

HƯỚNG DẪN CHẤM THI

Môn thi chuyên: **TIN HỌC**

Thời gian làm bài: **150 phút** (không kể thời gian phát đề)

Ngày làm bài thi: **19/4/2026**

1 hướng dẫn chấm thi gồm 07 trang, 03 bài

Tổng quan đề thi

	Tên bài	File chương trình	File dữ liệu	File kết quả	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	La Grande Finale	LGF.*	LGF.INP	LGF.OUT	1 giây	1024 MB
Bài 2	May áo	MAYAO.*	MAYAO.INP	MAYAO.OUT	1 giây	1024 MB
Bài 3	Tổng số phép nhớ	TONGNHO.*	TONGNHO.INP	TONGNHO.OUT	1 giây	1024 MB

I. Hướng dẫn chung

- Bài thi của thí sinh được chấm thi bằng phần mềm chấm thi trực tuyến (online judge) trên website <https://oj.giftedbat.edu.vn/>, sử dụng bộ test của Tổ chức The Gifted Battlefield – Ban Tin học, đúng với đáp án và biểu điểm được nêu trong hướng dẫn chấm thi.
- Điểm bài thi được xuất từ phần mềm chấm thi; không quy tròn điểm thành phần của từng câu và điểm của bài thi.

II. Đáp án

Bài 1. La Grande Finale (4,0 điểm)

Tổng quan

- Subtask 1: 50% số test có $N, Q \leq 2000$.
- Subtask 2: 50% số test còn lại có $N, Q \leq 2 \times 10^5$.

Subtask 1

Mô phỏng trực tiếp quá trình nộp bài theo cách đơn giản nhất. Ta sử dụng hai mảng song song để lưu lại toàn bộ lịch sử Q lần nộp bài (một mảng lưu chỉ số bài toán, một mảng lưu trạng thái nộp bài).

- Đọc Q truy vấn và lưu tuần tự vào mảng lịch sử chung.
- Duyệt qua từng bài toán i từ 1 đến N . Với mỗi bài, ta duyệt qua **toàn bộ** Q phần tử của mảng lịch sử.
- Nếu gặp trạng thái **wa** của bài i , ta tăng biến đếm số lần sai. Nếu gặp trạng thái **ac** của bài i , ta đánh dấu bài đã được giải và cập nhật penalty hiện tại bằng đúng số lần **wa** đã đếm được tính đến thời điểm đó.

Độ phức tạp thời gian là $\mathcal{O}(N \times Q)$. Với $N, Q \leq 2000$, số phép toán tối đa khoảng 4×10^6 , chương trình hoàn toàn có thể chạy tốt trong giới hạn thời gian. Độ phức tạp không gian là $\mathcal{O}(Q)$.

Cài đặt (tham khảo)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAXQ = 200005;
5
6 int history_u[MAXQ];
7 string history_s[MAXQ];
8
9 int main() {
10     ios_base::sync_with_stdio(false);
11     cin.tie(NULL); cout.tie(NULL);
12     freopen("LGF.INP", "r", stdin);
13     freopen("LGF.OUT", "w", stdout);
14
15     int n, q;
16     cin >> n >> q;
17
18     for (int i = 0; i < q; i++) {
19         cin >> history_u[i] >> history_s[i];
20     }
21
22     int ac = 0;
23     int pen = 0;
24
25     // Duyệt qua từng bài toán từ 1 đến n
26     for (int i = 1; i <= n; ++i) {
27         int wa_count = 0;
28         int current_pen = 0;
29         bool solved = false;
30
31         // Quét lại toàn bộ lịch sử cho từng bài
32         for (int j = 0; j < q; ++j) {
33             if (history_u[j] == i) {
34                 if (history_s[j] == "wa") {
35                     wa_count++;
36                 } else if (history_s[j] == "ac") {
37                     solved = true;
38                     // Cập nhật penalty đến thời điểm ac cuối cùng
39                     current_pen = wa_count;
40                 }
41             }
42         }
43
44         if (solved) {
45             ac++;
46             pen += current_pen;
47         }
48     }
49
50     cout << ac << " " << pen;
51     return 0;
52 }
```

Subtask 2

Nhận xét rằng ta không cần phải lưu lại toàn bộ lịch sử nộp bài của hệ thống. Ta có thể duyệt qua Q lần nộp bài một lần duy nhất và duy trì các trạng thái hiện tại của từng bài toán.

Sử dụng hai mảng (hoặc vector):

- Mảng `temp[u]`: Lưu tổng số lần nộp `wa` của bài u tính đến thời điểm hiện tại.
- Mảng `pen[u]`: Lưu số penalty của bài u . Khởi tạo `pen[u] = -1` để đánh dấu bài chưa được giải (ac).

Duyệt qua lần lượt Q lần nộp bài (u, s) :

- Nếu $s = wa$: Ta tăng $temp[u]$ lên 1.
- Nếu $s = ac$: Ta cập nhật $pen[u] = temp[u]$. Do bài toán yêu cầu tính penalty đến lần ac **cuối cùng**, việc ta liên tục ghi đè $pen[u]$ bằng $temp[u]$ mỗi khi gặp một chữ ac sẽ đảm bảo: giá trị cuối cùng đọng lại trong $pen[u]$ chính là số wa xuất hiện trước chữ ac cuối cùng.

Sau khi xử lý xong Q truy vấn, ta chỉ cần duyệt qua N bài toán. Những bài có $pen[u] > -1$ là những bài được giải, ta cộng $pen[u]$ vào tổng penalty.

Độ phức tạp thời gian: $\mathcal{O}(N + Q)$. Độ phức tạp không gian: $\mathcal{O}(N)$. Thuật toán này vượt qua giới hạn 2×10^5 một cách dễ dàng.

Cài đặt (tham khảo)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     ios_base::sync_with_stdio(false);
6     cin.tie(NULL); cout.tie(NULL);
7     freopen("LGF.INP", "r", stdin);
8     freopen("LGF.OUT", "w", stdout);
9
10    int n, q;
11    cin >> n >> q;
12
13    int ac = 0, cnt = 0;
14
15    // pen khởi tạo -1 (pen[i] là số lần wa trước ac cuối), wa khởi tạo 0 (wa[i] là số lần wa của bài i)
16    vector<int> pen(n + 1, -1), wa(n + 1, 0);
17
18    while (q--) {
19        int i; cin >> i;
20        string s; cin >> s;
21        if (s == "ac") {
22            pen[i] = wa[i];
23        } else {
24            wa[i]++;
25        }
26    }
27
28    // Tổng hợp kết quả
29    for (int i = 1; i <= n; i++) {
30        if (pen[i] > -1) {
31            ac++;
32            cnt += pen[i];
33        }
34    }
35
36    cout << ac << ' ' << cnt;
37
38    return 0;
39 }

```

Bài 2. May áo (3,0 điểm)

Tổng quan

Ràng buộc cho mọi subtask: $-10^9 \leq a_i, b_i \leq 10^9$.

- Có 20% số test tương ứng với 20% số điểm có $1 \leq n \leq 20$.
- Có 30% số test tương ứng với 30% số điểm có $1 \leq n \leq 1000$.

- Có 50% số test tương ứng với 50% số điểm có $1 \leq n \leq 2 \times 10^5$.

Do b_i có thể âm, nên trong quá trình chọn ta sẽ không bao giờ muốn lấy các đơn hàng có $b_i < 0$ vì chúng làm giảm tổng lợi nhuận. Do đó, ta có thể loại bỏ trước tất cả các đơn hàng có $b_i < 0$. Tuy nhiên, đề bài yêu cầu phải chọn ít nhất một đơn hàng, nên trong trường hợp tất cả các b_i đều âm, ta sẽ chọn đơn hàng có b_i lớn nhất (ít âm nhất).

Subtask 1:

Vì $n \leq 20$, ta có thể duyệt toàn bộ các tập con S của các đơn hàng (tối đa 2^n cách). Với mỗi tập S , ta tính trực tiếp lợi nhuận bằng cách lấy tổng các b_i trong S , sau đó trừ đi $\max(a_i) - \min(a_i)$ với $i \in S$. Có thể dùng bitmask để biểu diễn các tập con, đảm bảo không bỏ sót trường hợp nào và lấy giá trị lớn nhất.

Subtask 2:

Nhận xét rằng nếu đã chọn giá trị nhỏ nhất và lớn nhất của a_i trong tập S , thì để tối ưu ta nên chọn tất cả các đơn hàng có kích cỡ nằm trong khoảng đó. Do đó, ta sắp xếp các đơn hàng theo a_i , rồi xét tất cả các đoạn con $[l, r]$. Với mỗi đoạn, lợi nhuận là tổng b từ l đến r trừ đi $a[r] - a[l]$. Ta có thể dùng prefix sum để tính nhanh tổng b , từ đó duyệt tất cả các đoạn và lấy giá trị lớn nhất.

Subtask 3:

Ta tiếp tục dựa trên mảng đã sắp xếp theo a_i và prefix sum của b . Cố định i là vị trí của phần tử có a_i lớn nhất, ta cần tìm $j \leq i$ sao cho giá trị $\sum_{k=j}^i b_k - (a_i - a_j)$ là lớn nhất. Biến đổi lại, ta được $a_i + \text{pref}[i] + \max_{j \leq i} (-a_j - \text{pref}[j - 1])$. Do đó, khi duyệt từ trái sang phải, ta duy trì một biến $mx = \max(-a_j - \text{pref}[j - 1])$, cập nhật dần theo j , và với mỗi i ta tính giá trị tốt nhất bằng công thức trên rồi lấy max toàn cục.

Cài đặt (tham khảo)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const long long INF = (long long)4e18;
5
6 struct Z {
7     long long a, b;
8 };
9
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(nullptr);
13     freopen("MAYAO.INP", "r", stdin);
14     freopen("MAYAO.OUT", "w", stdout);
15
16     int n;
17     cin >> n;
18
19     vector<Z> v(n);
20     for (int i = 0; i < n; i++) cin >> v[i].a >> v[i].b;
21
22     sort(v.begin(), v.end(), [](Z x, Z y) {
23         if (x.a != y.a) return x.a < y.a;
24         return x.b < y.b;
25     });
26
27     vector<long long> x, s, e;
28
29     for (int i = 0; i < n; ) {
30         int j = i;
31         long long a = v[i].a;
32         long long sumPos = 0;
```

```

33     long long mx = -INF;
34
35     while (j < n && v[j].a == a) {
36         if (v[j].b > 0) sumPos += v[j].b;
37         mx = max(mx, v[j].b);
38         j++;
39     }
40
41     x.push_back(a);
42     s.push_back(sumPos);
43     e.push_back(sumPos > 0 ? sumPos : mx);
44
45     i = j;
46 }
47
48 int k = x.size();
49 vector<long long> pref(k);
50
51 pref[0] = s[0];
52 for (int i = 1; i < k; i++) pref[i] = pref[i - 1] + s[i];
53
54 long long ans = -INF;
55
56 for (int i = 0; i < k; i++) ans = max(ans, e[i]);
57
58 long long bst = e[0] - pref[0] + x[0];
59
60 for (int r = 1; r < k; r++) {
61     long long cur = bst + e[r] + pref[r - 1] - x[r];
62     ans = max(ans, cur);
63     bst = max(bst, e[r] - pref[r] + x[r]);
64 }
65
66 cout << ans << '\n';
67 return 0;
68 }

```

Bài 3. Tổng số phép nhớ (3,0 điểm)

Tổng quan

Ràng buộc cho mọi subtask: $M \times K \leq N \leq 10^5$; $M \leq 100$; $a_i \leq 10^9$.

- Có 20% số test tương ứng với 20% số điểm có $N \leq 1000$; $M \leq 2$; $K \leq 3$.
- Có 20% số test tương ứng với 20% số điểm có $M \leq 2$; $K \leq 3$.
- Có 20% số test tương ứng với 20% số điểm có $N \leq 1000$; $M \leq 2$.
- Có 20% số test tương ứng với 20% số điểm có $M \leq 2$.
- Có 20% số test tương ứng với 20% số điểm không có ràng buộc gì thêm.

Subtask 1 (N nhỏ, $K \leq 3$)

Với giới hạn $a_i \leq 10^9$, việc tính số phép nhớ khi cộng hai số có thể xem như thực hiện trong $O(1)$. Vì K rất nhỏ ($K \leq 3$), ta có thể thử toàn bộ $K!$ thứ tự cộng các số trong đoạn $[i, i + K - 1]$ để xác định giá trị $cost[i]$ nhỏ nhất.

- Với $M = 1$: Lấy giá trị $cost[i]$ nhỏ nhất hợp lệ trên toàn bộ dãy.
- Với $M = 2$: Duyệt qua tất cả các cặp chỉ số (i, j) thỏa mãn điều kiện không ghi đè $i + K \leq j$ và tìm $\min(cost[i] + cost[j])$.

Độ phức tạp thời gian: $O(N \cdot K! + N^2)$.

Subtask 2 (Tối ưu việc tìm cặp)

Để tối ưu hóa việc tìm cặp (i, j) từ $O(N^2)$ xuống $O(N)$, ta sử dụng kỹ thuật duy trì giá trị nhỏ nhất trên tiền tố. Khi duyệt chỉ số j tăng dần, ta luôn cập nhật và giữ lại giá trị $cost[i]$ nhỏ nhất sao cho $i + K \leq j$, từ đó tính toán đáp án tối ưu một cách trực tiếp. Độ phức tạp thời gian: $O(N \cdot K! + N)$.

Subtask 3 (Tính toán trên từng cột chữ số)

Nhận xét quan trọng: Thứ tự thực hiện phép cộng các số trong cùng một đoạn không ảnh hưởng đến tổng số phép nhớ cuối cùng.

Thay vì cộng dồn từng cặp, ta xét tổng các chữ số trên từng cột d (từ hàng đơn vị trở lên). Gọi $sum[d]$ là tổng các chữ số ở cột d của tất cả các số trong đoạn và $carry[d]$ là số phép nhớ chuyển từ cột $d - 1$ sang cột d . Công thức xác định số phép nhớ mới là:

$$carry[d + 1] = \left\lfloor \frac{sum[d + 1] + carry[d]}{10} \right\rfloor$$

Độ phức tạp thời gian: $O(N \cdot K \cdot D)$ với D là số chữ số tối đa ($D \approx 14$).

Subtask 4 (Sử dụng mảng cộng dồn)

Để cải tiến tốc độ tính $cost[i]$, ta xây dựng mảng tổng tiền tố cho các chữ số tại mỗi cột d . Khi đó, tổng các chữ số $sum[d]$ của một đoạn độ dài K bắt đầu tại i được tính nhanh qua công thức:

$$sum[d] = prefsun[i + K - 1][d] - prefsun[i - 1][d]$$

Độ phức tạp thời gian: $O(N \cdot D + N)$.

Subtask 5 (Quy hoạch động)

Để chọn ra M đoạn tối ưu không giao nhau, ta sử dụng quy hoạch động. Gọi $dp[i][j]$ là tổng số phép nhớ ít nhất khi chọn j đoạn trong phạm vi i phần tử đầu tiên. Các trạng thái chuyển đổi:

- (1) Không chọn đoạn kết thúc tại i : $dp[i][j] = dp[i - 1][j]$.
- (2) Chọn đoạn độ dài K kết thúc tại i : $dp[i][j] = dp[i - K][j - 1] + cost[i - K + 1]$.

Độ phức tạp thời gian cuối cùng: $O(N \cdot D + N \cdot M)$.

Cài đặt (tham khảo)

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 1e5 + 1;
5
6 int a[N], S[N], cost[N], dp[2][N];
7
8 int calc_S(long long x) {
9     int res = 0;
10    while(x > 0) {
11        res += x % 10;
12        x /= 10;
13    }
14    return res;
15 }
16
17 int main() {
18     ios_base::sync_with_stdio(0);
19     cin.tie(0);
```

```

20
21     int n, m, k;
22     cin >> n >> m >> k;
23
24     int sum_S = 0;
25     long long sum_a = 0;
26     for(int i = 1; i <= n; ++i) {
27         cin >> a[i];
28         sum_a += a[i];
29
30         S[i] = calc_S(a[i]);
31         sum_S += S[i];
32
33         if(i >= k) {
34             sum_a -= a[i - k];
35             sum_S -= S[i - k];
36             cost[i] = (sum_S - calc_S(sum_a)) / 9;
37         }
38     }
39
40     bool cur = 0;
41     for(int j = 1; j <= m; ++j) {
42         bool prev = cur;
43         cur ^= 1;
44         fill(dp[cur], dp[cur] + n + 1, inf32);
45
46         for(int i = k * j; i <= n; ++i) dp[cur][i] = min(dp[cur][i - 1], dp[prev][i - k] +
47 cost[i]);
48     }
49     cout << dp[cur][n];
50
51     return 0;
52 }

```

— HẾT —

- Thí sinh **KHÔNG** được sử dụng tài liệu.
- Giám thị **KHÔNG** được giải thích gì thêm.