“subvector”

```
struct subvector {
     int *mas;
     unsigned int top;
     unsigned int capacity;
};

bool init(subvector *qv);                  //инициализация пустого недовектора

bool push_back(subvector *qv, int d);      //добавление элемента в конец недовектора
                                           //с выделением дополнительной памяти при необходимости

int pop_back(subvector *qv);               //удаление элемента с конца недовектора

bool resize(subvector *qv, unsigned int new_capacity); //увеличить емкость недовектора

void shrink_to_fit(subvector *qv);         //очистить неиспользуемую память

void clear(subvector *qv);                 //очистить содержимое недовектора, занимаемое место
                                           //при этом не меняется

void destructor(subvector *qv);            //очистить всю используемую память, инициализировать
                                           //недовектор как пустой

bool init_from_file(subvector *qv, char *filename); //инициализировать недовектор из файла
```

“subforwardlist”

```
struct subforwardlist {
     int data;
     subforwardlist* next;
};

bool init(subforwardlist *sfl);          //инициализация пустого недосписка

bool push_back(subforwardlist *sfl, int d);      //добавление элемента в конец недосписка

int pop_back(subforwardlist *sfl);      //удаление элемента с конца недосписка

bool push_forward(subforwardlist *sfl, int d);   //добавление элемента в начало недосписка

int pop_forward(subforwardlist *sfl);  //удаление элемента из начала недосписка

bool push_where(subforwardlist *sfl, unsigned int where, int d); //добавление элемента с поряд-
                                                                 //ковым номером where

bool erase_where *sfl, unsigned int where);      //удаление элемента с порядковым номером where

unsigned int size(subforwardlist *sfl);          //определить размер недосписка

void clear(subforwardlist *sfl);         //очистить содержимое недосписка

bool init_from_file(subforwardlist *sfl, char* filename); //инициализировать недосписок из файл
```

"subset"

```
struct subset_node { // дерево поиска
    int key;
    subset_node *left;
    subset_node *right;
}; // можете добавлять дополнительные поля и структуры данных (например, для BFS)

bool init(subset_node **sn);              //инициализация пустого дерева

bool insert(subset_node **sn, int k);     //добавление элемента в дерево, дубли игнорировать

bool remove(subset_node **sn, int k);     //удаление элемента из дерева

subset_node* find(subset_node *sn, int k);        //поиск элемента в дереве

unsigned int size(subset_node *sn);       //количество элементов в дереве

unsigned int height(subset_node *sn);     //высота дерева

void destructor(subset_node *sn);         //очистить всю используемую память

int* DFS (subset_node *sn);    //обход в глубину, возвращает указатель на массив

int* BFS (subset_node *sn);    //обход в ширину, возвращает указатель на массив

subset_node* merge(subset_node *sn1, subset_node *sn2); // слияние двух деревьев
```