SŁOWNIKI

Słownik składa się z elementów, które są różne między sobą.

Podstawowe operacje:

Member(x,A) - ma jako argumenty słownik A oraz obiekt x. Zwraca wartość true, gdy obiekt należy do słownika i false
w przeciwnym wypadku.

Insert(x,A) – Czyni x elementem A. Jeśli x już jest w A, to Insert nic nie zmienia.

Delete(x,A) – Usuwa x z A.

MAKENULL(x,A) – Czyni A słownikiem pustym.

Jak można zaimplementować słowniki?

- Jako **listy posortowane**
- Jako **listy nieposortowane**
- Jako **wektory bitowe**
  (przy założeniu, że elementy są z przedziału (1,..,n) lub ze zbioru dla którego można ustalić odpowiedniość z tym przedziałem

HASZOWANIE

- Ważną implementacją słowników jest **haszowanie**.
- **Haszowanie** daje średnio stały czas dostępu do elementu słownika
- W najgorszym przypadku daje czas proporcjonalny do rozmiaru zbioru

- **Tablica mieszająca** (**tablica haszująca)** to struktura danych służąca do przechowywania informacji, w taki sposób, aby możliwy był do nich szybki dostęp.
- Odwołania do przechowywanych obiektów dokonywane są na podstawie *klucza*, który jest identyfikatorem danego obiektu.
- Kluczem może być na przykład ciąg znaków zawierający nazwisko pracownika, a wyszukiwaną informacją jego adres domowy.

- Położenie informacji w tablicy określa się obliczając wartość **funkcji haszującej** (**funkcji mieszającej**) dla danego klucza

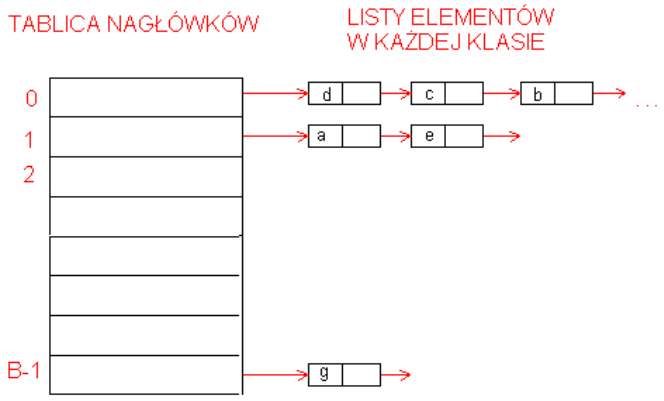
Wyróżniamy dwa rodzaje haszowania:

- **haszowanie otwarte** (ang. open lub external)
- **haszowanie zamknięte** (ang. Closed lub internal)

HASZOWANIE OTWARTE

- Zbiór uniwersalny zostaje podzielony na **skończoną** liczbę klas.
- Jeżeli chcemy mieć B klas ponumerowanych od 0 do B-1, to używamy **funkcji haszującej h**, która każdemu **obiektowi** (kluczowi) *x* przyporządkuje **h(x)**, będące **liczbą** z przedziału od 0 do B-1.
- Wartość **h(x)** jest **numerem klasy**, do której ***x*** należy.

- W tablicy klas znajdują się nagłówki list elementów należących do tych klas.



- Chcemy, aby klasy były w przybliżeniu równych rozmiarów.

- Nie zawsze da się tak dobrać h, aby elementy były równo rozdzielone na klasy.

- Jeżeli w zbiorze uniwersalnym jest ***N*** elementów, to długość listy wynosi średnio ***N/B***

- Jeśli określimy ***N*** i przyjmiemy ***B*** zbliżonej wielkości to każda klasa będzie miała w przybliżeniu 1-2 elementy- mały czas dostępu do elementów

```
const B=65;
typedef char * elementtype;
struct celltype
{
elementtype element;
celltype * next;
};
typedef celltype * position;

class dictionary
{
protected :
   position D[B];//tablica nagłówków list
public:
   dictionary();
   ~dictionary();
   void Makenul();
   bool Member(elementtype x);
   void Insert(elementtype x);
   void Delete(elementtype x);
   int H(elementtype x);
};
```

Zadanie. Zaimplementuj metody klasy dictionary

1. **Konstruktor()** – wypełnia tablice wartościami NULL

2. **Destruktor()** – dla każdej komórki tablicy:
   w pętli usuń elementy odpowiedniej listy

3. **Makenull()** - dla każdej komórki tablicy:
   - w pętli usuń elementy odpowiedniej listy
   - wstaw wartośc NULL do tablicy

4. **Member(x) –** oblicz, w której liście może znajdować się element *x*
   wywołując funkcję *H(x)*
   - przegladaj odpowiednią listę w poszukiwaniu elementu *x*

5. **Insert(x) –** sprawdź, czy element *x* jest w słowniku (wywołując *Member(x)*.
   Jeśli nie ma elementu *x* w słowniku, to:
   -oblicz, do której listy należy wstawic element *x* (wywołując funkcję *H(x))*
   -wstaw element *x* na poczatek odpowiedniej listy

6. **Delete(x) -** oblicz, w której liście może znajdować się element *x*
   wywołując funkcję *H(x) (int bucket=H(x))*
   - jeśli ta lista nie jest pusta to:
   - jesli *x* jest pierwszym elementem tej listy,usuń
   element z początku listy aktualizując *D[bucket]*
   - jesli *x* nie jest pierwszym elementem tej listy,
   przeglądaj listę aby znaleźć element I usuń go

7. **H(x) –** pobierz kod ASCII pierwszego znaku słowa *x (int (x[0]))* modulo
   rozmiar tablicy