Assignment 4 ADT

> 本次作业中建议大家使用`StanfordCppLib`中的ADT，关于如何使用`StanfordCppLib`在配置的文档中已经详细说明，接口的使用方法大家可以到StanfordCppLib搜索。

> judger的使用命令（以第一题为例）：
>
> ```
> python judger_batch.py -T 1_nat_pred
> ```
>
> 可以不指定`-I, -O`（其默认都为`data\task_name`）和`-S`（默认为`source\task_name`）；当然也可以显式指定
>
> 如果你使用了`StanfordCppLib`，那么需要将编译StanfordCppLib产生的cs1604文件夹的**绝对路径**复制到`source/cs1604.txt`下（需要自己创建），以让`judger`成功编译你的程序。如果不使用StanfordCppLib，则不需要创建`cs1604.txt`。
>
> 因为在评测的时候，judger会根据你`source`目录下是否有`cs1604.txt`来决定是否引入StanfordCppLib，所以如果使用了StanfordCppLib，那么在你提交的文件夹的目录下也需要有`cs1604.txt`这个文件（里面内容可以为空）。

> 本次作业难度按题目顺序**递增**，请合理安排时间。

1. 自然数前驱

课上我们讲了如何使用`vector`实现一个任意长度的自然数并实现了它的后继函数`succ`，即加一函数。在本题中你需要实现一个任意长度自然数的前驱`predecessor`，也就是减一函数，并使用这个函数实现一个自然数相减的功能。

题目描述

给定任意长度的自然数`n`，它的前缀函数`pred`的数学定义如下：

- `pred(n) = n - 1 (n > 0)`
- `pred(n) = 0 (n == 0)`

有了`pred`的定义后，我们可以定义一种简单的自然数相减函数`sub(n,m)`，即调用`m`次`pred(n)`，计算结果是自然数意义下的`n-m`的值，注意如果`n < m`，计算的结果为`0`（根据`pred`的定义）。

输入输出格式

在本题中，我们程序会输入两串数字序列分别代表上面的`n, m`，你需要输出自然数意义下`n-m`的值，例如：

输入

```
00001000000000000000000000000000000000000
10
```

输出

```
99999999999999999999999999999999999990
```

需要注意的是，代表`n`的数字序列可能会超出`c++`中整型的表示范围，所以你应该用`string`来接收`n`并转换成`vector`，另外你需要忽略输入的数字中开头连续的`0`

为了降低计算的复杂度，我们控制`m`的范围为`[0,10000]`

数据范围

对于100%的数据，`0<length(n)<200`，`0<=m<=10000`

2. 洪水

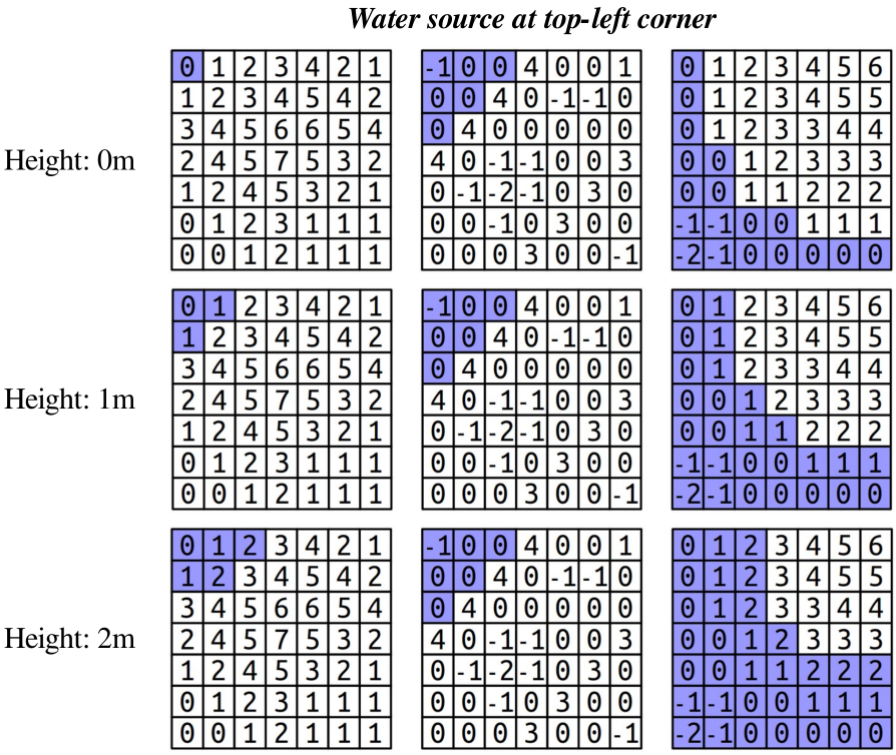
在本题中，会给你一个带有海拔数据的“地形图”，并给出洪水的高度，让你确定最终被淹没的陆地面积。你需要利用合适的`ADT`去表示海拔和泛洪的过程。

题目描述

我们的海拔地图可以用一个二维向量或者说网格（Grid）来表示，其中每个点的值表示这个地方的海拔，用整数表示。如下图所示：

0	1	2	3	4	2	1
1	2	3	4	5	4	2
3	4	5	6	6	5	4
2	4	5	7	5	3	2
1	2	4	5	3	2	1
0	1	2	3	1	1	1
0	0	1	2	1	1	1

我们假设有一个或多个洪水的源头，并且我们知道它的高度（洪水的高度固定），洪水只会朝四个方向（↑↓←→）进行扩散，而且扩散只会往比洪水海拔**低或相等**的方向。如下图所示，纵向表示不同洪水高度导致最后被淹没的面积，横向是不同的地形图，其中洪水的源头为左上角`(0,0)`。需要注意的是源头可能**不止一个**，源头的位置可能是网格中的任意一个点，但是保证每个源头的水位高度一样。



你的任务是计算出**被淹没的陆地面积**，即图中蓝色的部分，每一块蓝色的区域面积算`1`，所以你实现中的核心函数可能类似于下面的形式

```
int floodedArea (const Grid<int>& terrain, const Vector<pair<int,int>>& sources,
int height)
```

其中`terrain`为海拔网格，`sources`为洪水的源头，其中每个元素是一个点的坐标，`height`是洪水的高度。

> 我们在这里使用`pair<int,int>`类型来表示一个点对（当然你可以使用其他方法表示点对，比如说`vector<int>`）。如果`p`是一个点对，那么它的两个点的坐标可以通过`p.first`和`p.second`来获取，详细可以参考C++中pair的用法。下面是一些基本的初始化方法
>
> ```
> typedef pair<int,int> point;
> ...
> point p1(1,1);
> point p2;
> p2.first=2;
> p2.second=2;
> ```

输入输出格式

输入的第一行为`n m k h`表示地图为`n`行`m`列，有k个源头，洪水高度为`h`

接下来k行输入k个源头的坐标`xi yi`

接下来输入地图，每个值表示对应点的海拔高度

```
7 7 1 2
0 0
0 1 2 3 4 2 1
1 2 3 4 5 4 2
3 4 5 6 6 5 4
2 3 5 7 5 3 2
1 2 4 5 3 2 1
0 1 2 3 1 1 1
0 0 1 2 1 1 1
```

输出为被淹没的面积大小，上面例子的输出结果为

```
5
```

我们保证输入数据合法

数据范围

对于80%的数据，`0<n<10，0<m<10，0<k<5`

对于100%的数据，`0<n<1000，0<m<1000，0<k<5`

提示

你要实现的算法是如何模拟洪水扩散的过程，图算法**广度优先搜索（Breath First Search）**可以很好地解决这一个问题，这个算法可以用`queue`来实现。在本题中，用队列中的元素表示洪水已经扩散到的点，把队首元素出队，并判断它四周的点是否会被淹没，若会被淹没，则加入队尾。伪代码如下：

```
create an empty queue;
for (each water source at or below the water level) {
    flood that square;
    add that square to the queue;
}
while (the queue is not empty) {
```

```
    dequeue a position from the front of the queue;
    for (each square adjacent to the position in a cardinal direction) {
        if (that square is at or below the water level and isn't yet flooded) {
            flood that square;
            add that square to the queue;
        }
    }
}
```

3. ASM虚拟机

在这次作业中，你将会实现一个简单的"ASM"语言（Assembly）的虚拟机。虚拟机也是一个计算机程序，它可以模拟真实的计算机系统，而在这次作业中要实现的虚拟机功能是模拟一段程序的执行，不过这个程序不是用C, Java, 和Python等语言写的，而是用一个更加简单，更加接近计算机底层的语言——"ASM"

我们要实现的虚拟机的核心部件正是大家学过的`Stack`，基于栈的虚拟机称为**Stack Machine**[1]，例如课上介绍的RPN。实际中Stack Machine的例子有Java虚拟机JVM[2]和Python的字节码解释器CPython[3]等。

除此之外，另一种常见的计算模型是**Register Machine**[4]，它是一种更加符合现代计算机体系结构的模型。

栈机器

我们的虚拟机通过输入一段程序，对这段程序进行解析，并模拟运行，最终输出结果。

Stack Machine的核心是它的计算模型。

计算模型

Stack Machine的计算模型包含三部分：

- A program counter(pc)：指向现在执行的语句，值为整数，从0开始
- A state: 存储变量到其值的映射，变量的类型是`string`，值的类型是`int`
- An evaluation stack: 存储操作数，操作数都是`int`类型
- Language: 该栈机器的执行语言

语言

我们的机器输入的是一个小型的程序语言"ASM"，它是一种更接近计算机底层的语言，也可以称为指令。

具体我们要实现的指令有：

- `Add`: 将栈顶两个元素出栈，两者相加，并将结果放入栈顶
- `Sub`: 将栈顶两个元素出栈，第二个出栈的元素减去第一个出栈的元素，并将结果放入栈顶
- `Mul`: 乘法，与`Add`类似
- `Div`: 整数除法，与`Sub`类似
- `SetVar x`: 将栈顶元素a出栈，并更新state将变量x映射到a，其中x是一个类型为string的变量名
- `Var x`: 将state中变量x的值放入栈顶
- `Jmp n`: 跳转到pc=n处
- `JmpEq n`: 将栈顶两个元素出栈，若两者相等，则跳转到pc=n处
- `JmpGt n`: 将栈顶两个元素出栈，若第二个出栈的元素大于第一个出栈的元素，则跳转到pc=n处
- `JmpLt n`: 小于关系，执行方式与`JmpGt`类似
- `Const n`: 将整数n放入栈顶
- `Print x`: 打印变量x的值并换行
- `Halt`: 程序结束

程序执行

从pc=0开始，程序每一步会读取当前`pc`所指向的指令，并执行指令，每执行一条非跳转（不是`Jmp`类型）指令，`pc=pc+1`。若遇到`Jmp n, JmpEq n, JmpGt n, JmpLt n`这类的指令，则根据执行结果选择是否跳转到`pc=n`处，如果不跳转，则`pc=pc+1`。更新`pc`后重复这一过程，直到遇到`pc`指向`Halt`或者程序运行错误。

程序的执行伴随着**stack**和**state**的变化，在你实现Stack Machine的程序中，应该使用两个**ADT**来分别表示它们。

程序结果的正确性通过`Print x`指令的输出以及错误处理来判断，我们的测试会比较你的输出与正确的输出。

错误处理

用这个语言写的程序并不保证安全，它有可能出现各种各样的错误，本题需要你处理四种错误：除零，跳转越界，操作数缺失和输出未定义变量。我们需要你在出现这些错误时，输出`Error`并终止程序（注意这个错误并不会影响你之前输出的数据）

示例: factorial

计算10的阶乘，输入的第一行代表指令的数量。

Input

这里为了方便解释把指令所在的位置标出来，实际输入中不会有这些值。

```
18
0: Const 10
1: SetVar z
2: Const 1
3: SetVar y
4: Var z
5: Const 0
6: JmpEq 16
7: Var y
8: Var z
9: Mul
10: SetVar y
11: Var z
12: Const 1
13: Sub
14: SetVar z
15: Jmp 4
16: Print y
17: Halt
```

Output

```
3628800
```

其对应的C++代码如下，注释中标出每条语句对应输入的哪些指令

```
//C++ like language
//Every statement corresponds to serval commands
z = 10; //pc from 0 to 1
y = 1; //pc from 2 to 3
while(z != 0) { //pc from 4 to 6
      y = y * z; // pc from 7 to 10
      z = z - 1; // pc from 11 to 14
} // pc 15
cout<<y<<endl; // pc 16
```

其中`pc=4`到`pc=6`是判断`z`的值是否为`0`，如果是，则跳转到`pc=16`输出阶乘的结果，不是则进入循环。`pc=15`是执行完循环体并跳回到循环条件判断语句，也就是`pc=4`

输入输出格式

我们保证输入格式都是合法的

Input format

```
n: number of commands
line 1: command 0
line 2: command 1
...
line n: command n-1
```

Output

对于运行中遇到的`Print x`指令都要输出一行数据或者在运行错误时输出`Error`并终止程序

数据范围

对于100%的数据，`0<n<100`

提示

- 关于指令的读取，这里推荐一种方法：你可以通过`getline`方法读取一行指令，比如`"Const 10"`。接下来你需要将它分解（split），根据空格来将指令分成一个个的`token`，这里的`token`分别为`"Const"`，`"10"`，然后将之存储到`Vector`中。关于如何根据空格来分解`string`，这里贴出网上提供的解决方法How do I iterate over the words of a string?
- 关于如何将`string`转换为`int`，可以使用`stoi(str)`函数

```
string str = "10";
int a = stoi(str);
```

4. 文本编辑器

文本编辑器（text editor）是一种计算机软件，主要用于用来编写和查看文本文件。程序员写程序也是通过编辑器来编写程序，如今的各种集成开发环境都会自带文本编辑器。比较著名的文本编辑器包括Vim[5],GNU Emacs[6],notepad++以及现在很热门的VSCode，它们的设计理念各不相同，各有特色，有各自的用户群体。甚至在历史上，还发生过“编辑器之战”。[7]

文本编辑器的设计需要考虑性能，用户体验，功能性，可扩展性等等，大多数时候这些性质是互相冲突的，所以要设计出一个好用的编辑器并不简单[8]。不过编辑器的基本功能并不复杂，在本题中，我们的目标是实现一个通过命令行来进行操作的文本编辑器，它具有最简单的编辑功能。

题目描述

我们的编辑器会存储文本并带着光标（cursor），通过命令来修改文本，命令的规范如下：

- `c n m`：将光标移动到第`n`行第`m`个字符后，如果没有这个位置，则忽略此命令
- `i s`：在当前光标后插入字符串`s`（`s`不含换行符），插入结束后光标移动到`s`末尾
- `d k`：从当前光标开始，从后往前删除`max(k,m)`个字符（`m`表示当前光标到行首的字符数目），删除结束后光标移动到被删除字符串的开始位置
- `ENT`：在当前光标下换行，光标位置移动到新行的行首（相当于按下回车）
- `u`：undo操作，撤回最近一次可撤回的操作，如果没有这种操作，则忽略此命令。**可撤回**的操作有`c`,`i`,`d`,`ENT`和`r`
- `r`：redo操作，撤回最近一次没有被redo过的undo操作，如果没有这种undo操作，则忽略这次操作（类似于按下ctrl y）。**注意：**为了实现的方便，undo和redo操作之间不会有`c`,`i`,`d`,`ENT`这些操作（即不会出现`i s1 => u => i s2 => r`这种情况）
- `p`：打印当前编辑器的文本内容，并换行（该指令不能被撤回）

初始状态下，光标位于第1行第0个字符后。

> 鉴于undo和redo实现的难度较大，我们的测试数据有40%不会出现undo和redo，70%不会出现redo。让同学们可以先着手实现基本功能再去考虑进阶功能。

输入输出格式

我们保证输入格式都是合法的

输入的第一行为`k`，表示有k个命令

接下来的k行每行一条命令

```
k: number of commands
line 1: command 0
line 2: command 1
...
line n: command k-1
```

输出：对于运行中遇到的`p`，输出对应的文本

示例

输入

```
20
i #include<iostream>
ENT
i using namespace std
i ;
ENT
i int main(){
c 3 10
ENT
c 4 1
ENT
i cout<<"Hello World!";
d 14
u
u
r
ENT
i return 0; }
c 5 19
i \n
p
```

输出

```
#include<iostream>
using namespace std;
int main()
{
cout<<"Hello World!\n";
return 0; }
```

数据范围

对于40%的数据，不会出现undo和redo操作

对于70%的数据，不会出现redo操作

对于80%的数据，`0<k<100`；对于100%的数据，`0<k<200000`

提示

- undo和redo操作是否像某个ADT?
- 能否做到存储最小的历史信息来完成undo和redo？注意`c`,`i`,`d`,`ENT`这四个操作都是可逆的。

提交格式

你提交的文件结构应该类似如下形式：

```
<your student number>.zip
|- 1_nat_pred
|  |- main.cpp
|
|- 2_flooding
|  |- main.cpp
|
|- 3_asm_vm
|  |- main.cpp
|
|- 4_editor
|  |- main.cpp
|
|- cs1604.txt (include it if you use StanfordCppLib)
```

---

1. https://en.wikipedia.org/wiki/Stack_machine
2. https://en.wikipedia.org/wiki/Java_virtual_machine
3. https://en.wikipedia.org/wiki/CPython
4. https://en.wikipedia.org/wiki/Register_machine
5. https://github.com/vim/vim
6. https://www.gnu.org/software/emacs/
7. 编辑器之战
8. 怎么去实现一个简单文本编辑器？ . 知乎