Eksamen i softwareudvikling

Test af hashtabelmodul

Præsentation af Johannes Serup

8. april 2008

Introduktion

Oprindelige løsning

Nuværende løsning

Fremtidige forbedringer og udvidelser

Benchmark `insert`

Benchmark `find`

Opsummering

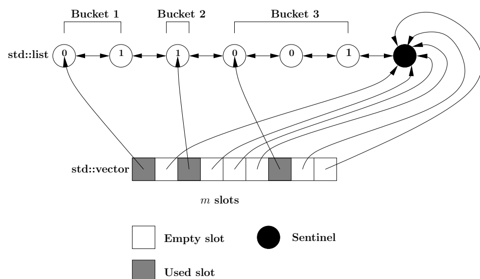
Oprindelige løsning

- Svært at overskue modulet (intet namespace eller opdeling i header og src filer).
- Brugt lang tid på at oversætte modulet. Masser af todos, memory leaks etc.
- Baseret på `vector`-modulet, der heller ikke oversatte.
- Meget omfattende refactoring.
- Konklusion: Skriv modulet fra bunden.

Nuværende løsning

- Anvendelse af bridge pattern.
- Få datastrukturer.
- Opdeling i header og src filer og anvendelse af namespace.
- Inkludering af unittest i hvert modul.

Hashtabel



- n nøgler hashes til m slots i tabellen. Minimere index-range.
- Elementer med kolliderende hashværdier placeres i *buckets*.
- Hver *bucket* højst $\alpha = \frac{n}{m}$ elementer (load-factor).
- Simpelt at understøtte `it++` operation.

Tabeludvidelse

- Når load-factor > 5 fordobles kapaciteten.
- Når load-factor < 2 halveres kapacitet.
- Genbrug hashværdi, når elementer flyttes (beregning af nye index).
- Under flytning deallokeres/allokeres elementer!
- Allokering/deallokering administeres af `std::vector`, `std::list`.

Hashfunktion

- En god hashfunktion opfylder **Simple Uniform Hashing** (lige fordeling af nøgler).
- Universal hashing sikrer denne egenskab.
- En universal klasse af hashfunktioner er givet ved:

  $$\mathscr{H}_{p,m} = \{h_{a,b} : a \in \mathbf{Z}_p^* \text{ and } b \in \mathbf{Z}_p\}$$

  hvor $p > m$ der giver $p(p-1)$ hashfunktioner i $\mathscr{H}_{p,m}$.
- $Pr\{h_{a,b}(k) = h_{a,b}(l)\} \leq \frac{1}{m}$.
- Universal hashing på heltal men ikke strenge. Hashfunktionen skiftes ikke.

Opbevaring af hashværdi

- Et index i tabellen bestemmes ved:

  `index = hash_value MOD capacity`

- Når tabellen udvides/indskrænkes, skal de nye index bestemmes.
- Gemmes hashværdierne, er det kun `MOD`, der skal udregnes.

Kompleksitet

- `INSERT(v)`, `DELETE(it)` tager $O(1)$ tid (dobbelthægtede liste).
- Antag for tabellen at $n = O(m)$ hvor n er elementer og m er slots i tabellen og load-factoren er givet ved $\alpha = \frac{n}{m}$.
- `SEARCH` er proportional med α.
- Da vi har antaget at $n = O(m)$, fås: $\frac{O(m)}{m} = O(1)$.
- Forudsætter **Simple Uniform Hashing**!

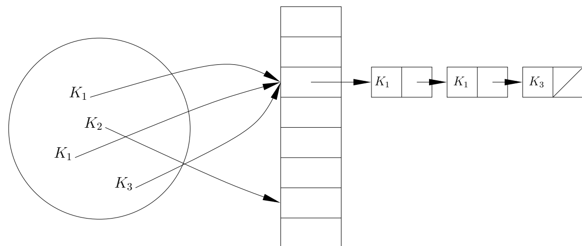
Fremtidige forbedringer og udvidelser

- Skift hashfunktion i forbindelse med udvidelse og sammentrækning.
- Et element består af:

```
pair<pair<K, V>, pair<bool, hash_value> >
```

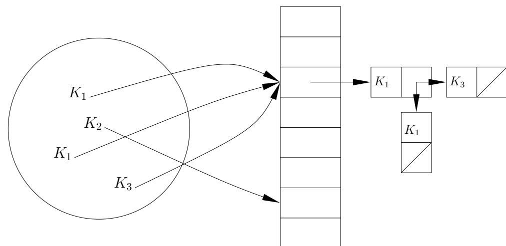
- Opret en elementklasse (`node`) i stedet.
- Udvid med dynamisk hashing i stedet og sammenlign ydelse.
- Slå opbevaring af hashværdi fra/til (plads contra hastighed).
- Implementer `multiset`.

Multiset og equal_range (1)



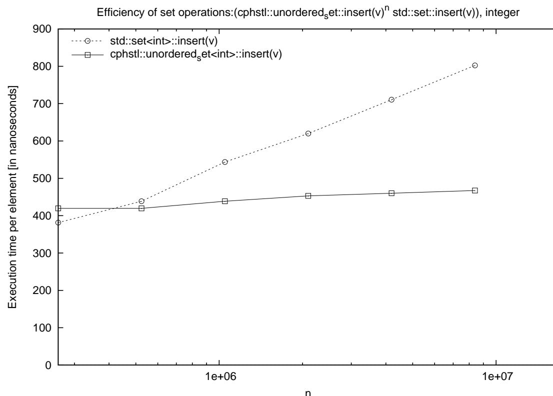
- Sorteret *bucket*.
- equal_range på *bucket*.

Multiset og equal_range (2)

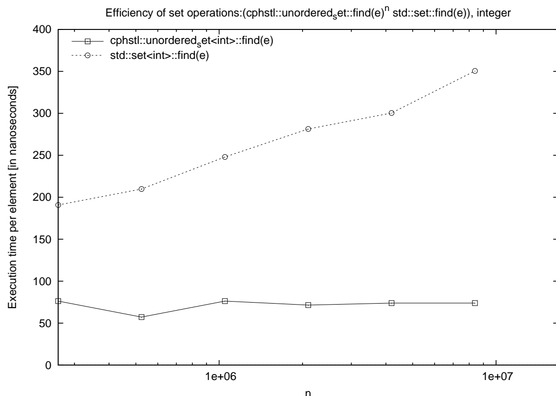


- Tilføj liste.
- equal_range på listen.

Benchmark insert



Benchmark find



Opsummering

- Har ikke nået linear hashing og universal hashfunktioner da refactoring har været meget omfattende.
- Har kostet blod, sved og tårer at implementere iteratorer (`it++` operation).
- Givet praktisk erfaring med anvendelse af design patterns (bridge, iterator).
- Nuværende modul er forhåbentlig væsentligt lettere at udvide og overskue.
- God ydelse.