C++ ext/pb_ds 用法小结

CodeWaySky

October 31, 2012

Contents

1 运行环境

本文所提到的代码的运行环境为 Ubuntu 10.04 LTS，g++ 版本如下：

```
g++ (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

更高版本的 g++ 的 ext/pb_ds 稍有更改，如果不能编译，请到 ext/pb_ds 目录下的头文件中查找原型

PS: **NOI Linux 下 pb_ds 貌似没有什么改动，以下代码全部能够正常编译执行。**

2 Basic Use

熟悉一下 pb_ds，主要是说明 pb_ds 中对应于 STL 的 std::map 和 std::set 的基本用法

2.1 map

在 pb_ds 中,基本上可以达到 std::map 的所有常用功能,并且可以自定义 map 的内部结构实现,可用的实现有 rb_tree_tag(红黑树),splay_tree_tag(伸展树),ov_tree_tag(ordered_vector tree),个人感觉,总体上还是 rb_tree_tag 性能更好。

下面这段代码展示了 __gnu_pbds::tree 的基本用法

```
#include <cstdio>
#include <cstdlib>
#include <cassert>
#include <string>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<string, int, less<string>, rb_tree_tag> Tree;

int main() {
   Tree t;

   t["abc"] = 6;
   t.insert(make_pair<string, int>("abc", 7));
   t.insert(make_pair<string, int>("ebc", 6));
   t["basicmap"] = -10;

   assert(t.find("basicmap") != t.end());
   t.erase("basicmap");

   for (Tree::const_iterator i = t.begin(); i != t.end(); ++i) {
      printf("%s %d\n", i->first.c_str(), i->second);
   }

   exit(EXIT_SUCCESS);
}
```

2.2 set

set 中,只有一个元素,所以只需要将 typedef 改一下就可以了

注意 typedef 中的 null_mapped_type,在新版本的实现中变为了 null_type,具体可以在 ext/pb_ds/tag_and_trait.hpp 中找到

```
#include <cstdio>
#include <cstdlib>
#include <cassert>
#include <string>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<string, null_mapped_type, less<string>, rb_tree_tag> Tree;

int main() {
   Tree t;

   t.insert("abc");
   t.insert("abc");
```

```
   t.insert("ebc");
   t.insert("basicmap");

   assert(t.find("basicmap") != t.end());
   t.erase("basicmap");

   for (Tree::const_iterator i = t.begin(); i != t.end(); ++i) {
      printf("%s\n", i->c_str());
   }

   exit(EXIT_SUCCESS);
}
```

3 Tree-Like Containers (Trees and Tries)

3.1 区间第 K 大值

pb_ds 中的 tree 非常实用的功能就是能够在树中查询第 K 大的元素是多少，并且可以查询一个给定的元素在树中是第几大的，相关的三个函数如下：

```
inline const_iterator find_by_order(size_type order) const
inline iterator find_by_order(size_type order)
inline size_type order_of_key(const_key_reference r_key) const
```

其中，find_by_order 的作用是查找第 order 大的元素并返回指向它的迭代器，order_of_key 的作用则是查找给定的 r_key 元素在树中是第几大的，两者的编号都从 0 开始。

一个简短的例子（注意 typedef）：

```
#include <cassert>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_mapped_type, less<int>, rb_tree_tag,
   tree_order_statistics_node_update> set_t;

int main() {
   set_t s;

   s.insert(12);
   s.insert(505);
   s.insert(30);
   s.insert(1000);
   s.insert(10000);
   s.insert(100);

   assert(*s.find_by_order(0) == 12);
   assert(*s.find_by_order(1) == 30);
   assert(*s.find_by_order(2) == 100);
   assert(*s.find_by_order(3) == 505);
   assert(*s.find_by_order(4) == 1000);
   assert(*s.find_by_order(5) == 10000);
   assert(s.find_by_order(6) == s.end());
```

```
   assert(s.order_of_key(10) == 0);
   assert(s.order_of_key(12) == 0);
   assert(s.order_of_key(15) == 1);
   assert(s.order_of_key(30) == 1);
   assert(s.order_of_key(99) == 2);
   assert(s.order_of_key(100) == 2);
   assert(s.order_of_key(505) == 3);
   assert(s.order_of_key(1000) == 4);
   assert(s.order_of_key(10000) == 5);
   assert(s.order_of_key(9999999) == 6);

   s.erase(30);

   assert(*s.find_by_order(0) == 12);
   assert(*s.find_by_order(1) == 100);
   assert(*s.find_by_order(2) == 505);
   assert(*s.find_by_order(3) == 1000);
   assert(*s.find_by_order(4) == 10000);
   assert(s.find_by_order(5) == s.end());

   assert(s.order_of_key(10) == 0);
   assert(s.order_of_key(12) == 0);
   assert(s.order_of_key(100) == 1);
   assert(s.order_of_key(505) == 2);
   assert(s.order_of_key(707) == 3);
   assert(s.order_of_key(1000) == 3);
   assert(s.order_of_key(1001) == 4);
   assert(s.order_of_key(10000) == 4);
   assert(s.order_of_key(100000) == 5);
   assert(s.order_of_key(9999999) == 5);

   exit(EXIT_SUCCESS);
}
```

3.2 树的合并

在这里，两棵树能合并的必要条件是合并的树中的所有元素都小于被合并的树中的所有元素，如果不满足这个规则，那么合并的时候会抛出异常。

具体看下面示例。

```
#include <cstdio>
#include <cstdlib>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/exception.hpp>
#include <cassert>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, char, less<int>, rb_tree_tag> map_t;

int main() {
   map_t h0, h1;

   for (int i0 = 0; i0 < 100; ++i0)
      h0.insert(make_pair(i0, 'a'));
   for (int i1 = 0; i1 < 100; ++i1)
      h1.insert(make_pair(1000 + i1, 'b'));
```

```
   h0.join(h1);

   // Check that h0 should now contain all entries, and h1 should be empty.
   assert(h0.size() == 200);
   assert(h1.empty());


   map_t h2, h3;
   h2[1] = 'a';
   h2[3] = 'c';
   h3[2] = 'b';
   // Now perform an illegal join.
   // It is not true that all elements in h2 are smaller than those in
   // h3, nor is it true that they are all larger. Hence, attempting to
   // join h2, and h3 should result in an exception.
   bool exception_thrown = false;
   try {
      h2.join(h3);
   }
   catch (__gnu_pbds::join_error& ) {
      exception_thrown = true;
   }
   assert(exception_thrown);

   // Since the operation was not performed, the tables should be unaltered.
   assert(h2.size() == 2);
   assert(h3[2] == 'b');

   return 0;
}
```

3.3 树的分割

```
#include <string>
#include <cassert>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
using namespace __gnu_pbds;

int main()
{
   // A PATRICIA trie table mapping strings to chars.
   typedef trie<string, char> map_type;

   // A map_type object.
   map_type r;

   // Inserts some entries into r.
   for (int i = 0; i < 100; ++ i)
      r.insert(make_pair(string(i, 'a'), 'b'));

   // Now split r into a different map_type object.

   // larger_r will hold the larger values following the split.
   map_type larger_r;
```

```
   // Split all elements with key larger than 'a'^1000 into larger_r.
   // This is exception free.
   r.split(string(1000, 'a'), larger_r);

   // Since there were no elements with key larger than 'a'^1000, r
   // should be unchanged.
   assert(r.size() == 100);
   assert(r.begin()->first == string(""));

   // Now perform a split which actually changes the content of r.

   // Split all elements with key larger than "aaa" into larger_r.
   r.split(string("aaa"), larger_r);

   assert(r.size() == 4);
   assert(larger_r.begin()->first == string("aaaa"));

   return 0;
}
```

3.4 Trie 搜索前缀

```
#include <cassert>
#include <iostream>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
#include <ext/pb_ds/tag_and_trait.hpp>
#include <string>

using namespace std;
using namespace __gnu_pbds;

typedef trie<string, null_mapped_type, string_trie_e_access_traits<>,
    pat_trie_tag, trie_prefix_search_node_update> trie_type;

// The following helper function takes a trie object and r_key, a
// const reference to a string, and prints all entries whose key
// includes r_key as a prefix.
void print_prefix_match(const trie_type& t, const string& key) {
   cout << "All keys whose prefix matches \"" << key << "\":" << endl;
   pair<trie_type::const_iterator, trie_type::const_iterator> match_range =
      t.prefix_range(key);
   for (trie_type::const_iterator it = match_range.first; it !=
      match_range.second; ++it)
      cout << *it << ' ';
   cout << endl;
}

int main() {
   trie_type t;

   // Insert some entries.
   assert(t.insert("I").second == true);
   assert(t.insert("wish").second == true);
   assert(t.insert("that").second == true);
   assert(t.insert("I").second == false);
```

```
    assert(t.insert("could").second == true);
    assert(t.insert("ever").second == true);
    assert(t.insert("see").second == true);
    assert(t.insert("a").second == true);
    assert(t.insert("poem").second == true);
    assert(t.insert("lovely").second == true);
    assert(t.insert("as").second == true);
    assert(t.insert("a").second == false);
    assert(t.insert("trie").second == true);

    // Now search for prefixes.
    print_prefix_match(t, "");
    print_prefix_match(t, "a");
    print_prefix_match(t, "as");
    print_prefix_match(t, "ad");
    print_prefix_match(t, "t");
    print_prefix_match(t, "tr");
    print_prefix_match(t, "trie");
    print_prefix_match(t, "zzz");

    return 0;
}
```

3.5 重复元素的插入

默认情况下，tree 里面只能插入不重复元素，当需要插入重复元素的时候，可以用不同的实数代替一个整数来多次插入。

下面是 Vijos P1081 动物园的代码。

注意 erase 的时候要用 lower_bound 函数，不能用 find 函数

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/tag_and_trait.hpp>

using namespace std;
using namespace __gnu_pbds;

const int kMaxN = 100010;
const int kMaxM = 50050;

typedef tree<double, null_mapped_type, less<double>, rb_tree_tag,
    tree_order_statistics_node_update> Tree;

struct Ques {
    bool operator < (const Ques &r) const {
        if (x != r.x) {
            return x < r.x;
        } else {
            return y < r.y;
        }
    }

    int x, y;
```

```
   int k;
   int no;
};

int n, m;
int value[kMaxN];
int ans[kMaxN];
Ques ques[kMaxM];
Tree t;

void Work() {
   sort(ques + 1, ques + 1 + m);

   int last_s = 0, last_t = 0;
   int order = 0;
   for (int i = 1; i <= m; i++) {
      int x = ques[i].x;
      int y = ques[i].y;
      int k = ques[i].k;
      int no = ques[i].no;

      if (x > last_t) {
         t.clear();
         order = 0;
         for (int j = x; j <= y; j++) {
            t.insert((double)value[j] + 0.0000005 * order);
            order++;
         }
      } else {
         for (int j = last_s; j < x; j++) {
            t.erase(t.lower_bound((double)value[j]));
         }
         for (int j = last_t + 1; j <= y; j++) {
            t.insert((double)value[j] + 0.0000005 * order);
            order++;
         }
      }

      ans[no] = *t.find_by_order(k - 1);
      last_s = x;
      last_t = y;
   }
}

int main() {
   freopen("zoo.in", "r", stdin);
   freopen("zoo.out", "w", stdout);

   scanf("%d%d", &n, &m);
   for (int i = 1; i <= n; i++) {
      scanf("%d", &value[i]);
   }

   for (int i = 1; i <= m; i++) {
      scanf("%d%d%d", &ques[i].x, &ques[i].y, &ques[i].k);
      ques[i].no = i;
   }

   Work();
   for (int i = 1; i <= m; i++) {
```

```
        printf("%d\n", ans[i]);
    }

    return 0;
}
```

4 Priority Queue

4.1 优先队列的合并

```
#include <functional>
#include <iostream>
#include <cassert>
#include <ext/pb_ds/priority_queue.hpp>

using namespace std;
using namespace __gnu_pbds;

int main() {
    __gnu_pbds::priority_queue<int> even_p, odd_p;

    for (size_t i = 0; i < 10; ++i) {
        even_p.push(2 * i);
        odd_p.push(2 * i + 1);
    }

    assert(even_p.size() == 10);
    assert(even_p.top() == 18);

    cout << "Initial values in even priority queue:" << endl;
    __gnu_pbds::priority_queue<int>::const_iterator it;
    for (it = even_p.begin(); it != even_p.end(); ++it)
        cout <<* it << endl;

    assert(odd_p.size() == 10);
    assert(odd_p.top() == 19);

    cout << "Initial values in odd priority queue:" << endl;
    for (it = odd_p.begin(); it != odd_p.end(); ++it)
        cout <<* it << endl;

    even_p.join(odd_p);

    assert(even_p.size() == 20);
    assert(even_p.top() == 19);

    cout << "After-join values in even priority queue:" << endl;
    for (it = even_p.begin(); it != even_p.end(); ++it)
        cout <<* it << endl;

    assert(odd_p.size() == 0);

    return 0;
}
```

4.2 优先队列内部元素的修改

以 Dijkstra 为例,用 modify 函数进行修改

```
#include <vector>
#include <iostream>
#include <ext/pb_ds/priority_queue.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef std::pair<size_t, size_t> pq_value;

struct pq_value_cmp : public binary_function<pq_value, pq_value, bool>
{
   inline bool
   operator()(const pq_value& r_lhs, const pq_value& r_rhs) const
   {
      return r_rhs.second < r_lhs.second;
   }
};

int main() {
   enum {
      num_vertices = 5,
      graph_inf = 1000
   };

   const size_t a_a_edge_legnth[num_vertices][num_vertices] =
         {
            {0, 5, 3, 7, 6},
            {2, 0, 2, 8, 9},
            {2, 1, 0, 8, 0},
            {1, 8, 3, 0, 2},
            {2, 3, 4, 2, 0}
         };

   typedef __gnu_pbds::priority_queue< pq_value, pq_value_cmp> pq_t;
   pq_t p;
   vector<pq_t::point_iterator> a_it;

   for (size_t i = 0; i < num_vertices; ++i)
      a_it.push_back(p.push(pq_value(i, graph_inf)));
   p.modify(a_it[0], pq_value(0, 0));
   while (!p.empty())
   {
      const pq_value& r_v = p.top();
      const size_t node_id = r_v.first;
      const size_t dist = r_v.second;

      cout << "The distance from 0 to " << node_id << " is " << dist << endl;

      for (size_t neighbor_i = 0; neighbor_i < num_vertices; ++neighbor_i) {
         const size_t pot_dist = dist + a_a_edge_legnth[node_id][neighbor_i];
         if (a_it[neighbor_i] == a_it[0])
            continue;
         if (pot_dist < a_it[neighbor_i]->second)
            p.modify(a_it[neighbor_i], pq_value(neighbor_i, pot_dist));
      }
      a_it[node_id] = a_it[0];
```

```
        p.pop();
    }
    return 0;
}
```