6. Структуры

Владимир Верстов

Б. Керниган, Д. Ритчи. Язык программирования C

Структура

- — это совокупность нескольких переменных, часто различных типов, сгруппированных под единым именем для удобства обращения и обработки

Пример

Студент
Фамилия
Имя
Отчество
Курс
Группа
Факультет
...

- С помощью структур удобно моделировать сложные данные

- Структуры можно:

  ➡ копировать

  ➡ присваивать

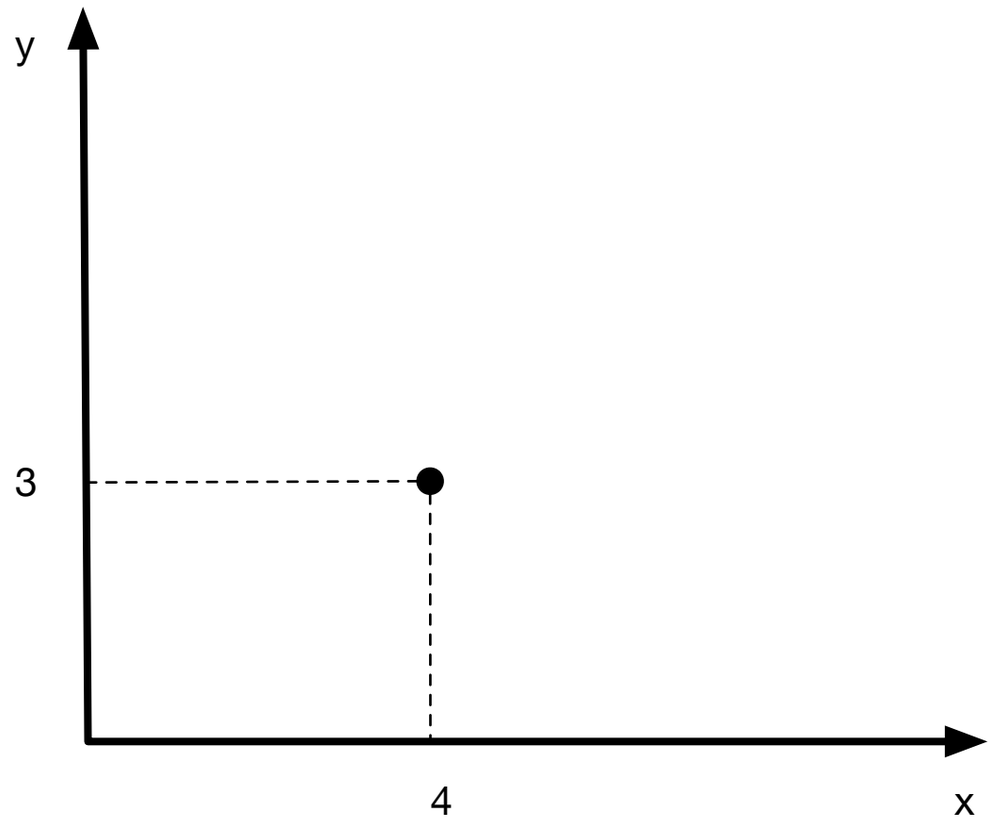
  ➡ передавать и возвращать из функций

- Так же можно получать адрес структур в памяти и создавать указатели на структуры

Основы работы со структурами

Структура “точка”

```
struct point {
    int x;
    int y;
};
```



- `struct` — начало объявления структуры

- После `struct` следует **метка структуры** (например, `point`) — **структурный тип**, который можно использовать для объявлений

- В фигурных скобках перечисляются **поля структуры**

- Объявление структуры фактически вводит новый тип данных

Объявление структуры

```
struct point {
    int x;
    int y;
} p1, p2, p3, point_array[100];
```

Инициализация структуры

```
struct point origin = {0, 0};
```

Обращение к полю структуры

```
имя-структуры.имя-поля
```

- Точка — знак операции обращения к полю структуры
- Точка ставиться между именем структуры и именем поля структуры

Обращение к полю структуры

```
struct point origin = {0, 0};
struct point pt = {5, 3};

printf("(%d,%d)\n", origin.x, origin.y);

// distance between origin and pt
dist = sqrt((double)(pt.x * pt.x + pt.y * pt.y));
```

Вложенность структур

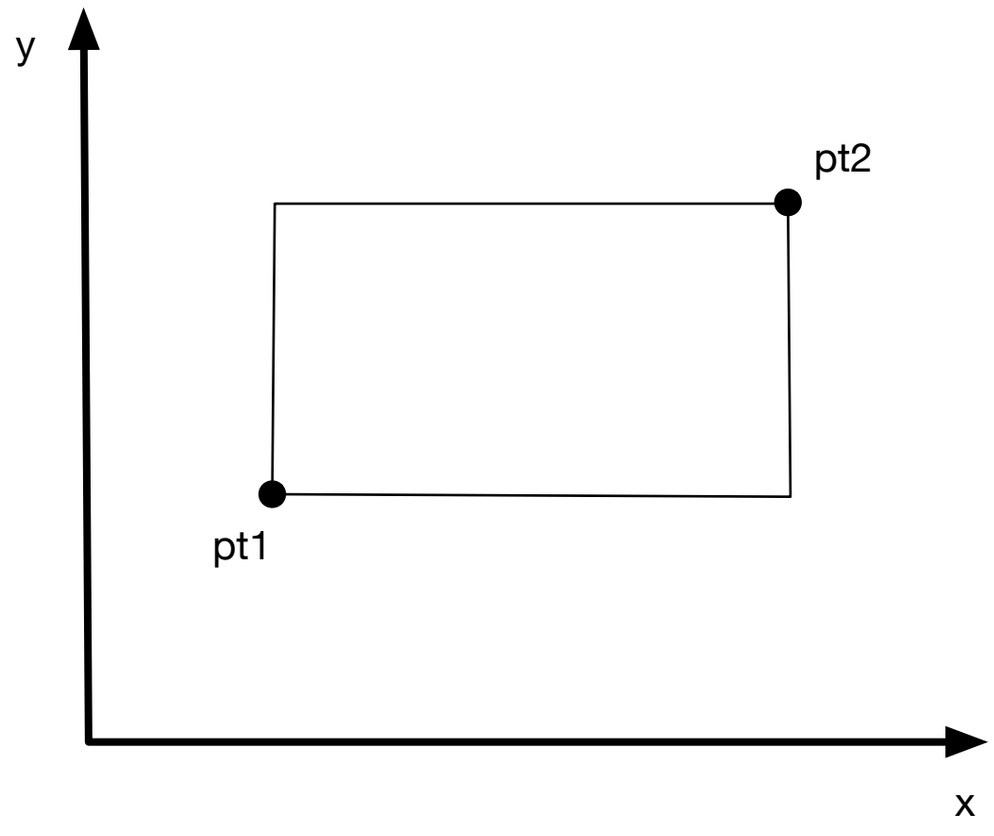
```
struct rect {
    struct point pt1;
    struct point pt2;
};

// ...

struct rect r1;

// ...

printf("(%d,%d)\n", r1.pt1.x, r1.pt1.y);
```



Функции и структуры

```
struct point makepoint(int x, int y) {
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

Функции и структуры

```
int point_in_rect(struct point p, struct rect r) {
    return p.x >= r.pt1.x && p.x < r.pt2.x
        && p.y >= r.pt1.y && p.y < r.pt2.y;
}
```

Указатели на структуры

```
struct point *pp;
struct point origin;

pp = &origin;
printf ("origin: (%d,%d)\n", (*pp).x, (*pp).y);
```

- `pp` — указатель на структуру
- `*pp` — сама структура
- `(*pp).x, (*pp).y` — поля структуры

Доступ к полям через указатель на структуру

```
указатель-на-структуру->поле-структуры
```

Примеры

```
struct point *pp;
struct point origin;

pp = &origin;

printf("origin: (%d,%d)\n", pp->x, pp->y);

struct rect r, *rp;
rp = &r;

// записи ниже эквиваленты
r.pt1.x;
rp->pt1.x;
(r.pt1).x;
(rp->pt1).x;
```

Еще пример

```
struct {
    int len;
    char *str;
} *p;
```

```
++p->len;
++(p->len);
(++p)->len;
(p++)->len;
*p->str;
*p->str++;
(*p->str)++;
*p++->str;
```

* -> ++ & ???

```
struct {
    int len;
    char *str;
} *p;
```

```
++p->len;
++(p->len);
(++p)->len;
(p++)->len;
*p->str;
*p->str++;
(*p->str)++;
*p++->str;
```

Происходит следующее…

```
struct {
    int len;
    char *str;
} *p;
```

```
++p->len; // инкремент len, а не p
++(p->len); // инкремент len, а не p
(++p)->len; // инкремент p, до обращения к len
(p++)->len; // инкремент p, после обращения к len
*p->str; // значение указателя str, а не p
*p->str++; // инкремент str, а не p
(*p->str)++; // инкремент значения, на которое указывает str
*p++->str; // инкремент p, после обращения к данным str
```

- `->` `.` `[]` `()` — операторы с самым высоким приоритетом в языке C

Подсчет количества ключевых слов в исходном коде

```c
#include "stdio.h"
#include "ctype.h"
#include "string.h"

struct key {
    char *word;
    int count;
};

struct key keytab[NKEYS] = {
   "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    // ...
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
}
```

```c
#define MAXWORD 100
#define NKEYS (sizeof keytab / sizeof(struct key))

int getword(char *, int);
int binsearch(char *, struct key *, int);

int main() {
    int n;
    char word[MAXWORD];
    while(getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((n = binsearch(word, keytab, NKEYS)) >= 0)
                keytab[n].count++;
    for (n = 0; n < NKEYS; n++)
        if (keytab[n].count > 0)
    printf("%4d %s\n", keytab[n].count, keytab[n].word);
    return 0;
}
```

Размер `keytab`

```
размер  keytab / размер struct key
```

Оператор `sizeof`

```
sizeof объект
sizeof(имя-типа-данных)

#define NKEYS (sizeof keytab / sizeof(struct key))
#define NKEYS (sizeof keytab / sizeof keytab[0])
```

```c
int getword (char *word, int lim) {
    int c, getch(void);
    void ungetch(int);
    char *w = word;
    while (isspace(c = getch())) {
        ;
    }
    if (c != EOF) {
        *w++ = c;
    }
    if (!isalpha(c)) {
        *w = '\0';
        return c;
    }
    for ( ; --lim > 0; w++) {
        if (!isalnum(*w = getch())) {
            ungetch(*w);
            break;
        }
    }
    *w = '\0';
    return word[0];
}
```

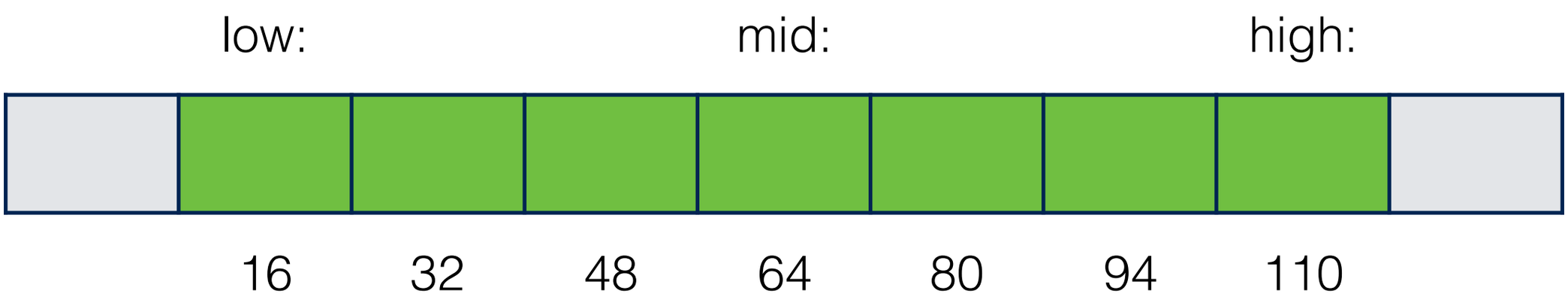
Применение указателей

```c
int getword(char *, int);
struct key *binsearch(char *, struct key *, int);

int main() {
    char word [MAXWORD];
    struct key *p;
    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((p = binsearch(word, keytab, NKEYS)) != NULL)
                p->count++;
    for (p = keytab; p < keytab + NKEYS; p++)
        if (p->count > 0)
            printf("%4d %s\n", p->count, p->word);
    return 0;
}
```

```
struct key *binsearch(char *word, struct key *tab, int n) {
    int cond;
    struct key *low = &tab[0];
    struct key *high = &tab[n];
    struct key *mid;
    while (low < high) {
        mid = low + (high - low) / 2;
        if ((cond = strcmp(word, mid->word)) < 0)
            high = mid;
        else if (cond > 0) ,
            low = mid + 1 ;
        else
            return mid;
    }
    return NULL;
}
```

Вычисление `mid`




```
mid = low + (high - low) / 2;
// OK: high - low = 7, 7 / 2 = 3, low + 3 = 16 + 16*3 = 64

mid = (high + low) / 2;
// ERROR: (16 + 110) / 2 = 63
```

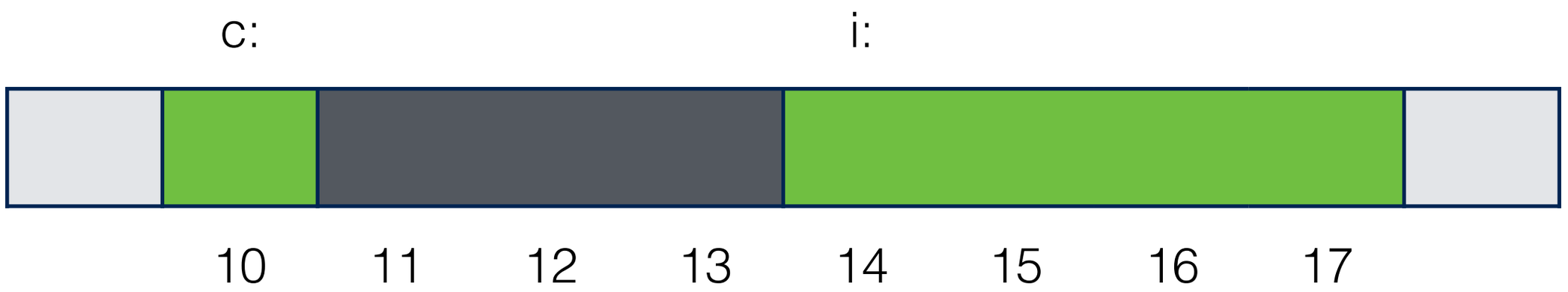
Длинна структуры в памяти

- Длинна структуры в памяти может быть не равна сумме длин её полей
- Объекты могут быть выровнены в памяти по машинным словам
- Внутри структуры могут быть “дыры”

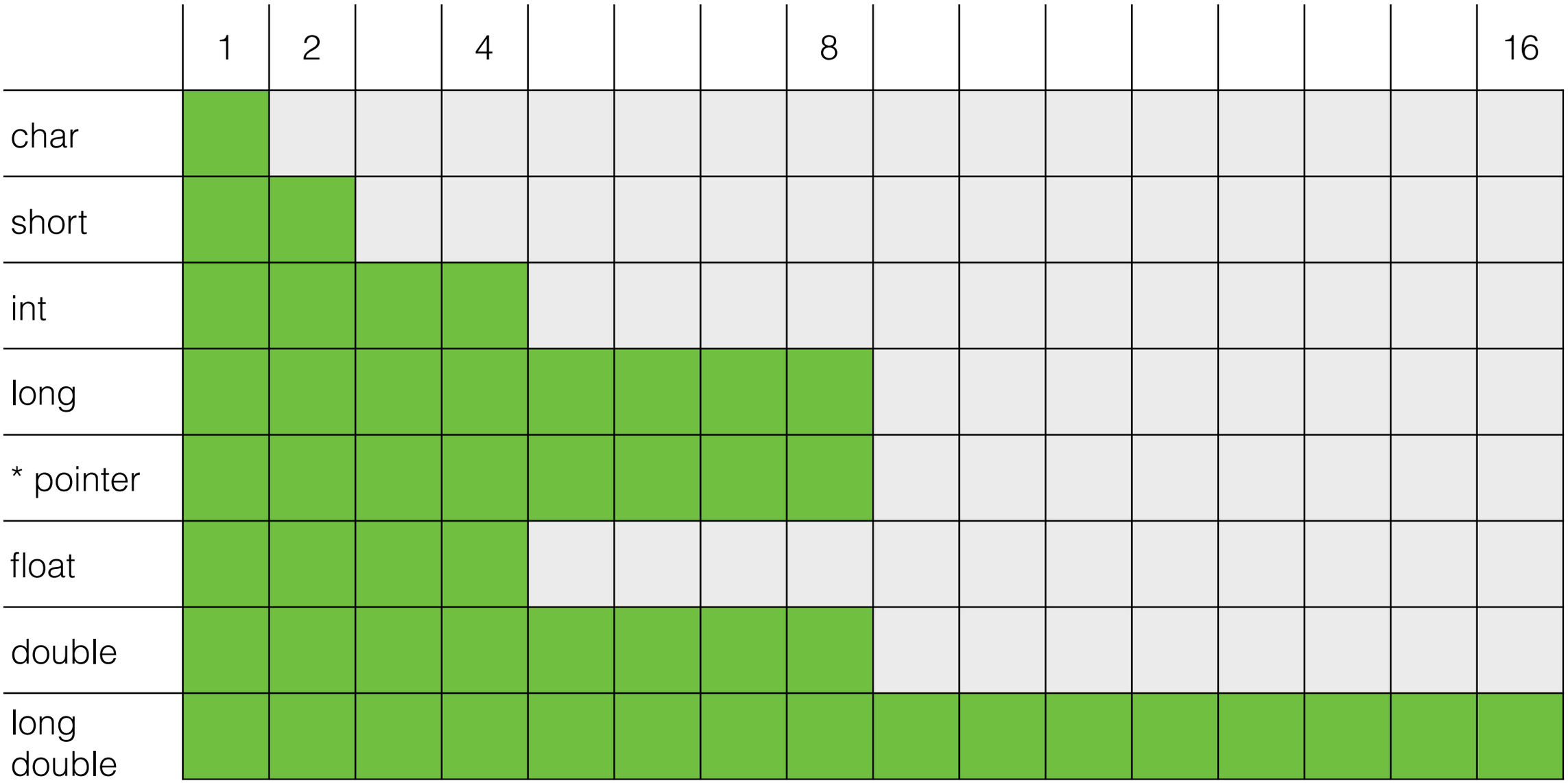
Пример

```
struct example {
    char c;
    int i;
}
```

```
sizeof(struct example); // 8
```



Размеры типов



Binary Search Tree

Бинарное дерево поиска

Задача

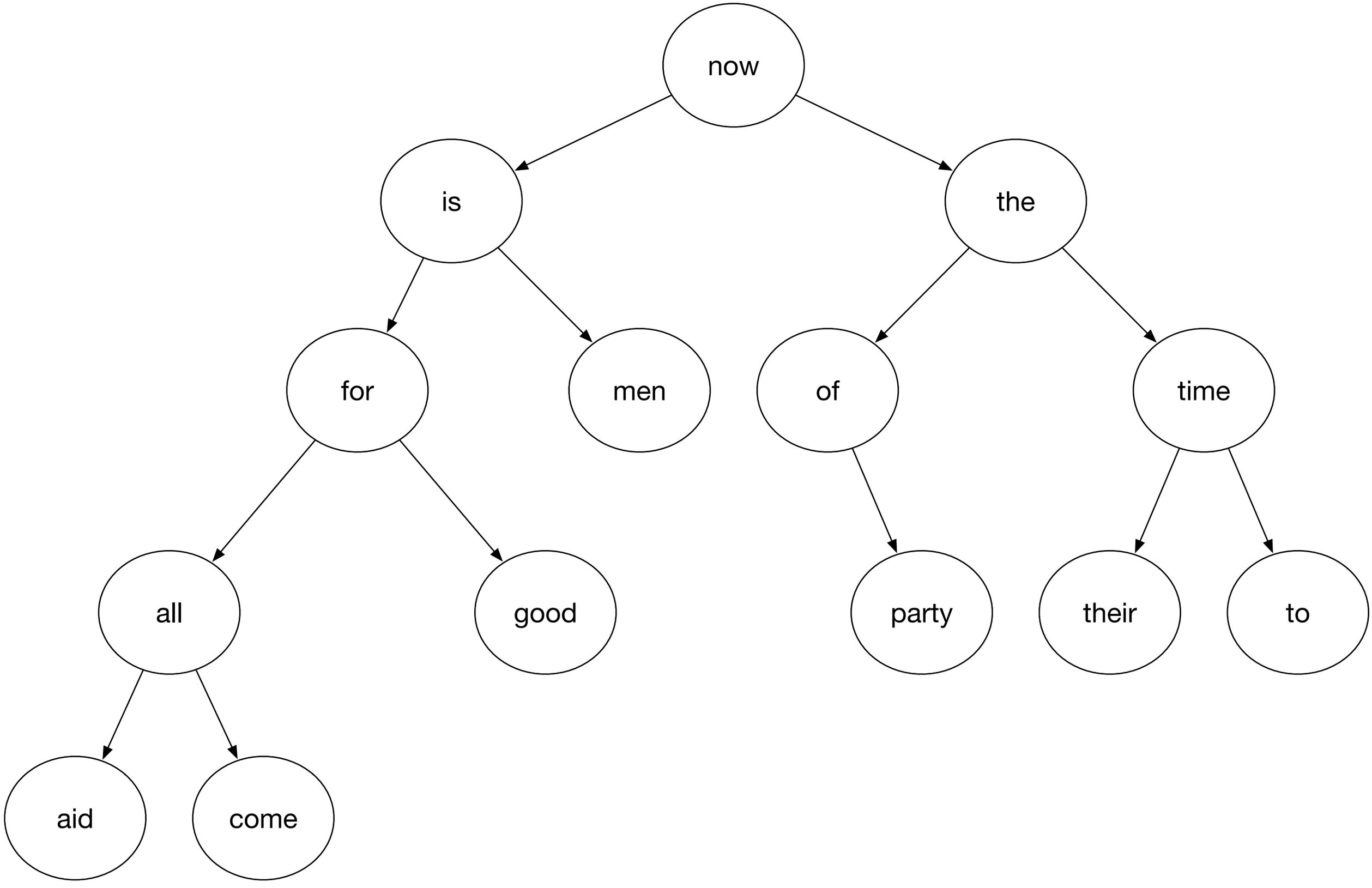
- Подсчитать частоту, с которой встречаются слова во входном потоке
- Количество слов и сами слова заранее не известны

Пример

“now is the time for all good men to come to the aid of their party”

“настало время всем хорошим людям прийти на помощь своей стороне”
Чарльз Э. Уэллер.

Фраза содержит почти все буквы латинского алфавита и часто применяется для тестирования различных технологий обработки текста


now
is
the
for
men
of
time
all
good
party
their
to
aid
come

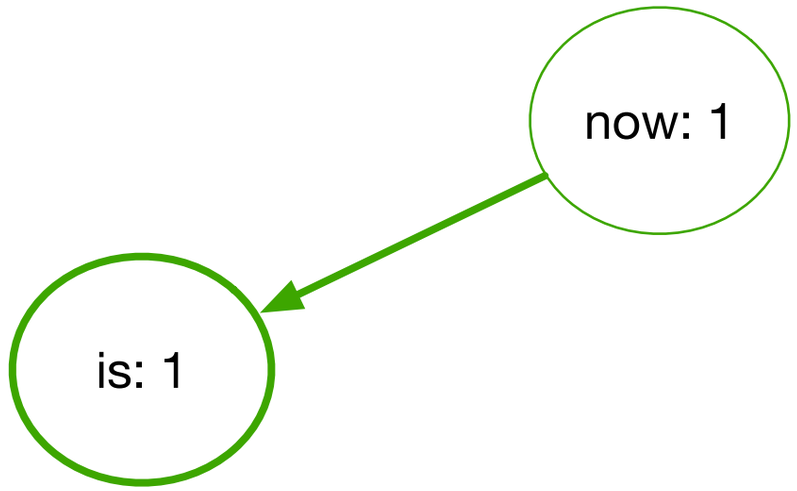
- Дерево содержит один узел для каждого слова
- Узел не может иметь больше 2-х дочерних узлов
- Левое поддерево узла содержит слова, меньшие по алфавиту, чем слово в узле
- Правое поддерево узла содержит слова, большие по алфавиту, чем слово в узле

Узел дерева содержит:

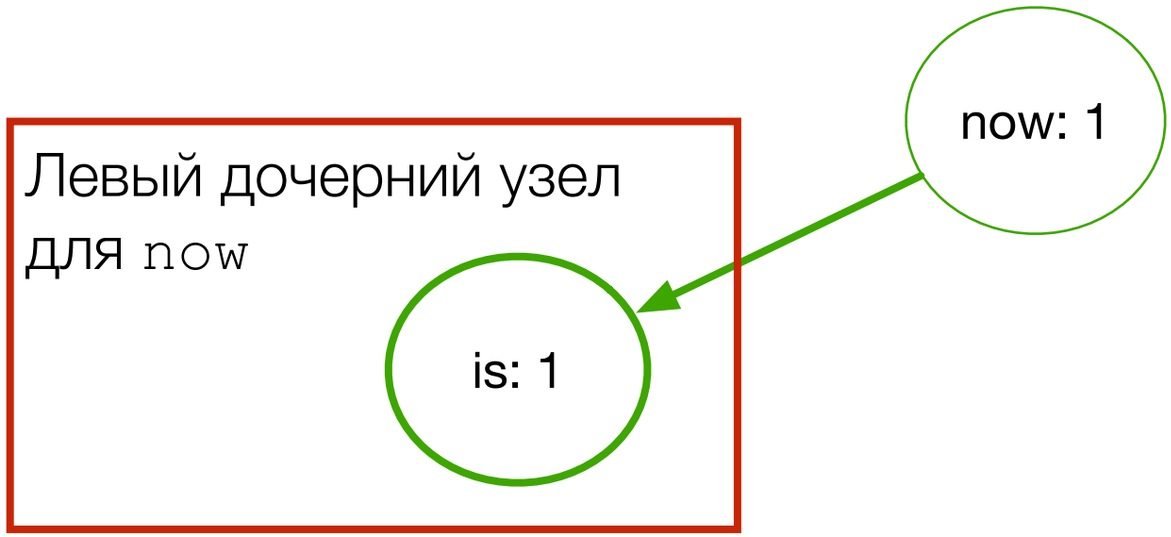
- Указатель на слово
- Частота, с которой слово встречается в тексте
- Указатель на левый дочерний узел
- Указатель на правый дочерний узел

now is the time for all good men
to come to the aid of their party

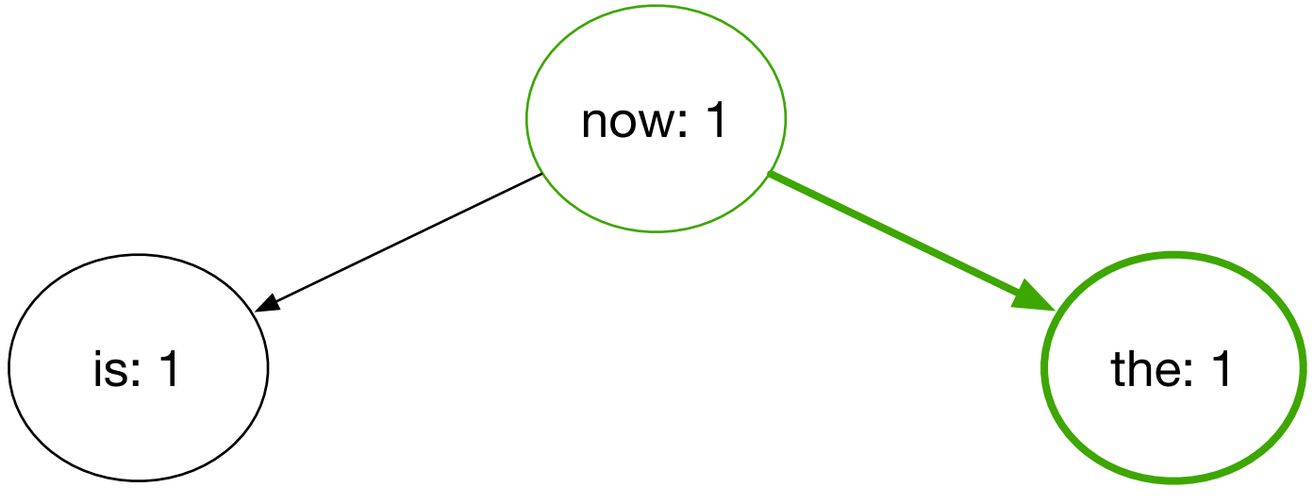
now is the time for all good men
to come to the aid of their party



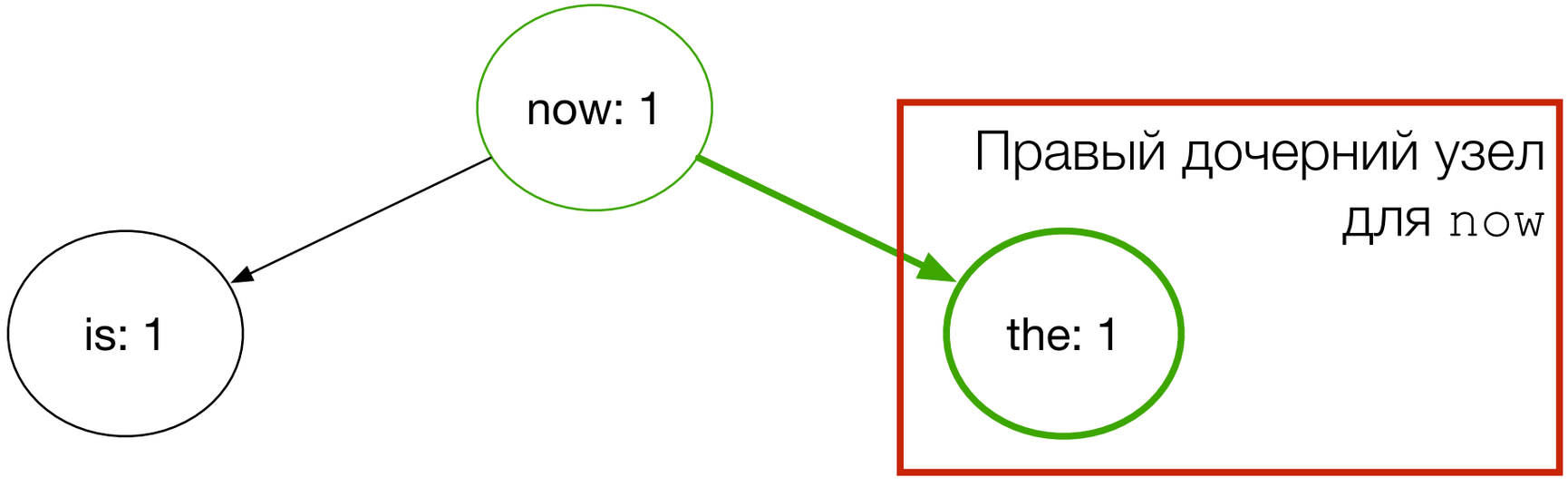
now is the time for all good men
to come to the aid of their party



now is the time for all good men
to come to the aid of their party



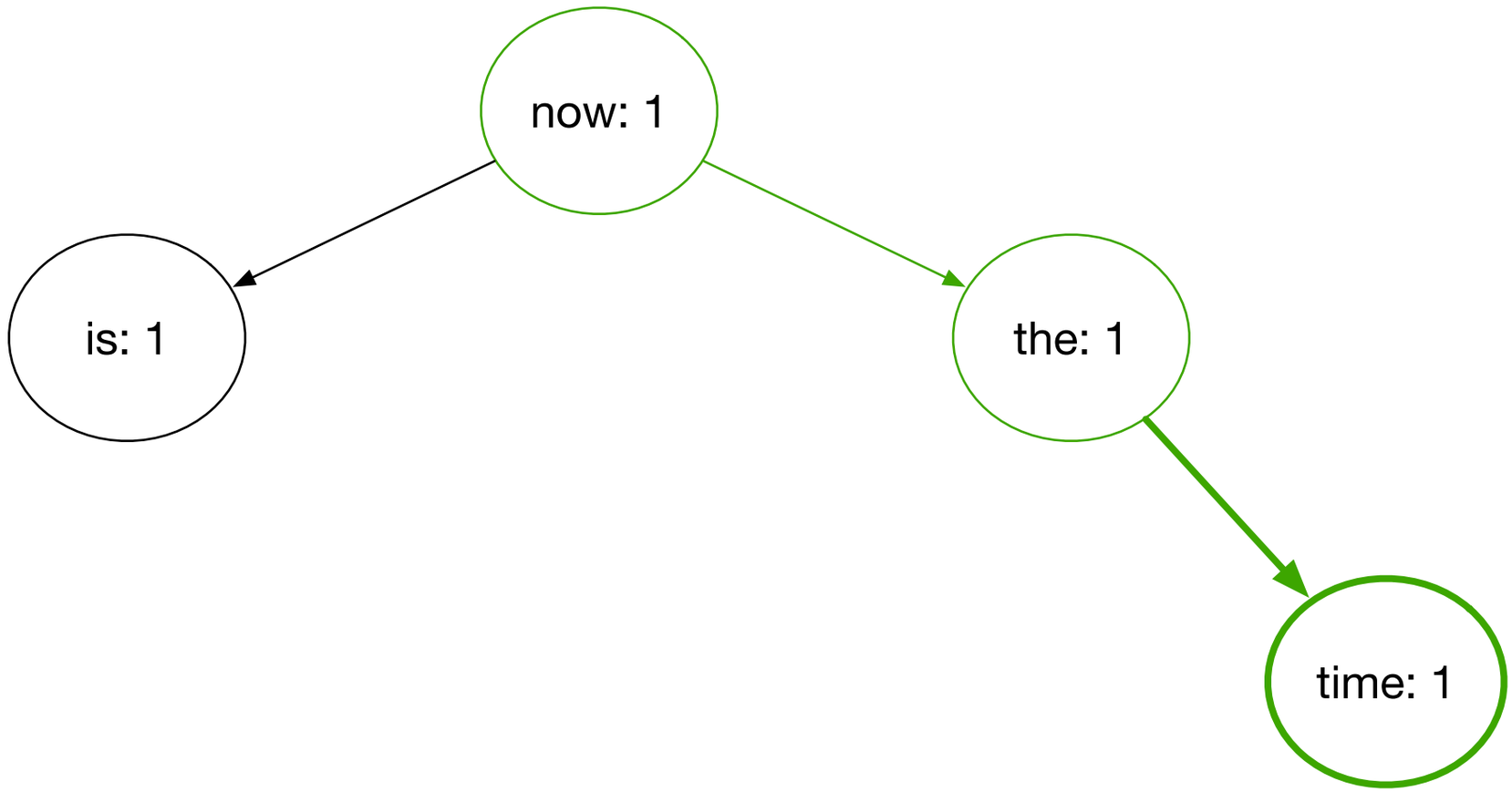
now is the time for all good men
to come to the aid of their party



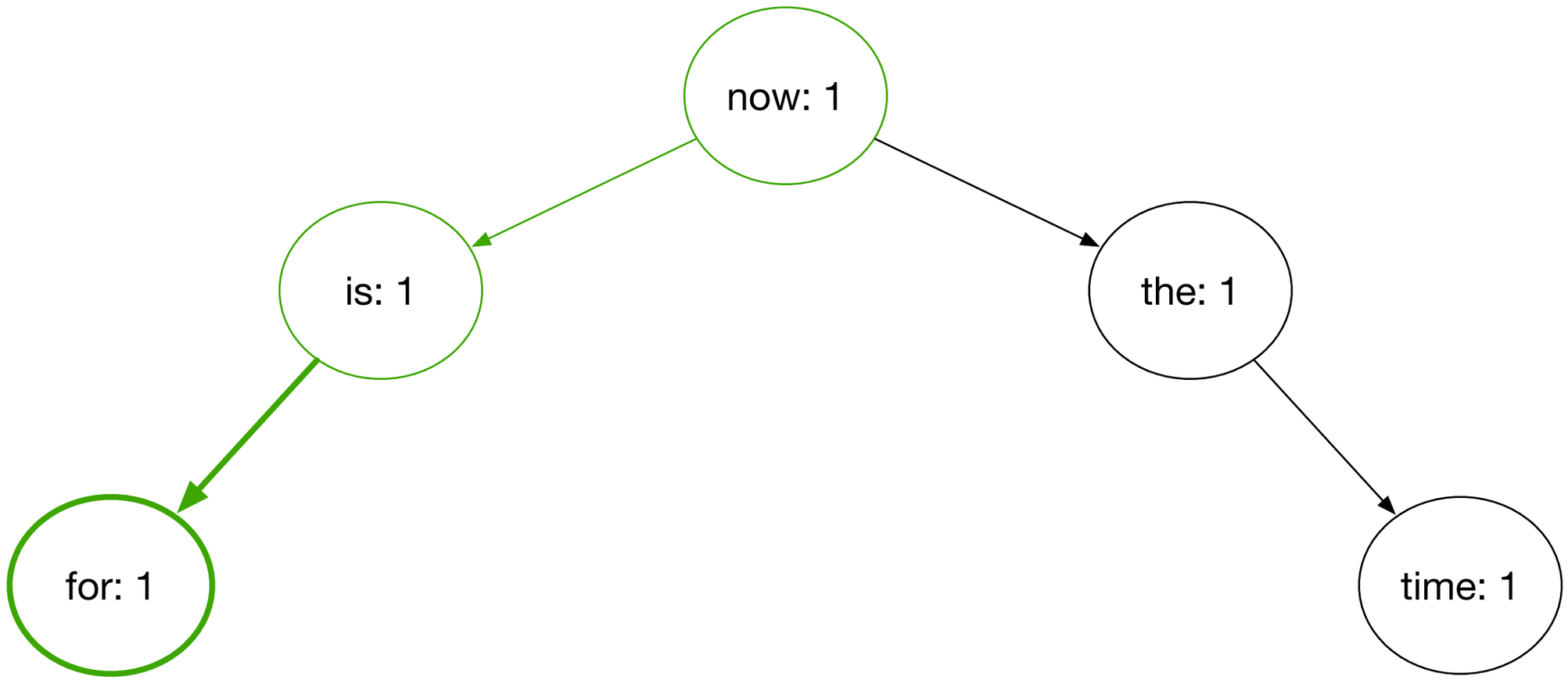
now is the time for all good men
to come to the aid of their party

- Корень дерева — узел, для которого не существует родительского узла

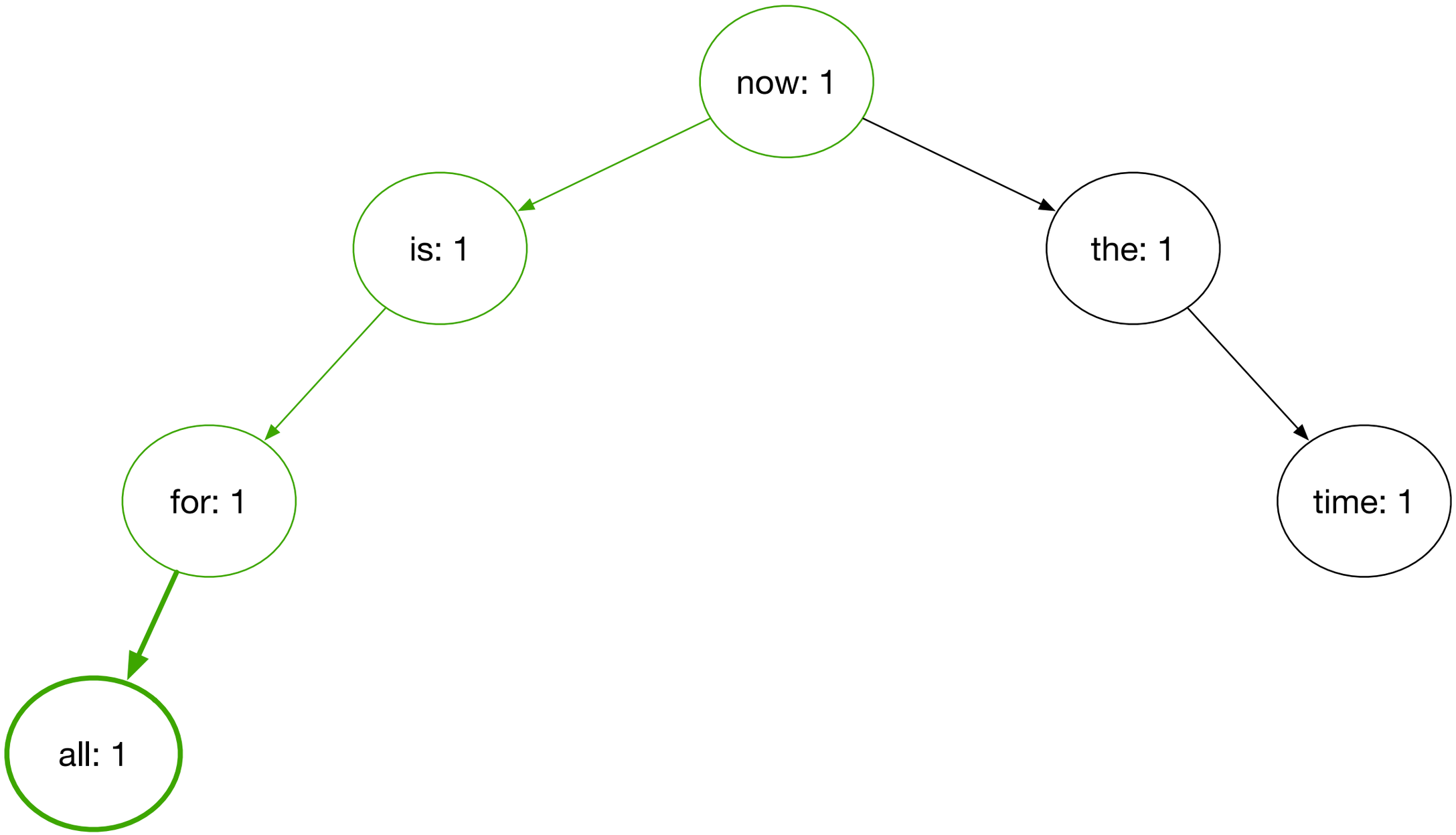
now is the time for all good men
to come to the aid of their party



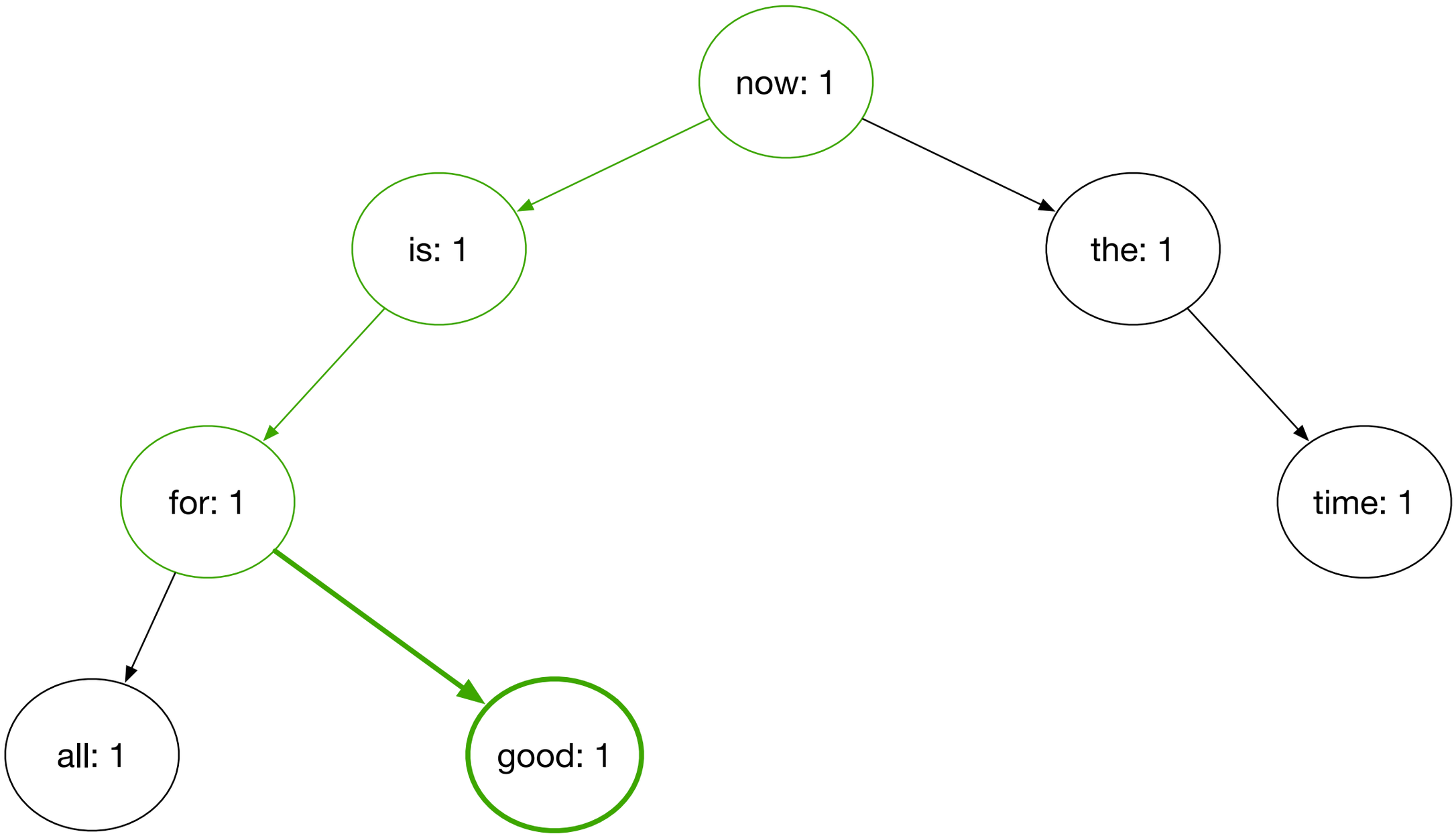
now is the time for all good men
to come to the aid of their party



now is the time for all good men
to come to the aid of their party

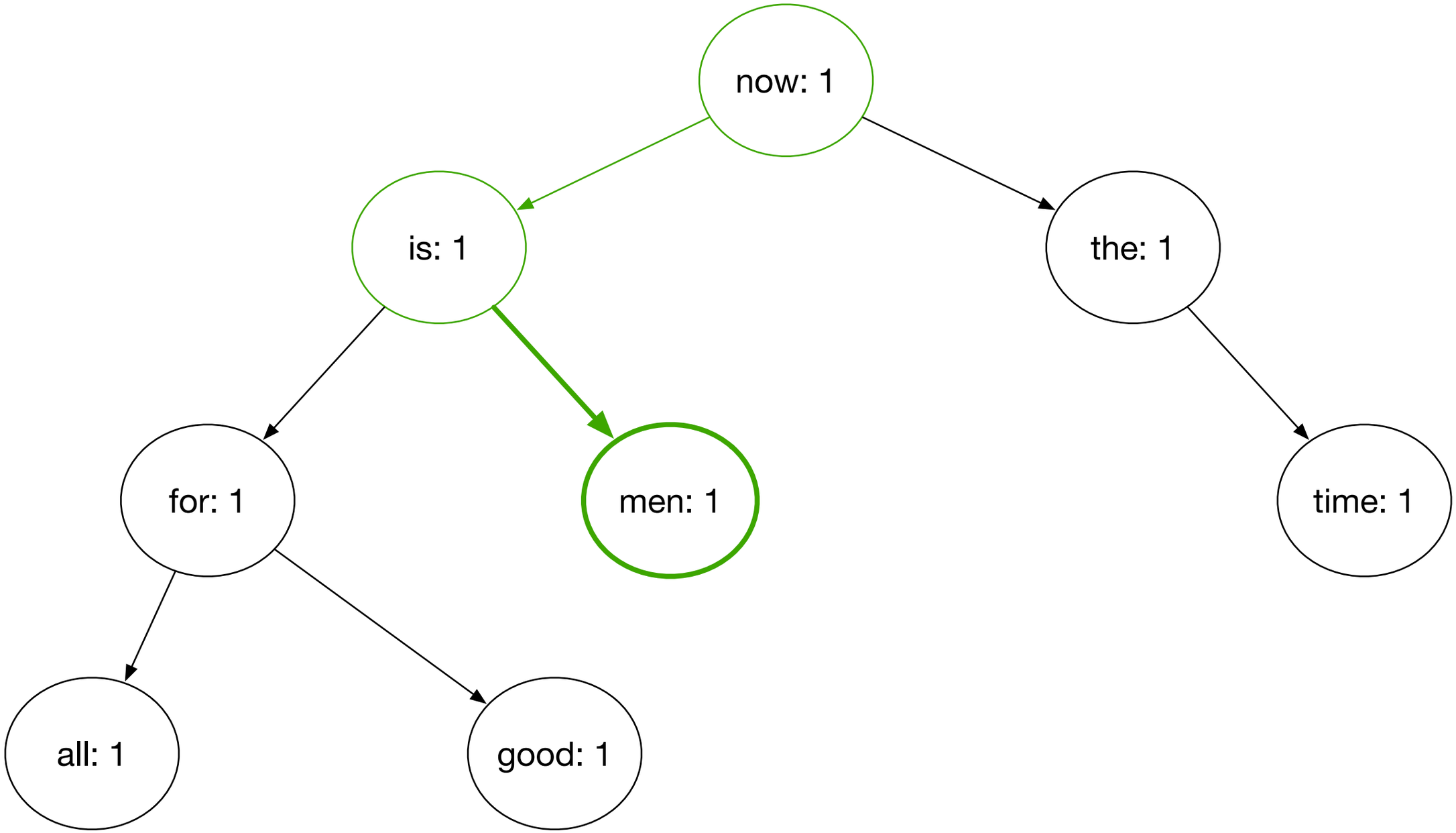


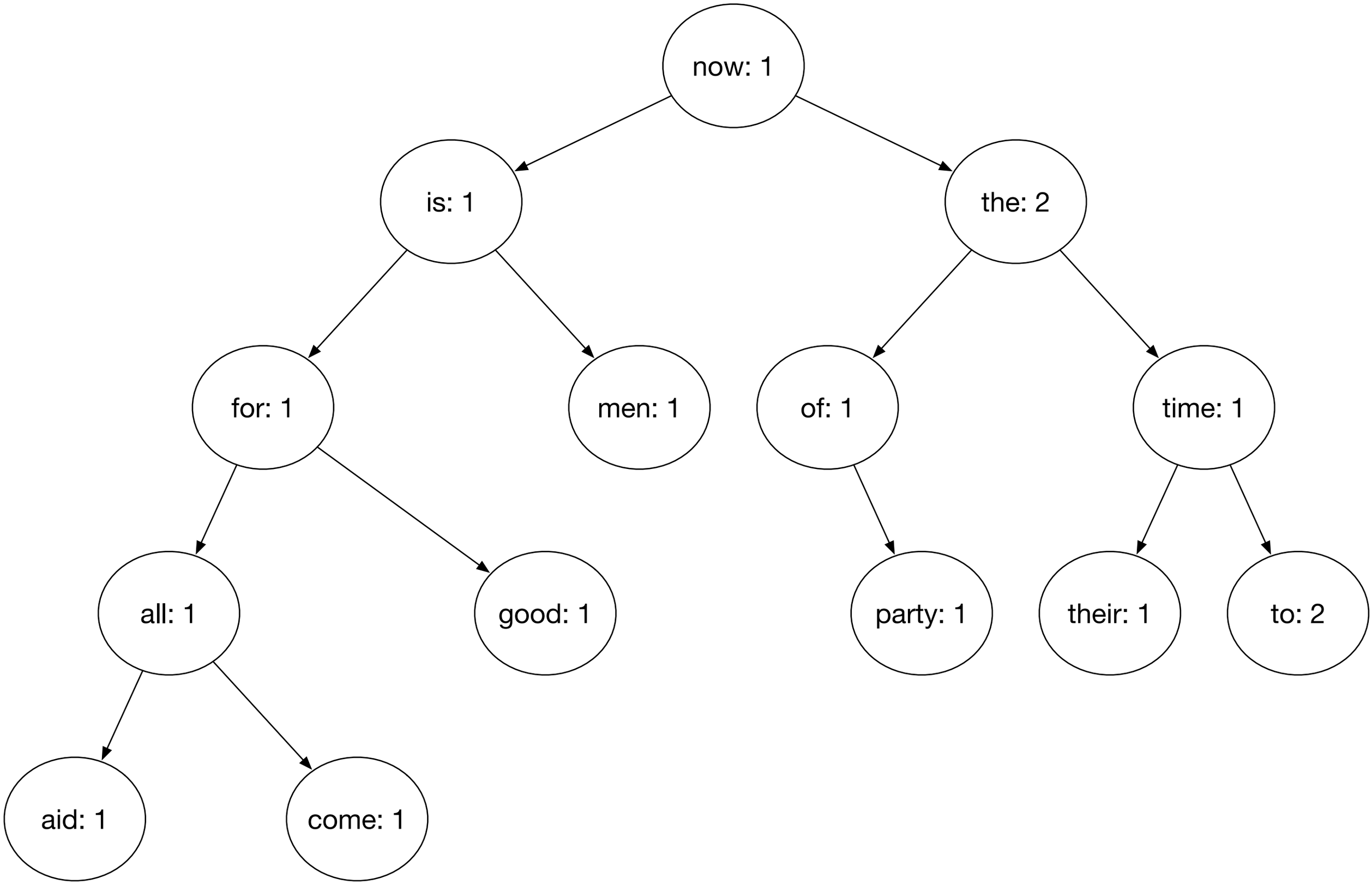
now is the time for all good men
to come to the aid of their party

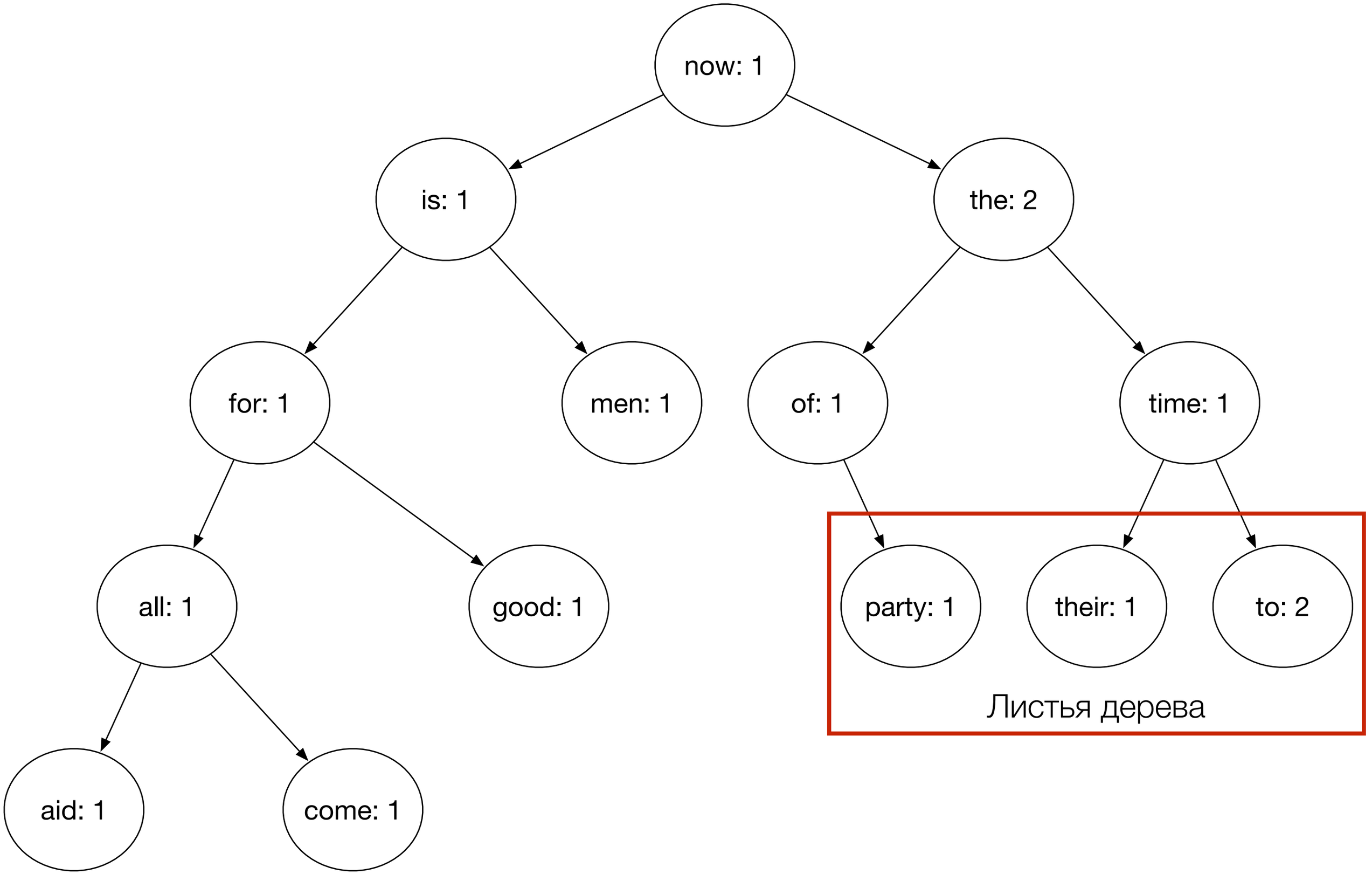


now is the time for all good men

to come to the aid of their party




now: 1
is: 1
the: 2
for: 1
men: 1
of: 1
time: 1
all: 1
good: 1
party: 1
their: 1
to: 2
aid: 1
come: 1


now: 1
is: 1
the: 2
for: 1
men: 1
of: 1
time: 1
all: 1
good: 1
party: 1
their: 1
to: 2
aid: 1
come: 1
Листья дерева

- Лист дерева — узел, для которого не существует дочерних узлов

Узел дерева

```
struct tnode {
    char *word;
    int count;
    struct tnode *left;
    struct tnode *right;
};
```

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAXWORD 100

struct tnode *addtree(struct tnode *, char *);
void treeprint(struct tnode *);
int getword(char *, int);
```

```c
int main() {
    struct tnode *root;
    char word[MAXWORD];
    root = NULL;
    while (getword (word, MAXWORD) != EOF)
        if (lsalpha(word[0]))
            root = addtree(root, word);
    treeprint(root);
    return 0;
}
```

```c
struct tnode *talloc(void);
char *strdup(char *);

struct tnode *addtree(struct tnode *p, char *w) {
    int cond;
    if (p == NULL) {
        p = talloc();
        p->word = strdup(w);
        p->count = 1;
        p->left = p->right = NULL;
    } else if ((cond = strcmp(w, p->word)) == 0)
        p->count++;
    else if (cond < 0)
        p->left = addtree(p->left, w);
    else
        p->right = addtree(p->right, w);
    return p;
}
```

```
void treeprint(struct tnode *p) {
    if (p != NULL) {
        treeprint(p->left);
        printf("%4d %s\n", p->count, p->word);
        treeprint(p->right);
    }
}
```

```c
#include <stdlib.h>

struct tnode *talloc(void) {
   return (struct tnode *) malloc(sizeof(struct tnode));
}
```

```c
char *strdup(char *s) {
    char *p;
    p = (char *) malloc(strlen(s) + 1);
    if (p != NULL)
        strcpy(p, s);
    return p;
}
```

Hash Table

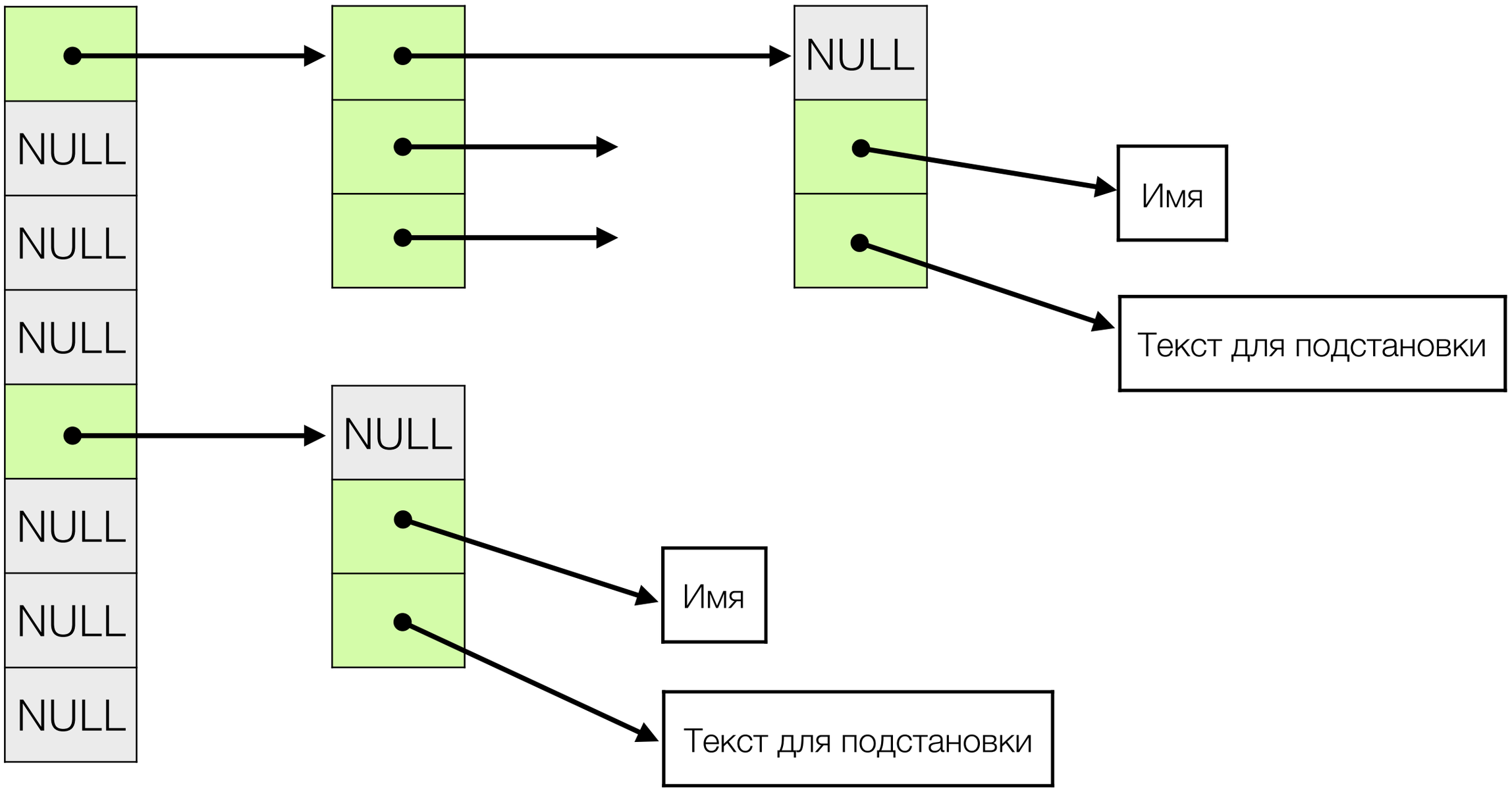
Хэш-таблица

Задача

- Реализовать структуру данных для замены констант из директивы `#define` на их значения

- Структура данных должна обеспечить минимальную временную сложность добавления новых записей, поиска записей и их удаление

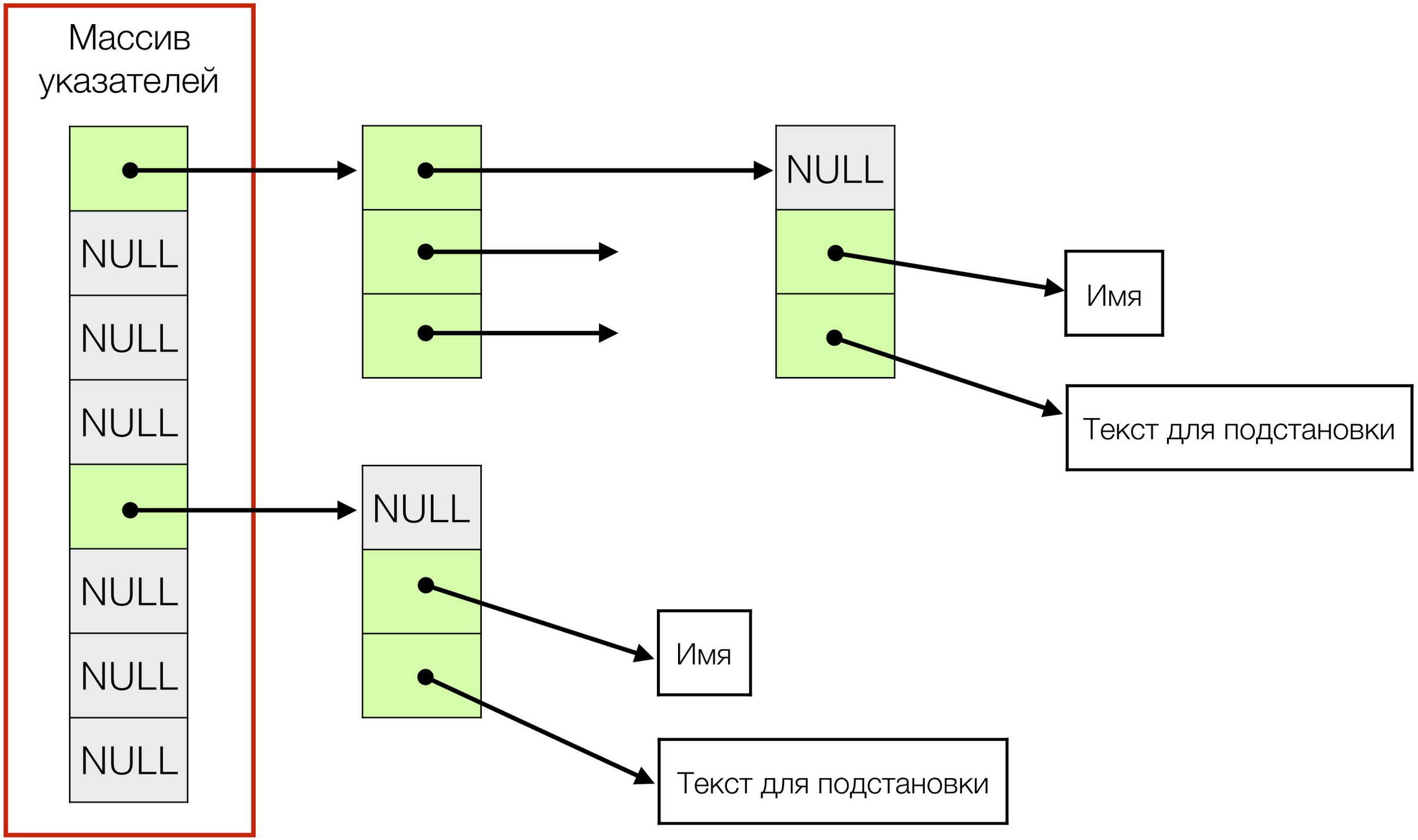
Hash Table

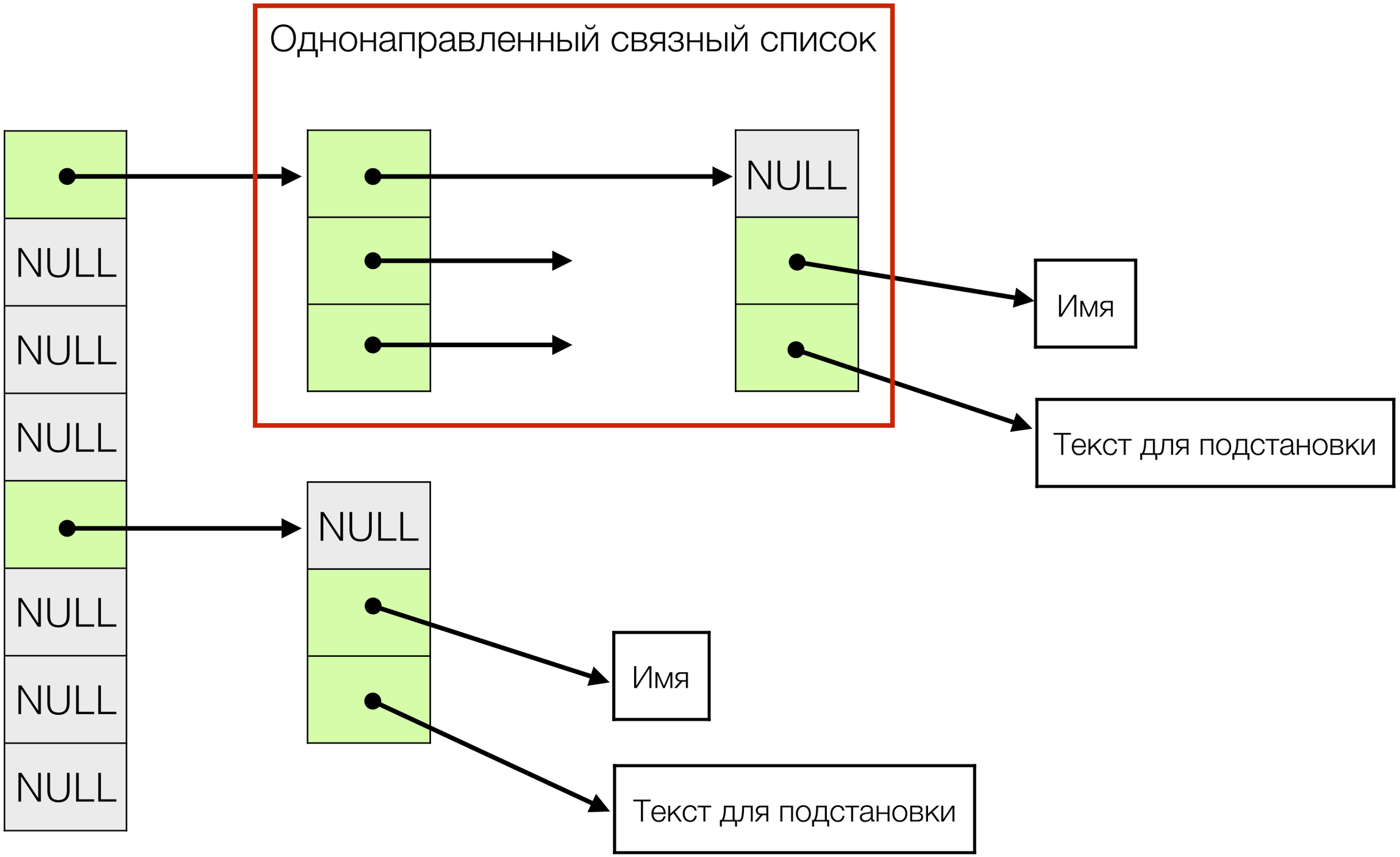

NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
Имя
Текст для подстановки
NULL
Имя
Текст для подстановки

- Имя конвертируется в неотрицательное целое число (`hash`), которое является индексом в массиве указателей

- Каждый указатель в массиве указывает на начало однонаправленного связного списка

- Каждый элемент списка содержит имя и текст для подстановки, которые соответствуют одному и тому же значению `hash`


Массив
указателей
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
Имя
Текст для подстановки
NULL
Имя
Текст для подстановки


Однонаправленный связный список
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
Имя
Текст для подстановки
Имя
Текст для подстановки

Список и таблица

```
struct nlist {
   struct nlist *next;
   char *name;
   char *defn;
};

#define HASHSIZE 101

static struct nlist *hashtab[HASHSIZE];
```

Хэш-функция

```
unsigned hash(char *s) {
    unsigned hashval;
    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}
```

Поиск в таблице

```
struct nlist *lookup(char *s) {
    struct nlist *np;
    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
          if (strcmp(s, np->name) == 0)
               return np;
      return NULL;
}
```

Добавление записи

```
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;
    if ((np = lookup(name)) == NULL) {
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    } else
        free((void *) np->defn);
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```

Определение новых типов

```
typedef int Length;

Length len, maxlen;
Length *lengths[];

typedef char *String;

String p, lineptr[MAXLINES];
int strcmp(String, String);
p = (String) malloc(100);
```

```
typedef struct tnode *Treeptr;

typedef struct tnode {
    char *word;
    int count;
    Treeptr left;
    Treeptr right;
};

Treeptr talloc(void) {
    return (Treeptr) malloc(sizeof (Treenode));
}
```

Объединение

- … — это переменная, которая может содержать объекты различных типов и размеров.
- Требования к размеру и выравниванию определяет компилятор
- Размер объединения достаточен для хранения переменной любого типа из указанных
- При помощи объединений можно работать с данными различных типов в рамках одного участка памяти

```
union u_tag {
    int ival;
    float fval;
    char *sval;
} u;

struct {
    char *name;
    int flags;
    int utype;
    union {
        int ival;
        float fval;
        char *sval;
    } u;
} symtab[NSYM];

symtab[i].u.ival;
*symtab[i].u.sval;
symtab[i].u.sval[0];
```

Обращение к полю объединения

```
имя-объединения.поле

указатель-на-объединение->поле
```

- Объединения можно использовать в массивах и структурах

- Фактически объединение — структура вся поля, которой имеют нулевое смещение относительно ее начала

- Для объединения справедливы все операции, которые возможны над структурами

- Объединение можно инициализировать только данными того типа, который имеет его первое поле

Битовое поле. Bit Field

- … — совокупность идущих подряд битов внутри системно-зависимой единицы памяти (внутри машинного слова)

```c
#define KEYWORD 01
#define EXTERNAL 02
#define STATIC 04

enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };

flags |= EXTERNAL | STATIC; // == 1
flags &= ~(EXTERNAL | STATIC); // == 0

if ((flags & (EXTERNAL | STATIC)) == 0) { ... }
```

```c
struct {
    unsigned int is_keyword : 1;
    unsigned int is_extern : 1;
    unsigned int is_static : 1;
} flags;

flags.is_extern = flags.is_static = 1; // == 1
flags.is_extern = flags.is_static = 0; // == 0

if (flags.is_extern == 0 && flags.is_static == 0) { ... }
```