Size\_T bi = hash\_to\_bucket(m);

mBuckerts[bi].clear();

4.

Size\_t count = 0;

For(auto bucket:mBuckets){

If(bucket.size()>count){

Count = bucket.size();

}

}

Return count;

5.queue<size\_t> q; สำหรับเช็ก least recently used

Size\_t mSize; สำหรับเช็กว่ามีข้อมูลใน cache เท่าไหร่

Size\_t mSizeMax; สำหรับ ระบุขนาดข้อมูล n ที่ cache สามารถเก็บได้

Map<size\_t,char> map\_mem; เก็บข้อมูล char;

B

Auto it = Map\_mem.find(addr);

if(it==map\_mem.end()){

if(mSize==mSizeMax){

map\_mem.erase(q.front());

mSize--;

}

Map\_mem[addr] = get\_memory(addr);

mSize++;

Return get\_memory(addr);

}

Else{

Return it->second;

}

LRUCache(size\_t n){

mSize = 0;

mSizeMax = n;

}

6.

If(it.ptr!=NULL){

Node\* newnode = new Node(element,it.ptr->prev,it.ptr);

It.ptr->prev->next = newnode;

It.ptr->prev = newnode;

mSize++;

return list\_iterator(newnode);

}

Node\* n = mHeader->next;

While(position--){

n = n->next;

}

Node\* ndata = mHeader->next;

While(ndata->data!=data){

Ndata = ndata->next;

}

Ndata->prev->next = ndata->next;

Ndata->next->prev = ndata->prev;

Ndata->prev = n->prev;

Ndata->next = n;

n->prev->next = ndata;

n->prev = ndata;

return list\_Iterator(ndata);

7.

Void pop(){

mData[0] = mData[mSize-1];

mSize--;

size\_t idx = 0;

//fixdown(mData[0]);

int tmp = mData[0];

size\_t c;

while((c = idx\*2+1)<mSize){

if(c+1<mSize&&mLess(mData[c],mData[c=1]))c++;

if(mLess(mData[c],tmp)) break;

mData[idx] = mData[c];

idx = c;

}

mData[idx] = tmp;

}

B

Void changeData(int p, int data){

Int up = (p-1)/2

mData[p] = data;

If(up>=0&&mLess(mData[up],data){

fixup(p);  
}

Else{

fixDown(p);

}

}