**C++ priority\_queue的自定义比较方式**

**另一个：https://blog.csdn.net/qq\_40691051/article/details/102874220?spm=1001.2101.3001.6650.5&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5-102874220-blog-78267716.pc\_relevant\_default&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5-102874220-blog-78267716.pc\_relevant\_default&utm\_relevant\_index=7**

经常需要用[优先队列](https://so.csdn.net/so/search?q=%E4%BC%98%E5%85%88%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "_blank)来进行一些算法的优化，于是今天对优先队列的排序方式做一个小结：

首先，先手写一个小根堆，再分析代码：

#include<queue>

#include<vector>

#include<iostream>

using namespace std;

struct node

{

int x, y;

node(int x,int y):x(x),y(y){}

};

struct cmp

{

bool operator()(node a,node b)

{

if(a.x == b.x) return a.y >= b.y;

else return a.x > b.x;

}

};

int main()

{

priority\_queue<node,vector<node>,cmp> pq; //带有三个参数的优先队列;

for(int i = 1; i <= 5; i++)

for(int j = 1; j <= 5; j++)

pq.push(node(i,j));

while(!pq.empty())

{

cout<<pq.top().x<<" "<<pq.top().y<<endl;

pq.pop();

}

return 0;

}

最近学习STL，发现STL默认都是使用()比较的，默认比较使用less（即'<'运算符），如sort(a,a+n)，默认将数组按照递增的顺序来排序（前面的元素<后面的嘛），但是优先队列的源码比较奇特，虽然按道理使用less比较应该默认是小根堆（即堆顶元素最小），但是priority\_queue<>默认是大根堆的，这是因为优先队列队首指向最后，队尾指向最前面的缘故！每次入队元素进去经排序调整后，优先级最大的元素排在最前面，也就是队尾指向的位置，这时候队首指向优先级最小的元素！所以虽然使用less但其实相当于greater，我们重载运算符的时候比较函数里面写>就相当于<排序方式，这点需要花点时间想想，再来说一说优先队列的这个类型，其实有三个参数：priority\_queue<class Type,class Container,class Compare>，即类型，容器和比较器，后两个参数可以缺省，这样默认的容器就是vector，比较方法是less，也就是默认大根堆，可以自定义写比较方法，但此时若有比较方法参数，则容器参数不可省略！priority\_queue<>的可支持的容器必须是用数组实现的容器，如vector，deque，但不能是list（推荐vector），比较方法可以写结构体重载()运算符，也可以用less，greater这些语言实现了的，但是灵活性不够，建议手写重载结构体，或者——如果不想写比较结构体的话，就将后面的两个参数缺省，直接重载类型的<运算符，如：

#include<queue>

#include<iostream>

using namespace std;

struct node

{

int x, y;

node(int x,int y):x(x),y(y){}

};

bool operator< (node a,node b)

{

if(a.x == b.x) return a.y >= b.y;

else return a.x > b.x;

}

int main()

{

priority\_queue<node> pq; //只传node，但是node结构体的<运算符已被改变

for(int i = 1; i <= 5; i++)

for(int j = 1; j <= 5; j++)

pq.push(node(i,j));

while(!pq.empty())

{

cout<<pq.top().x<<" "<<pq.top().y<<endl;

pq.pop();

}

return 0;

}

也可以写在结构体里面，只用一个参数，但是需要常量修饰符修饰和引用修饰参数，并且函数外面也需要const：

#include<queue>

#include<iostream>

using namespace std;

struct node

{

int x, y;

node(int x, int y):x(x),y(y){}

bool operator< (const node &b) const //写在里面只用一个b，但是要用const和&修饰，并且外面还要const修饰;

{

if(x == b.x) return y >= b.y;

else return x > b.x;

}

};

int main()

{

priority\_queue<node> pq;

for(int i = 1; i <= 5; i++)

for(int j = 1; j <= 5; j++)

pq.push(node(i,j));

while(!pq.empty())

{

cout<<pq.top().x<<" "<<pq.top().y<<endl;

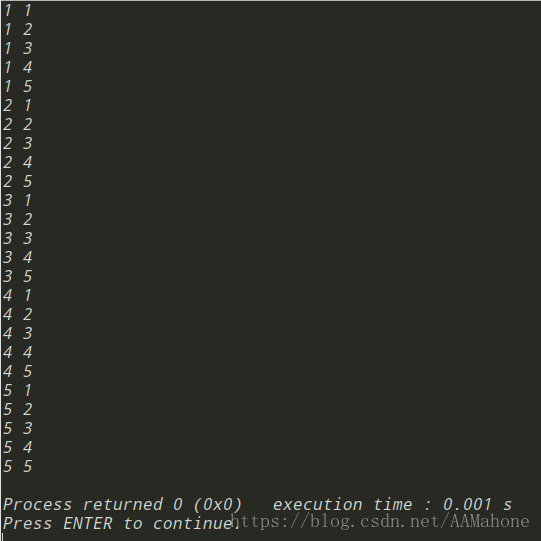
pq.pop();

}

return 0;

}

最后说一句，以上三段代码运行结果都是一样的，为：



# 优先队列（priority\_queue）四种自定义排序方法：

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct cmp{

bool operator () ( int r , int l ){

return r > l;

}

};

struct cmp1{

bool operator ()( int a ,int b ){

return a<b;

}

};

int main(void){

// priority\_queue< int > q;// 默认是 从大到小。

// priority\_queue < int , vector<int> ,less<int> > q;//从大到小

// priority\_queue < int , vector<int>, greater<int> > q; //从小到大，需要vector

// priority\_queue < int , vector<int> , cmp1 > q;//从大到小，需要vector

// priority\_queue < int , vector<int> , cmp > q;//从小到大，需要vector

q.push( 1 );

q.push( 2 );

q.push( 3 );

while( !q.empty() ){

int t =q.top();

q.pop();

printf("%d ",t);

}

return 0;

}