**Дополнительно задание на тему «Простейшие структуры данных»**

Ответьте *РАЗВЕРНУТО* на следующие вопросы. Документ с ответами загрузите в свой репозиторий, после чего создайте коммит с названием «C3 additional complete». Напишите мне в дискорде о выполнении, чтобы получить баллы.

1. Что такое Хэш-таблица? Какие способы реализации Хэш-таблиц вам известны?

Это структура данных, позволяющая хранить пары ключей и их значений. Такая структура данных помогает экономить память, храня не сами значения, а ключи, при этом сохраняя возможность быстрого доступа к ячейке (О(1)). Она эффективна для создания словаря (словарь – это структура данных, хранящая пары ключ-значение с доступом по ключу в среднем за О(1)).

Хэш-функция: функция h(x), которая переводит аргумент x в некоторое конечное множество R, являющееся подмножеством Z. Аргумент может иметь любой тип. Хэш-функции не предназначены для обратного преобразования. Хэш-функция подвержена коллизиям (ситуации, когда значение хэш-функции на двух разных аргументах совпадает). Реализации, применяемые для разрешения коллизии в хэш-таблицах:

* Хранение элементов с одинаковым хэшем в односвязных списках (в ячейках хранится не само значение, не сам ключ, а указатель на односвязный список, который будет хранить все значения)
* Смещение (использование для хранения последующих свободных ячеек. K – натуральное число, смещение)

Основные операции: добавление, удаление и поиск элементов.

Просто удалять элемент безвозвратно нельзя. Ведь если так сделать, то при попытке его найти, т.е. значение хеш-функции которого совпадет с ее значением у удаленного элемента), сразу наткнемся на пустую ячейку. А это значит, что такого элемента и не было никогда, но он есть, просто лежит где-то дальше в массиве. Это основная сложность использования второго метода решения коллизий.

2. Что такое граф? Какие способы реализации графа вам известны?

**Граф** — это абстрактное представление множества объектов и связей между ними. Графом называют пару (V, E), где V это множество **вершин**, а E множество пар, каждая из которых представляет собой связь (эти пары называют **рёбрами**). Путь в графе – это множество ребер, которые приводят из одной вершин в другую. Цикл – это путь, начинающийся и заканчивающийся в одной и той же вершине. Связный граф – граф, в котором существует путь из любой вершины в любую другую.

Граф может быть **ориентированным** или **неориентированным**. В ориентированном графе, связи являются направленными (то есть пары в E являются упорядоченными, например пары (a, b) и (b, a) это две разные связи). В свою очередь в неориентированном графе, связи ненаправленные, и поэтому если существует связь (a, b) то значит что существует связь (b, a).

Взвешенный граф – граф, для которого определена функция весов w(E) -> R.

Деревом называется связный граф без циклов.

Степень вершины – число вершин, с которыми данная варшина связана.

Реализации:

1. **Матрица смежности**

Этот способ является удобным для представления **плотных** графов, в которых количество рёбер (|E|) примерно равно количеству вершин в квадрате (|V|2).   
В данном представлении заполняется матрица размером |V| x |V| следущим образом:   
A[i][j] = 1 (Если существует ребро из i в j)  
A[i][j] = 0 (Иначе)  
Данный способ подходит для ориентированных и неориентированных графов. Для неориентированных графов матрица A является симметричной (то есть A[i][j] == A[j][i). Благодаря этому свойству можно сократить почти в два раза использование памяти, храня элементы только в верхней части матрицы, над главной диагональю)  
С помощью данного способа представления, можно быстро проверить есть ли ребро между вершинами v и u, просто посмотрев в ячейку A[v][u].  
С другой стороны этот способ очень громоздкий, так как требует O (|V|2) памяти для хранения матрицы.

1. **Списки смежности**

Данный способ представления больше подходит для разреженных графов, то есть графов у которых количество рёбер гораздо меньше чем количество вершин в квадрате (|E| << |V|2).  
В данном представлении используется массив Adj содержащий |V| списков. В каждом списке Adj[v] содержатся все вершины u, так что между v и u есть ребро. Память требуемая для представления равна O (|E| + |V|) что является лучшим показателем, чем матрица смежности для разреженных графов.  
Главный недостаток этого способа представления в том, что нет быстрого способа проверить существует ли ребро (u, v).

1. **Список ребер**

Под списком вершин можно указать просто их количество, потому что по сути

все вершины будем кодировать числами от 0 до n минус 1, где n - количество вершин, поэтому достаточно будет показывать их количество. список ребер надо бы предоставить переписка пар чисел.

Для графа с Е ребрами такой способ хранения требует порядка Е памяти, но при этом основные операции — проверка наличия ребра между двумя вершинами и перебор вершин, смежных с данной — выполняются также за время порядка Е, поскольку требуют полного перебора всех списков ребер.

Иногда удобно хранить каждое ребро неориентированного графа дважды: как [u, v] и как [v, u].

3. Какие алгоритмы на графах вам известны? Опишите их.

* *Обход (поиск) в глубину*

Из названия этого метода обхода графа ясно, что в процессе поиска мы идем «вглубь» графа настолько, насколько возможно. Следуя алгоритму, мы последовательно обойдем все вершины графа, которые доступны из начальной вершины. Если ребро ведет в не пройдённую до этого момента вершину, то алгоритм запускается с нее. В случае если ребер, которые ведут в не рассмотренную вершину, больше нет, то происходит возврат назад. В этом алгоритме мы не стремимся найти всех потомков рассматриваемой вершины, а берем только одного потомка, ищем его потомка и т.д. Для реализации используется стэк.

* *Обход (поиск) в ширину*

Алгоритм позволяет найти кратчайший (содержащий наименьшее число ребер) путь из одной вершины графа до всех остальных вершин. В нем сначала посещаются все вершины, смежные с текущей, а затем их потомки. Другими словами, мы обходим в порядке уровней, где мы посещаем каждый узел на уровне прежде чем перейти на следующий уровень. Для реализации используется очередь.