buglab实验报告

2021201709 李俊霖

问题1 shuffle

bug1

- **问题**：输入的m为元素个数，n为询问组数。在判断交换的元素是否越界时应该为（原代码为n）
- **发现**：当m=5，n=2时发现无法交换第3和5个元素时无法实现。
- **修改**：

```
if (a < 0 || a >= m || b < 0 || b >= m)
```

bug2

- **问题**：用异或来实现两个数交换时，如果要交换的两个数是同一个数，即a==b时，经过一次异或操作后这个数始终为0。
- **发现**：当交换相同元素时，该数会清零。（某次尝试时偶然在询问中输入了2 2，发现该结论。）
- **修改**：在交换前加上：

```
if (a == b) continue;
```

或：在交换函数中，修改如下（在swap函数内部改动，使其适应更多情况，笔者认为更合理）。

```
void swap(int &a, int &b)
{
    if(a == b) return;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
}
```

问题2 polycalc

bug1

- **问题**：在累加求和之前，未将求和答案置零。
- **发现**：观察代码可得出，求和之前要对求和存放的值置零。
- **修改**：在求和计算前加上

```
ElemTypeB result = 0;
```

bug2

- **问题**：用朴素的一个一个相乘求幂的办法会时间超限，应采用更快速的求幂方法，或直接调用pow函数（正好头文件已引入cmath）。
- **发现**：提交后时间超限。
- **修改**：在 `calcOnce` 函数中：

```
result = pow(node.base, node.exp);
```

问题3 violetStore

bug1

- **问题**：price的指针对象用构造函数来new一个新的对象，不需要mallow，否则会造成内存问题，此时调用构造函数可以各个类对象的值初始化。
- **发现**：阅读代码可得发现。
- **修改**：

```
price *test = new price;
```

bug2

- **问题**：初始化应将n的值赋为3，否则输出会出错。
- **发现**：无法得到正确输出。
- **修改**：

```
n = 3;
```

bug3

- **问题**：在加入元素的时候，使用 `*prices++` 的形式会导致指针不断后移，导致在输出项目时再加i越界出错。
- **发现**：无法得到正确输出，一直在add函数中报 `segmentation fault` 。
- **修改**：类似输出的形式，用flag下标来访问。flag构造时赋初值0，每次++更新。

```
flag = 0;
```

```
void add_item(std::string name, int price)
    {
        items[flag] = name;
        prices[flag] = price;
        flag++;
    }
```

问题4 swapCase

bug1

- **问题**：用scanf输入遇到空格会终止，无法读入带有空格的字符串。
- **发现**：输入一段有空格的字符串，发现输出只有空格之前的一部分。
- **修改**：使用getline代替

```
std::cin.getline(s, MAXN - 9);
```

bug2

- **问题**：在循环过程中，每次循环时都计算一次strlen，会大大增加时间复杂度，导致超时。而且这个值只需要计算一次，每次都是一样的，因此将该值提前求出存在 `len` 中。
- **发现**：提交报错超时，分析时间复杂度容易看出循环中的超时细节。
- **修改**：

```
int len = strlen(s);
for (int i = 0; i < len; ++i)
```

问题5 xorsum

bug1

- **问题**：调用函数 `Replace(ReadInt(), ReadInt())` 时，后面的参数首先压栈，因此首先 `ReadInt()` 读到的值应当是第二个参数，其次读到的值才是第一个参数。
- **发现**：阅读代码时，对 `ReadInt()` 函数的度读入函数格外关注。后来在数据测试时，发现前后在 `replace` 时两个传入的参数错位了。
- **修改**： 将 `void Replace` 函数中的 `value` 和 `pos` 两个参数互换。

```
void Replace(int value,int pos)
```

问题6 mergeIntervals

bug1

- **问题**： compare 函数中，实际的比较格式是正反各比较一次，两次比较的结果需要为一真一假，因此应该改为严格的小于号 x.l < y.l 。
- **发现**：以往写 compare 函数都是严格大于或小于，因对此处错误有所留心，去查了相关 sort 函数的 compare 操作细节后得知了错误的原因。
- **修改**：

```
bool compare(const Range &x, const Range &y)
{
    return x.l < y.l;
}
```

bug2

- **问题**：在判断是否需要进行区间合并的时候，需要进行两个判断：1.此时区间的左端点**小于等于**之前区间的右端点（原代码已给出）；2.此时区间的右端点取当前区间和之前区间右端点的**更大值**。
- **发现**：构造了一组需要合并的区间，比如区间1：1-4；区间2：2-3。如果只有判断1而无判断2，则会合并成1-3，实际应为1-4。
- **修改**：

```
last.r = it->r > last.r ? it->r : last.r;
```

问题7 8num

bug1

- **问题**： stack 类型的容器在进行 pop 操作时，之前的栈顶元素依然存储在之前的空间内，相当于只是栈顶指针做了移动。当新的元素 push 入栈时，此时覆盖掉之前的栈顶元素。因此， IDS 函数在每次搜索并入栈新的元素时，会把原来 pop 出来的栈顶元素的 parent 信息给覆盖掉，产生自己的 parent 是自己的逻辑错误。
  有的元素入栈时就会将自己的父节点信息覆，从而在逻辑上产生自己的父节点是自己的情况，
- **发现过程如下**：
  - 首先尝试在程序中输出，在输出解决八数码变化路径的过程中的矩阵状态和最终状态。发现最终解决的步骤和最终状态的判断是没有问题的，问题出在查找路径时程序进入了死循环。
  - 查找死循环原因，发现每一次的 parent 都是自己，应该是parent赋值的时候出错了。

```
while (curState != nullptr)
{
    // cout << "in while" << endl;
    stack_path.push(*curState);
    curState = curState->parent;
    cout << "parent"<<curState->parent << endl;
}
```

- 观察逻辑没有发现问题，遂做了许多尝试输出每一轮过程中的每一个状态对应的矩阵和对应的 parent 的值，过程大致如下。

```
State curs = State(que.top().first);
if (curs.parent != nullptr)
{
    cout << "now curs.parent: " << matrix2string(curs.parent->matrix) << endl;
    cout << "now que.top: " << matrix2string(que.top().first.matrix) << endl;
    cout << "now que.top.parent: " << matrix2string(que.top().first.parent->matrix) <<
}
State *curState = &curs;
cout << "now matrix2string(curState->matrix)" << matrix2string(curState->matrix) << en
if (curState->parent != nullptr)
    cout << "now matrix2string_parent" << matrix2string(curState->parent->matrix) << e
```

```
nextState->g = curState->g + 1;
nextDepth = curDepth + 1;
nextState->parent = curState;
if (curState->parent != nullptr)
cout << "2 curState_parent: " << matrix2string(curState->parent->matrix) << endl;
cout << "now nextstate_parent" << matrix2string(nextState->parent->matrix) << endl;
que.push(std::make_pair(*nextState, nextDepth));
cout << "now que.pushin.parent: " << matrix2string(que.top().first.parent->matrix) <<
```

- 发现每一轮入栈的时候状态对应的 parent 是正确的，但是在 curState 取出来的时候就不正确了。
- 怀疑IDS函数中 `State* curState = &curs;` 的行为有一些未考虑到的情况。
- 于是自己写了一段小程序尝试检验正确性，发现问题所在之处。

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    stack<int> t;
    t.push(1);
    t.push(2);
    t.push(3);
    t.push(4);
    t.push(5);

    int *p = &t.top();
    cout << *p << endl;
    t.pop();
    cout << *p << endl;
    t.push(6);
    cout << *p << endl;
    return 0;
}
```

输出为： `5 5 6` 。
可以得知， `stack` 类型的容器在进行 `pop` 操作时，之前的栈顶元素依然存储在之前的空间内，相当于只是栈顶指针做了移动。当新的元素 `push` 入栈时，此时覆盖掉原先的栈顶元素。

- **修改**：代码做了如下修改。
  - 为了将父节点parent的信息储存起来，保证每次parent的信息在新元素入栈后不会被覆盖掉，使用一个全局数组Tmp存储取出来的值，用全局变量cnt来作为数组的寻找下标。

```cpp
Tmp[cnt] = State(que.top().first);
State* curState = &Tmp[cnt];
cnt++;
```

  - 用 `State` 类型的指针 `nowstate` 来存储变量，指针指向父节点之后指针可以通过 `nowstate` 来释放内存。

```cpp
while(curState){
    State *nowstate = curState;
    stack_path.push(*curState);
    curState = curState->parent;
}
```

问题8 segtree

bug1

- **问题**：sum的数据和各类结果的数据会超出整型 `int` 的表示范围，因此需要改成 `long long` 类型。
- **发现**：观察题目数据类型，每一个a_i的范围最大是10^9,n、m的范围最大是10^5，试想如果每个都是取最大时得到的sum和结果必然会超出 `int` 的表示范围。

- **修改**：
  相应的struct修改：

```
struct Node{
        long long int l;
        long long int r;
        long long int add;
        long long int sum;
        long long int size;
        Node* lch;
        Node* rch;
    ~Node();
        Node(long long int,long long int);
        void Maintain();
        void PushDown();
        long long int Query(const long long int&,const long long int&);
        void Add(const long long int&,const long long int&,const long long int&);
};
```

主函数的输出需要做修改：

```
else if(t==2){
        scanf("%d%d",&l,&r);
        printf("%lld\n",N->Query(l,r));
}
```

问题9 antbuster

bug1

- **问题**：若某点信息素为0，该点的信息素不可能再减少了。
- **发现**：从题目原话可以找到 每一秒，地图所有点上的信息素会损失1单位，如果那个点上有信息素的话。
- **修改**：加一个大于0的判断。

```
    void DecreaseSignal(){
        for(int i=0;i<=n;i++){
            for(int j=0;j<=m;j++){
                if(sign[i][j]>0) --sign[i][j];
            }
        }
    }
```

bug2

- **问题**：在 `Tower::Fire()` 函数中，对于斜率的计算，要考虑$dx = 0$的情况。
- **发现**：观察代码可以发现没有$dx = 0$的特判，会发生 `k=dy/dx` 分母为0的情况。
- **修改**：

```
    if(dx!=0){
        double k=dy/dx;
        double b=this->y-k*this->x;
        for(std::list<Ant>::iterator i=ants.begin();i!=ants.end();++i){
        if(Cross(k,-1.0,b,i->x,i->y)&&InSegment(this->x,this->y,target->x,target->y,i->x,i->y)
            i->HP-=d;
            }
        }
    }
    else{
        for(std::list<Ant>::iterator i=ants.begin();i!=ants.end();++i){
            if(Sqr(i->x - target->x) <= 1 &&InSegment(this->x,this->y,target->x,target->y,i->x
             i->HP-=d;
            }
        }
    }
```

bug3

- **问题**：在 `SpecialMove(int dir)` 函数中，把 `dir-1` 改为 `dir+3` ，同样可以实现整除的效果，同时防止了出现数组越界问题出现。
- **发现**：找遍了其他错误点但依然无法得到正确的结果，仔细观察发现此处-1会导致某次数组下标为负值。
- **修改**：

```
void Ant::SpecialMove(int dir){
    dir=(dir+3)%4;
    while(!this->CheckAvailable(this->x+dx[dir],this->y+dy[dir]))
        dir=(dir+3)%4;
    this->NormalMove(dir);
}
```

问题10 softDouble

- **问题**：在所有形如 `1 << 54` 的位运算表达式中，1会被默认设为32位，但此处我们需要的是64位的1，因此需要做类型提升，否则左移32位以上会得到零，得不到位运算需要的掩码效果。
- **发现**：写附加程序对位运算部分进行验算时发现， `1 << 54` 、 `1 << 53` 等表达式的值为零。
- **修改**：
  将64位的1定义为 `ui1` ，然后将所有需要左移相应位数的1改为 `ui1` 。

```
#define ui1 (uint64_t) 1
```

bug2

- **问题**：程序无法相应输出负无穷（-inf） 。

- **发现**：验证边界条件，如$-1/0$，发现程序只能输出无穷（inf），无法输出负无穷。
- **修改**：
  - 在除法函数中，加入除以零得到正负无穷的判断。

```
if(isZero(rhs)){ // divided by zero
      if(isZero(lhs))
          return NaN;
      else
          return Sign(lhs) * Sign(rhs) > 0 ? INF : NINF;
  }
```

  - 被除数是无穷时加入判断。

```
if(isINF(lhs))
      return Sign(lhs) * Sign(rhs) > 0 ? INF : NINF;
```

  - `write_to_string(uint64_t x)` 函数中，字符串输出时加入负无穷分支。

```
else if(isINF(x) && Sign(x) == -1)
      strcpy(ans, "-inf");
  else if(isINF(x))
      strcpy(ans, "inf");
```

bug3

- **问题**：加减法函数中，需要将两个数的指数差值计入 `ediff` 中，然后将两计算数对齐操作时，需要判断指数是否大于64位。如果大于64位，再将cur右移时会出现移动 `ediff mod 64` 位的效果，与我们此时想达到的将64位全部移除清零目标不符。
- **发现**：在模拟ediff大于64位时，输出中间计算结果发现cur的值不为零。
- **修改**：

```
uint64_t cur = ediff>=64? LowBit(rhsf) >> 63 >> 1 : LowBit(rhsf) >> ediff;
```

bug4

- **问题**：在乘法函数判是否需要 `Rounding` 舍入时，处理 `roundup=1` 的条件，还需要判断末尾位数的情况。
- **发现**：受到 `datalab` 启发，模拟判断了一下舍入的情况。
- **修改**：
  沿用 `datalab` 中进位处理的操作方法，枚举考虑真值表可知，只有在末位为11的情况下需要加一舍入，其余诸如01、00、10均不需要（向偶数舍入）。

```
// Rounding
  if((ansf & 1) == 0)
      ansf >>= 1;
  else{
      ansf >>= 1;
      if(roundup||(ansf&0x3>0x2))
          ++ansf;
  }
```