**STL系列之四 heap 堆**

原创 2011年11月14日 01:16:13

* 标签：
* [测试](http://so.csdn.net/so/search/s.do?q=测试&t=blog) /
* [vector](http://so.csdn.net/so/search/s.do?q=vector&t=blog) /
* [算法](http://so.csdn.net/so/search/s.do?q=算法&t=blog) /
* [iterator](http://so.csdn.net/so/search/s.do?q=iterator&t=blog) /
* [algorithm](http://so.csdn.net/so/search/s.do?q=algorithm&t=blog)

下面再介绍STL中与堆相关的4个函数——建立堆make\_heap()，在堆中添加数据push\_heap()，在堆中删除数据pop\_heap()和堆排序sort\_heap()：

头文件 #include <algorithm>

下面的\_First与\_Last为可以随机访问的迭代器（指针），\_Comp为比较函数（仿函数），其规则——如果函数的第一个参数小于第二个参数应返回true，否则返回false。

**建立堆**

make\_heap(\_First, \_Last, \_Comp)

默认是建立最大堆的。对int类型，可以在第三个参数传入greater<int>()得到最小堆。

**在堆中添加数据**

push\_heap (\_First, \_Last)

要先在容器中加入数据，再调用push\_heap ()

**在堆中删除数据**

pop\_heap(\_First, \_Last)

要先调用pop\_heap()再在容器中删除数据

**堆排序**

sort\_heap(\_First, \_Last)

排序之后就不再是一个合法的heap了

有关堆与堆排序的更详细介绍请参阅——《[白话经典算法系列之七 堆与堆排序](http://blog.csdn.net/morewindows/article/details/6709644" \o "白话经典算法系列之七 堆与堆排序)》

下面给出STL中heap相关函数的使用范例：

**[cpp]** [view plain](http://blog.csdn.net/morewindows/article/details/6967409) [copy](http://blog.csdn.net/morewindows/article/details/6967409)

1. //by MoreWindows( http://blog.csdn.net/MoreWindows )
2. #include <cstdio>
3. #include <vector>
4. #include <algorithm>
5. #include <functional>
6. using namespace std;
7. void PrintfVectorInt(vector<int> &vet)
8. {
9. for (vector<int>::iterator pos = vet.begin(); pos != vet.end(); pos++)
10. printf("%d ", \*pos);
11. putchar('\n');
12. }
13. int main()
14. {
15. const int MAXN = 20;
16. int a[MAXN];
17. int i;
18. for (i = 0; i < MAXN; ++i)
19. a[i] = rand() % (MAXN \* 2);
21. //动态申请vector 并对vector建堆
22. vector<int> \*pvet = new vector<int>(40);
23. pvet->assign(a, a + MAXN);
25. //建堆
26. make\_heap(pvet->begin(), pvet->end());
27. PrintfVectorInt(\*pvet);
29. //加入新数据 先在容器中加入，再调用push\_heap()
30. pvet->push\_back(25);
31. push\_heap(pvet->begin(), pvet->end());
32. PrintfVectorInt(\*pvet);
34. //删除数据  要先调用pop\_heap()，再在容器中删除
35. pop\_heap(pvet->begin(), pvet->end());
36. pvet->pop\_back();
37. pop\_heap(pvet->begin(), pvet->end());
38. pvet->pop\_back();
39. PrintfVectorInt(\*pvet);
41. //堆排序
42. sort\_heap(pvet->begin(), pvet->end());
43. PrintfVectorInt(\*pvet);
45. delete pvet;
46. return 0;
47. }

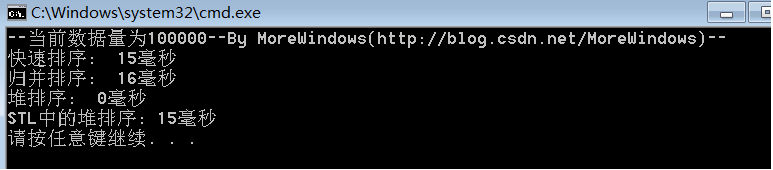
掌握其基本用法后，我们用这个堆排序和[《白话经典算法系列》](http://blog.csdn.net/morewindows/article/category/859207)中的[堆排序](http://blog.csdn.net/morewindows/article/details/6709644)、[快速排序](http://blog.csdn.net/morewindows/article/details/6684558)，[归并排序](http://blog.csdn.net/morewindows/article/details/6678165)来进行个性能测试（Win7 + VS2008 Release下），测试代码如下：

**[cpp]** [view plain](http://blog.csdn.net/morewindows/article/details/6967409) [copy](http://blog.csdn.net/morewindows/article/details/6967409)

1. // by MoreWindows( http://blog.csdn.net/MoreWindows )
2. #include <cstdio>
3. #include <algorithm>
4. #include <ctime>
5. using namespace std;
6. //------------------------快速排序----------------------------
7. void quick\_sort(int s[], int l, int r)
8. {
9. if (l < r)
10. {
11. int i = l, j = r, x = s[l];
12. while (i < j)
13. {
14. while(i < j && s[j] >= x) // 从右向左找第一个小于x的数
15. j--;
16. if(i < j)
17. s[i++] = s[j];
19. while(i < j && s[i] < x) // 从左向右找第一个大于等于x的数
20. i++;
21. if(i < j)
22. s[j--] = s[i];
23. }
24. s[i] = x;
25. quick\_sort(s, l, i - 1); // 递归调用
26. quick\_sort(s, i + 1, r);
27. }
28. }
29. //------------------------归并排序----------------------------
30. //将有二个有序数列a[first...mid]和a[mid...last]合并。
31. void mergearray(int a[], int first, int mid, int last, int temp[])
32. {
33. int i = first, j = mid + 1;
34. int m = mid,   n = last;
35. int k = 0;
37. while (i <= m && j <= n)
38. {
39. if (a[i] < a[j])
40. temp[k++] = a[i++];
41. else
42. temp[k++] = a[j++];
43. }
45. while (i <= m)
46. temp[k++] = a[i++];
48. while (j <= n)
49. temp[k++] = a[j++];
51. for (i = 0; i < k; i++)
52. a[first + i] = temp[i];
53. }
54. void mergesort(int a[], int first, int last, int temp[])
55. {
56. if (first < last)
57. {
58. int mid = (first + last) / 2;
59. mergesort(a, first, mid, temp);    //左边有序
60. mergesort(a, mid + 1, last, temp); //右边有序
61. mergearray(a, first, mid, last, temp); //再将二个有序数列合并
62. }
63. }
64. bool MergeSort(int a[], int n)
65. {
66. int \*p = new int[n];
67. if (p == NULL)
68. return false;
69. mergesort(a, 0, n - 1, p);
70. return true;
71. }
72. //------------------------堆排序---------------------------
73. inline void Swap(int &a, int &b)
74. {
75. int c = a;
76. a = b;
77. b = c;
78. }
79. //建立最小堆
80. //  从i节点开始调整,n为节点总数 从0开始计算 i节点的子节点为 2\*i+1, 2\*i+2
81. void MinHeapFixdown(int a[], int i, int n)
82. {
83. int j, temp;
85. temp = a[i];
86. j = 2 \* i + 1;
87. while (j < n)
88. {
89. if (j + 1 < n && a[j + 1] < a[j]) //在左右孩子中找最小的
90. j++;
92. if (a[j] >= temp)
93. break;
95. a[i] = a[j];     //把较小的子结点往上移动,替换它的父结点
96. i = j;
97. j = 2 \* i + 1;
98. }
99. a[i] = temp;
100. }
101. //建立最小堆
102. void MakeMinHeap(int a[], int n)
103. {
104. for (int i = n / 2 - 1; i >= 0; i--)
105. MinHeapFixdown(a, i, n);
106. }
107. void MinheapsortTodescendarray(int a[], int n)
108. {
109. for (int i = n - 1; i >= 1; i--)
110. {
111. Swap(a[i], a[0]);
112. MinHeapFixdown(a, 0, i);
113. }
114. }
115. const int MAXN = 5000000;
116. int a[MAXN];
117. int b[MAXN], c[MAXN], d[MAXN];
118. int main()
119. {
120. int i;
121. srand(time(NULL));
122. for (i = 0; i < MAXN; ++i)
123. a[i] = rand() \* rand(); //注rand()产生的数在0到0x7FFF之间
125. for (i = 0; i < MAXN; ++i)
126. d[i] = c[i] = b[i] = a[i];
128. clock\_t ibegin, iend;
130. printf("--当前数据量为%d--By MoreWindows(http://blog.csdn.net/MoreWindows)--\n", MAXN);
131. //快速排序
132. printf("快速排序:  ");
133. ibegin = clock();
134. quick\_sort(a, 0, MAXN - 1);
135. iend = clock();
136. printf("%d毫秒\n", iend - ibegin);

139. //归并排序
140. printf("归并排序:  ");
141. ibegin = clock();
142. MergeSort(b, MAXN);
143. iend = clock();
144. printf("%d毫秒\n", iend - ibegin);
146. //堆排序
147. printf("堆排序:  ");
148. ibegin = clock();
149. MakeMinHeap(c, MAXN);
150. MinheapsortTodescendarray(c, MAXN);
151. iend = clock();
152. printf("%d毫秒\n", iend - ibegin);
154. //STL中的堆排序
155. printf("STL中的堆排序: ");
156. ibegin = clock();
157. make\_heap(d, d + MAXN);
158. sort\_heap(d, d + MAXN);
159. iend = clock();
160. printf("%d毫秒\n", iend - ibegin);
161. return 0;
162. }

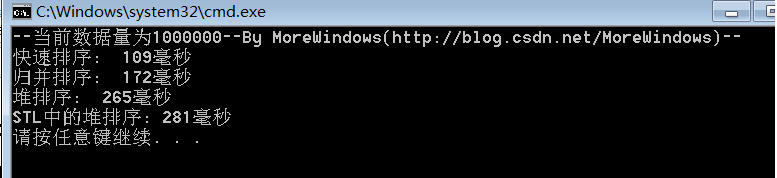
对100000（十万）个数据的测试结果：



对500000（五十万）个数据的测试结果：



对1000000（一百万）个数据的测试结果：



对5000000（五百万）个数据的测试结果：



从中可以看出快速排序的效率确实要比其它同为O(N \* logN)的排序算法要高，而STL中堆操作函数的性能与《[白话经典算法系列之七 堆与堆排序](http://blog.csdn.net/morewindows/article/details/6709644)》一文中堆操作函数的性能是相差无几的。

转载请标明出处，原文地址：<http://blog.csdn.net/morewindows/article/details/6967409>