Task Overview

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. The task is to answer shortest path queries on a changing graph, as quickly as possible.

Our approach

Appropriate data structure + Efficient Implementation

Strategic solutions

- **Optimize the data structure**: adjacency list
  - Each node is identified by a number.
  - All incoming/outgoing nodes of a node are stored in an adjacency array.
  - An index array is used to store the starting position of trunk in incoming/outgoing arrays.
  - Increase the hit rate in CPU Cache

- **Optimize the update actions**
  - Adding a new edge: 2 ideas
    - Preallocate a bucket
      Each 32 nodes has a reserved buffer for adding new nodes, this buffer is called BUCKET. By default, BUCKET size is 8. This means a maximum of 8 nodes can be added in the interval of 32 vertices.
    - Move to the end
      Move the adjacent elements array to the end, and then add a new node at the end of the array.
  - Deleting an edge: Update only the number of incoming/outgoing in the index arrays.

- **Optimize the query processing**
  - **Algorithm**: Breadth First Search (BFS) is used in both directions: from incoming and outgoing array.
  - 2 bitmap arrays are used for remarking travelled incoming/outgoing nodes.
  - Strategies to reduce the search space
    - Predict the next number of nodes in incoming/outgoing queues: calculate the total number of incoming and outgoing in each iteration.
    - Follow the direction having the smaller queue element number.
  - More jobs for multicore/multichip CPUs
    - Parallel the action batch: only for query actions: all consecutive queries will be performed in parallel in multicore/multichip CPUs.
    - Update and delete actions will be handled sequentially.
  - **Graph processing**: Use Cilk Plus to parallel queries.

Implementation

DATA STRUCTURE:

ADD: edge(2 1)

DELETE: edge(3 1)

QUERY:

Global incoming queue and outgoing queue are used for each time searching the shortest path. Then, each searching thread will only use proper in/out queues determined by an interval of global in/out queues.

Each searching thread will also own the proper in/out map slots computed from the global in/out maps. Once the searching is finished, these in/out slots will be cleared for the next search.

Cilk Plus is used for performing queries in parallel.

Experimental result

- **Language**: C
- **Evaluation Machine**:
  - CPUs: 2 x Intel E5-2620v2
  - Main Memory: 20GB
  - Operating System: Ubuntu 14.04 Linux
  - Software: Automake 1.15, gcc5.2.1

<table>
<thead>
<tr>
<th>Size</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.118</td>
</tr>
<tr>
<td>Medium</td>
<td>0.494</td>
</tr>
<tr>
<td>X-Large</td>
<td>1.284</td>
</tr>
<tr>
<td>XX-Large</td>
<td>2.878</td>
</tr>
</tbody>
</table>

Final test result