Pg. 253

Cormen Notes - Hash tables

Dictionary operations = INSERT, SELECT, DELETE
e.g. No order so not sort, loop etc

many applications only req this. e.g. compiler using a symbol table to translate languages

Hash table is an EFFective data structure for implementing dictionaries

Theoretically, a hash table takes $O(n)$, the same as a linked list

but w/ reasonable assumptions it is $O(1)$ & this is reflected in practice

(for searching an element)

HTs emphasise the useage of direct addressing

Direct addressing is possible when we can allocate AN array which has a position (memory?) for each key

Instead of using a key as an index directly, the array index is computed from the key

DIRECT ADDRESSING

works best when universe of keys is reasonable small

elements are matched to keys from a set $\{0, 1, \ldots, m-1\}$

Assumed that no two elements have same key

Slot k points to an element in the dynamic set w/ key k

Dyamic set may be array or direct address table

Dict ops:
- direct address search: T[k]
- DA insert T[x.key] = x
- DA Del T[x.key] = NIL

each operation only takes O(1)

11.2
HASH TABLES

downside of DA = if universe of keys is large then storing table T of size |U| may be impos or too costy / impractical ↓ comp vs memory of PC

Need to store all keys in universe (waste)
→ only actually need set of keys used K
—
if K much less U
→ hash table reqs much less space than direct access

Can reduce storage to O(|K|)
but retain search time of O(1)

w/ direct adds element w/ key k is stored in slot k

w/ hash, element is stored in slot h(k)

we use the hash func to compute the slot from key k

h maps universe of keys to the slots of the hash function

$$h : U \longrightarrow \{0, 1, \ldots, m-1\}$$

size of M (hash table) is much less than U

h an element w/ key k hashes to slot (~~k~~) h(k)

h(k) is the hash value of k key

hash functions reduce the range of array indices

Array = size M, not |U|

Collision

2 keys may hash to the same slot

Main goal is to avoid collision. do this using good hash func

One idea is to invoke "randomness" hence never paraphrase map to same point

but a hash func must be deterministic

k must always map to the same output

because $|U| > M$ there must be keys that prod the same output $h(k)$

not k still comes from the set of the universe.
i.e. we can select any number k

its just that we dont need to store U in memory

Note there are two comparsions to be made on ~~the~~ DA vs HT:

Speed
→ search, insert & del should all be $O(1)$ for both

however, hash tables rely on good hash func and proper collision resolution

w/o this worst case scenario, hash table = $O(n)$
- $O(1)$ = hash average

Memory
→ hash tables real benefit is in memory
indicies stored goes from $|U|$ in DA to M, size of HT

DA may be impossible for large U

Collision reso by chain

Place elements that hash into same spot into a linked list

Slot contains a pointer to head of list

Insertion into A LL in a slot is O(1)

↳ Assuming element is new & list does need to be checked first

Search worst case is full length of ~~O(LL)~~ list O(len(LL))

Delete
- by default would be same as search
- but if DEC is req, the LL should be doable linked to make this quicker

Analysis of hash w/ chain

worst case scenario is that all keys hash to the same point & have to search through LL to obtain anything
→ O(n) + hash func comp time

The assump is that HT would never be used this way

Ar HT depends on how well the hash func h distributes the keys among the M slots