
Kiểm tra đúng đắn &
hiệu năng bằng bộ
test



CONTENTS ✨

01

Mở đầu

02

Phân loại test

03

Công cụ sinh test

04

Kiểm thử phần mềm

05

Quiz&BTVN

NEXT

PART

01

Mở đầu

Vai trò của bộ test

Bộ test đóng vai trò then chốt trong việc kiểm tra chương trình vừa đúng vừa nhanh. Nó giúp phát hiện lỗi sớm, tránh tối ưu mang tính may rủi và đảm bảo chương trình hoạt động như mong muốn.

Liên hệ thực tế

Trong thực tế môn học và kỳ thi, bộ test giúp sinh viên kiểm tra kết quả, kịp thời phát hiện sai sót và điều chỉnh. Nó là công cụ không thể thiếu trong quá trình học và thi.

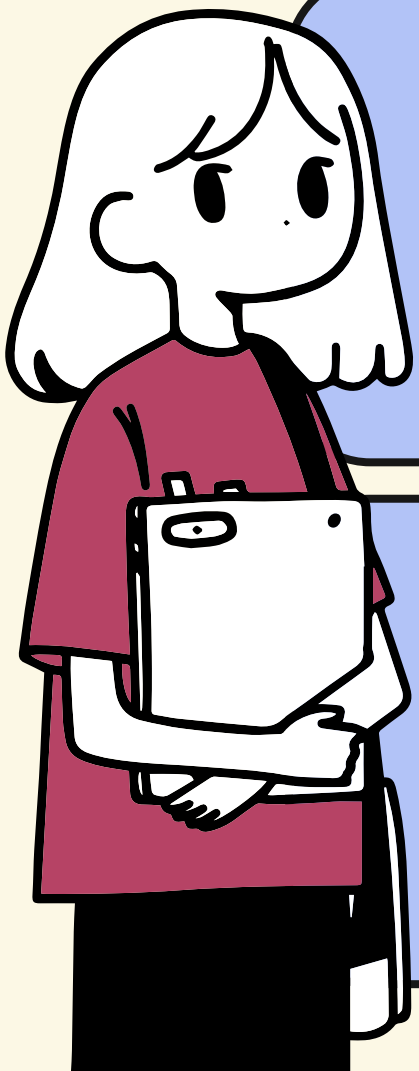


Tại sao cần bộ test?

Tầm quan trọng của test

Không có test đầy đủ, mọi tối ưu đều mang tính may rủi. Test giúp đảm bảo chương trình không chỉ chạy được mà còn chạy chính xác và hiệu quả, tránh các lỗi tiềm ẩn.

Correctness & Performance



Tính đúng đắn

01

Correctness là tiêu chí cơ bản của chương trình, yêu cầu chương trình trả về kết quả chính xác với mọi input hợp lệ. Logic thuật toán phải chuẩn xác, không sai lệch.

Hiệu năng

02

Performance yêu cầu chương trình chạy nhanh, sử dụng bộ nhớ tiết kiệm và có khả năng mở rộng tốt khi dữ liệu lớn. Đây là yếu tố quan trọng để đảm bảo chương trình hoạt động hiệu quả.



NEXT

PART



02

Phân loại test

Phân loại test cơ bản

Nhóm Normal gồm các input điển hình và phổ biến mà người dùng sẽ gặp. Mục đích là kiểm tra logic chính của thuật toán trong điều kiện bình thường.

Normal

01

Edge

02

Nhóm Edge gồm các input ở ranh giới như min/max, 0, 1, giới hạn kiểu dữ liệu. Mục đích là tìm lỗi logic hay lỗi off-by-one, kiểm tra điều kiện ranh giới.



Nhóm Special gồm các input hợp lệ nhưng bất thường hoặc ít gặp, thường làm lộ logic không lường trước được của chương trình.

03

Special

04

Stress

Nhóm Stress gồm các input cực đại, dùng để đo hiệu năng của chương trình, tìm các giải pháp không tối ưu về thời gian và tài nguyên.

Ví dụ: Bộ test tìm max mảng pair

(không sử dụng phép so sánh có sẵn của pair)

Ràng buộc: $1 \leq n \leq 10^6$; $-10^9 \leq a[i].first, a[i].second \leq 10^9$.

Đặt tên file theo thứ tự: test01.in, test02.in,...



Normal

test01: $n=5, [(1, 2), (2, 3), (1, 3)] \rightarrow (2, 3)$

test02: $n=2, [(1, 2), (5, 1)] \rightarrow (5, 1)$

Special

test05: $n=3, [(3, 6), (6, 3), (6, 9)] \rightarrow (6, 9)$

test06: $n=2, [(1, 1), (1, 1)] \rightarrow (1, 1)$

Edge

test03: $n=2, [(1000000000, 0), (-1, -1)] \rightarrow (1000000000, 0)$

test04: $n=10^6, a[] = \text{random}$

Stress

test07: $n=10^5, a[] = \text{random}$

test08: $n=10^6, a[] = \text{random}$

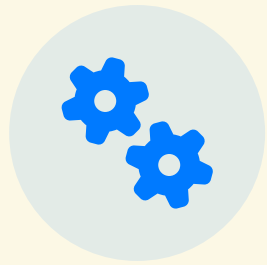
NEXT

PART

03

Công cụ sinh test

Bộ ba công cụ: Generator, Validator, Checker



Generator

Sinh input theo seed và tham số.



Validator

Kiểm tra input đúng format và ràng buộc.



Checker

So sánh output thí sinh với đáp án chuẩn.

Để đảm bảo rằng test đúng và chấm nhanh chính xác.

NEXT

PART

3.1

Generator testlib.h

Generator & testlib.h



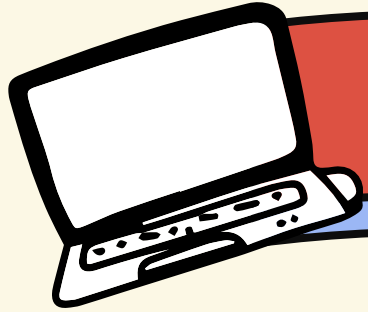
01 Vai trò của generator

Generator là công cụ sinh input có cấu trúc và ngẫu nhiên. Nó giúp tạo ra các trường hợp test đa dạng, đảm bảo kiểm tra toàn diện chương trình.

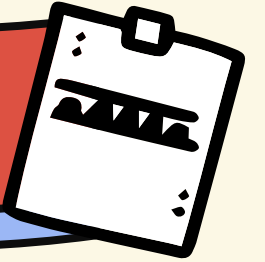
Sử dụng testlib.h

Testlib.h cung cấp các hàm như `rnd.next`, `rnd.perm`, `shuffle`, `setSeed` để sinh dữ liệu test. Đặt seed cố định giúp tái lập test khi cần thiết.

02



Sinh dữ liệu đặc biệt



01

Mảng gần sorted

Tạo mảng đã sắp xếp rồi xáo trộn một phần nhỏ để tạo dữ liệu gần sorted, giúp kích bug ở thuật toán sắp xếp.

02

Phân phối chuông

Sử dụng hàm `rnd.nextGaussian` để sinh số tập trung quanh mean với độ lệch sigma, phù hợp với các trường hợp cần dữ liệu có xu hướng.

03

Chọn có trọng số

Hàm `rnd.any` giúp chọn ngẫu nhiên phần tử từ vector với trọng số, tạo dữ liệu thiên về nhỏ hoặc lớn tùy thuộc vào yêu cầu.



NEXT

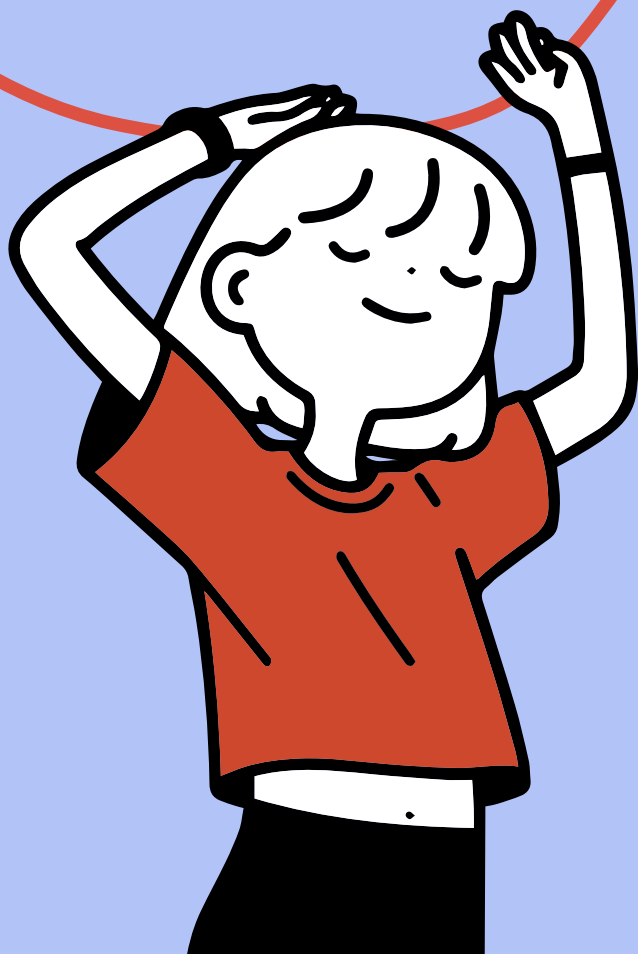
PART



3.2

Checker

Checker: so output



01 So sánh output

Checker so sánh output của thí sinh với đáp án chuẩn, hỗ trợ so sánh chính xác (exact) cho số nguyên, dung sai (tolerance) cho số thực và special judge cho trường hợp đặc biệt.

02 Các lệnh sử dụng

Checker sử dụng lệnh `ouf.readInt` và `ans.readInt` để đọc output và đáp án, kết luận bằng `quitf(_ok/_wa/_pe)` tùy theo kết quả so sánh.

03 Tích hợp vào bash loop

Checker có thể tích hợp vào bash loop để chấm hàng loạt test, giúp tiết kiệm thời gian và đảm bảo tính chính xác của kết quả.

Ví dụ về cách Checker hoạt động

A. Tuyến đường ngắn nhất

time limit per test: 1 second

memory limit per test: 256 megabytes

input: duongdingannhat.inp

output: duongdingannhat.out

Cho một bảng A kích thước $m * n$, ô ở hàng i và cột j có giá trị $a_{i,j}$ ($1 \leq i \leq m, 1 \leq j \leq n$). Tùng, người xuất phát tại ô nào đó của cột 1, cần sang cột n (tại ô nào cũng được).

Quy tắc đi: Từ ô (i, j) chỉ được quyền sang một trong 3 ô $(i, j + 1); (i - 1, j + 1); (i + 1, j + 1)$.

Hãy giúp Tùng lựa chọn tuyến đường có tổng trọng số từ các ô đi qua là bé nhất có thể.

Input

- Dòng đầu tiên gồm hai số m, n là số hàng và số cột của bảng. ($1 \leq m, n \leq 800$)
- n dòng tiếp theo, dòng thứ i ghi đủ n số trên hàng i của bảng theo đúng thứ tự từ trái qua phải ($-1000 \leq a_{i,j} \leq 1000$).

Output

- Dòng đầu tiên in ra tổng bé nhất tìm được.
- n dòng tiếp theo, mỗi dòng ghi các ô mà Tùng đã đi qua

Scoring

- Subtask 1 (40 điểm): $m, n \leq 10$.
- Subtask 2 (60 điểm): Không có giới hạn gì thêm.

Example

input


```
5 7
9 -2 6 2 1 3 4
0 -1 6 7 1 3 1
8 -2 8 2 5 0 2
1 -1 6 2 1 6 1
7 -2 6 2 1 3 7
```


output


```
8
2 1
1 2
1 3
1 4
2 5
3 6
2 7
```


Checker: Chấm bài tự động

So sánh output thí sinh với đáp án chuẩn và đưa ra kết luận.

 Đọc output thí sinh (ouf) và đáp án (ans).

 So sánh: exact (int), tolerance (float), hoặc special judge.

 Kết luận: `quitf(_ok)` hoặc `quitf(_wa)`.

 Tích hợp vào bash loop để chấm hàng loạt.



Output Thí Sinh




Đáp Án Chuẩn


Checker ví dụ


```
1  #include <bits/stdc++.h>
2  #include "testlib.h"
3
4  using namespace std;
5  const int MAX = 807;
6
7  int m, n;
8  int a[MAX][MAX];
9  int sumans; bool calchecker = false;
10 bool partial = false;
11
12 inline void readAndCheckAnswer(InStream &in) {
13     if (calchecker == false) {
14         sumans = in.readInt();
15         calchecker = true;
16         return;
17     }
18
19     int nowans = in.readInt();
20
21     if (nowans != sumans) {
22         in.quitf(_wa, "Wrong answer, find = %d, answer = %d", nowans, sumans);
23     }
24 }
```

Checker: Chấm bài tự động

So sánh output thí sinh với đáp án chuẩn và đưa ra kết luận.

 Đọc output thí sinh (ouf) và đáp án (ans).

 So sánh: exact (int), tolerance (float), hoặc special judge.

 Kết luận: `quitf(_ok)` hoặc `quitf(_wa)`.

 Tích hợp vào bash loop để chấm hàng loạt.



Output Thí Sinh




Đáp Án Chuẩn


Checker ví dụ


```
1
2  int sum(0);
3  int prei = -1;
4
5  for (int j = 1; j <= n; j ++){
6      if (in.eof()) {
7          partial = true;
8          break;
9      }
10     int u = in.readInt(), v = in.readInt();
11
12     if (prei != -1 && abs(prei - u) > 1) {
13         partial = true;
14     }
15     if ((u < 1 || u > m) || v != j) {
16         partial = true;
17     }
18     prei = u;
19     sum += a[u][v];
20 }
21 if (sum != sumans) {
22     partial = true;
23 }
24 }
```

Checker: Chấm bài tự động

So sánh output thí sinh với đáp án chuẩn và đưa ra kết luận.

 Đọc output thí sinh (ouf) và đáp án (ans).

 So sánh: exact (int), tolerance (float), hoặc special judge.

 Kết luận: `quitf(_ok)` hoặc `quitf(_wa)`.

 Tích hợp vào bash loop để chấm hàng loạt.



Output Thí Sinh




Đáp Án Chuẩn


Checker ví dụ


```
1 int main(int argc, char *argv[]) {
2     registerTestlibCmd(argc, argv);
3
4     m = inf.readInt();
5     n = inf.readInt();
6     for (int i = 1; i <= m; i++) {
7         for (int j = 1; j <= n; j++) {
8             a[i][j] = inf.readInt();
9         }
10    }
11
12    readAndCheckAnswer(ans);
13    readAndCheckAnswer(ouf);
14
15    if (partial) quitp(0.17, "Output is close to the correct answer");
16    quitf(_ok, "answer = %d", sumans);
17 }
```

Checker: Chấm bài tự động

So sánh output thí sinh với đáp án chuẩn và đưa ra kết luận.

 Đọc output thí sinh (ouf) và đáp án (ans).

 So sánh: exact (int), tolerance (float), hoặc special judge.

 Kết luận: `quitf(_ok)` hoặc `quitf(_wa)` .

 Tích hợp vào bash loop để chấm hàng loạt.



Output Thí Sinh



Đáp Án Chuẩn

Checker ví dụ

```
for (int i = 1; i <= m; i++) {
    dp[i][1] = a[i][1];
}
for (int j = 2; j <= n; j++) {
    for (int i = 1; i <= m; i++) {
        dp[i][j] = dp[i][j - 1];
        trace[i][j] = i;
        if (i < m && minimize(dp[i][j], dp[i + 1][j - 1])) trace[i][j] = i + 1;
        if (i > 1 && minimize(dp[i][j], dp[i - 1][j - 1])) trace[i][j] = i - 1;
        dp[i][j] += a[i][j];
    }
}
int ans = inf;

int u = 0;
for (int i = m; i >= 1; i--) {
    if (minimize(ans, dp[i][n])) u = i;
}
cout << ans << '\n';
}
```

1

Time: 31 ms, memory: 7608 KB

Verdict: OK

Input

```
5 7
9 -2 6 2 1 3 4
0 -1 6 7 1 3 1
8 -2 8 2 5 0 2
1 -1 6 2 1 6 1
7 -2 6 2 1 3 7
```

Participant's output

8

Jury's answer

```
8
2 1
1 2
1 3
1 4
2 5
3 6
2 7
```

Checker comment

points 0.17 Output is close to the correct answer

PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS STANDINGS ADM. EDIT CUSTOM INVOCATION

General

#	Author	Problem	Lang	Verdict	Sent	Judged	
247751579	Manager: Giangcoder	506220A - 8	C++17 (GCC 9-64)	Partial result: 51.17 points	2024-02-22 16:59:39	2025-10-02 21:16:45	<button>Rejudge</button> <button>Ignore</button> <button>Reject</button> <button>To problem</button>

► Show Judge Protocol

→ Source

```
#include<bits/stdc++.h>
using namespace std;

const int MAX = 8e2 + 7;
const int inf = 0x3f3f3f3f;

bool maximize(int &a, const int &b) {
    if (a < b) {a = b; return true;}
    return false;
}

bool minimize(int &a, const int &b) {
    if (a > b) {a = b; return true;}
    return false;
}
```

Bảng lệnh checker thường dùng

Lệnh	Ý nghĩa
ouf.readInt()	Đọc output thí sinh
ans.readInt()	Đọc đáp án chuẩn
quitf(_ok, "...")	Kết luận Accepted
quitf(_wa, "...")	Kết luận Wrong Answer
quitf(_pe, "...")	Kết luận Presentation Error

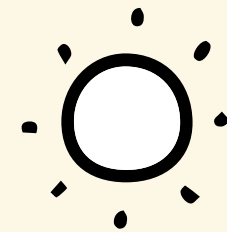
NEXT

PART

3.3

Validator

Validator: kiểm input



01

Nhiệm vụ của validator

Validator đảm bảo input đúng format, đúng range, không thừa ký tự và logic nhất quán. Nó giúp phát hiện lỗi sớm, tránh việc chạy test với dữ liệu không hợp lệ.

02

Cách thực hiện

Validator đọc từng token, sử dụng `ensuref` để bắt lỗi logic phức tạp và kết thúc file đúng lúc bằng `readEof`, đảm bảo tính hợp lệ của input.



Ví dụ về Generator, Validation và Bash

Ví dụ: Bộ test tìm max mảng pair

(không sử dụng phép so sánh có sẵn của pair)

Ràng buộc: $1 \leq n \leq 10^6$; $-10^9 \leq a[i].first, a[i].second \leq 10^9$.

Đặt tên file theo thứ tự: test01.in, test02.in,...

Generator ví dụ

```
1  #include "testlib.h"
2  #include <iostream>
3  #include <cstdlib>
4
5  using namespace std;
6
7  int main(int argc, char* argv[]) {
8      registerGen(argc, argv, 1);
9      int n = atoi(argv[1]);
10     int seed = (argc >= 3 ? atoi(argv[2]) : 712367);
11     rnd.setSeed(seed);
12
13     const int VMIN = 1;
14     const int VMAX = 1001;
15
16     println(n);
17     for (int i = 0; i < n; i++) {
18         int a = rnd.next(VMIN, VMAX);
19         int b = rnd.next(VMIN, VMAX);
20         println(a, b);
21     }
22     return 0;
23 }
24
```

Validation ví dụ

```
1  #include "testlib.h"
2  using namespace std;
3
4  int main(int argc, char* argv[]) {
5      registerValidation(argc, argv);
6
7      const int N_MIN = 1;
8      const int N_MAX = 100000;
9      const int V_MIN = 1;
10     const int V_MAX = 1000;
11
12     int n = inf.readInt(N_MIN, N_MAX, "n");
13     inf.readEoln();
14
15     for (int i = 0; i < n; i++) {
16         int a = inf.readInt(V_MIN, V_MAX, "a_i");
17         inf.readSpace();
18         int b = inf.readInt(V_MIN, V_MAX, "b_i");
19         inf.readEoln();
20     }
21
22     inf.readEof();
23 }
24
```

Generator ví dụ

```
1 #include "testlib.h"
2 #include <iostream>
3 #include <cstdlib>
4
5 using namespace std;
6
7 int main(int argc, char* argv[]) {
8     registerGen(argc, argv, 1);
9     int n = atoi(argv[1]);
10    int seed = (argc >= 3 ? atoi(argv[2]) : 712367);
11    rnd.setSeed(seed);
12
13    const int VMIN = 1;
14    const int VMAX = 1001;
15
16    println(n);
17    for (int i = 0; i < n; i++) {
18        int a = rnd.next(VMIN, VMAX);
19        int b = rnd.next(VMIN, VMAX);
20        println(a, b);
21    }
22    return 0;
23 }
24
```

Bash ví dụ

```
1 #!/bin/bash
2 set -e # nếu lệnh nào lỗi thì dừng luôn
3
4 # biên dịch
5 g++ -std=c++17 -O2 -o validator validator.cpp
6 g++ -std=c++17 -O2 -o gen gen.cpp
7
8 # số lượng cặp n cho mỗi test
9 sizes=(10 20 50 100 200 500 1000 2000 5000 10000)
10
11 # tạo thư mục lưu test
12 mkdir -p tests
13
14 for i in {1..10}
15 do
16     n=${sizes[$((i-1))]}
17     seed=$(date +%s%N) # seed lấy theo thời gian (nano giây)
18     filename=$(printf "tests/%02d.in" $i)
19
20     # sinh file test
21     ./gen $n $seed > "$filename"
22     echo "Generated $filename (n=$n, seed=$seed)"
23
24     # check bằng validator
25     ./validator < "$filename"
26     echo "✅ $filename passed validation"
27 done
```

Generator ví dụ

```
1  # Generated tests/01.in (n=10, seed=1759426636910952000)
2  # ✓ tests/01.in passed validation
3  # Generated tests/02.in (n=20, seed=1759426637482493000)
4  # ✓ tests/02.in passed validation
5  # Generated tests/03.in (n=50, seed=1759426637498260000)
6  # ✓ tests/03.in passed validation
7  # Generated tests/04.in (n=100, seed=1759426637510680000)
8  # ✓ tests/04.in passed validation
9  # Generated tests/05.in (n=200, seed=1759426637521520000)
10 # FAIL Integer parameter [name=b_i] equals to 1001, violates the range [1, 1000] (stdin, line 123)
11
12 int main(int argc, char* argv[]) {
13     registerGen(argc, argv, 1);
14     int n = atoi(argv[1]);
15     int seed = (argc >= 3 ? atoi(argv[2]) : 712367);
16     rnd.setSeed(seed);
17
18     const int VMIN = 1;
19     const int VMAX = 1001;
20
21     println(n);
22     for (int i = 0; i < n; i++) {
23         int a = rnd.next(VMIN, VMAX);
24         int b = rnd.next(VMIN, VMAX);
25         println(a, b);
26     }
27     return 0;
28 }
```

Validation ví dụ

```
1  / const int N_MIN = 1;
2  / const int N_MAX = 100000;
3  / const int V_MIN = 1;
4  / const int V_MAX = 1000;
5
6  int n = inf.readInt(N_MIN, N_MAX, "n");
7  inf.readEoln();
8
9  for (int i = 0; i < n; i++) {
10     int a = inf.readInt(V_MIN, V_MAX, "a_i");
11     inf.readSpace();
12     int b = inf.readInt(V_MIN, V_MAX, "b_i");
13     inf.readEoln();
14 }
15
16 inf.readEof();
17
18 }
```

NEXT

PART

04

Kiểm thử phần mềm

Unit test với Google Test

01

Định nghĩa Unit test

Unit test là phương pháp kiểm tra từng hàm (đơn vị nhỏ nhất) một cách độc lập để đảm bảo chúng hoạt động đúng. Nó giúp phát hiện lỗi sớm, dễ refactor và làm tài liệu sống cho hàm.

02

Ví dụ với hàm factorial

Ví dụ: Viết TEST cho hàm factorial với các trường hợp zero input, positive input và negative input. Lệnh biên dịch g++ kèm -lgtest -lpthread và cách chạy giúp đảm bảo hàm hoạt động chính xác.



Unit Test

- Phát hiện lỗi sớm: Giúp tiết kiệm chi phí sửa lỗi.
- Dễ bảo trì và refactor: Làm tài liệu sống cho hàm.
- Tăng độ tin cậy: Đảm bảo từng module nhỏ hoạt động đúng.

```
1 // factorial.cpp
2 int factorial(int n) {
3     if (n < 0) throw std::invalid_argument("Negative input not allowed");
4     if (n == 0) return 1;
5     int result = 1;
6     for (int i = 1; i <= n; i++) {
7         result *= i;
8     }
9     return result;
10 }
```

```
1 // test_factorial.cpp
2 #include <gtest/gtest.h>
3 #include "factorial.cpp"
4
5 TEST(FactorialTest, HandlesZeroInput) {
6     EXPECT_EQ(factorial(0), 1);
7 }
8
9 TEST(FactorialTest, HandlesPositiveInput) {
10     EXPECT_EQ(factorial(5), 120);
11     EXPECT_EQ(factorial(3), 6);
12 }
13
14 TEST(FactorialTest, HandlesNegativeInput) {
15     EXPECT_THROW(factorial(-1), std::invalid_argument);
16 }
17
18 int main(int argc, char **argv) {
19     ::testing::InitGoogleTest(&argc, argv);
20     return RUN_ALL_TESTS();
21 }
```

Ví dụ: Viết TEST cho hàm factorial với Google test

Lệnh biên dịch & chạy:
g++ test.cpp -lgtest -lpthread -o test
./test

Unit test

- Độc lập, không phụ thuộc vào hệ thống lớn.
- Phát hiện bug sớm, giảm chi phí fix.
- Làm tài liệu “sống” mô tả hành vi code.
- Hỗ trợ bảo trì, mở rộng dễ dàng.

Hạn chế

- Tốn thời gian viết ban đầu.
- Không thay thế được integration test/system test.

```
└─ results/                # Experiment outputs. See detail in results/README.md
└─ scripts/                # Utility scripts
└─ src/                    # Python modules & scripts
    └─ dataloaders/        # Data loading modules
    └─ networks/           # Model definitions
    └─ utils/              # Losses, metrics, etc.
    └─ methods/            # Method implementations
        └─ method_name/    # Method name, e.g., 'AugSeg', 'DGCL'
            └─ __init__.py
            └─ README.md    # Detail described how to run this method and update experi
            └─ method_name.py # Main method implementation, e.g., 'dgcl.py'
            └─ train_semi.py  # Training script
            └─ train_sup.py   # Supervised training script
            └─ evaluate.py    # Evaluation script
    └─ experiments/        # Experiment configurations & scripts
        └─ data_name/      # Data-specific code & configs, e.g., Pascal
            └─ experiment_name/ # Should be number of label partitions, e.g., 732
                └─ method_name/ # Method name, e.g., 'AugSeg', 'DGCL'
                    └─ config.yaml # Specific config for this experiment
                    └─ train.sh    # Training script
                    └─ evaluate.sh # Evaluation script
                    └─ logs/       # Automatically created
                    └─ checkpoints/ # Automatically created. Note: Save best only as 'ckpt_
└─ tests/                  # Unit tests
```

01

Khái niệm Black-box test

Black-box test xem phần mềm như hộp đen, chỉ quan tâm input/output, không cần biết code. Nó giúp phát hiện lỗi chức năng, giao diện, dữ liệu mà không cần hiểu logic bên trong.

03

Phân tích giá trị biên

Phân tích giá trị biên kiểm tra các giá trị ở ranh giới của input, giúp phát hiện lỗi logic hay lỗi off-by-one, thường xảy ra ở các điều kiện ranh giới.

02

Phân vùng tương đương

Phân vùng tương đương chia input thành các lớp tương đương, chọn một giá trị đại diện trong mỗi lớp để kiểm tra, giúp giảm số lượng test case mà vẫn đảm bảo bao phủ đầy đủ.

04

Bảng quyết định

Bảng quyết định giúp kiểm tra các trường hợp phức tạp với nhiều điều kiện và hành động, đảm bảo phần mềm xử lý chính xác trong mọi trường hợp.

Chi tiết Black-box test



Chi tiết White-box test



Khái niệm White-box test

White-box test dựa vào cấu trúc code, yêu cầu người kiểm tra biết rõ logic, lệnh, nhánh, vòng lặp.

Mục tiêu đạt statement, branch, condition, path coverage, loại dead code, phát hiện lỗi logic ẩn.

```
1 # factorial.py
2 def factorial(n: int) -> int:
3     if n < 0:
4         raise ValueError("Negative input not allowed")
5     if n == 0:
6         return 1
7     result = 1
8     for i in range(1, n + 1):
9         result *= i
10    return result
```

```
1 # test_factorial_whitebox.py
2 import unittest
3 from factorial import factorial
4
5 class TestFactorialWhiteBox(unittest.TestCase):
6     def test_negative_input(self): # kiểm tra nhánh n < 0
7         with self.assertRaises(ValueError):
8             factorial(-5)
9
10    def test_zero_input(self): # kiểm tra nhánh n == 0
11        self.assertEqual(factorial(0), 1)
12
13    def test_positive_input(self): # kiểm tra nhánh n > 0 + vòng lặp
14        self.assertEqual(factorial(5), 120) # vòng lặp chạy nhiều lần
15        self.assertEqual(factorial(1), 1) # vòng lặp chạy đúng 1 lần
16
17 if __name__ == "__main__":
18     unittest.main()
```

So sánh: Black-box vs White-box Testing



Black-box (Hộp đen)

Không cần biết code bên trong. Tập trung vào chức năng và input/output.

- Phân vùng tương đương
- Phân tích giá trị biên
- Bảng quyết định
- Use-case testing



White-box (Hộp trắng)

Cần biết code. Tập trung vào cấu trúc bên trong, bao phủ logic.

- Statement coverage
- Branch coverage
- Condition coverage
- Path coverage



NEXT

PART



05

Quiz&BTVN

Quiz Kiểm Tra Kiến Thức

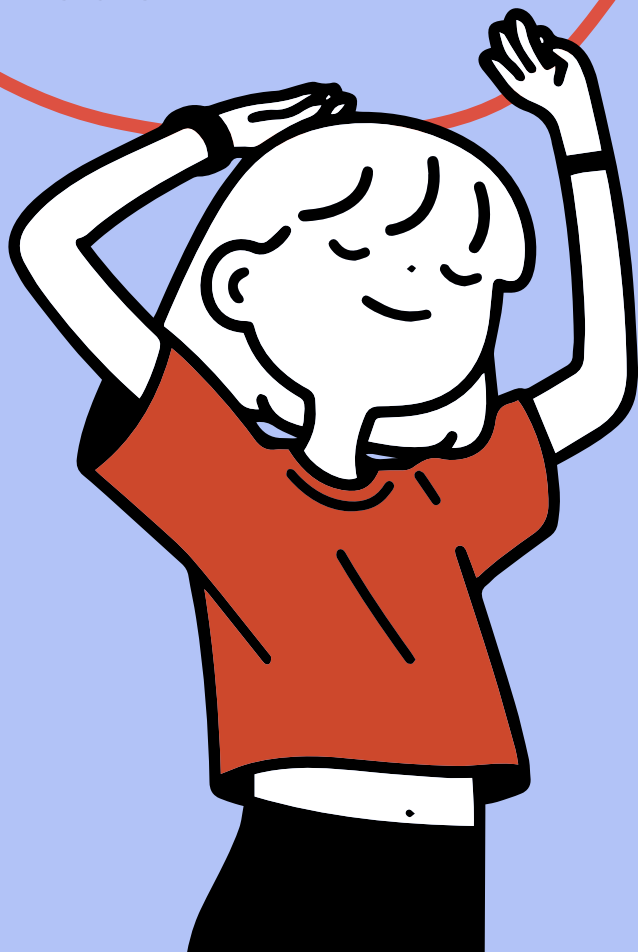
Hãy cùng làm một bài quiz ngắn để ôn lại những kiến thức vừa học!

Link: wayground.com

Mã: 960562



Tổng dãy số – Mô tả bài toán



01 Mô tả bài toán

Cho mảng n số nguyên, tính tổng tất cả phần tử. Input: n và dãy $a_1..a_n$. Output: tổng. Giới hạn $1 \leq n \leq 100\,000$, $-10^9 \leq a_i \leq 10^9$. Subtask 30 điểm $n \leq 1000$, 70 điểm $n \leq 100\,000$.

02 Thuật toán

Thuật toán $O(n)$ đơn giản, duyệt qua mảng một lần để tính tổng. Dễ benchmark và dễ phát hiện overflow 64-bit khi tổng vượt quá giới hạn của kiểu int.

03 Ví dụ

Ví dụ: Input: $n=5$, $a=[1, 2, 3, 4, 5]$. Output: tổng=15. Input: $n=3$, $a=[-1, 0, 1]$. Output: tổng=0.

Tổng dãy số – Bài tập về nhà

01

Generator

Sinh 30 test chia easy ($n \leq 1000$) và hard ($n \leq 1e5$), random giá trị $-1e9..1e9$, thêm case $sum=0$, max/min. Đảm bảo test đa dạng và bao phủ nhiều trường hợp.

02

Validator và Checker

Validator: kiểm tra n nằm trong range, đúng n số trên dòng, không thừa ký tự.
Checker: exact so sánh tổng 64-bit với đáp án chuẩn, báo WA nếu sai, PE nếu thừa khoảng trắng.



Mô tả bài toán



Đồ thị có hướng n đỉnh m cạnh trọng số dương, tìm độ dài đường đi ngắn nhất từ 1 đến n . Input: n m và m dòng $u_i v_i w_i$. Output: độ dài hoặc -1 . Giới hạn $2 \leq n \leq 2000, m \leq 5000, 1 \leq w_i \leq 10^9$.

Thuật toán

Thuật toán Dijkstra hoặc SPFA để tìm đường đi ngắn nhất. Dijkstra phù hợp với đồ thị không có cạnh âm, SPFA có thể xử lý đồ thị có cạnh âm.



Đường đi ngắn nhất – Mô tả bài toán

Subtask

Subtask 30 điểm $n \leq 200, m \leq 1000$; 70 điểm full. Đảm bảo test bao phủ các trường hợp nhỏ và lớn.

Ví dụ

Ví dụ: Input: $n=3, m=2$, cạnh $(1, 2, 10)$ và $(2, 3, 5)$. Output: 15.
Input: $n=3, m=1$, cạnh $(1, 2, 10)$. Output: -1 .





Đường đi ngắn nhất – Bài tập về nhà

Generator, Validator và Checker

Generator: sinh 40 test chia small ($n \leq 200$) và large ($n \leq 2000$), đồ thị ngẫu nhiên, thêm case không đường, cạnh song song. Validator: kiểm tra n m trong range, cạnh không trùng, không self-loop, trọng số dương. Checker: numeric exact so độ dài shortest path hoặc -1 , chấp nhận tolerance 0.

Đường Hamilton – Mô tả bài toán



01 Mô tả bài toán

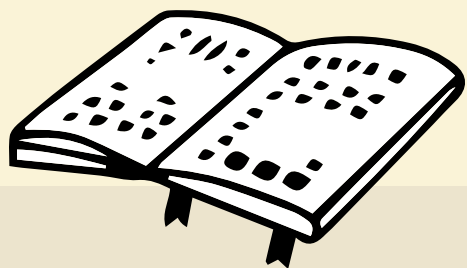
Đồ thị vô hướng n đỉnh m cạnh, in một đường đi Hamilton (qua mỗi đỉnh đúng 1 lần) hoặc -1 . Input: n và m cạnh ui vi. Output: dãy n đỉnh hoặc -1 . Giới hạn $1 \leq n \leq 12$, $m \leq n(n-1)/2$.

Thuật toán

Thuật toán bitmask DFS hoặc dynamic programming on path. Đảm bảo kiểm tra tất cả các đỉnh và cạnh để tìm đường đi Hamilton.

02

Đường Hamilton – Bài tập về nhà



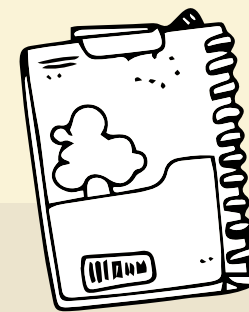
Generator

Sinh 50 test chia nhóm $n \leq 8$ và $n \leq 12$, đồ thị ngẫu nhiên, thêm case không cạnh, complete graph. Đảm bảo test đa dạng và bao phủ nhiều trường hợp.



Validator

Validator: kiểm tra n m hợp lệ, cạnh không lặp, không self-loop, đỉnh trong $1..n$. Đảm bảo input luôn hợp lệ.



Checker

Checker: special judge xác minh hoán vị đủ n đỉnh khác nhau, mọi cặp liên tiếp tồn tại cạnh, hoặc -1 đúng khi không có đường.

THANK YOU

