Praktikum Sekundärspeicheralgorithmen
Algorithmik II

Sanders, Dementiev, Schultes, Singler SS 06

http://algo2.iti.uka.de/ioprakt06.php

Experiment 5

Deadline: 13:00 — July 5, 2006

Exercise 1

Extend you list ranking code from Experiment 4 such that it works for *any* node type with defined orderings $<$ and $>$ and corresponding `min_value` sentinels. Make the node type a template parameter of your list ranking function.

Exercise 2

Read Section 3.6.2 from [MSS03] about computing Euler tours for trees in external memory. Based on the proposed techniques, devise an algorithm that computes for each node of an undirected tree its distance from the root of the tree. The algorithm must run in $\mathcal{O}(\text{sort}(n))$ I/Os, where n is the number of nodes in the tree.

Exercise 3

Implement the I/O-efficient Euler tour algorithm. The prototype of the function is:

```
template <class InIterator_, class OutIterator_>
void euler_tourV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output iterator (output graph)
   unsigned M );                          // bytes of main memory to use
```

The input range `[first,beyond)` is the range in a container (e.g. `stxxl::vector`) with the edges of the input tree $T = (V_T, E_T)$. `M` is the number of internal memory bytes the function can use. This memory is to be shared between the STXXL data structures and algorithms you will use. `result` is the output iterator for storing the output edges $(e, \text{succ}(e))$, where $e \in E_T$. The edge $((v, r), \text{succ}((v, r)) =$ `null`$)$ should not be output. It is assumed that the following assignment operation works:

```
typedef typename InIterator_::value_type  T_edge_type;
typedef typename OutIterator_::value_type out_edge_type;

// e and succ_e are of type T_edge_type

*out = out_edge_type(e,succ_e);
```

Find a formula for the exact I/O volume in bytes $\text{Vol}_{\text{et}}(n)$ required by your implementation. Assume that the multiway merge sorting needs only one merging pass (Section 3.2.2 of [MSS03]), such that it makes only $4N/B$ I/Os, where N is the input size in *bytes* and B is the block size in *bytes*. Assume also that the node type of the input tree T is `int`, taking four bytes. Do not forget the I/O volume for reading the input and writing the output. You can find the I/O volume for sequential reading or writing `stxxl::vector` in the tutorial [Dem]. For the calculations, assume that the output iterator is an `stxxl::vector` iterator with record size 16 bytes: `sizeof(out_edge_type)=16`.

Exercise 4

Using the Euler tour implementation from Exercise 3 and your list ranking implementation from Exercise 1, implement the tree rooting, postorder, preorder, subtree size [MSS03] and root distance (Exercise 2) algorithms. The prototypes of the functions are:

```
template <class InIterator_, class OutIterator_>
void tree_rootingV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output (forward edges)
   unsigned M );                          // bytes of main memory to use

template <class InIterator_, class OutIterator_>
void tree_postorderV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output (pairs (node,postorder number) )
   unsigned M );                          // bytes of main memory to use

template <class InIterator_, class OutIterator_>
void tree_preorderV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output (pairs (node,preorder number) )
   unsigned M );                          // bytes of main memory to use

template <class InIterator_, class OutIterator_>
void tree_subtree_sizeV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output (pairs (node,subtree size) )
   unsigned M );                          // bytes of main memory to use

template <class InIterator_, class OutIterator_>
void tree_root_distanceV1(
   InIterator_ first, InIterator_ beyond, // input range (input graph T)
   int root_id,                           // id of the root node
   OutIterator_ result,                   // output (pairs (node,root distance) )
   unsigned M );                          // bytes of main memory to use
```

Try to reuse the code in the functions to minimize the implementation efforts.

Find formulas for the exact I/O volume in bytes $\mathrm{Vol}_{\mathrm{XX}}(n)$ required by these implementation. Make the same assumptions about the I/O volume of sorting and scanning as in Exercise 3. Also, assume that the output iterator is an `stxxl::vector` iterator with the record size 8 bytes. The STXXL priority queue in the list ranking implementation transfers at most $8x \cdot \mathrm{sizeof(ElementType)}$ bytes for all operations, if x elements have been inserted.

The *node* data structure which will be used in the list ranking algorithm is essentially the edge of the original tree T. In the list ranking and other parts of the implementations, you can use the following *union* data structure as the node type (this type maps the (source,destination) pair to a 64-bit node id):

```
union node_edge
{
        int64 node;
        struct {
                int source, destination;
        } edge;

        bool operator < (const node_edge & obj ) const {
                return node < obj.node;
        }

        node_edge() {}
        // Construct from node id
        node_edge(int64 node_): node(node_) {}
        // Construct from edge
        node_edge(int source_, int destination_) {
                edge.source=source_;
                edge.destination=destination_;
        }
};
```

Exercise 5

Measurements:

- Choose $M = 256\,\text{MB}$.
- Use `int` as the node type of the input tree.
- Measure the running time of your Euler tour implementation and the implementations from Exercise 4 for random trees with n nodes, where $n = 2^{20}, 2^{21}, \ldots, 2^{28}$ (powers of 2). You can find the input files with random trees in the directory `/home/dementiev/courses/instances/` as `spf.bin` files. **Copy** them to your local disk for further usage. The files store the undirected edges in binary format as pairs of C++ `int`s. Each edge is stored once either as (u, v) or (u, v). To easily access such file from STXXL you can map it to an `stxxl::vector` as in the example `stxxl_dir/containers/copy_file.cpp`
- Besides the running time, measure the I/O volume and I/O wait time using `stxxl::stats` class (methods `get_read_volume` and `get_write_volume`) [1].
- Try to tune your implementations changing the block sizes.

Write a report that includes *at least* the following:

- A pseudocode of the algorithm for Exercise 2.
- The analyses for Exercise 2, 3, and 4.
- The running time of implementations drawn in a single figure (plot, as usual, time per input item).
- The measured and computed by the formulas I/O volume of the implementations drawn in a single figure (bytes per input item).
- Discussion of the results:
  - Do the computed and measured I/O volumes match (closely)? Why?
  - Which implementations are compute and I/O bound respectively? Why?

Send your source code and your report with figures to `dementiev@ira.uka.de` before the deadline. Also, make an appointment with Roman Dementiev for the defense of your work.

[1] A call of `stats::reset()` of implementations starts a measurement of I/O counter values.

References

[Dem] R. Dementiev. The STXXLlibrary. Documentation and download at `http://stxxl.sourceforge.net/`.

[MSS03] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003.