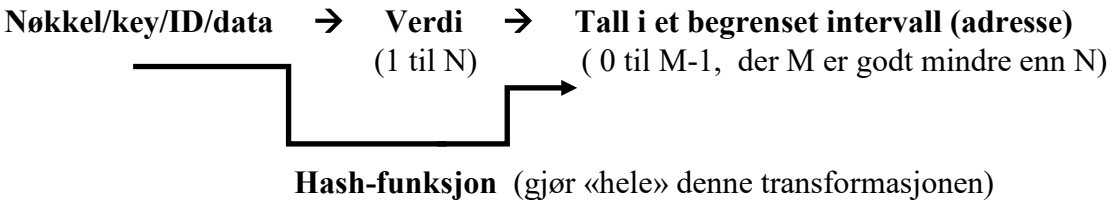
Hashing:

Hashing - **er en aritmetisk/matematisk transformasjoner på en nøkkel/key/ID/data for å få/finne en *direkte adresse i en tabell*.**

- er en god balanse mellom tids- og plassforbruk.

Dvs:



Vi ønsker oss (selvsagt) at hash-funksjonen sprer ulike nøkler jevnt utover hele området fra 0 (null) og opp til M-1. (Men, dette er det ingen triviell sak å lage/få til…..)

Både like og ulike nøkler vil komme til å bli hashet til samme adresse. Da får vi en kollisjon, og dette må løses! Vi skal her se på tre ulike strategier for å løse slike kollisjoner:

- Separate Chaining
- Linear Probing
- Double Hashing

Separate Chaining:

Det brukes en array/vector med Stacker eller LIFO-lister. Når en nøkkel hashes til en indeks i arrayen/vectoren, så settes den bare inn aller først i stacken/listen. Er det derfor N nøkler som skal hashes inn i en array/vector som er M lang, så vil det gjennomsnittlig være N/M elementer/nøkler i hver stack/liste. Greit å bruke denne metoden når N er så stor at det er lite hensiktsmessig i bruke *en* array/vector der det er plass til *alle* nøklene/elementene. (Emnet inneholder ikke noe kode for denne metoden.)

Er det derimot plass til *alle* nøklene i en (stor) array/vector, så er de to neste metodene ofte å foretrekke. Vi setter av/lager en array/vector stor nok til å inneholde *alle* nøklene.

Linear Probing:

Nøkkelen hashes til indeksen der den *bør* legges. *Er det allerede opptatt der, forsøkes den lagt inn i første etterfølgende ledige indeks. Når man arrayens slutt, så startes det med leting forfra igjen.* Er arrayens lengde satt stor nok, så er vi garantert å finne en ledig plass!

Double Hashing:

Den store ulempen med Linear Probing er «clustering». Dvs. sammenklumping av nøkler som har blitt hashet til omtrent de samme indeksene. Dette kan forbedres ved at når en krasj oppstår, så letes det ikke bare i en og en fortløpende indeks etterpå. I stedet får de ulike nøklene litt forskjellige tilleggsverdier som det sjekkes om vedkommende indeks er ledig i stedet. F.eks. at *en* nøkkel sjekker hver andre indeks utover, mens *en annen* sjekker hver sjette. Nøklene får ulike tilleggsverdier ved å kjøre den også igjennom en annen hash-funksjon. Denne kan f.eks. være: 6 - (nøkkel % 6) - som altså blir et tall i intervallet 1 til 6

Rent praktisk for oss:

For enkelhets og eksemplenes del, så lar vi i både **EKS_29_Hashing.cpp** og (eksamens)oppgaver:

- nøklene være bokstaver i intervallet A til Z
- 'verdi' være bokstavens (nøklens) nummer i alfabetet, altså: 1 til 26
- M være et primtall mindre enn 20

(Hadde nøkkelen vært et ord/en tekst, så kunne f.eks. hash-funksjonen ha tatt hver enkelt bokstavs nummer i alfabetet, ganget hver med ulike tall, summert opp alt dette, og til slutt tatt modulo M.)