**Policy Based Data Structure or Order Set -> Built-in Functions:**

1. **Constructor**

*#include*<ext/pb\_ds/assoc\_container.hpp>

*#include*<ext/pb\_ds/tree\_policy.hpp>

*using* *namespace* \_\_gnu\_pbds;

*using* *namespace* std;

template <typename T> *using* pbds *=* tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

less -> small to big : without duplicate

less\_equal -> small to big : with duplicate

greater -> big to small : without duplicate

greater\_equal -> big to small : without duplicate

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** |  |
| **Insert(val)** | **insert(value)**: Inserts an element into the tree. |  |
| **Erase(val)** |  **erase(value)**: Removes an element from the tree. |  |
| **Find(val)** |  **find(value)**: Finds an element in the tree. Returns an iterator to the element if it exists, otherwise returns end(). |  |
| **Order\_of\_key(val)** |  **order\_of\_key(value)**: Returns the number of items in the tree that are strictly smaller than value. |  |
| **Find\_by\_order()** |  **find\_by\_order(order)**: Returns an iterator to the order-th smallest element (0-indexed) in the tree. |  |
| **Lower\_boudn(val)** |  **lower\_bound(value)**: Returns an iterator to the first element that is not less than value. |  |
| **Upper\_bound(val)** |  **upper\_bound(value)**: Returns an iterator to the first element that is greater than value. |  |
| **Size()** |  **size()**: Returns the number of elements in the tree. |  |
| **Clear()** |  **clear()**: Removes all elements from the tree. |  |
| **Begin(), end()** |  **begin()**: Returns an iterator to the first element.   **end()**: Returns an iterator to the past-the-end element.   |  |

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** |  |
| Access first value | pbds*<*pair*<*int, int*>>* pb;      pb.insert({1, 3});      pb.insert({1, 4});      pb.insert({1, 5});      pb.insert({2, 4});      pb.insert({2, 5});  যদি আমি এখানে 2 এর under এ প্রথম element টি কি তা জানতে;  int n *=* pb.order\_of\_key({2, -1});  auto it *=* *\**pb.find\_by\_order(n);  print(it.X, " ", it.Y); // output : 2 4 |  |
| Custom pbds sorting | struct custom\_cmp {  bool *operator()*(const  pair<int,int>a , const pair<int,int> b) const      {  *if*(a.first *==* b.first) *return* a.second *>* b.second;  *return* a.first *<* b.first;      }  };  template <typename T>  *using* pbds *=* tree<T, null\_type, custom\_cmp, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;  int main()  {      pbds*<*pair*<*int, int*>>* pb;      pb.insert({1, 3}); pb.insert({1, 4}); pb.insert({1, 5});      pb.insert({2, 4}); pb.insert({2, 5});  *for*(auto c:pb) print(c, ln);      checkmate 0;  } | Output:  1 5  1 4  1 3  2 5  2 4 |
|  |  |  |