>>a[i];
23             m[a[i]]++;
24         }
25         int ans=0;
26         for(int i=1;i<=n;i++){
27             ans+=m[a[i]-x];
28         }
29         cout<<ans<<'\n';
30         for(int i=1;i<=n;i++){
31             m[a[i]]=0;
32         }
33     }
34     return 0;
35 }
36
```

J. Journey of Liyu Beneath the Sea

Idea by `liyuff`

Editorial by `liyuff`

思路

注意到如果固定查询区间左端点为 1，随着 r 往右移动，查询到的结果是单调不减的，具备单调性，所以可以二分出第一个答案位置，同理可得第二个位置。

Code

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main() {
5      int n;
6      cin >> n;
7
8      auto ask = [&](int r) -> int { // lambda表达式，建议学一下
9          cout << "? " << 1 << " " << r << endl;
10         int res;
11         cin >> res;
12         if(res == -1) exit(0);
13         return res;
14     };
15
16     int l = 0, r = n + 1;
17     while(l + 1 < r) {
18         int mid = (l + r) >> 1;
19         if(ask(mid) >= 1) r = mid;
20         else l = mid;
21     }
22     int x = r;
23
24     l = x, r = n + 1;
25     while(l + 1 < r) {
26         int mid = (l + r) >> 1;
27         if(ask(mid) >= 2) r = mid;
28         else l = mid;
29     }
30     int y = r;
31
32     cout << "! " << x << " " << y << endl;
33 }
```

K. Keep RiNG in Order

Idea by Hailuo

Editorial by Hailuo

思路

把成员看作点，要求看作从 u 指向 v 的有向边，题目就变成寻找一个顺序，使每条边的起点都排在终点之前。

我们统计每个点的**入度**，即当前这个点被多少条边指向。显然入度为 0 的点没有约束，要立即安排出场。先把所有入度为零的点放入队列。每次从队列中取出一个点，加入答案，并删去它指向的所有边：将对应终点的入度减 1，如果终点的入度变成 0，就将其入队。这个过程叫拓扑排序。

考虑结束时的情况，如果答案中的点数少于 n 个，说明剩余的点的入度全部 > 0 ，图中一定存在有向环，所以此时无解。

时间复杂度 $O(n + m)$ 。

Code

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  void solve()
5  {
6      int n, m;
7      cin >> n >> m;
8
9      vector<vector<int>> g(n + 1);
10     vector<int> din(n + 1);
11
12     for (int i = 1, u, v; i <= m; i++)
13         cin >> u >> v, g[u].push_back(v), din[v]++;
14
15     queue<int> q;
16     for (int i = 1; i <= n; i++)
17         if (din[i] == 0)
18             q.push(i);
19
20     vector<int> res;
21     while (!q.empty())
22     {
23         int u = q.front();
24         q.pop();
25
26         res.push_back(u);
27
28         for (auto v: g[u])
29         {
30             if (--din[v] == 0)
31                 q.push(v);
32         }
33     }
34
35     if (res.size() != n)
36         cout << -1 << endl;
37     else
38         for (auto x: res)
39             cout << x << ' ';
40 }
41
42 int main()
```

```

43 {
44     ios::sync_with_stdio(false), cin.tie(0);
45
46     int T = 1;
47     // cin >> T;
48
49     while (T--)
50         solve();
51
52     return 0;
53 }
54

```

L. Look, It's Yesterday Once More

Idea by [Liang_SYEA](#)

Editorial by [Liang_SYEA](#)

思路

首先思考第一个问题：如何求解最短花费时间。

考虑对于一个阶段 u ，存在两种到达工作阶段 u 的方案，到达 u 的时间分别记为 t_1 和 t_2 ，且 $t_1 \leq t_2$ 。

那么第一种方案可以在原地等到 $t_2 - t_1$ 单位时间，然后做出与第二个方案完全相同的后续操作。

因此较早到达一定不劣于较晚到达的方案，故只需要记录每个点的最早到达时间即可。

此时考虑第二个问题：如何判定是否存在一种最优路径经过指定的边。

一个很容易想到但明显错误的做法是：对于边 (u, v, w) ， $dis_u + \Delta + w = dis_v$ ，其中 Δ 表示在当前阶段的等待时间。

因为在某个最快方案可能以一个严格大于 dis_v 的时间到达 v ，虽然当前方案没有尽早到达 v ，但更早到达 v 的方案可能还需要等待，因此两者仍然可能都在最优时间到达 n 。

因此我们还需要知道最晚到达什么时间到达 v ，依然能以最优时间到达 n 。

记 lim_u 表示能够最终以最优时间到达 n 的前提下，可以最晚到达 u 的时刻。

在反图上做一次与第一轮过程对称的 *Dijkstra* 即可求得 lim 。

对一条边 (u, v, w) ，存在一条最短路经过它，当且仅当 $dis_u + \Delta + w \leq lim_v$ 。

时间复杂度 $O((n + m) \log m)$ 。

Code

```

1  /*
2   吾日三省吾身
3   “怎么这都不会做”
4   “怎么做这么慢”
5   “写了个什么勾巴”
6  */
7  #include <bits/stdc++.h>

```

```

8
9 using i64 = long long;
10 using u64 = unsigned long long;
11 using u32 = unsigned int;
12
13 using i128 = __int128;
14 using u128 = unsigned __int128;
15
16 void solve() {
17     int n, m;
18     std::cin >> n >> m;
19
20     std::vector<std::vector<std::array<int, 2>>> adj(n);
21     std::vector<std::array<int, 3>> edges(m);
22     for (int i = 0; i < m; i++) {
23         int u, v, w;
24         std::cin >> u >> v >> w;
25         u--;
26         v--;
27         adj[u].push_back({v, w});
28         edges[i] = {u, v, w};
29     }
30
31     std::vector<std::vector<std::array<int, 3>>> nadj(n);
32     for (int u = 0; u < n; u++) {
33         for (int j = 0; j < adj[u].size(); j++) {
34             auto [v, w] = adj[u][j];
35             nadj[v].push_back({u, w, j});
36         }
37     }
38
39     constexpr i64 inf = 1e18;
40     std::vector<i64> dis(n, inf);
41     dis[0] = 0;
42
43     std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>,
std::greater<std::pair<i64, int>>> q;
44     q.emplace(0, 0);
45
46     auto f = [](int a, int b, int l) -> int {
47         int ans = b - a;
48         if (ans < 0) {
49             ans += l;
50         }
51         return ans;
52     };
53     while (!q.empty()) {
54         auto [d, u] = q.top();
55         q.pop();
56
57         if (d > dis[u]) {
58             continue;
59         }

```

```

60
61     for (int j = 0; j < adj[u].size(); j++) {
62         auto [v, w] = adj[u][j];
63         w += f(d % adj[u].size(), j, adj[u].size());
64         if (dis[v] > dis[u] + w) {
65             dis[v] = dis[u] + w;
66             q.emplace(dis[v], v);
67         }
68     }
69 }
70
71 std::vector<i64> lim(n);
72 lim[n - 1] = dis[n - 1];
73
74 std::priority_queue<std::pair<i64, int>>> que;
75 que.emplace(dis[n - 1], n - 1);
76
77 while (!que.empty()) {
78     auto [d, u] = que.top();
79     que.pop();
80
81     if (d < lim[u]) {
82         continue;
83     }
84
85     for (auto [v, w, j] : nadj[u]) {
86         w += f(j, (d - w) % adj[v].size(), adj[v].size());
87         if (lim[v] < lim[u] - w) {
88             lim[v] = lim[u] - w;
89             que.emplace(lim[v], v);
90         }
91     }
92 }
93
94 std::vector<int> cnt(n);
95 std::string s;
96 for (auto [u, v, w] : edges) {
97     s += dis[u] + w + f(dis[u] % adj[u].size(), cnt[u]++, adj[u].size()) <= lim[v]
? '1' : '0';
98 }
99
100 std::cout << dis[n - 1] << '\n';
101 std::cout << s << '\n';
102 }
103
104 int main() {
105     std::ios::sync_with_stdio(false);
106     std::cin.tie(nullptr);
107
108     int t = 1;
109     // std::cin >> t;
110
111     while (t--) {

```

```
112     solve();
113 }
114
115 return 0;
116 }
```