小马加编信息学教案(三十六)

STL（vector，stack，queue， priority_queue）

一. 课程内容

1. 什么是STL?
2. $vector$
3. $stack$
4. $queue$
5. $priority\_queue$

二. 知识讲解

1. 什么是STL?

> 编写程序不只可以用C++一种语言，那我们为什么要使用C++呢?

> 一个原因是它效率高且容易编写.
> 另一个原因是它自带了标准模板库（Standard Template Library），也就是我们所说的STL，其中封装了很多实用的

> 容器，也可以理解成能够实现很多功能的系统函数，或者说用来存放数据的对象，开发者可以根据接口规范（调用格式）直接调用，而不用关心其内部实现的原理和具体代码，十分方便快捷。

> 我们常用的容器有`vector、stack、queue、map、set`等。使用STL中的函数，一般都要在头文件后加上`using namespace std`。

2. $vector$

2. 1 $vector$的介绍

> $vector$直接翻译为**向量**，一般说成**变长数组**，即**长度根据需要而自动改变的数组**。

> 在信息学竞赛中，有些题目需要定义很大的数组，这样会出现**超出内存限制**的错误。比如，有n个容器，m个球，告诉你每个球是在哪个容器，问每个容器里面有哪些球，且n, m都等于100000。

> 第一种方法：对每个容器，遍历完所有的球，看一些有哪些球是属于这个容器，那么需要$O(nm)$的时间复杂度。

> 第二种方法：对每个容器，开一个长度为m的数组，记录里面有哪些球，虽然只用对所有球遍历一次，但是需要$O(nm)$的空间复杂度。

> 在第二种方法的存储中使用$vector$就很适合，由于每个数组可以任意改变长度，我总共只需要开$O(m)$的数组存放每个桶里有哪些球。

2. 2 $vector$的定义

> 使用$vector$，首先需要添加$vector$头文件，即`#include <vector>`，同时，必须要有`using namespace std`（下面所有STL的使用都需要加）。

```
#include <vector>
#include <cstdio>

using namespace std;

int main() {

}
```

> 定义一个 vector 的方法如下：

```
vector<typename> name;
```

> 以上定义相当于定义了一个一维数组$name[size]$，只是$size$不确定，其长度可以根据需要而变化。其中$typename$可以是任何基本类型或容器，例如`int、double、char`等，当然里面也可以是结构体和其他STL容器。

2. 3 $vector$的访问

> $vector$的访问很简单，使用时可以就把他视为一个数组。直接用下标进行访问。

> 假如一个$vector$类型变量a的长度是len，将其下表为i的位置赋值成i，可以通过如下方法实现。

```
    for (int i = 0; i < len; i ++)
        a[i] = i;
```

2. 4 $vector$的常用函数

> 定义a是一个$vector$类型变量，在下文中理解成一个边长数组

`vector<int> a;`

- `a.push_back(x)`
  令a的数组大小加一，并在最后一个位置放一个元素x，时间复杂度为$O(1)$。
- `a.size()`
  函数返回当前a数组的大小，时间复杂度为$O(1)$。
- `a.pop_back()`
  删除a中最后一个元素，并使数组大小减一，时间复杂度为$O(1)$。
- `a.clear()`
  清空a中所有元素，令a数组大小为o，时间复杂度是$O(n)$，n为a数组中原来的元素个数。

2. 5 $vector$的应用

> 一般当需要开n个数组，每个数组需要开的长度未知，但是总长度已知的情况下，可以开n个$vector$类型变量来进行存储，已达到节省空间的效果。

3. $stack$

3. 1 $stack$的介绍

> stack翻译为**栈**，是STL中实现的一个**后进先出**的容器，与$vector$类似，也能看作一个变长的数组，但只能通过函数访问栈顶元素，不能对中间元素进行访问。

3. 2 $stack$的定义

> 类似与$vector$，使用$stack$前，要先添加$stack$头文件，即`#include <stack>`。

> 定义一个$stack$的方法如下：

```
stack<typename> name;
```

> 其中，$typename$可以是任何基本类型或者容器，$name$是栈的名字。

3. 3 $stack$的常用函数

> 定义a是一个$stack$类型变量。

```
stack<int> a;
```

- `a.push(x)`
  将x压入a的栈顶，时间复杂度为$O(1)$。
- `a.top(x)`
  获取a中栈顶元素，时间复杂度为$O(1)$。
- `a.pop(x)`
  弹出a中栈顶元素，时间复杂度为$O(1)$。
- `a.empty()`
  用于检测a是否为空栈，空则返回$true$，否则返回$false$，时间复杂度为$O(1)$。
- `a.size()`
  返回栈a中元素个数，时间复杂度为$O(1)$。

4. $queue$

4. 1 $queue$的介绍

> $queue$翻译为**队列**，是STL中实现的一个**先进先出**的容器，与$vector$类似，可以看作是一个变长数组，但只能通过函数来访问队首或队尾元素，中间元素不能访问。

4. 2 $queue$的定义

> 类似与$vector$，使用$queue$前，要先添加$queue$头文件，即`#include <queue>`。

> 定义一个$queue$的方法如下：

```
queue<typename> name;
```

> 其中，$typename$可以是任何基本类型或者容器，$name$是栈的名字。

4. 3 $queue$的常用函数

> 定义a是一个$queue$类型变量。

```
queue<int> a;
```

- `a.push(x)`
  将x元素加入队列a队尾，时间复杂度为$O(1)$。
- `a.front(),a.back()`
  分别返回队首和队尾元素的值，时间复杂度为$O(1)$。
- `a.pop()`
  让队首元素出队，时间复杂度为$O(1)$。
- `a.empty()`
  用来检测队列a是否为空，若为空返回$true$，否则返回$false$，时间复杂度为$O(1)$

5. $priority\_queue$

5. 1 $prority_queue$的介绍

> $priority_queue$翻译为**优先队列**，一般用来解决一些贪心问题，其底层是用“堆”来实现的。

> 在优先队列中，任何时刻，队首元素默认是优先级最大的元素（大根堆）。可以不断往优先队列中添加某个优先级的元素，也可以不断弹出优先级最高的那个元素，每次操作其会自动调整结构，始终保证队首元素的优先级最高。

5. 2 $priority_queue$的定义

> $priority_queue$是$queue$库中的容器，所以y而需要添加$queue$的头文件，即`#include <queue>`。

> 定义一个$priority_queue$的方法如下：

```
priority_queue<typename> name;
```

> 其中，$typename$可以是任何基本类型或者容器，$name$为优先队列的名字。

5. 3 $priority_queue$的常用函数

> 和$queue$不一样的是，$priority_queue$没有$front()$和$back()$，而只能通过$top()$或$pop()$访问队首元素（也称堆顶元素），也就是优先级最高的元素。

> 定义a是一个$queue$类型变量。

```
priority_queue<int> a;
```

- `a.push(x)`
  将x加入优先队列a，时间复杂度为$O(log_2^n)$，n为当前优先队列中的元素个数。加入后会自动调整$priority_queue$的内部结构，以保证队首元素(堆顶元素)的优先级最高。
- `a.top()`
  返回优先队列a中给的队首元素，时间复杂度为$O(1)$。
- `a.pop()`
  让优先队列a中队首元素(堆顶元素)出队，时间复杂度为$O(log_2^n)$，n为当前优先队列中的元素个数。出队后会自动

调整$priority_queue$的内部结构，以保证队首元素(堆顶元素)的优先级最高。

三. 经典例题

1. 放球

> 有n个容器，m个球，告诉你每个球是在哪个容器，问每个容器里面有哪些球?

> 输入格式:
> 第一行两个整数n, m。
> 接下来m行，每行一个整数a_i表示第i个球属于哪个容器。
> $n, m \leq 10^5$

> 输出格式:
> 一共m行，对于第i行，第一个整数num_i表示第i个容器里球的数量，接下来num_i个数，从小到大依次输出在第i个容器中球的编号。

样例输入	样例输出
3 5 1 2 1 3 3	2 1 3 1 2 2 4 5

2. 栈

> 给定一个栈，$1 \sim n$依次进栈。有两种操作，一种是"IN"，表示将一个数进栈。另一个是"OUT"，表示将栈顶元素出栈，并输出它的值。读入n和操作顺序，并模拟这个过程。
> 对于任何"OUT"操作，保证栈中存在元素。"IN"和"OUT"分别有n个。

> 输入格式:
> 第一行一个正整数n，表示元素个数。
> 接下$2n$行，每行为"IN"或"OUT"中的一种操作。
> $n \leq 1000$

> 输出格式:
> 对于每个"OUT"操作，输出一行一个整数，表示栈顶元素。

样例输入	样例输出
4 IN IN OUT IN OUT OUT IN OUT	2 3 1 4

3. 最大元素

> 现在有两种操作，一种是加入一个数x，另一种是去掉当前最大的数并输出它的值。现在有n次操作，要求对于所有第二种操作，输出删去数的值。
> 保证对于第二种操作，当前元素数量大于0。

> 输入格式:
> 第一行一个整数n表示操作数。
> 接下来n行，每行第一个正整数$type$等于1或2表示操作类型，若为1则为第一种操作，再读入一个数x表示要加入的数。若为2则为第二种操作。
> $n \leq 10^5$
> $a_i \leq 10^9$

> 输出格式:
> 对于所有$type = 2$的输入，输出一个正整数表示当前的最大值。

样例输入	样例输出
5 1 3 1 2 2 1 4 1 4 2 2	3 4 4

四. 提高巩固

1. 括号匹配

> 给定一个只包含左右小括号的字符串s，若s中所有左右括号恰好能完全匹配，则输出YES，否则输出NO。

> 样例输入:
> 一行一个字符串s。
> s的长度小于10^5

> 样例输出:
> 若能恰好匹配则输出YES，否则输出NO。

样例输入	样例输出
()(())	YES

2. 关系网络

> 有n个人，他们的编号为$1 \sim n$，其中有一些人相互认识，现在x想要认识y，可以通过他所认识的人来认识更多的人（如果x认识y、y认识z，那么x可以通过y来认识z），求出x最少需要通过多少人才能认识y。

> 输入格式:
> 第一行三个正整数n, x, y
> 接下来一个$n \times n$的矩阵a，若$a[i][j] = 1$则表示第i个人认识第j个人，$a[i][j] = 0$则不认识。
> 保证$i = j$时$a[i][j] = 0$。且对于任意i, j，$a[i][j] = a[j][i]$
> $x, y \leq n \leq 100$

> 输出格式:
> 输出一行一个数，表示x认识y最少需要通过的人数。

样例输入	样例输出
5 1 5 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0	2

3. 序列合并

> 有两个长度都是n的序列A和B，在A和B中各取一个数相加可以得到n^2个和，求这n^2个和中最大的n个。

> 输入格式:
> 第一行一个正整数n。
> 第二行n个从大到小的正整数表示序列A。
> 第三行n个从大到小的正整数表示序列B。
> $n \leq 10^5$
> 序列A, B中的数不超过10^9

> 输出格式:
> 一行n个从大到小的正整数，表示和最大的n个数。

样例输入	样例输出
3 5 3 3 6 4 1	11 9 9