A Catalogue of Weak-Heap Programs

Stefan Edelkamp[1], Amr Elmasry[2], and Jyrki Katajainen[3]

[1] *TZI, Universität Bremen*
Am Fallturm 1, 28357 Bremen, Germany
[2] *Computer and Systems Engineering Department, Alexandria University*
Alexandria 21544, Egypt
[3] *Department of Computer Science, University of Copenhagen*
Universitetsparken 1, 2100 Copenhagen East, Denmark

Abstract. This report is an electronic appendix to the paper "A Catalogue of Algorithms for Building Weak Heaps" presented at the 23nd International Workshop on Combinatorial Algorithms held in Tamil Nadu in July 2012, and to the journal version of this paper entitled "Weak Heaps Engineered". This report together with an accompanying `tar` ball gives the source code used in the experiments reported in these papers.

Keywords. Data structures, weak heaps, branch mispredictions, cache misses

CPH STL Report 2012-2, August 2012. Revised December 2012, July 2013.

Release date

2013-07-29

Included files

Construction

§ 1 *standard.h++*

```
// Standard weak–heap construction

#include <algorithm> // std::swap
#include <iterator> // std::iterator_traits

namespace standard {

  template <typename index, typename bits>
  index d_ancestor(index j, bits const & r) {
    while ((j & 1) == r[j / 2]) {
      j = j / 2;
    }
    return j / 2;
  }

  template <typename items, typename bits, typename index, typename comparator>
  void join(items a, bits & r, index i, index j, comparator less) {
    if (less(a[j], a[i])) {
      std::swap(a[i], a[j]);
      r[j] = 1 - r[j];
    }
  }

  template <typename items, typename bits, typename comparator>
  void make_weak_heap(items a, items beyond, bits & r, comparator less) {
    typedef typename std::iterator_traits<items>::difference_type index;
    index const n = beyond - a;
    for (index i = 0; i != n; ++i) {
      r[i] = 0;
    }
    for (index j = n - 1; j > 0; --j) {
      index i = d_ancestor(j, r);
      join(a, r, i, j, less);
    }
  }
}
```

§ 2 *instruction_optimized.h++*

```
// Instruction–optimized weak–heap construction

#include <algorithm> // std::swap
#include <cstddef> // std::size_t
#include <iterator> // std::iterator_traits

namespace instruction_optimized {

#if defined(builtin_ctz)

  template <typename bits>
  unsigned int d_ancestor(unsigned int j, bits const &) {
    return j >> (__builtin_ctz(j) + 1);
  }

  template <typename bits>
  unsigned long d_ancestor(unsigned long j, bits const &) {
    return j >> (__builtin_ctzl(j) + 1);
  }

```

```
21   template <typename bits>
22   unsigned long long d_ancestor(unsigned long long j, bits const &) {
23     return j >> (__builtin_ctzll(j) + 1);
24   }
25
26 # elif defined(builtin_popcount)
27
28   template <typename bits>
29   unsigned int d_ancestor(unsigned int j, bits const &) {
30     return j >> (__builtin_popcount(~j & (j - 1)) + 1);
31   }
32
33   template <typename bits>
34   unsigned long d_ancestor(unsigned long j, bits const &) {
35     return j >> (__builtin_popcountl(~j & (j - 1)) + 1);
36   }
37
38   template <typename bits>
39   unsigned long long d_ancestor(unsigned long long j, bits const &) {
40     return j >> (__builtin_popcountll(~j & (j - 1)) + 1);
41   }
42
43 # else
44
45   template <typename index, typename bits>
46   index d_ancestor(index j, bits const &) {
47     while ((j & 1) == 0) {
48       j = j / 2;
49     }
50     return j / 2;
51   }
52
53 # endif
54
55   template <typename items, typename bits, typename index, typename comparator>
56   void join(items a, bits & r, index i, index j, comparator less) {
57     if (less(a[j], a[i])) {
58       std::swap(a[i], a[j]);
59       r[j] = 1;
60     }
61   }
62
63   template <typename items, typename bits, typename comparator>
64   void make_weak_heap(items a, items beyond, bits & r, comparator less) {
65     typedef std::size_t index;
66     index const n = beyond - a;
67     if (n == 0) {
68       return;
69     }
70     for (index i = 0; i != n; ++i) {
71       r[i] = 0;
72     }
73     for (index j = n - 1; j > 0; --j) {
74       index i = d_ancestor(j, r);
75       join(a, r, i, j, less);
76     }
77   }
78 }
```

§ 3 *builtin_ctz.h++*

```
1 # define builtin_ctz
2 # define instruction_optimized builtin_ctz
3
```

```
4 #include "instruction_optimized.h++"
```

§ 4 *builtin_popcount.h++*

```
1 #define builtin_popcount
2 #define instruction_optimized builtin_popcount
3
4 #include "instruction_optimized.h++"
```

§ 5 *builtin_while.h++*

```
1 #define builtin_while
2 #define instruction_optimized builtin_while
3
4 #include "instruction_optimized.h++"
```

§ 6 *branch_optimized.h++*

```
1 // Branch–optimized weak–heap construction
2
3 #include <algorithm> // std::swap
4 #include <climits> // defines CHAR_BIT
5 #include <cstddef> // std::size_t
6 #include <iterator> // std::iterator_traits
7
8 #if defined(__x86_64__)
9
10 #define builtin_popcount
11
12 #elif defined(__i386__)
13
14 #define builtin_ctz
15
16 #else
17
18 #define builtin_while
19
20 #endif
21
22 namespace branch_optimized {
23
24 #if defined(builtin_ctz)
25
26   template <typename bits>
27   unsigned int d_ancestor(unsigned int j, bits const &) {
28     return j >> (__builtin_ctz(j) + 1);
29   }
30
31   template <typename bits>
32   unsigned long d_ancestor(unsigned long j, bits const &) {
33     return j >> (__builtin_ctzl(j) + 1);
34   }
35
36   template <typename bits>
37   unsigned long long d_ancestor(unsigned long long j, bits const &) {
38     return j >> (__builtin_ctzll(j) + 1);
39   }
40
41 #elif defined(builtin_popcount)
42
43   template <typename bits>
44   unsigned int d_ancestor(unsigned int j, bits const &) {
45     return j >> (__builtin_popcount(~j & (j - 1)) + 1);
```

```
46    }
47
48    template <typename bits>
49    unsigned long d_ancestor(unsigned long j, bits const &) {
50      return j >> (__builtin_popcountl(~j & (j - 1)) + 1);
51    }
52
53    template <typename bits>
54    unsigned long long d_ancestor(unsigned long long j, bits const &) {
55      return j >> (__builtin_popcountll(~j & (j - 1)) + 1);
56    }
57
58  # else
59
60    template <typename index, typename bits>
61    index d_ancestor(index j, bits const &) {
62      while ((j & 1) == 0) {
63        j = j / 2;
64      }
65      return j / 2;
66    }
67
68  # endif
69
70  # ifdef IF_FREE
71
72    template <typename items, typename bits, typename index, typename comparator>
73    void join(items a, bits & r, index i, index j, comparator less) {
74      typedef typename std::iterator_traits<items>::value_type element;
75      bool smaller = less(a[j], a[i]);
76      index delta = smaller * (j - i);
77      index k = i + delta;
78      index l = j - delta;
79      element t = a[l];
80      a[i] = a[k];
81      a[j] = t;
82      r[j] = smaller;
83    }
84
85  # else
86
87    template <typename items, typename bits, typename index, typename comparator>
88    void join(items a, bits & r, index i, index j, comparator less) {
89      typedef typename std::iterator_traits<items>::value_type element;
90      bool smaller = less(a[j], a[i]);
91      index k = i;
92      index l = j;
93      if (smaller) {
94        k = j;
95        l = i;
96      }
97      element t = a[l];
98      a[i] = a[k];
99      a[j] = t;
100     r[j] = smaller;
101   }
102
103 # endif
104
105   template <typename items, typename bits, typename comparator>
106   void make_weak_heap(items a, items beyond, bits & r, comparator less) {
107     typedef typename std::iterator_traits<items>::difference_type index;
108     index const n = beyond - a;
109     if (n < 2) {
110       return;
```

```
111     }
112     for (index i = 0; i != n; ++i) {
113       r[i] = 0;
114     }
115     for (std::size_t j = n - 1; j > 0; --j) {
116       auto i = d_ancestor(j, r);
117       join(a, r, i, j, less);
118     }
119   }
120 }
```

§ 7 *floyd.h++*

```
1  // Floyd's heap-construction algorithm for weak heaps
2
3  #include <algorithm> // std::swap
4  #include <iterator> // std::iterator_traits
5
6  namespace floyd {
7
8    template <typename items, typename bits, typename index, typename comparator>
9    void siftdown(items a, bits& r, index j, index n, comparator less) {
10     index k = 2 * j + 1 - r[j];
11     if (k >= n) {
12       return;
13     }
14     while (2 * k + r[k] < n) {
15       k = 2 * k + r[k];
16     }
17     while (k != j) {
18       if (less(a[k], a[j])) {
19         std::swap(a[j], a[k]);
20         r[k] = 1 - r[k];
21       }
22       k /= 2;
23     }
24   }
25
26   template <typename items, typename bits, typename comparator>
27   void make_weak_heap(items a, items beyond, bits& r, comparator less) {
28     typedef typename std::iterator_traits<items>::difference_type index;
29     index const n = beyond - a;
30     for (index i = 0; i != n; ++i) {
31       r[i] = 0;
32     }
33     for (index j = n / 2 - 1; j >= 0; --j) {
34       siftdown(a, r, j, n, less);
35     }
36   }
37 }
```

§ 8 *cache_optimized.h++*

```
1  // Bojesen et al.'s heap-construction algorithm for weak heaps
2
3  #include <algorithm> // std::swap
4  #include <cmath> // ilogb
5  #include <iterator> // std::iterator_traits
6
7  extern int ilogb(double) throw();
8
9  namespace cache_optimized {
10
11   template <typename items, typename bits, typename index, typename comparator>
```

```
12   void siftdown(items a, bits & r, index j, index n, comparator less) {
13     index k = 2 * j + 1 - r[j];
14     if (k >= n) {
15       return;
16     }
17     while (2 * k + r[k] < n) {
18       k = 2 * k + r[k];
19     }
20     while (k != j) {
21       if (less(a[k], a[j])) {
22         std::swap(a[j], a[k]);
23         r[k] = 1 - r[k];
24       }
25       k /= 2;
26     }
27   }
28
29   template <typename items, typename bits, typename comparator>
30   void make_weak_heap(items a, items beyond, bits & r, comparator less) {
31     typedef typename std::iterator_traits<items>::difference_type index;
32
33     index const  n = beyond - a;
34     if (n == 0) {
35       return;
36     }
37     index j = (1 << ilogb(n)) - 1;
38     index const i = j / 2;
39     while (j > i) {
40       r[j] = 0;
41       siftdown(a, r, j, n, less);
42       index z = j;
43       while ((z & 1) == 0) {
44         z /= 2;
45         siftdown(a, r, z, n, less);
46       }
47       --j;
48     }
49     r[0] = 0;
50     siftdown(a, r, index(0), n, less);
51   }
52 }
```

§ 9 *move_optimized.h++*

```
1  // Move-optimized version of the recursive construction
2
3  #include <algorithm> // std::swap
4  #include <iterator> // std::iterator_traits
5
6  namespace move_optimized {
7
8    template <typename items, typename bits, typename indices, typename index,
         typename comparator>
9    void construct(items a, bits r, indices path, index i, index j, index n,
         comparator less) {
10     typedef typename std::iterator_traits<items>::value_type element;
11     if (j >= n) {
12       return;
13     }
14     index d = j;
15     do {
16       construct(a, r, path, d, 2 * d + 1 - r[d], n, less);
17       d = 2 * d + r[d];
18     } while (d < n);
```

```
19      index size = 0;
20      index current = i;
21      do {
22        d /= 2;
23        if (less(a[d], a[current])) {
24          path[size++] = current = d;
25          r[d] = 1 - r[d];
26        }
27      } while (d > j);
28      if (size != 0) {
29        element const v = a[i];
30        a[i] = a[path[size - 1]];
31        while (--size) {
32          a[path[size]] = a[path[size - 1]];
33        }
34        a[path[0]] = v;
35      }
36    }
37
38    template <typename items, typename bits, typename comparator>
39    void make_weak_heap(items a, items beyond, bits r, comparator less) {
40      typedef typename std::iterator_traits<items>::difference_type index;
41      index const n = beyond - a;
42      if (n < 2) {
43        return;
44      }
45      index stack[64];
46      construct(a, r, stack, index(0), index(1), n, less);
47    }
48  }
```

Insertion

§ 10 *repeated_insertion.h++*

```
1  // Constructing a weak heap by repeated insertions
2
3  #include <algorithm> // std::swap
4  #include <iterator> // std::iterator_traits
5
6  namespace repeated_insertion {
7
8    template <typename index, typename bits>
9    index d_ancestor(index j, bits r) {
10     while ((j & 1) == r[j / 2]) {
11       j = j / 2;
12     }
13     return j / 2;
14   }
15
16   template <typename items, typename bits, typename index, typename comparator>
17   void join(items a, bits r, index i, index j, comparator less) {
18     if (less(a[j], a[i])) {
19       std::swap(a[i], a[j]);
20       r[j] = 1 - r[j];
21     }
22   }
23
24   template <typename items, typename bits, typename index, typename comparator>
25   void insert(items a, index j, bits r, comparator less) {
26     r[j] = 0;
27     if ((j & 1) == 0) {
28       r[j / 2] = 0;
29     }
```

```
30      while (j != 0) {
31        index i = d_ancestor(j, r);
32        bool before = r[j];
33        join(a, r, i, j, less);
34        if (r[j] == before) {
35          break;
36        }
37        j = i;
38      }
39    }
40
41    template <typename items, typename bits, typename comparator>
42    void make_weak_heap(items a, items beyond, bits r, comparator less) {
43      typedef typename std::iterator_traits<items>::difference_type index;
44      index const N = beyond - a;
45      for (index i = 0; i < N; i++) {
46        insert(a, i, r, less);
47      }
48    }
49  }
```

§ 11 *bulk_insertion.h++*

```
1  // Constructing a weak heap by regular bulk insertions
2
3  #include <algorithm> // std::swap
4  #include <cmath> // ilogb
5  #include <iterator> // std::iterator_traits
6
7  extern int ilogb(double) throw();
8
9  namespace bulk_insertion {
10
11   unsigned int length;
12   unsigned int n = 0;
13   unsigned int n_prime = 0;
14   unsigned int overall_minimum = 0;
15
16   template <typename index, typename bits>
17   index d_ancestor(index j, bits const & r) {
18     while ((j & 1) == r[j / 2]) {
19       j = j / 2;
20     }
21     return j / 2;
22   }
23
24   template <typename items, typename bits, typename index, typename comparator>
25   void join(items a, bits & r, index i, index j, comparator less) {
26     if (less(a[j], a[i])) {
27       std::swap(a[i], a[j]);
28       r[j] = 1 - r[j];
29     }
30   }
31
32   template <typename items, typename bits, typename index, typename comparator>
33   void sift_down(items a, bits & r, index j, index n, comparator less) {
34     index k = 2 * j + 1 - r[j];
35     if (k >= n) {
36       return;
37     }
38     while (2 * k + r[k] < n) {
39       k = 2 * k + r[k];
40     }
41     while (k != j) {
```

```
42      join(a, r, j, k, less);
43      k = k / 2;
44    }
45  }
46
47  template <typename items, typename bits, typename index, typename comparator>
48  void sift_up(items a, bits& r, index j, comparator less) {
49    while (j != 0) {
50      index i = d_ancestor(j, r);
51      bool before = r[j];
52      join(a, r, i, j, less);
53      if (r[j] == before) {
54        break;
55      }
56      j = i;
57    }
58  }
59
60  template <typename items, typename bits, typename index, typename comparator>
61  void bulk_insert(items a, bits& r, index k, index n, comparator less) {
62    index right = n - 1;
63    index left = std::max(k, right / 2 + 1);
64    while (right > 1 + left) {
65      left = left / 2;
66      right = right / 2;
67      for (index j = right; j >= left; --j) {
68        sift_down(a, r, j, n, less);
69      }
70    }
71    index j = 0;
72    if (right != 0) {
73      j = d_ancestor(right, r);
74      sift_down(a, r, j, n, less);
75    }
76    if (left != 0) {
77      index i = d_ancestor(left, r);
78      sift_down(a, r, i, n, less);
79      sift_up(a, r, i, less);
80    }
81    sift_up(a, r, j, less);
82  }
83
84  template <typename items, typename bits, typename item, typename comparator>
85  void insert(items a, bits& r, item const& x, comparator less) {
86    typedef typename std::iterator_traits<items>::difference_type index;
87    r[n] = 0;
88    a[n] = x;
89    if (n - n_prime > index(ilogb(n + 1)) + 1) {
90      bulk_insert(a, r, n_prime, n + 1, less);
91      n_prime = n + 1;
92      overall_minimum = 0;
93    }
94    else if (less(a[n], a[n_prime])) {
95      std::swap(a[n_prime], a[n]);
96      if (less(a[n_prime], a[0])) {
97        overall_minimum = n_prime;
98      }
99      else {
100       overall_minimum = 0;
101     }
102   }
103   n += 1;
104 }
105
106 template <typename items, typename bits, typename comparator>
```

```
107   void make_weak_heap(items a, items beyond, bits & r, comparator less) {
108     typedef typename std::iterator_traits<items>::difference_type index;
109     length = beyond - a;
110     n = 0;
111     n_prime = 0;
112     for (index i = 0; i != length; ++i) {
113       insert(a, r, a[i], less);
114     }
115     if (n_prime != n) {
116       bulk_insert(a, r, n_prime, n, less);
117     }
118   }
119 }
```

§ 12 *worst_case_insertion.h++*

```
1  // Constructing a weak heap by worst–case constant–time insertions
2
3  #include <algorithm> // std::swap
4  #include <cmath> // ilogb
5  #include <iterator> // std::iterator_traits
6
7  extern int ilogb(double) throw();
8
9  namespace worst_case_insertion {
10
11   unsigned int length;
12   unsigned int n = 0;
13   unsigned int n_prime = 0;
14   unsigned int n_two_primes = 0;
15   unsigned int submersion_minimum = 0;
16   unsigned int overall_minimum = 0;
17   unsigned int state = 0;
18   unsigned int const kappa = 30;
19   unsigned int left;
20   unsigned int right;
21   unsigned int i;
22   unsigned int j;
23   unsigned int k;
24   bool before;
25
26   template <typename items, typename bits, typename index, typename comparator>
27   void join(items a, bits & r, index i, index j, comparator less) {
28     if (less(a[j], a[i])) {
29       std::swap(a[i], a[j]);
30       r[j] = 1 - r[j];
31     }
32   }
33
34   template <typename items, typename bits, typename natural, typename comparator>
35   natural submerge_step(items a, bits & r, natural state, comparator less) {
36     switch (state) {
37     case 1:
38       right = n_two_primes - 1;
39       left = std::max(n_prime, right / 2 + 1);
40     case 2:
41       if (right > 1 + left) {
42         return 3;
43       }
44       else {
45         return 12;
46       }
47     case 3:
48       left = left / 2;
```

```
49        right = right / 2;
50        j = right;
51      case 4:
52        if (j >= left) {
53          return 5;
54        }
55        else {
56          return 2;
57        }
58      case 5:
59        k = 2 * j + 1 - r[j];
60      case 6:
61        if (k < n_two_primes) {
62          return 7;
63        }
64        else {
65          return 11;
66        }
67      case 7:
68        if (2 * k + r[k] < n_two_primes) {
69          return 8;
70        }
71        else {
72          return 9;
73        }
74      case 8:
75        k = 2 * k + r[k];
76        return 7;
77      case 9:
78        if (k != j) {
79          return 10;
80        }
81        else {
82          return 11;
83        }
84      case 10:
85        join(a, r, j, k, less);
86        k = k / 2;
87        return 9;
88      case 11:
89        --j;
90        return 4;
91      case 12:
92        j = 0;
93        if (right != 0) {
94          return 13;
95        }
96        else {
97          return 21;
98        }
99      case 13:
100         j = right;
101     case 14:
102       if ((j & 1) == r[j / 2]) {
103         return 15;
104       }
105       else {
106         return 16;
107       }
108     case 15:
109       j = j / 2;
110       return 14;
111     case 16:
112       j = j / 2;
113       k = 2 * j + 1 - r[j];
```

```
114       if (k < n_two_primes) {
115         return 17;
116       }
117       else {
118         return 21;
119       }
120     case 17:
121       if (2 * k + r[k] < n_two_primes) {
122         return 18;
123       }
124       else {
125         return 19;
126       }
127     case 18:
128       k = 2 * k + r[k];
129       return 17;
130     case 19:
131       if (k != j) {
132         return 20;
133       }
134       else {
135         return 21;
136       }
137     case 20:
138       join(a, r, j, k, less);
139       k = k / 2;
140       return 19;
141     case 21:
142       if (left != 0) {
143         return 22;
144       }
145       else {
146         return 36;
147       }
148     case 22:
149       i = left;
150     case 23:
151       if ((i & 1) == r[i / 2]) {
152         return 24;
153       }
154       else {
155         return 25;
156       }
157     case 24:
158       i = i / 2;
159       return 23;
160     case 25:
161       i = i / 2;
162       k = 2 * i + 1 - r[i];
163       if (k < n_two_primes) {
164         return 26;
165       }
166       else {
167         return 30;
168       }
169     case 26:
170       if (2 * k + r[k] < n_two_primes) {
171         return 27;
172       }
173       else {
174         return 28;
175       }
176     case 27:
177       k = 2 * k + r[k];
178       return 26;
```

```
179     case 28:
180       if (k != i) {
181         return 29;
182       }
183       else {
184         return 30;
185       }
186     case 29:
187       join(a, r, i, k, less);
188       k = k / 2;
189       return 28;
190     case 30:
191       k = i;
192     case 31:
193       if (k != 0) {
194         return 32;
195       }
196       else {
197         return 36;
198       }
199     case 32:
200       i = k;
201     case 33:
202       if ((i & 1) == r[i / 2]) {
203         return 34;
204       }
205       else {
206         return 35;
207       }
208     case 34:
209       i = i / 2;
210       return 33;
211     case 35:
212       i = i / 2;
213       before = r[k];
214       join(a, r, i, k, less);
215       if (r[k] == before) {
216         return 36;
217       }
218       k = i;
219       return 31;
220     case 36:
221       if (j != 0) {
222         return 37;
223       }
224       else {
225         return 0;
226       }
227     case 37:
228         i = j;
229     case 38:
230       if ((i & 1) == r[i / 2]) {
231         return 39;
232       }
233       else {
234         return 40;
235       }
236     case 39:
237       i = i / 2;
238       return 38;
239     case 40:
240       i = i / 2;
241       before = r[j];
242       join(a, r, i, j, less);
243       if (r[j] == before) {
```

```
244         return 0;
245       }
246       j = i;
247       return 36;
248     default:
249       return 0;
250     }
251     return 0;
252   }
253
254   template <typename items, typename bits, typename item, typename comparator>
255   void insert(items a, bits & r, item const & x, comparator less) {
256     a[n] = x;
257     r[n] = 0;
258     if (less(x, a[n_two_primes])) {
259       std::swap(a[n_two_primes], a[n]);
260       if (less(a[n_two_primes], a[overall_minimum])) {
261         overall_minimum = n_two_primes;
262       }
263     }
264     n += 1;
265     if (n - n_two_primes > (unsigned int)(ilogb(n_prime + 1)) + 1) {
266       state = 1;
267       n_prime = n_two_primes;
268       n_two_primes = n;
269       submersion_minimum = n_prime;
270     }
271     for (unsigned int t = 0; t < kappa; ++t) {
272       if (state == 0) {
273         break;
274       }
275       state = submerge_step(a, r, state, less);
276     }
277   }
278
279   template <typename items, typename bits, typename comparator>
280   void make_weak_heap(items a, items beyond, bits & r, comparator less) {
281     length = beyond - a;
282     if (length == 0) {
283       return;
284     }
285     n = 0;
286     n_two_primes = 0;
287     n_prime = 0;
288     state = 0;
289     for (unsigned int i = 0; i != length; ++i) {
290       insert(a, r, a[i], less);
291     }
292     for (unsigned int t = 0; t < kappa * ilogb(1 + n); ++t) {
293       state = submerge_step(a, r, state, less);
294     }
295     if (n != n_two_primes) {
296       n_prime = n_two_primes;
297       n_two_primes = n;
298       state = 1;
299       for (unsigned int t = 0; t < kappa * ilogb(1 + n); ++t) {
300         state = submerge_step(a, r, state, less);
301       }
302     }
303   }
304 }
```

Drivers

§ 13 *char-driver.c++*

```
1  #if ! defined(MAXSIZE)
2
3  #define MAXSIZE (64 * 1024 * 1024)
4
5  #endif
6
7  #include <algorithm> // std::random_shuffle
8  #include <functional> // std::less
9  #include <iostream> // std::cout and std::cerr
10 #include <iterator> // std::iterator_traits
11 #include "timer.h++"
12
13 #if ! defined(NDEBUG)
14
15 template <typename position>
16 bool is_permutation(position first, position beyond) {
17   typedef typename std::iterator_traits<position>::value_type element;
18   std::sort(first, beyond);
19   for (position q = first; q != beyond; ++q) {
20     element e = element(q - first);
21     if (! (*q == e)) {
22       std::cerr << e << ": element missing " << *q << " instead" << std::endl;
23       return false;
24     }
25   }
26   return true;
27 }
28
29 template <typename position, typename bits, typename comparator>
30 bool is_weak_heap(position first, position beyond, bits r, comparator less) {
31   position a = first;
32   typedef typename std::iterator_traits<position>::difference_type index;
33   index n = beyond - first;
34   for (index i = n - 1; i > 0; --i)  {
35     index j = i;
36     while ((j & 1) == r[j / 2]) {
37       j = j / 2;
38     };
39     j = j / 2;
40     if (less(a[i], a[j])) {
41       std::cerr << "X :" << j << ": d-ancestor=" << a[j] << "; "
42                 << i << " current=" << a[i] << std::endl;
43       return false;
44     }
45   }
46   return true;
47 }
48
49 #endif
50
51 #include "algorithm.h++"
52
53 #ifdef MEASURE_COMPARISONS
54
55 long long volatile comparisons = 0;
56
57 template <typename T>
58 class counting_comparator {
59 public:
60
```

```
61    typedef T first_argument_type;
62    typedef T second_argument_type;
63    typedef bool result_type;
64
65    bool operator()(T const & a, T const & b) const {
66      ++comparisons;
67      return a < b;
68    }
69  };
70
71  # endif
72
73  # ifdef MEASURE_MOVES
74
75  long long volatile moves = 0;
76
77  template <typename T>
78  class move_counter {
79  private:
80
81    T element;
82
83  public:
84
85    explicit move_counter()
86      : element(0) {
87    }
88
89    template <typename number>
90    explicit move_counter(number x = 0)
91      : element(x) {
92    }
93
94    move_counter(move_counter const & x) {
95      element = x;
96      moves += 1;
97    }
98
99    move_counter & operator=(move_counter const & other) {
100     element = other.element;
101     moves += 1;
102     return *this;
103   }
104
105   operator T() const {
106     return element;
107   }
108
109   template <typename U>
110   friend bool operator<(move_counter<U> const &, move_counter<U> const &);
111
112   template <typename U>
113   friend bool operator==(move_counter<U> const &, move_counter<U> const &);
114
115 };
116
117 template <typename T>
118 bool operator<(move_counter<T> const & x, move_counter<T> const & y) {
119   return x.element < y.element;
120 }
121
122 template <typename T>
123 bool operator==(move_counter<T> const & x, move_counter<T> const & y) {
124   return x.element == y.element;
125 }
```

```
126
127 #endif
128
129 template <typename position>
130 void generate(position p, position r, char z) {
131   typedef typename std::iterator_traits<position>::value_type element;
132   switch (z) {
133   case 'd':
134     for (position q = p; q != r; ++q) {
135       *q = element((r - 1) - q);
136     }
137     break;
138   case 'i':
139     for (position q = p; q != r; ++q) {
140       *q = element(q - p);
141     }
142     break;
143   case 'r':
144     for (position q = p; q != r; ++q) {
145       *q = element(q - p);
146     }
147     std::random_shuffle(p, r);
148     break;
149   case 's':
150     for (position q = p; q != r; ++q) {
151       if (p == q) {
152         *q = element(q - p);
153       }
154       else {
155         *q = element(r - q);
156       }
157     }
158     break;
159   case 'b':
160     bool t = false;
161     for (position q = p; q != r; ++q) {
162       *q = element(t);
163       t = ! t;
164     }
165     break;
166   }
167 }
168
169 void usage(int argc, char **argv) {
170   std::cerr << "Usage: " << argv[0]
171     << " <N> <'i'ncreasing | 'd'ecreasing | 'r'andom | 's'pecial | 'b'ool>"
172     << std::endl;
173   exit(1);
174 }
175
176 int main(int argc, char** argv) {
177
178 #ifdef MEASURE_MOVES
179
180   typedef move_counter<int> element;
181
182 #elif defined(ELEMENT)
183
184   typedef ELEMENT element;
185
186 #else
187
188   typedef long long element;
189
190 #endif
```

```
191
192 #ifdef MEASURE_COMPARISONS
193
194   typedef counting_comparator<element> C;
195
196 #else
197
198   typedef std::less<element> C;
199
200 #endif
201
202   unsigned int N = 15;
203   char type = 'i';
204   if (argc == 2) {
205     N = atoi(argv[1]);
206     type = 'i';
207   }
208   else if (argc != 3) {
209     usage(argc, argv);
210   }
211   else {
212     N = atoi(argv[1]);
213     type = *argv[2];
214   }
215   if (N < 1 || N > MAXSIZE) {
216     std::cerr << "N out of bounds [1.."
217               << MAXSIZE
218               << "]"
219               << std::endl;
220     usage(argc, argv);
221   };
222   switch (type) {
223   case 'd':
224   case 'i':
225   case 'r':
226   case 's':
227   case 'b':
228     break;
229   default:
230     std::cerr << "Type not in ['d','i','r','s','b']" << std::endl;
231     usage(argc, argv);
232   }
233
234   element* a = new element[MAXSIZE];
235   element* b = a;
236   for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
237     generate(b, b + N, type);
238     b = b + N;
239   }
240
241   bool* r = (bool*) calloc(MAXSIZE, 1);
242   bool* s = r;
243
244   b = a;
245   r = s;
246   timer clock;
247
248 #if defined(MEASURE_MOVES)
249
250   moves = 0;
251
252 #elif defined(MEASURE_COMPARISONS)
253
254   comparisons = 0;
255
```

```
#endif

  clock.start();
  for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t)  {
    NAME::make_weak_heap(b, b + N, r, C());
    b = b + N;
    r = r + N;
  }
  clock.stop();

#if ! defined(NDEBUG)

  b = a;
  r = s;
  for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
    bool ok = is_weak_heap(b, b + N, r, std::less<element>());
    if (! ok) {
      return 1;
    }
    if (type == 'd' || type == 'i' || type == 'r') {
      ok = is_permutation(b, b + N);
      if (! ok) {
        return 2;
      }
    }
    b = b + N;
    r = r + N;
  }

#endif

  unsigned int t = (MAXSIZE / N) * N;

#if defined(MEASURE_COMPARISONS)

  std::cout << N << '\t' << double(comparisons) / double(t) << std::endl;

#elif defined(MEASURE_MOVES)

  std::cout << N << '\t' << double(moves) / double(t) << std::endl;

#else

  double nano_seconds = clock.getElapsedTimeNanoSec();
  std::cout << N << '\t' << nano_seconds / double(t) << std::endl;

#endif

  delete[] a;
  free(s);
  return 0;
}
```

§ 14 *byte-driver.c++*

```
#if ! defined(BIT)

#define BIT unsigned char

#else

#define BIT bool

#endif
```

```
10
11 #if ! defined(MAXSIZE)
12
13 #define MAXSIZE (64 * 1024 * 1024)
14
15 #endif
16
17 #include <algorithm> // std::random_shuffle
18 #include <functional> // std::less
19 #include <iostream> // std::cout and std::cerr
20 #include <iterator> // std::iterator_traits
21 #include "timer.h++"
22 #include <vector>
23
24 #if ! defined(NDEBUG)
25
26 template <typename position>
27 bool is_permutation(position first, position beyond) {
28   typedef typename std::iterator_traits<position>::value_type element;
29   std::sort(first, beyond);
30   for (position q = first; q != beyond; ++q) {
31     element e = element(q - first);
32     if (! (*q == e)) {
33       std::cerr << e << ": element missing " << *q << " instead" << std::endl;
34       return false;
35     }
36   }
37   return true;
38 }
39
40 template <typename position, typename bits, typename comparator>
41 bool is_weak_heap(position first, position beyond, bits const & r, comparator
       less) {
42   position a = first;
43   typedef typename std::iterator_traits<position>::difference_type index;
44   index n = beyond - first;
45   for (index i = n - 1; i > 0; --i)  {
46     index j = i;
47     while ((j & 1) == r[j / 2]) {
48       j = j / 2;
49     };
50     j = j / 2;
51     if (less(a[i], a[j])) {
52       std::cerr << "X :" << j << ": d-ancestor=" << a[j] << "; "
53                 << i << " current=" << a[i] << std::endl;
54       return false;
55     }
56   }
57   return true;
58 }
59
60 #endif
61
62 #include "algorithm.h++"
63
64 #ifdef MEASURE_COMPARISONS
65
66 long long volatile comparisons = 0;
67
68 template <typename T>
69 class counting_comparator {
70 public:
71
72   typedef T first_argument_type;
73   typedef T second_argument_type;
```

```
74    typedef bool result_type;
75
76    bool operator()(T const & a, T const & b) const {
77      ++comparisons;
78      return a < b;
79    }
80  };
81
82  # endif
83
84  # ifdef MEASURE_MOVES
85
86  long long volatile moves = 0;
87
88  template <typename T>
89  class move_counter {
90  private:
91
92    T element;
93
94  public:
95
96    explicit move_counter()
97      : element(0) {
98    }
99
100   template <typename number>
101   explicit move_counter(number x = 0)
102     : element(x) {
103   }
104
105   move_counter(move_counter const & x) {
106     element = x;
107     moves += 1;
108   }
109
110   move_counter & operator=(move_counter const & other) {
111     element = other.element;
112     moves += 1;
113     return *this;
114   }
115
116   operator T() const {
117     return element;
118   }
119
120   template <typename U>
121   friend bool operator<(move_counter<U> const &, move_counter<U> const &);
122
123   template <typename U>
124   friend bool operator==(move_counter<U> const &, move_counter<U> const &);
125
126 };
127
128 template <typename T>
129 bool operator<(move_counter<T> const & x, move_counter<T> const & y) {
130   return x.element < y.element;
131 }
132
133 template <typename T>
134 bool operator==(move_counter<T> const & x, move_counter<T> const & y) {
135   return x.element == y.element;
136 }
137
138 # endif
```

```
139
140 template <typename position>
141 void generate(position p, position r, char z) {
142   typedef typename std::iterator_traits<position>::value_type element;
143   switch (z) {
144   case 'd':
145     for (position q = p; q != r; ++q) {
146       *q = element((r - 1) - q);
147     }
148     break;
149   case 'i':
150     for (position q = p; q != r; ++q) {
151       *q = element(q - p);
152     }
153     break;
154   case 'r':
155     for (position q = p; q != r; ++q) {
156       *q = element(q - p);
157     }
158     std::random_shuffle(p, r);
159     break;
160   case 's':
161     for (position q = p; q != r; ++q) {
162       if (p == q) {
163         *q = element(q - p);
164       }
165       else {
166         *q = element(r - q);
167       }
168     }
169     break;
170   case 'b':
171     bool t = false;
172     for (position q = p; q != r; ++q) {
173       *q = element(t);
174       t = ! t;
175     }
176     break;
177   }
178 }
179
180 void usage(int argc, char **argv) {
181   std::cerr << "Usage: " << argv[0]
182     << " <N> <'i'ncreasing | 'd'ecreasing | 'r'andom | 's'pecial | 'b'ool>"
183     << std::endl;
184   exit(1);
185 }
186
187 int main(int argc, char** argv) {
188
189 #ifdef MEASURE_MOVES
190
191   typedef move_counter<int> element;
192
193 #else
194
195   typedef long long element;
196
197 #endif
198
199 #ifdef MEASURE_COMPARISONS
200
201   typedef counting_comparator<element> C;
202
203 #elif defined(ELEMENT)
```

```
204
205   typedef ELEMENT element;
206
207 # else
208
209   typedef std::less<element> C;
210
211 # endif
212
213   unsigned int N = 15;
214   char type = 'i';
215   if (argc == 1) {
216   }
217   else if (argc == 2) {
218     N = atoi(argv[1]);
219     type = 'i';
220   }
221   else if (argc != 3) {
222     usage(argc, argv);
223   }
224   else {
225     N = atoi(argv[1]);
226     type = *argv[2];
227   }
228   if (N < 1 || N > MAXSIZE) {
229     std::cerr << "N out of bounds [1.."
230               << MAXSIZE
231               << "]"
232               << std::endl;
233     usage(argc, argv);
234   };
235   switch (type) {
236   case 'd':
237   case 'i':
238   case 'r':
239   case 's':
240   case 'b':
241     break;
242   default:
243     std::cerr << "Type not in ['d','i','r','s','b']" << std::endl;
244     usage(argc, argv);
245   }
246
247   element* a = new element[MAXSIZE];
248   element* b = a;
249   for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
250     generate(b, b + N, type);
251     b = b + N;
252   }
253
254   typedef std::vector<BIT> bitvector;
255   std::vector<bitvector> r;
256   r.resize(MAXSIZE / N);
257   for (unsigned int t = 0; t < MAXSIZE / N; ++t) {
258     r[t].resize(N);
259   }
260   b = a;
261   timer clock;
262
263 # if defined(MEASURE_MOVES)
264
265   moves = 0;
266
267 # elif defined(MEASURE_COMPARISONS)
268
```

```
269   comparisons = 0;
270
271 # endif
272
273   clock.start();
274   for (unsigned int t = 0; t < MAXSIZE / N; ++t)  {
275     NAME::make_weak_heap(b, b + N, r[t], C());
276     b = b + N;
277   }
278   clock.stop();
279
280 # if ! defined(NDEBUG)
281
282   b = a;
283   for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
284     bool ok = is_weak_heap(b, b + N, r[t], std::less<element>());
285     if (! ok) {
286       return 1;
287     }
288     if (type == 'd' || type == 'i' || type == 'r') {
289       ok = is_permutation(b, b + N);
290       if (! ok) {
291         return 2;
292       }
293     }
294     b = b + N;
295   }
296
297 # endif
298
299   unsigned int t = (MAXSIZE / N) * N;
300
301 # if defined(MEASURE_COMPARISONS)
302
303   std::cout << N << '\t' << double(comparisons) / double(t) << std::endl;
304
305 # elif defined(MEASURE_MOVES)
306
307   std::cout << N << '\t' << double(moves) / double(t) << std::endl;
308
309 # else
310
311   double nano_seconds = clock.getElapsedTimeNanoSec();
312   std::cout << N << '\t' << nano_seconds / double(t) << std::endl;
313
314 # endif
315
316   delete[] a;
317   return 0;
318 }
```

§ 15 *bit-driver.c++*

```
1  # define BIT bool
2
3  # if ! defined(MAXSIZE)
4
5  # define MAXSIZE (64 * 1024 * 1024)
6
7  # endif
8
9  # include <algorithm> // std::random_shuffle
10 # include <climits> // defines CHAR_BIT
11 # include <functional> // std::less
```

```
12 #include <iostream> // std::cout and std::cerr
13 #include <iterator> // std::iterator_traits
14 #include "timer.h++"
15 #include <vector>
16
17 typedef long long ELEMENT;
18 enum word_size { w = CHAR_BIT * sizeof(ELEMENT) };
19 enum { tail_mask = w - 1 };
20
21 template <typename bits, typename index>
22 void set(bits B, index I,  bool x) {
23   index i = I / w;
24   index lambda = I  & tail_mask;
25   ELEMENT v = ELEMENT(1) << lambda;
26   v = ~v;
27   v = B[i]  & v;
28   ELEMENT u = ELEMENT(x) << lambda;
29   B[i] = v | u;
30 }
31
32 template <typename bits, typename index>
33 bool get(bits B, index I) {
34   index i = I / w;
35   index lambda = I  & tail_mask;
36   ELEMENT v = ELEMENT(1) << lambda;
37   v = B[i]  & v;
38   return (v > ELEMENT(0));
39 }
40
41 #if ! defined(NDEBUG)
42
43 template <typename position>
44 bool is_permutation(position first, position beyond) {
45   typedef typename std::iterator_traits<position>::value_type element;
46   std::sort(first, beyond);
47   for (position q = first; q != beyond; ++q) {
48     element e = element(q - first);
49     if (! (*q == e)) {
50       std::cerr << e << ": element missing " << *q << " instead" << std::endl;
51       return false;
52     }
53   }
54   return true;
55 }
56
57 template <typename position, typename bits, typename comparator>
58 bool is_weak_heap(position first, position beyond, bits r, comparator less) {
59   position a = first;
60   typedef typename std::iterator_traits<position>::difference_type index;
61   index n = beyond - first;
62   for (index i = n - 1; i > 0; --i)  {
63     index j = i;
64     while ((j  & 1) == get(r, j / 2)) {
65       j = j / 2;
66     };
67     j = j / 2;
68     if (less(a[i], a[j])) {
69       std::cerr << "X :" << j << ": d-ancestor=" << a[j] << "; "
70                 << i << " current=" << a[i] << std::endl;
71       return false;
72     }
73   }
74   return true;
75 }
76
```

```
77 #endif
78
79 #include "algorithm.h++"
80
81 #ifdef MEASURE_COMPARISONS
82
83 long long volatile comparisons = 0;
84
85 template <typename T>
86 class counting_comparator {
87 public:
88
89   typedef T first_argument_type;
90   typedef T second_argument_type;
91   typedef bool result_type;
92
93   bool operator()(T const & a, T const & b) const {
94     ++comparisons;
95     return a < b;
96   }
97 };
98
99 #endif
100
101 #ifdef MEASURE_MOVES
102
103 long long volatile moves = 0;
104
105 template <typename T>
106 class move_counter {
107 private:
108
109   T element;
110
111 public:
112
113   explicit move_counter()
114     : element(0) {
115   }
116
117   template <typename number>
118   explicit move_counter(number x = 0)
119     : element(x) {
120   }
121
122   move_counter(move_counter const & x) {
123     element = x;
124     moves += 1;
125   }
126
127   move_counter & operator=(move_counter const & other) {
128     element = other.element;
129     moves += 1;
130     return *this;
131   }
132
133   operator T() const {
134     return element;
135   }
136
137   template <typename U>
138   friend bool operator<(move_counter<U> const &, move_counter<U> const &);
139
140   template <typename U>
141   friend bool operator==(move_counter<U> const &, move_counter<U> const &);
```

```
142
143 };
144
145 template <typename T>
146 bool operator<(move_counter<T> const & x, move_counter<T> const & y) {
147   return x.element < y.element;
148 }
149
150 template <typename T>
151 bool operator==(move_counter<T> const & x, move_counter<T> const & y) {
152   return x.element == y.element;
153 }
154
155 # endif
156
157 template <typename position>
158 void generate(position p, position r, char z) {
159   typedef typename std::iterator_traits<position>::value_type element;
160   switch (z) {
161   case 'd':
162     for (position q = p; q != r; ++q) {
163       *q = element((r - 1) - q);
164     }
165     break;
166   case 'i':
167     for (position q = p; q != r; ++q) {
168       *q = element(q - p);
169     }
170     break;
171   case 'r':
172     for (position q = p; q != r; ++q) {
173       *q = element(q - p);
174     }
175     std::random_shuffle(p, r);
176     break;
177   case 's':
178     for (position q = p; q != r; ++q) {
179       if (p == q) {
180         *q = element(q - p);
181       }
182       else {
183         *q = element(r - q);
184       }
185     }
186     break;
187   case 'b':
188     bool t = false;
189     for (position q = p; q != r; ++q) {
190       *q = element(t);
191       t = ! t;
192     }
193     break;
194   }
195 }
196
197 void usage(int argc, char **argv) {
198   std::cerr << "Usage: " << argv[0]
199     << " <N> <'i'ncreasing | 'd'ecreasing | 'r'andom | 's'pecial | 'b'ool>"
200     << std::endl;
201   exit(1);
202 }
203
204 int main(int argc, char** argv) {
205
206 # ifdef MEASURE_MOVES
```

```
207
208   typedef move_counter<int> element;
209
210 # else
211
212   typedef long long element;
213
214 # endif
215
216 # ifdef MEASURE_COMPARISONS
217
218   typedef counting_comparator<element> C;
219
220 # elif defined(ELEMENT)
221
222   typedef ELEMENT element;
223
224 # else
225
226   typedef std::less<element> C;
227
228 # endif
229
230   unsigned int N = 15;
231   char type = 'i';
232   if (argc == 1) {
233   }
234   else if (argc == 2) {
235     N = atoi(argv[1]);
236     type = 'i';
237   }
238   else if (argc != 3) {
239     usage(argc, argv);
240   }
241   else {
242     N = atoi(argv[1]);
243     type = *argv[2];
244   }
245   if (N < 1 || N > MAXSIZE) {
246     std::cerr << "N out of bounds [1.."
247               << MAXSIZE
248               << "]"
249               << std::endl;
250     usage(argc, argv);
251   };
252   switch (type) {
253   case 'd':
254   case 'i':
255   case 'r':
256   case 's':
257   case 'b':
258     break;
259   default:
260     std::cerr << "Type not in ['d','i','r','s','b']" << std::endl;
261     usage(argc, argv);
262   }
263
264   element* a = new element[MAXSIZE];
265   element* b = a;
266   for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
267     generate(b, b + N, type);
268     b = b + N;
269   }
270
271   ELEMENT* r = new ELEMENT[MAXSIZE / w];
```

```
272   ELEMENT* s = r;
273   b = a;
274
275   timer clock;
276
277 # if defined(MEASURE_MOVES)
278
279   moves = 0;
280
281 # elif defined(MEASURE_COMPARISONS)
282
283   comparisons = 0;
284
285 # endif
286
287   clock.start();
288   for (unsigned int t = 0; t < MAXSIZE / N; ++t)  {
289     NAME::make_weak_heap(b, b + N, r, C());
290     b = b + N;
291     r = r + N / w;
292   }
293   clock.stop();
294
295 # if ! defined(NDEBUG)
296
297   b = a;
298   r = s;
299   for (volatile unsigned int t = 0; t < MAXSIZE / N; ++t) {
300     bool ok = is_weak_heap(b, b + N, r, std::less<element>());
301     if (! ok) {
302       return 1;
303     }
304     if (type == 'd' || type == 'i' || type == 'r') {
305       ok = is_permutation(b, b + N);
306       if (! ok) {
307         return 2;
308       }
309     }
310     b = b + N;
311     r = r + N / w;
312   }
313
314 # endif
315
316   unsigned int t = (MAXSIZE / N) * N;
317
318 # if defined(MEASURE_COMPARISONS)
319
320   std::cout << N << '\t' << double(comparisons) / double(t) << std::endl;
321
322 # elif defined(MEASURE_MOVES)
323
324   std::cout << N << '\t' << double(moves) / double(t) << std::endl;
325
326 # else
327
328   double nano_seconds = clock.getElapsedTimeNanoSec();
329   std::cout << N << '\t' << nano_seconds / double(t) << std::endl;
330
331 # endif
332
333   delete[] a;
334   return 0;
335 }
```

§ 16 *test-driver.c++*

```
1  #if !defined(MAXSIZE)
2  #define MAXSIZE 200
3  #endif
4
5  #include <algorithm> // std::random_shuffle
6  #include <functional> // std::less
7  #include <iostream> // std::cout and std::cerr
8  #include <iterator> // std::iterator_traits
9  #include "timer.h++"
10
11 template <typename position, typename index>
12 void show(position a, index n) {
13   index i = 0;
14   while (i < n) {
15     std::cout << int(a[i]) << " ";
16     ++i;
17   }
18   std::cout << std::endl;
19 }
20 template <typename position>
21 bool is_permutation(position first, position beyond) {
22   typedef typename std::iterator_traits<position>::value_type element;
23   std::sort(first, beyond);
24   for (position q = first; q != beyond; ++q) {
25     element e = element(q - first);
26     if (! (*q == e)) {
27       std::cerr << "n: " << (beyond - first) << " element " << e << " missing, "
             << *q << " instead" << std::endl;
28       return false;
29     }
30   }
31   return true;
32 }
33
34 template <typename position, typename bits, typename comparator>
35 bool is_weak_heap(position first, position beyond, bits r, comparator less) {
36   position a = first;
37   typedef typename std::iterator_traits<position>::difference_type index;
38   index n = beyond - first;
39   for (index i = n - 1; i > 0; --i)  {
40     index j = i;
41     while ((j & 1) == r[j / 2]) {
42       j = j / 2;
43     };
44     j = j / 2;
45     if (less(a[i], a[j])) {
46       std::cerr << "n: " << n << " d-ancestor a[" << j << "]: " << a[j] << "; "
47                 << "current a[" << i << "]: " << a[i] << std::endl;
48       show(r, n);
49       show(a, n);
50       return false;
51     }
52   }
53   return true;
54 }
55
56 #include "algorithm.h++"
57
58 template <typename position>
59 void generate(position p, position r, char z) {
60   typedef typename std::iterator_traits<position>::value_type element;
61   switch (z) {
62   case 'd':
```

```
63      for (position q = p; q != r; ++q) {
64        *q = element((r - 1) - q);
65      }
66      break;
67    case 'i':
68      for (position q = p; q != r; ++q) {
69        *q = element(q - p);
70      }
71      break;
72    case 'r':
73      for (position q = p; q != r; ++q) {
74        *q = element(q - p);
75      }
76      std::random_shuffle(p, r);
77      break;
78    case 's':
79      for (position q = p; q != r; ++q) {
80        if (p == q) {
81          *q = element(q - p);
82        }
83        else {
84          *q = element(r - q);
85        }
86      }
87      break;
88    case 'b':
89      bool t = false;
90      for (position q = p; q != r; ++q) {
91        *q = element(t);
92        t = ! t;
93      }
94      break;
95    }
96  }
97
98  void usage(int argc, char **argv) {
99    std::cerr << "Usage: " << argv[0]
100     << " <n> <'i'ncreasing | 'd'ecreasing | 'r'andom | 's'pecial | 'b'ool>"
101     << std::endl;
102   exit(1);
103 }
104
105 int main(int argc, char** argv) {
106   typedef long long element;
107   typedef std::less<element> C;
108
109   unsigned int n = MAXSIZE;
110   char type = 'r';
111   if (argc == 1) {
112   }
113   else if (argc == 2) {
114     n = atoi(argv[1]);
115     type = 'i';
116   }
117   else if (argc == 3) {
118     n = atoi(argv[1]);
119     type = *argv[2];
120   }
121   else {
122     usage(argc, argv);
123   }
124   if (n < 1 || n > MAXSIZE) {
125     std::cerr << "n out of bounds [1.."
126               << MAXSIZE
127               << "]"
```

```
128               << std::endl;
129     usage(argc, argv);
130   };
131   switch (type) {
132   case 'd':
133   case 'i':
134   case 'r':
135   case 's':
136   case 'b':
137     break;
138   default:
139     std::cerr << "Type not in ['d','i','r','s','b']" << std::endl;
140     usage(argc, argv);
141   }
142
143   element* a = new element[MAXSIZE];
144   bool* r = (bool*) calloc(MAXSIZE, 1);
145   for (unsigned i = 0; i != n; ++i) {
146     for (unsigned int t = 0; t != MAXSIZE; ++t) {
147       generate(a, a + i, type);
148       NAME::make_weak_heap(a, a + i, r, C());
149       bool ok = is_weak_heap(a, a + i, r, C());
150       if (! ok) {
151         return 1;
152       }
153       if (type == 'd' || type == 'i' || type == 'r') {
154         ok = is_permutation(a, a + i);
155         if (! ok) {
156           return 2;
157         }
158       }
159     }
160   }
161
162   std::cout << "Tests passed!" << std::endl;
163
164   delete[] a;
165   free(r);
166   return 0;
167 }
```

Timer

§ 17 *timer.h++*

```
1  /*
2    This high-resolution timer is able to measure the elapsed time with
3    one microsecond accuracy
4
5    Author: Song Ho Ahn (song.ahn@gmail.com) © 2003, 2006
6  */
7
8  #include <sys/time.h>
9
10 class timer {
11 public:
12
13   timer();
14   ~timer();
15
16   void start();
17   void stop();
18   double getElapsedTime();        // get elapsed time in seconds
19   double getElapsedTimeSec();     // same as getElapsedTime
```

```
20   double getElapsedTimeMilliSec();// get elapsed time in milliseconds
21   double getElapsedTimeMicroSec();// get elapsed time in microseconds
22   double getElapsedTimeNanoSec(); // get elapsed time in nanoseconds
23
24 private:
25
26   double startTimeMicroSec;       // starting time in microseconds
27   double endTimeMicroSec;         // ending time in microseconds
28   int    stopped;                 // stop flag
29   timeval startCount;
30   timeval endCount;
31 };
32
33 #include "timer.i++"
```

§ 18 *timer.i++*

```
1  /*
2    This high-resolution timer is able to measure the elapsed time with
3    one micro-second accuracy
4
5    Author: Song Ho Ahn (song.ahn@gmail.com) © 2003, 2006
6  */
7
8  #include <stdlib.h>
9
10 /*
11   constructor
12 */
13 timer::timer() {
14   startCount.tv_sec = startCount.tv_usec = 0;
15   endCount.tv_sec = endCount.tv_usec = 0;
16   stopped = 0;
17   startTimeMicroSec = 0;
18   endTimeMicroSec = 0;
19 }
20
21 /*
22   destructor
23 */
24 timer::~timer() {
25 }
26
27 /*
28   start timer; startCount will be set at this point
29 */
30 void timer::start() {
31   stopped = 0; // reset stop flag
32   gettimeofday( &startCount, NULL);
33 }
34
35 /*
36   stop the timer; endCount will be set at this point
37 */
38 void timer::stop() {
39   stopped = 1; // set timer stopped flag
40   gettimeofday( &endCount, NULL);
41 }
42
43 /*
44   multiply elapsedTimeMicroSec by 1000
45 */
46 double timer::getElapsedTimeNanoSec() {
47   return getElapsedTimeMicroSec() * 1000.0;
```

```
48 }
49
50 /*
51   compute elapsed time in micro-second resolution;
52   other getElapsedTime will call this, then convert to correspond resolution
53 */
54 double timer::getElapsedTimeMicroSec() {
55   if (! stopped) {
56     gettimeofday( &endCount, NULL);
57   }
58   startTimeMicroSec = (startCount.tv_sec * 1000000.0) + startCount.tv_usec;
59   endTimeMicroSec = (endCount.tv_sec * 1000000.0) + endCount.tv_usec;
60   return endTimeMicroSec - startTimeMicroSec;
61 }
62
63 /*
64   divide elapsedTimeMicroSec by 1000
65 */
66 double timer::getElapsedTimeMilliSec() {
67   return getElapsedTimeMicroSec() * 0.001;
68 }
69
70 /*
71   divide elapsedTimeMicroSec by 1000000
72 */
73 double timer::getElapsedTimeSec() {
74   return getElapsedTimeMicroSec() * 0.000001;
75 }
76
77 /*
78   same as getElapsedTimeSec()
79 */
80 double timer::getElapsedTime() {
81   return getElapsedTimeSec();
82 }
```

Makefile

§ 19 *benchmark.mk*

```
1 CXX=g++-4.7
2 CXXFLAGS=-O3 -Wall -std=c++11 -msse4.2 -mabm -DIF_FREE --param case-values-
      threshold=50# -DNDEBUG -march=core2 -mtune=generic -fif-conversion -fif-
      conversion2 -DMEASURE_COMPARISONS -DMEASURE_MOVES
3 SEQUENCE=r
4 INSTRUCTION=builtin_popcount
5
6 header-files:= $(wildcard *.h++)
7 versions:= $(basename $(header-files))
8 correctness-tests:= $(addsuffix .test, $(versions))
9 time-tests:= $(addsuffix .time, $(versions))
10 comp-tests:= $(addsuffix .comp, $(versions))
11 move-tests:= $(addsuffix .move, $(versions))
12 branch-tests:= $(addsuffix .branch, $(versions))
13 cache-tests:= $(addsuffix .cache, $(versions))
14 instruction-tests:= $(addsuffix .count, $(versions))
15
16 N = 1024 32768 1048576 33554432
17
18 $(correctness-tests): %.test : %.h++
19         @cp $*.h++ algorithm.h++
20         @$(CXX) -Wall -std=c++0x -g -DNAME=$* test-driver.c++
21         @./a.out
22 #       rm -f ./a.out
```

```
23
24 $(time_tests): %.time : %.h++
25         @cp $*.h++ algorithm.h++
26         @$(CXX) $(CXXFLAGS) -DNAME="$*" driver.c++
27         @for n in $(N) ; do \
28           ./a.out $$n $(SEQUENCE) ; \
29         done; \
30 #       rm -f ./a.out
31
32 $(comp_tests): %.comp : %.h++
33         @cp $*.h++ algorithm.h++
34         @$(CXX) $(CXXFLAGS) -DMEASURE_COMPARISONS -D$(INSTRUCTION) -DNAME="$*"
               driver.c++
35         @for n in $(N) ; do \
36           ./a.out $$n $(SEQUENCE) ; \
37         done; \
38 #       rm -f ./a.out
39
40 $(move_tests): %.move : %.h++
41         @cp $*.h++ algorithm.h++
42         @$(CXX) $(CXXFLAGS) -DMEASURE_MOVES -DNDEBUG -D$(INSTRUCTION) -DNAME="$*"
               driver.c++
43         @for n in $(N) ; do \
44           ./a.out $$n $(SEQUENCE) ; \
45         done; \
46         rm -f ./a.out
47
48 $(branch_tests): %.branch : %.h++
49         @cp $*.h++ algorithm.h++
50         @for n in $(N) ; do \
51           python branch_mispredictions.py $* $$n ; \
52           rm -f ./a.out ; \
53           rm -f ./cachegrind.out.* ; \
54         done
55
56 $(cache_tests): %.cache : %.h++
57         @cp $*.h++ algorithm.h++
58         @for n in $(N) ; do \
59           python cache_misses.py $* $$n ; \
60           rm -f ./a.out ; \
61           rm -f ./cachegrind.out.* ; \
62         done
63
64 $(instruction_tests): %.count : %.h++
65         @cp $*.h++ algorithm.h++
66         @for n in $(N) ; do \
67           python instruction_count.py $* $$n ; \
68           rm -f ./a.out ; \
69           rm -f ./cachegrind.out.* ; \
70         done
71
72 clean:
73         - rm  -f a.out algorithm.h++ cachegrind.out.* *.o 2>/dev/null
74
75 veryclean: clean
76         - rm -f *~ */*~ 2>/dev/null
77
78 find:
79         find . -type f -print -exec grep $(word) {} \; | less
```

Shell scripts

§ 20 *table2.sh*

```
rm table2.data 2>/dev/null
echo "standard" >>table2.data
cp char-driver.c++ driver.c++
make -f benchmark.mk standard.count >>table2.data
echo "while loop" >>table2.data
make -f benchmark.mk builtin_while.count >>table2.data
echo "builtin-ctz" >>table2.data
make -f benchmark.mk builtin_ctz.count >>table2.data
echo "builtin-popcount" >>table2.data
make -f benchmark.mk builtin_popcount.count >>table2.data
rm cachegrind.out.* 2>/dev/null
```

§ 21 *table3.sh*

```
rm table3.data 2>/dev/null
echo "standard" >>table3.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk standard.time >>table3.data
echo "while loop" >>table3.data
make -f benchmark.mk 'INSTRUCTION=builtin_while' builtin_while.time >>table3.data
echo "builtin-ctz" >>table3.data
make -f benchmark.mk 'INSTRUCTION=builtin_ctz' builtin_ctz.time >>table3.data
echo "builtin-popcount" >>table3.data
make -f benchmark.mk 'INSTRUCTION=builtin_popcount' builtin_popcount.time >>
    table3.data
rm cachegrind.out.* 2>/dev/null
```

§ 22 *table4.sh*

```
rm table4.data 2>/dev/null
echo "builtin-ctz" >>table4.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk builtin_ctz.time >>table4.data
echo "branch optimized" >>table4.data
make -f benchmark.mk 'INSTRUCTION=builtin_ctz' branch_optimized.time >>table4.
    data
rm cachegrind.out.* 2>/dev/null
```

§ 23 *table5.sh*

```
rm table5.data 2>/dev/null
echo "standard" >>table5.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk standard.branch >>table5.data
echo "builtin-ctz" >>table5.data
make -f benchmark.mk builtin_ctz.branch >>table5.data
echo "branch optimized" >>table5.data
make -f benchmark.mk branch_optimized.branch >>table5.data
rm cachegrind.out.* 2>/dev/null
```

§ 24 *table6.sh*

```
rm table6.data 2>/dev/null
echo "standard" >>table6.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk standard.time >>table6.data
echo "alternative" >>table6.data
cp char-driver.c++ driver.c++ # this driver is faster; why?
make -f benchmark.mk floyd.time >>table6.data
rm cachegrind.out.* 2>/dev/null
```

§ 25 *table7.sh*

```
rm table7.data 2>/dev/null
echo "standard---byte array" >>table7.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk standard.time >>table7.data
echo "standard---bit array" >>table7.data
cp bit-driver.c++ driver.c++
make -f benchmark.mk standard.time >>table7.data
cp char-driver.c++ driver.c++
echo "cache optimized" >>table7.data
make -f benchmark.mk cache_optimized.time >>table7.data
rm cachegrind.out.* 2>/dev/null
```

§ 26 *table8.sh*

```
rm table8.data 2>/dev/null
echo "standard---byte array" >>table8.data
cp char-driver.c++ driver.c++
make -f benchmark.mk standard.cache >>table8.data
echo "standard---bit array" >>table8.data
cp bit-driver.c++ driver.c++
make -f benchmark.mk standard_bit_version.cache >>table8.data
cp char-driver.c++ driver.c++
echo "builtin-ctz" >>table8.data
make -f benchmark.mk builtin_ctz.cache >>table8.data
echo "alternative" >>table8.data
cp char-driver.c++ driver.c++
make -f benchmark.mk floyd.cache >>table8.data
echo "cache optimized" >>table8.data
cp char-driver.c++ driver.c++
make -f benchmark.mk cache_optimized.cache >>table8.data
rm cachegrind.out.* 2>/dev/null
```

§ 27 *table9.sh*

```
rm table9.data 2>/dev/null
echo "standard---random" >>table9.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' standard.time >>table9.data
echo "standard---decreasing" >>table9.data
make -f benchmark.mk 'SEQUENCE=d' standard.time >>table9.data
echo "move optimized---random" >>table9.data
cp char-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' move_optimized.time >>table9.data
echo "move optimized---decreasing" >>table9.data
make -f benchmark.mk 'SEQUENCE=d' move_optimized.time >>table9.data
rm cachegrind.out.* 2>/dev/null
```

§ 28 *table10.sh*

```
rm table10.data 2>/dev/null
echo "standard---random" >>table10.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' standard.move >>table10.data
echo "standard---decreasing" >>table10.data
make -f benchmark.mk 'SEQUENCE=d' standard.move >>table10.data
echo "move optimized---random" >>table10.data
cp char-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' move_optimized.move >>table10.data
echo "move optimized---decreasing" >>table10.data
make -f benchmark.mk 'SEQUENCE=d' move_optimized.move >>table10.data
rm cachegrind.out.* 2>/dev/null
```

§ 29 *table11.sh*

```
rm table11.data 2>/dev/null
echo "standard---random" >>table11.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' standard.time >>table11.data
echo "standard---special" >>table11.data
make -f benchmark.mk 'SEQUENCE=s' standard.time >>table11.data
echo "repeated insertions---random" >>table11.data
cp char-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' repeated_insertion.time >>table11.data
echo "repeated insertions---special" >>table11.data
make -f benchmark.mk 'SEQUENCE=s' repeated_insertion.time >>table11.data
rm cachegrind.out.* 2>/dev/null
```

§ 30 *table12.sh*

```
rm table12.data 2>/dev/null
echo "standard---random" >>table12.data
cp byte-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' standard.comp >>table12.data
echo "standard---special" >>table12.data
make -f benchmark.mk 'SEQUENCE=s' standard.comp >>table12.data
echo "move optimized---random" >>table12.data
cp char-driver.c++ driver.c++
make -f benchmark.mk 'SEQUENCE=r' repeated_insertion.comp >>table12.data
echo "move optimized---decreasing" >>table12.data
make -f benchmark.mk 'SEQUENCE=s' repeated_insertion.comp >>table12.data
rm cachegrind.out.* 2>/dev/null
```

§ 31 *table13.sh*

```
rm table13.data 2>/dev/null
cp char-driver.c++ driver.c++
echo "bulk insertion---time" >>table13.data
make -f benchmark.mk bulk_insertion.time >>table13.data
echo "bulk insertion---instructions" >>table13.data
make -f benchmark.mk bulk_insertion.count >>table13.data
echo "bulk insertion---comparisons" >>table13.data
make -f benchmark.mk bulk_insertion.comp >>table13.data
echo "bulk insertion---moves" >>table13.data
make -f benchmark.mk bulk_insertion.move >>table13.data
echo "bulk insertion---mispredictions" >>table13.data
make -f benchmark.mk bulk_insertion.branch >>table13.data
echo "bulk insertion---misses" >>table13.data
make -f benchmark.mk bulk_insertion.cache >>table13.data
rm cachegrind.out.* 2>/dev/null
```

§ 32 *table14.sh*

```
rm table14.data 2>/dev/null
cp char-driver.c++ driver.c++
echo "worst-case constant-time insertion---time" >>table14.data
make -f benchmark.mk worst_case_insertion.time >>table14.data
echo "worst-case constant-time insertion---instructions" >>table14.data
make -f benchmark.mk worst_case_insertion.count >>table14.data
echo "worst-case constant-time insertion---comparisons" >>table14.data
make -f benchmark.mk worst_case_insertion.comp >>table14.data
echo "worst-case constant-time insertion---moves" >>table14.data
make -f benchmark.mk worst_case_insertion.move >>table14.data
echo "worst-case constant-time insertion---mispredictions" >>table14.data
make -f benchmark.mk worst_case_insertion.branch >>table14.data
echo "worst-case constant-time insertion---misses" >>table14.data
```

```
14 make -f benchmark.mk worst_case_insertion.cache >>table14.data
15 rm cachegrind.out.* 2>/dev/null
```