Высшая школа экономики
Факультет компьютерных наук
Департамент программной инженерии

Алгоритмы и алгоритмические языки
Лекция 9

25 ноября 2025 г.

Динамические структуры данных. Стек

Стек (stack) — это динамическая последовательность элементов, количество которых изменяется, причем как добавление, так и удаление элементов возможно только с одной стороны последовательности (вершина стека).

Работа со стеком осуществляется с помощью функций:

`push(x)` — затолкать элемент x в стек;
`x = pop()` — вытолкнуть элемент из стека.

Стек можно организовать на базе (примеры):

- фиксированного массива `stack[MAX]`, где константа `MAX` задает максимальную глубину стека;
- динамического массива, текущий размер которого хранится отдельно.

В обоих случаях необходимо хранить позицию текущей вершины стека.
Можно использовать и другие структуры данных (список).

Организация стека на динамическом массиве

```
struct stack {
  int sp;       /* Текущая вершина стека */
  int sz;       /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static void push (char c) {
  if (stack.sz == stack.sp + 1) {
    stack.sz = 2*stack.sz + 1;
    stack.stack = (char *) realloc (stack.stack,
                                    stack.sz*sizeof (char));
  }
  stack.stack[++stack.sp] = c;
}
```

Организация стека на динамическом массиве

```
struct stack {
  int sp;       /* Текущая вершина стека */
  int sz;       /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static char pop (void) {
  if (stack.sp < 0) {
    fprintf (stderr, "Cannot␣pop:␣stack␣is␣empty\n");
    return 0;
  }
  return stack.stack[stack.sp--];
}
```

Дома. Сделайте, чтобы результат записывался по указателю-аргументу, а функция возвращала код успеха операции.

Организация стека на динамическом массиве

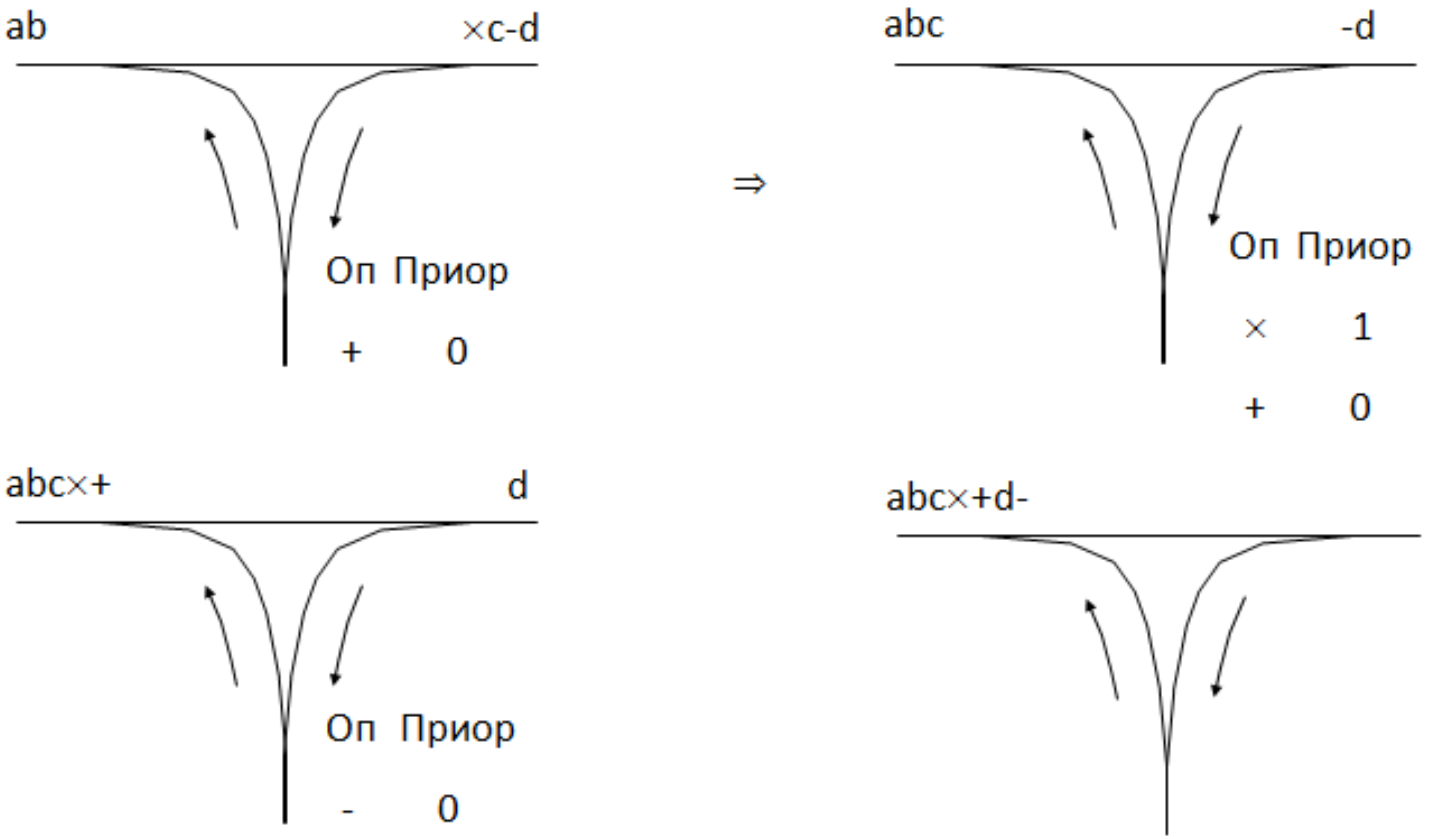
```
struct stack {
  int sp;       /* Текущая вершина стека */
  int sz;       /* Размер массива */
  char *stack;
} stack = { .sp = -1, .sz = 0, .stack = NULL };

static int isempty (void) {
  return stack.sp == -1;
}
```

Пример работы со стеком

Перевод арифметического выражения в обратную польскую запись (постфиксную).

$$a + b \times c - d \quad \rightarrow \quad abc \times +d-$$
$$c \times (a + b) - (d + e)/f \quad \rightarrow \quad cab + \times de + f/-$$



Пример работы со стеком

$$c \times (a + b) - (d + e)/f \quad \rightarrow \quad cab + \times de + f/-$$

Перевод арифметического выражения в обратную польскую запись

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

#include "stack.c"

/* Считывание символа-операции или переменной */
static char getop (void) {
  int c;
  while ((c = getchar ()) != EOF && isblank (c))
    ;
  return c == EOF || c == '\n' ? 0 : c;
}
```

Перевод арифметического выражения в обратную польскую запись

```
/* Является ли символ операцией */
static int isop (char c) {
  return (c == '+') || (c == '-') || (c == '*')
         || (c == '/');
}

/* Каков приоритет символа-операции */
static int prio (char c) {
  if (c == '(')
    return 0;
  if (c == '+' || c == '-')
    return 1;
  if (c == '*' || c == '/')
    return 2;
  return -1;
}
```

Перевод арифметического выражения в обратную польскую запись

```
int main (void) {
  char c, op;

  while (c = getop ()) {
    /* Переменная-буква выводится сразу */
    if (isalpha (c))
      putchar (c);
    /* Скобка заносится в стек операций */
    else if (c == '(')
      push (c);
    else <...>
```

Перевод арифметического выражения в обратную польскую запись

```
/* Операция заносится в стек в зависимости от
приоритета */
else if (isop (c)) {
       while (! isempty ()) {
         op = pop ();
         /* Заносим, если больший приоритет */
         if (prio (c) > prio (op)) {
           push (op); break;
         } else
         /* Иначе выталкиваем операцию из стека */
         putchar (op);
       }
       push (c);
     } else <...>
```

Перевод арифметического выражения в обратную польскую запись

```
/* Скобка выталкивает операции до парной скобки */
} else if (c == ')')
      while ((op = pop ()) != '(')
        putchar (op);
  }
/* Вывод остатка операций из стека */
  while (! isempty ())
    putchar (pop ());
  putchar ('\n');
  return 0;
}
```

Дома. Введите операцию **peek()** и перепишите код с ее помощью. Обработайте случай непарных скобок.

Организация стека как библиотеки

stack.h:

```
extern void push (char);
extern char pop (void);
extern int isempty (void);
```

stack.c:

```
#include "stack.h"
struct stack {
  <...>
};
static struct stack stack
        = { <...> };
```

main.c:

```
#include "stack.h"
int main (void) {
  <...push (c), pop (), ...>
}
```

```
$gcc main.c stack.c –o main
```

Организация стека как библиотеки

stack.h:

```
struct stack;   // forward declaration
extern void push (struct stack *, char);
extern char pop (struct stack *);
extern int isempty (struct stack *);
extern struct stack* new_stack (void);
extern void free_stack (struct stack *);
```

stack.c:

```
#include "stack.h"
struct stack {
  <...>
};
void push (struct stack *stack, char c ) {
  if (stack->sz == stack->sp + 1) <...>
}
<...>
```

Организация стека как библиотеки

stack.c:

```
struct stack* new_stack (void) {
  struct stack *s = malloc (sizeof (struct stack));
  *s = (struct stack) { .sp = -1, .sz = 0, .stack = NULL };
  return s;
}
void free_stack (struct stack *s) {
  free (s->stack);
  free (s);
}
```

main.c:

```
#include "stack.h"
int main (void) {
  struct stack *s = new_stack ();
  <...push (s, c), pop (s), ...>
  free_stack (s);          <...>
}
```

Очередь

Очередь (queue) — это линейный список информации, работа с которой происходит по принципу FIFO.
Для списка можно использовать статический массив: количество элементов массива (`MAX`) — наибольшей допустимой длине очереди.

Работа с очередью осуществляется с помощью двух функций:

- `qstore()` — поместить элемент в конец очереди;
- `qretrieve()` — удалить элемент из начала очереди;

и двух глобальных переменных:

- `spos` — индекс первого свободного элемента очереди, его значение `<` `MAX`;
- `rpos` — индекс очередного элемента, подлежащего удалению: “кто первый?”

Пример реализации

```
int queue[MAX]; //  Заведите enum
int spos = 0, rpos = 0;

int qstore (int q) {
  if (spos == MAX) {
    /*  Можно расширить очередь, см. реализацию стека */
    printf ("Очередь переполнена\n");
    return 0;
  }
  queue[spos++] = q;
  return 1;
}

int qretrieve (void) {
  if (rpos == spos) { //  Очередь пуста
    return -1;
  }
  return queue[rpos++];
}
```

Улучшение — «зацикленная» очередь

```
int queue[MAX]; //  Заведите enum
int spos = 0, rpos = 0;

int qstore (int q) {
  if (spos + 1 == rpos
      || (spos + 1 == MAX && !rpos) {
    printf ("Очередь␣переполнена␣\n");
    // Дома. Реализуйте очередь на динамическом массиве.
    return 0;
  }
  queue[spos++] = q;
  if (spos == MAX)
    spos = 0;
  return 1;
}
```

Улучшение — «зацикленная» очередь

```
int queue[MAX]; //  Заведите enum
int spos = 0, rpos = 0;

int qretrieve (void) {
  if (rpos == spos) {
    printf ("Очередь␣пуста␣\n");
    return -1;
  }
  if (rpos == MAX - 1) {
    rpos = 0;
    return queue[MAX – 1];
  }
  return queue[rpos++];
}
```

Зацикленная очередь переполняется, когда **spos** находится непосредственно перед **rpos**, так как в этом случае запись приведёт к **rpos == spos**, т.е. к пустой очереди.

Списки

Односвязный список — это динамическая структура данных, каждый элемент которой содержит ссылку на следующий элемент (либо **NULL**, если следующего элемента нет).
Доступ к списку осуществляется с помощью указателя на его первый элемент.

```
struct list {
  struct data info;  /* Данные */
  struct list *next; /* Ссылка на след. элемент */
};
```

Выделение элемента:

```
struct list *phead = NULL;
phead = (struct list *) malloc (sizeof (struct list));
```

Списки: добавление элемента в начало

```
struct list *phead = NULL;
struct list *add_element (struct list *phead,
  struct data *elem) {
  struct list *new = malloc (sizeof (struct list));
  new->info = *elem;
  new->next = phead;
  return new;
}
```

Списки: добавление элемента в конец

```
struct list *phead = NULL;
struct list *add_element (struct list *phead,
  struct data *elem) {
  if (! phead) {
    phead = malloc (sizeof (struct list));
    phead->info = *elem;
    phead->next = NULL;
    return phead;
  }
  struct list *ph = phead; // сохраним голову
  while (phead->next != NULL)
    phead = phead->next;
  phead->next = malloc (sizeof (struct list));
  phead->next->info = *elem;
  phead->next->next = NULL;
  return ph; // phead затёрт, вернём сохранённый указатель
}
```

Списки: поиск элемента

```
struct list * phead;

int equals (struct data *, struct data *);
struct list * search (struct list *phead,
  struct data *elem) {
  while (phead && ! equals (&phead->info, elem))
    phead = phead->next;
  return phead;
}
```

Списки: удаление элемента

```
struct list *remove (struct list *phead,
  struct data *elem) {
  struct list *prev = NULL, *ph = phead;
  while (phead && ! equals (&phead->info, elem)) {
    prev = phead;
    phead = phead->next;
  }
  if (! phead)
    return ph;
  if (prev)
    prev->next = phead->next;
  else
    ph = phead->next;
  free (phead);
  return ph;
}
```

Списки: удаление элемента (двойной указатель)

```
void remove (struct list **pphead,
  struct data *elem) {
  struct list *prev = NULL, *phead = *pphead;
  while (phead && ! equals (&phead->info, elem)) {
    prev = phead;
    phead = phead->next;
  }
  if (! phead)
    return;
  if (prev)
    prev->next = phead->next;
  else
    *pphead = phead->next;
  free (phead);
}
```

Дома. Напишите добавление элемента с двойным указателем.

Топологическая сортировка узлов ациклического ориентированного графа

Ациклический граф можно использовать для графического изображения *частично упорядоченного множества*.

Цель топологической сортировки: преобразовать частичный порядок в линейный. Графически это означает, что все узлы графа нужно расположить на одной прямой таким образом, чтобы все дуги графа были направлены в одну сторону.

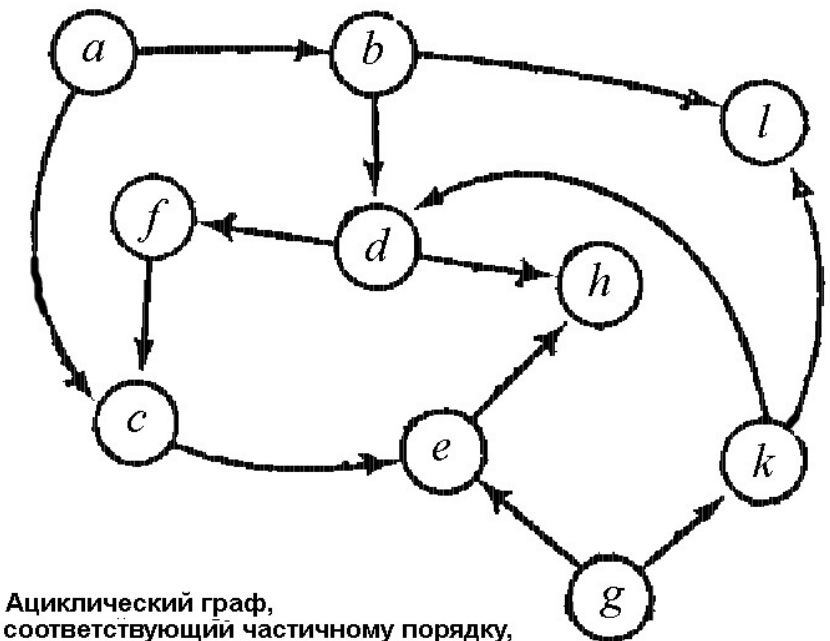
Топологическая сортировка узлов ациклического ориентированного графа

Пример. Частичный порядок ($<$) задается следующим набором отношений:

$$a < b, b < d, d < f, b < l, d < h, f < c, a < c,$$

$$c < e, e < h, g < e, g < k, k < d, k < l.$$

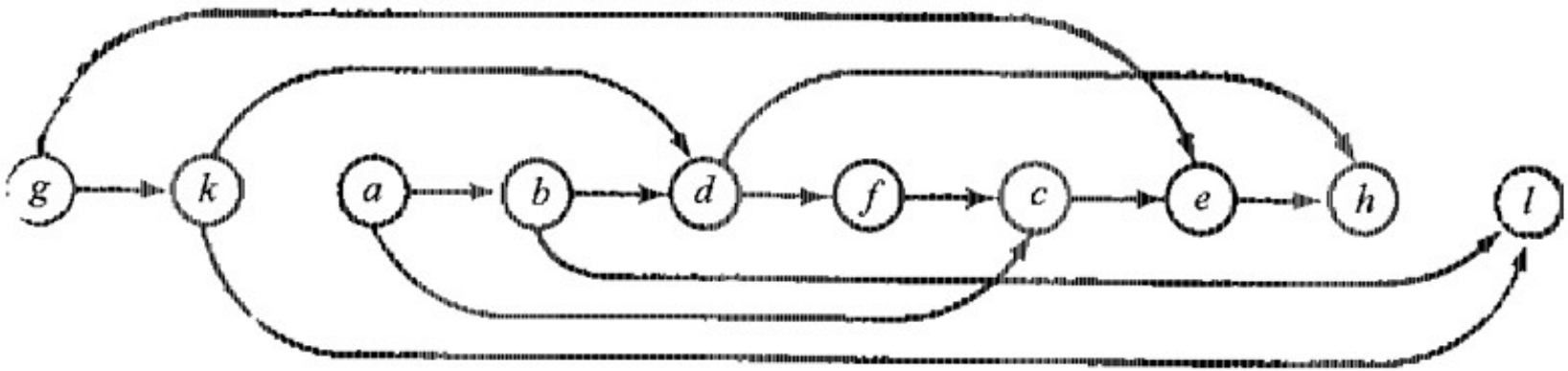
Его можно представить в виде такого графа:



Ациклический граф, соответствующий частичному порядку, заданному набором отношений (*).

Топологическая сортировка узлов ациклического ориентированного графа

Требуется привести рассматриваемый граф к линейному графу:



На этом графе ключи расположены в следующем порядке:

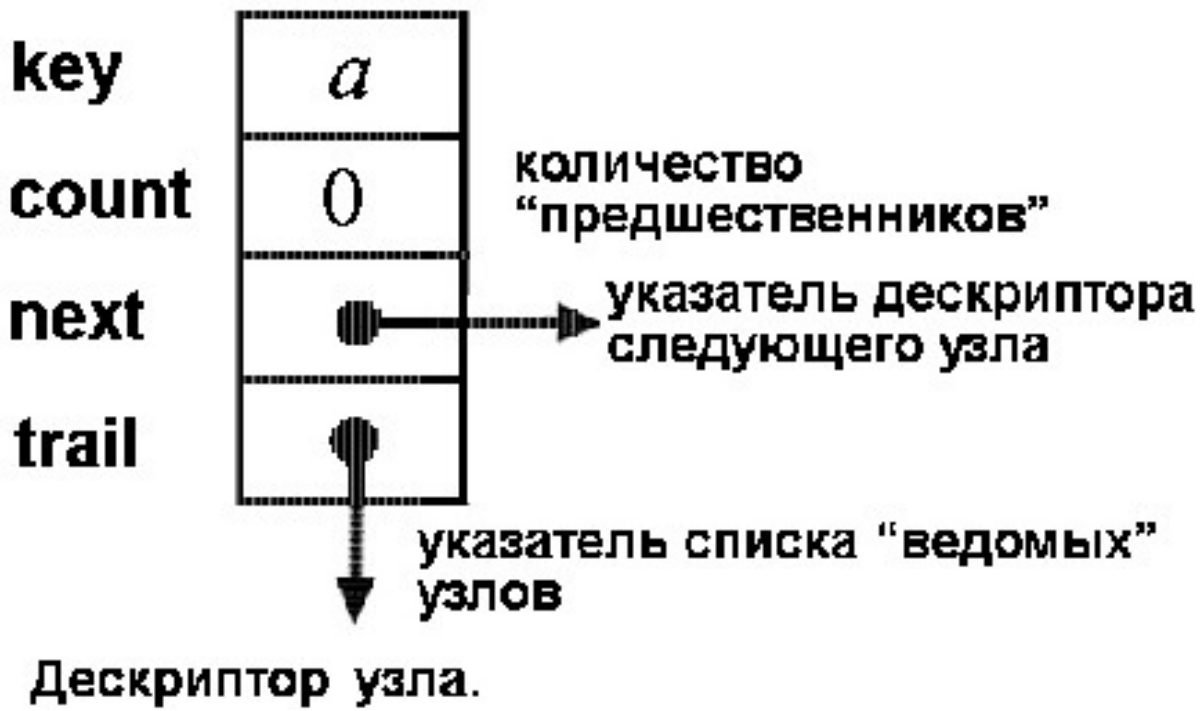
$$g, k, a, b, d, f, c, e, h, l.$$

(поскольку топологическая сортировка неоднозначна, это один из возможных топологических порядков).

Последовательная обработка полученного линейного списка узлов графа эквивалентна их обработке в порядке обхода графа.

Структуры данных для представления узлов

Каждый узел исходного графа представляется с помощью дескриптора узла, который имеет вид:

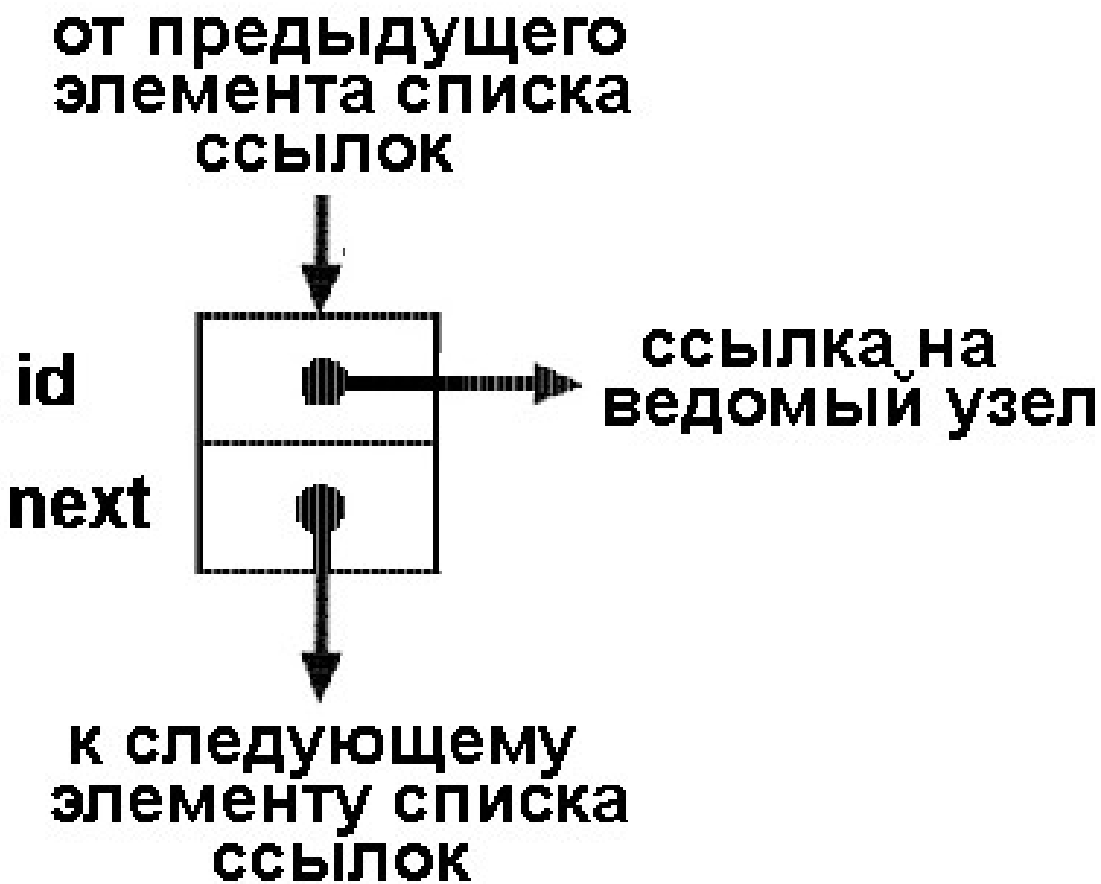


Дескриптор узла.

Ведомыми для узла n будут узлы, для которых n является предшественником. Каждый узел графа (не только ведущий) может иметь один или несколько ведомых узлов.

Структуры данных для представления узлов

Дескриптор каждого узла содержит ссылки на ведомые узлы. Так как заранее неясно, сколько у узла будет ведомых узлов, эти ссылки помещаются в список. На рисунке представлен элемент списка ссылок.

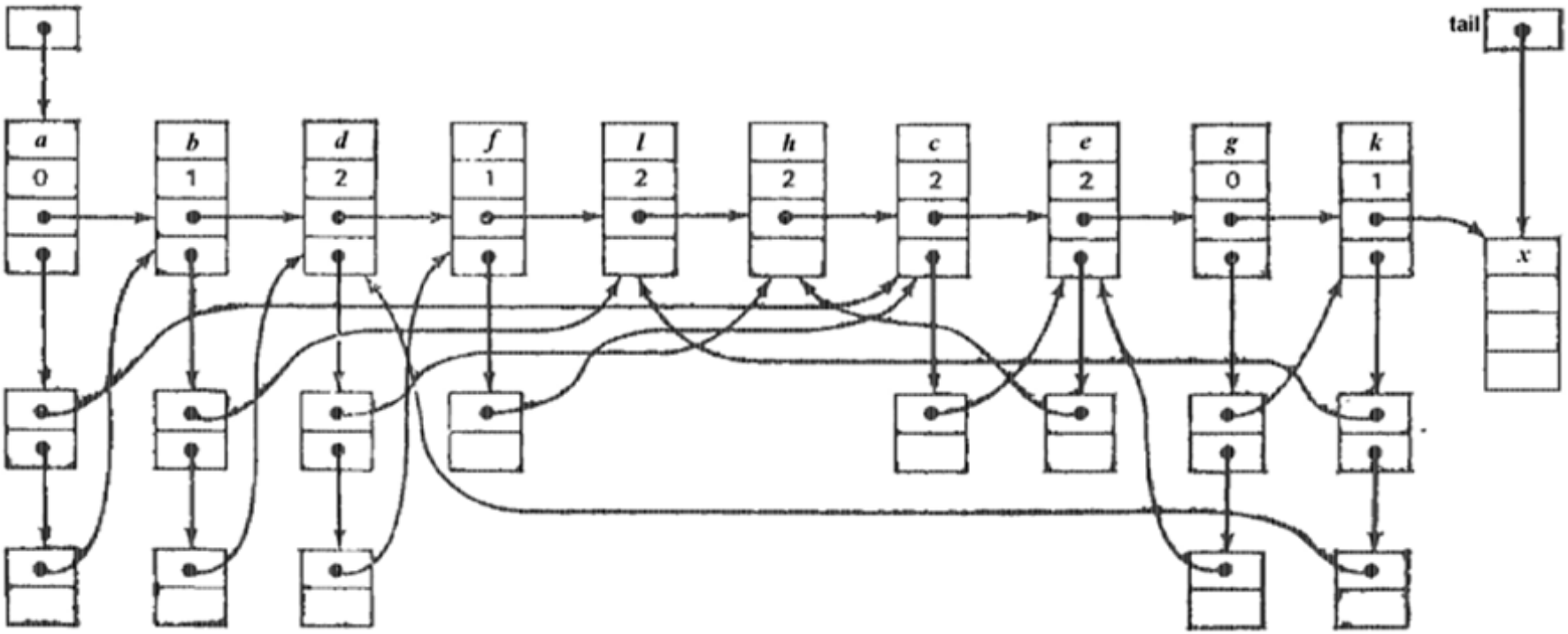


Первая фаза алгоритма: ввод исходного графа

На этой фазе вводятся пары ключей и из них формируется представление ациклического графа через дескрипторы узлов и списки ведомых узлов.

- Исходные данные представлены в виде множества пар ключей (*), которые вводятся в произвольном порядке.
- После ввода очередной пары $x < y$ ключи x и y ищутся в списке «ведущих» и в случае отсутствия добавляются к нему.
- В список ведомых узлов узла x добавляется ссылка на y, а счётчик предшественников y увеличивается на 1 (начальные значения всех счетчиков равны 0).

Пример: результат первой фазы



Вторая фаза алгоритма: сортировка

- В списке «ведущих» находим дескриптор узла z, у которого значение поля count равно 0.
- Включаем узел z в результирующую цепочку.
- Если у узла z есть «ведомые» узлы (значение поля `trail` не `NULL`):
  - просматриваем очередной элемент списка «ведомых» узлов;
  - корректируем поле `count` дескриптора соответствующего «ведомого» узла.
- Переходим к шагу 1.

Так как с каждой коррекцией поля `count` его значение уменьшается на 1, постепенно все узлы включаются в результирующую цепочку.

Топологическая сортировка на Си

```
#include <stdio.h>
#include <stdlib.h>
typedef struct ldr { /* дескриптор ведущего узла */
  char key;
  int count;
  struct ldr *next;
  struct trl *trail;
} leader;
typedef struct trl { /* дескриптор ведомого узла */
  struct ldr *id;
  struct trl *next;
} trailer;

leader *head, *tail; /* два вспомогательных узла */
int lnum; /* счётчик ведущих узлов */
```

Топологическая сортировка на Си: поиск по ключу

```
leader *find (char w) {
  leader *h = head;
  /* барьер на случай отсутствия w */
  tail->key = w;
  while (h->key != w)
    h = h->next;
  if (h == tail) {
    /* генерация нового ведущего узла */
    tail = malloc (sizeof (leader));
    /* старый tail становится новым элементом списка */
    lnum++;
    h->count = 0;
    h->trail = NULL;
    h->next = tail;
  }
  return h;
}
```

Топологическая сортировка на Си: формирование списка

```
void init_list() {
  leader *p, *q;
  trailer *t;
  char x, y;

  head = (leader *) malloc (sizeof (leader));
  tail = head;
  lnum = 0;                  /* начальная установка */
  while (1) {
    if (scanf ("%c␣%c", &x, &y) != 2)
      break;
    /* включение пары в список */
    p = find (x);
    q = find (y);
    <...>
```

Топологическая сортировка на Си: формирование списка

```
      <...>
      /* коррекция списка */
      t = malloc (sizeof (trailer));
      t->id = q;
      t->next = p->trail;
      p->trail = t;
      q->count += 1;
    }
}
```

Топологическая сортировка на Си: новый список

```
void sort_list() {
  leader *p, *q;
  trailer *t;
  /*  В выходной список включаются все узлы с count == 0 */
  p = head;
  head = NULL; /* голова выходного списка */
  while (p != tail) {
    q = p;
    p = q->next;
    if (q->count == 0) {
      /* включение q в выходной список */
      q->next = head;
      head = q;
    }
  }
<...>
```

Топологическая сортировка на Си: новый список

```
q = head; /* есть ведущий узел -> head != NULL */
while (q != NULL) {
  printf ("%c\n", q->key);
  lnum--;
  t = q->trail;
  q = q->next;
  while (t != NULL) {
    p = t->id;
    p->count -= 1;
    if (p->count == 0) {
      p->next = q;  // достаточно для
      q = p;        // правильной сортировки
    }
    t = t->next;
  }
}
/* lnum == 0 */
```

Топологическая сортировка на Си: основная функция

```
int main (void) {
  init_list ();
  sort_list ();
  return 0;
}
```

Дома. Что поменяется, если узлы идентифицируются не одним символом, а именем (строкой)? Сделайте нужные изменения в коде. Добавьте определение циклов в исходных данных.