```
  }
  return a + rand() % (b - a + 1);
}

static idx_t rng_state = SEED;

static inline idx_t
rand_idx(idx_t a, idx_t b) {
  rng_state ^= rng_state << 13;
  rng_state ^= rng_state >> 17;
  rng_state ^= rng_state << 5;
  return a + rng_state % (b - a + 1);
}

static inline int
max(int a, int b) {
  return (a > b) ? a : b;
}

static key_t*
arr_gen_conj_a(const key_t size) {
  key_t* new_arr = (key_t*) malloc( sizeof(key_t) * size);

  if (new_arr) {
    key_t offset = 0;
    new_arr[0] = 0; // não podes saltar um item atrás de 0
    for (key_t  i = 1; i < size; i++) {
      if (randint(0,9) == 0) offset += 1;
      new_arr[i] = i - offset;
    }
  }
  return new_arr;
}

static key_t*
arr_gen_conj_b(const key_t size) {
  key_t* new_arr = (key_t*) malloc( sizeof(key_t) * size);

  if (new_arr) {
    key_t offset = 0;
    new_arr[0] = 0; // não podes saltar um item atrás de 0
    for (key_t i = 1; i < size; i++) {
      if (randint(0,9) == 0) offset += 1;
      new_arr[i] = size+1-i+offset;
    }
  }

  return new_arr;
}

static key_t*
arr_gen_conj_c(const key_t size) {

  /* Array crescente com repetição minima */
  key_t* new_arr = arr_gen_conj_a(size);

  if (new_arr) {
    /* Knuth Shuffle */
    int i, j;
    for (j = size-1; j > 0; j--) {
      i = randint(0, j-1);
```