ТЕМА 5. Хэширование

5.1. Понятие хеширования

```
H[mas[i].key] = mas[i];    // Запись

x = H[key];                // Извлечение
```

5.2. Схемы хеширования

5.3. Хеш-таблица на основе перемешанной таблицы

(hash – перемешивание)

Функция хеширования:

$$i = Key \% M + C*p.$$

p – номер попытки,

C – шаг перебора позиций

Пример 5.1. Хэш-таблица, использующая алгоритм размещения с линейной адресацией (C = 1).

```
#include <iostream>
using namespace std;

void sv_add(int key, int m, int* H) {
    int i = abs(key % m);
      while (H[i] != -1) {
            i++;
            if (i == m) i = 0;
        }
    H[i] = key;
}
```

```
int sv_seach(int key, int m, int *H) {
    int i = abs(key % m);
    while (H[i] != -1)
    {
        if (H[i] == key) return i;
           i++;
           if (i == m) i = 0;
    }
    return -1;
}
```

```
void main()  {
    const int n = 8; // Число элементов в массиве
    int mas[n] = { 5, 15, 3, 10, 125, 333, 11, 437 };
    const int m = 10; // Число эл-тов в хеш-таблице
    int H[m];    int i;
for (i = 0; i < m; i++) H[i] = -1; // Позиции не заняты
    for (i = 0; i < n; i++) sv_add(mas[i], m, H);
    // Поиск элемента с ключом 333
    int key = 333;
    int k = sv_seach(key, m, H);
      if (k == -1) cout << "Item not found" << endl;
        else cout << H[k] << endl;  }
```

5.4. Хеш-таблицы с квадратичной и произвольной адресацией

Квадратичная адресация:

$$i = Key \,\%\, M + p^2$$

(p – номер попытки).

Произвольная адресация:

$$i = Key \,\%\, M + r_p$$

(r – заранее сгенерированный массив случайных чисел; p – номер попытки).

5.5. Хеш-таблица с двойным хешированием

Алгоритм метода:

1. Найти позицию элемента в хеш-таблице по формуле $i = key \% m$.
2. Если i-я ячейка свободна, то перейти к п. 6.
3. Вычислить $c = 1 + (key \% (m - 2))$.
4. Найти позицию элемента в хеш-таблице по формуле $i = i - c$ (если $i < 0$, то $i = i + m$).
5. Если i-я ячейка занята, то перейти к п. 4.
6. Вставить элемент в найденную позицию.

5.6. Хеш-таблица на основе связанных списков

Пример 19.2. Хэш-таблица, использующая алгоритм размещения на основе связанных списков.

```
# include <iostream>
using namespace std;

struct TNode {  // Описание элемента стека
int inf;    // Информационная часть структуры
    TNode* a;   // Адресная часть структуры
};

 TNode** sv_create(int m) {
     TNode** H = new  TNode * [m];
    for (int i = 0; i < m; i++) H[i] = nullptr;
    return H;
}
```

```
void sv_add(int inf, int m,  TNode** H) {
     TNode* spt = new  TNode;
    spt->inf = inf;
    int i = abs(inf % m);
    if (H[i]) { spt->a = H[i]; H[i] = spt; }
    else  { H[i] = spt; spt->a = nullptr; }
}
```

```
TNode* sv_seach(int inf, int m,  TNode** H) {
    int i = abs(inf % m);
     TNode* spt = H[i];
    while (spt)
    {
        if (spt->inf == inf) return spt;
        spt = spt->a;
    }
    return nullptr;
}
```

```
void sv_delete(int m,  TNode** H) {
     TNode* spt, * sp;
    for (int i = 0; i < m; i++)
    {
         sp = H[i];
        while (sp)  {
            spt = sp;
            sp = sp->a;
            delete spt;
        }
    }
    delete[]H;
}
```

```
void main() {
    int n = 8; // Число элементов  в массиве
    int mas[] = { 5, 15, 3, 10, 125, 333, 11, 437 };
    int m = 10; // Число эл-тов  в хеш-таблице
     TNode** H = sv_create(m);
    for (int i = 0; i < n; i++) sv_add(mas[i], m, H);
     int key = 333;
     TNode* p = sv_seach(key, m, H);
     if (!p)  cout << "Item not found" << endl;
     else cout << p->inf << endl;
    sv_delete(m, H);
}
```

5.7. Метод блоков

Используется массив одномерных массивов одинакового размера (блоков).

Вначале находится номер блока, в который помещается элемент. Если блок переполнен, то элемент помещается в специальный блок переполнения. Поиск ведется в найденном блоке и в блоке переполнения.