OI/ACM 数据生成库使用说明文档

📋 目录

1. 概述
2. 环境要求
3. 快速开始
4. 核心功能详解
    - 随机数生成
    - 随机字符串生成
    - 随机树生成
    - 随机图生成
    - 梯度数据生成
5. 示例代码解析
6. 特判程序(SPJ)编写
7. 高级配置
8. 常见问题

概述

这是一个专为OI/ACM题目设计的C++数据生成库，提供以下核心功能：

- ✅ **多种随机分布**：均匀分布、偏左/偏右分布、自定义分布
- ✅ **数据结构生成**：随机树、图、森林、DAG、二分图等
- ✅ **梯度数据生成**：支持多种递增策略生成渐进难度的测试数据
- ✅ **文件批量生成**：自动生成多组输入输出文件
- ✅ **高精度计算**：内置大整数类
- ✅ **内存管理工具**：类型安全的清零/填充工具

环境要求

- C++17 或更高版本
- 支持标准库（STL）
- Windows/Linux/macOS 均可

快速开始

1. 包含头文件

```
#include "DataGenerator.h"
using namespace std;
```

2. 基本使用示例

```
// 生成随机整数（均匀分布）
long long num = random(1, 100);  // [1, 100]之间的随机整数

// 生成偏右分布的随机数
long long right_biased = random(1, 100, 2.0);  // 偏向100

// 生成随机字符串
string s = randomString(10, "abcdefghijklmnopqrstuvwxyz");

// 生成随机树（10个节点）
auto tree_edges = RandomTree.random_tree(10);
```

3. 生成测试数据文件

见 test.cpp 示例。

核心功能详解

随机数生成

`random(long long l, long long r, double opt = 1.0)`

生成区间 `[l, r]` 内的随机整数。

参数说明：

- `l, r`：区间端点
- `opt`：分布参数
  - `opt = 1.0`：均匀分布
  - `opt > 1.0`：偏向 `r`
  - `opt < 1.0`：偏向 `l`

示例：

```
// 均匀分布
int a = random(1, 100);

// 偏向大数的分布（更可能接近100）
int b = random(1, 100, 3.0);

// 偏向小数的分布（更可能接近1）
int c = random(1, 100, 0.3);
```

`randomDouble(double l, double r, double opt = 1.0)`

生成区间 `[l, r]` 内的随机实数，分布规则与 `random` 相同。

随机字符串生成

`randomString(int n, string chars, bool allowZero = false)`

生成指定长度的随机字符串。

参数说明：

- `n`：字符串长度
- `chars`：字符集
- `allowZero`：是否允许前导零（当字符集包含'0'时）

示例：

```
// 生成10位数字字符串，避免前导零
string s1 = randomString(10, "0123456789");

// 生成8位小写字母字符串
string s2 = randomString(8, "abcdefghijklmnopqrstuvwxyz");

// 生成随机选择字符串
vector<string> options = {"YES", "NO", "MAYBE"};
string s3 = randomString(options);
```

随机树生成

`RandomTree.random_tree(int n, int base = 1, int root = 0, double rho = 2.0, bool vertices_rand = true)`

生成随机树，返回边列表。

参数说明：

- `n`：节点数
- `base`：节点编号偏移量（默认为1，即节点从1开始编号）
- `root`：根节点编号
- `rho`：深度控制参数
  - `rho → 0`：星形树（最浅）
  - `rho = 1`：均匀随机树

- `rho = 2`：较深的树（默认）
- `rho → ∞`：链（最深）

- `vertices_rand`：是否随机打乱节点编号

示例：

```
// 生成10个节点的随机树
auto edges = RandomTree.random_tree(10);

// 生成链式树
auto chain = RandomTree.random_tree_chain(10);

// 生成星形树
auto star = RandomTree.random_tree_star(10);

// 生成二叉树
auto binary_tree = RandomTree.random_binary_tree(10);

// 生成森林（3棵树，共20个节点）
auto forest = RandomTree.random_forest_tree(20, 3);
```

`RandomTree.random_tree_parent(...)`

生成随机树，返回父节点数组。参数与 `random_tree` 相同，但返回值格式不同。

返回值： `vector<int>`，其中 `parent[i]` 表示节点 `i` 的父节点（根节点为-1）

随机图生成

`RandomGraph.random_graph(int n, int m, ...)`

生成随机图，支持多种图类型。

完整参数：

```
vector<pair<int, int>> random_graph(
    int n,                      // 节点数
    int m,                      // 边数
    bool connected = true,      // 是否连通
    bool repeated_edges = false, // 是否允许重边
    bool self_rings = false,    // 是否允许自环
    bool directional = false,   // 是否为有向图
    int base = 1                // 节点偏移
);
```

示例：

```
// 生成无向连通图（默认）
auto graph1 = RandomGraph.random_graph(10, 15);

// 生成有向图
auto graph2 = RandomGraph.random_graph(10, 20, true, false, false, true);

// 生成DAG（有向无环图）
auto dag = RandomGraph.random_dag_graph(10, 15);

// 生成二分图
auto bipartite = RandomGraph.random_binary_graph(5, 5, 10);
```

`RandomGraph.valid_nm(...)`

验证并调整图的节点数和边数参数，确保参数合法。

梯度数据生成

`batchGenerateFiles(...)`

批量生成多组测试数据文件，支持梯度递增策略。

参数说明：

```
bool batchGenerateFiles(
    int startIndex,                                     // 起始文件编号
    int endIndex,                                       // 结束文件编号
    function<void(long long, long long, double)> solve, // 数据生成逻辑
    long long Limit,                                    // 数据规模上限
    stringstream &ins,                                  // 输入流
    stringstream &ous,                                  // 输出流
    int multiple = 1,                                   // 每组数据重复次数
    GradientStrategy strategy = GradientStrategy::SIGMOID // 递增策略
);
```

递增策略（GradientStrategy）：

- `LINEAR`：线性递增（均匀）
- `EXPONENTIAL`：指数递增（前期慢后期快）
- `LOGARITHMIC`：对数递增（前期快后期慢）
- `QUADRATIC`：二次递增
- `SQRT`：平方根递增
- `SIGMOID`：S型曲线（推荐默认）
- `UNIFORM`：所有文件均匀分布

`GradientScaleGenerator` 类

用于自定义梯度配置。

```
// 自动生成梯度配置
GradientScaleGenerator gen(maxScale, totalFiles);

// 获取第index个文件的数据范围
auto [minVal, maxVal] = gen.getGradientRange(index);

// 获取具体数值（可随机化）
long long scale = gen.getScale(index, true, 1.0);
```

示例代码解析

test.cpp 完整解析

```
#include "DataGenerator.h"
using namespace std;
#define int long long

// 输入输出字符串流
stringstream ins;
stringstream ous;

void solve(int DATAL, int DATAR, double OFFSET) {
    // 1. 生成随机整数n，范围在[DATAL, DATAR]之间
    //    使用OFFSET作为分布参数（梯度策略决定）
    int n = random(DATAL, DATAR, OFFSET);

    // 2. 将n写入输入流
    ins << n << '\n';

    // 3. 生成n个随机数
    for (int i = 0; i < n; i++) {
        int x = random(1, 1000);
        ins << x << (i == n-1 ? '\n' : ' ');
    }

    // 4. 计算答案（示例：求和）
    //    实际应根据题目逻辑计算
    int sum = 0;
    // 这里省略计算过程...

    // 5. 将答案写入输出流
    ous << sum << '\n';
}

void init() {
    // 初始化代码（如果需要）
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    init();
```

```
    // 批量生成20个测试文件
    // 数据规模上限为1e5
    // 使用SIGMOID递增策略
    if (batchGenerateFiles(1, 20, solve, 1e5, ins, ous)) {
        cout << "数据文件生成成功\n";
    } else {
        cerr << "数据文件生成失败\n";
    }
    return 0;
}
```

生成的文件结构

运行上述程序会生成以下文件：

```
D:\Desktop\data\1.in
D:\Desktop\data\1.out
D:\Desktop\data\2.in
D:\Desktop\data\2.out
...
D:\Desktop\data\20.in
D:\Desktop\data\20.out
```

文件编号1-20，难度逐渐递增（由SIGMOID策略控制）。

特判程序(SPJ)编写

spj.cpp 结构解析

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

// 评测结果代码
#define PC 99   // 部分正确
#define AC 100  // 全部正确
#define PE 101  // 格式错误
#define WA 102  // 答案错误
#define ERROR 103 // 系统错误

// 文件流对象
ifstream FIn;    // 输入数据
ifstream FUser;  // 用户输出
ifstream FOut;   // 标准输出

void initFiles(int argc, char* args[]) {
    // 参数说明:
    // args[1] = 输入文件路径
    // args[2] = 用户输出文件路径
    // args[3] = 标准输出文件路径

    FIn.open(args[1]);
    FUser.open(args[2]);
    FOut.open(args[3]);
```

```
}

int solve() {
    /***** 特判逻辑 *****/

    // 示例1：简单比较
    /*
    long long a, b;
    FUser >> a;
    FOut >> b;
    if (a == b) return AC;
    else return WA;
    */

    // 示例2：浮点数比较（相对误差1e-6）
    /*
    double x, y;
    FUser >> x;
    FOut >> y;
    if (abs(x - y) / max(1.0, abs(y)) < 1e-6) return AC;
    else return WA;
    */

    // 示例3：多行输出比较
    /*
    string line1, line2;
    while (getline(FUser, line1) && getline(FOut, line2)) {
        if (line1 != line2) return WA;
    }
    return AC;
    */

    return AC; // 默认返回AC
}

int main(int argc, char* args[]) {
    // 初始化文件流
    initFiles(argc, args);

    // 执行特判
    int result = solve();

    // 关闭文件
    FIn.close();
    FUser.close();
    FOut.close();

    return result; // 返回评测结果
}
```

特判程序使用

1. **编译SPJ**

```
g++ -o spj spj.cpp -O2
```

2. **在评测系统中配置**
   - 输入文件：`test.in`
   - 用户输出：`user.out`
   - 标准输出：`std.out`
   - 特判程序：`spj`
3. **SPJ返回值含义**
   - `100`：答案正确(AC)
   - `102`：答案错误(WA)
   - `101`：格式错误(PE)
   - `99`：部分正确(PC)
   - `103`：系统错误

高级配置

自定义文件路径

在 `DataGenerator.h` 中修改：

```
const std::string dataFilePath = "D:\\Desktop\\data\\"; // 数据文件保存路径
const std::string inFileType = ".in";  // 输入文件后缀
const std::string outFileType = ".out"; // 输出文件后缀
```

自定义字符集

```
// 内置字符集常量
const std::string charNumber = "0123456789";
const std::string charLowerCase = "abcdefghijklmnopqrstuvwxyz";
const std::string charUpperCase = "ABCDEFGHIJKLMNOPQRSTUVWxyz";

// 使用示例
string digits = randomString(10, charNumber);
string lowercase = randomString(8, charLowerCase);
```

类型别名

库中定义了一些常用类型别名：

```cpp
using vi = vector<long long>;
using vvi = vector<vector<long long>>;
using vs = vector<string>;
using vpii = vector<pair<long long, long long>>;
using PII = pair<long long, long long>;
```

常用常量

```cpp
const long long INF = 0X7FFFFFFF;
const long long MAX = 0x3f3f3f3f3f3f3f3f;
const long long mod = 1e9 + 7;
const long long MOD = 998244353;
```

常见问题

Q1: 编译错误 "undefined reference to `AMPLE_RAND::rng_64'"

A: 确保在某个.cpp文件中定义随机数引擎：

```cpp
// 在DataGenerator.cpp中已经定义
std::mt19937_64
AMPLE_RAND::rng_64(std::chrono::steady_clock::now().time_since_epoch().count());
```

Q2: 如何生成特定分布的随机数?

A: 使用 `opt` 参数控制分布：

- `opt = 1.0`：均匀分布
- `opt = 0.5`：偏向小值
- `opt = 2.0`：偏向大值
- `opt = 0.1`：极偏小值
- `opt = 10.0`：极偏大值

Q3: 如何生成一棵深度较浅的树?

A: 调整 `rho` 参数：

```cpp
// 生成星形树（深度最浅）
auto star = RandomTree.random_tree(10, 1, 0, 0.01);

// 生成较浅的树
auto shallow = RandomTree.random_tree(10, 1, 0, 0.5);
```

Q4: 如何生成特定类型的图?

A: 使用相应函数：

```
// 连通无向图（默认）
RandomGraph.random_graph(n, m);

// 有向图
RandomGraph.random_graph(n, m, true, false, false, true);

// 有向无环图
RandomGraph.random_dag_graph(n, m);

// 二分图
RandomGraph.random_binary_graph(n1, n2, m);
```

Q5: 如何控制测试数据的梯度?

A: 使用 `GradientStrategy` 参数:

```
// 线性递增（简单→困难）
batchGenerateFiles(1, 10, solve, limit, ins, ous, 1, GradientStrategy::LINEAR);

// 指数递增（前期简单，后期快速变难）
batchGenerateFiles(1, 10, solve, limit, ins, ous, 1,
GradientStrategy::EXPONENTIAL);

// S型曲线（推荐，兼顾简单和困难数据）
batchGenerateFiles(1, 10, solve, limit, ins, ous, 1, GradientStrategy::SIGMOID);
```

Q6: 如何生成多组数据?

A: 使用 `multiple` 参数:

```
// 每个文件包含3组数据
batchGenerateFiles(1, 5, solve, limit, ins, ous, 3);
```

Q7: 如何自定义梯度配置?

A: 使用 `GradientScaleGenerator`:

```
// 自定义梯度区间
vector<GradientScaleGenerator::GradientRange> custom = {
    {1, 100, 5},     // 文件1-5：规模1-100
    {100, 1000, 5},  // 文件6-10：规模100-1000
    {1000, 10000, 5} // 文件11-15：规模1000-10000
};

GradientScaleGenerator gen(custom);
```

总结

本数据生成库提供了完整的OI/ACM题目测试数据生成解决方案，主要特点：

1. **功能全面**：覆盖常见的随机数据结构生成
2. **灵活配置**：支持多种分布、梯度策略
3. **易于使用**：简洁的API接口
4. **高效稳定**：优化的算法实现

通过合理使用本库，可以快速生成高质量、渐进难度的测试数据，大大提高出题效率。

文档版本: 1.0
最后更新: 2026/01/31
作者: Ample