

Курс «Алгоритмы и алгоритмические языки»

Лекция 11

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 конец

```
struct list *phead = NULL;
```

```
struct list *add_element (struct list *phead, struct  
                           data *elem) {
```

```
    if (! phead) {
```

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

```
        phead->info = *elem;
```

```
        phead->next = NULL;
```

```
        return phead;
```

```
    }
```

```
    while (phead->next != NULL)
```

```
        phead = phead->next;
```

```
    phead->next = (struct list *) malloc (sizeof (struct list));
```

```
    phead->next->info = *elem;
```

```
    phead->next->next = NULL;
```

```
    return phead;
```

```
}
```

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;
}
```

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.

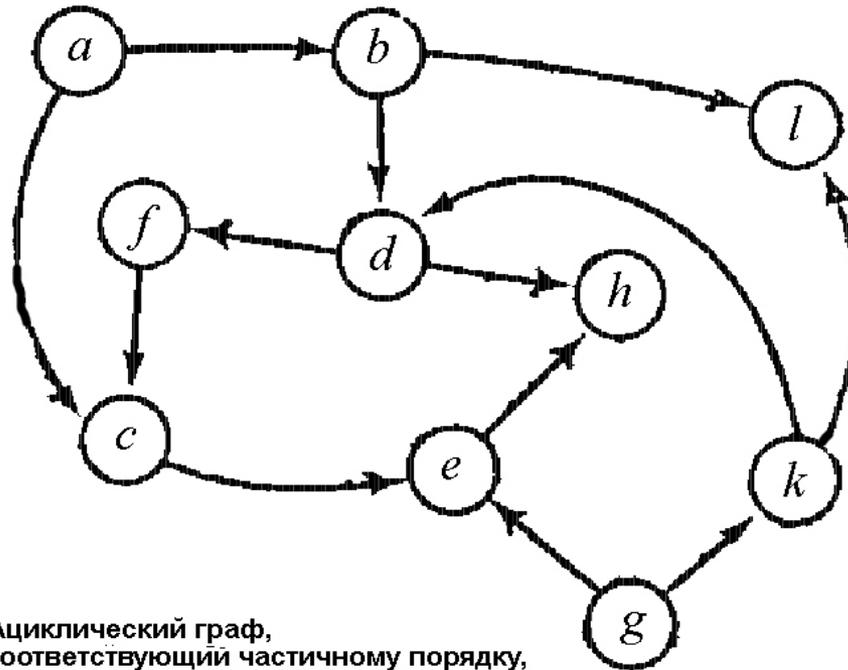
Топологическая сортировка узлов ациклического ориентированного графа

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

$$a < b, b < d, d < f, b < l, d < h, f < c, a < c, \quad (*)$$

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

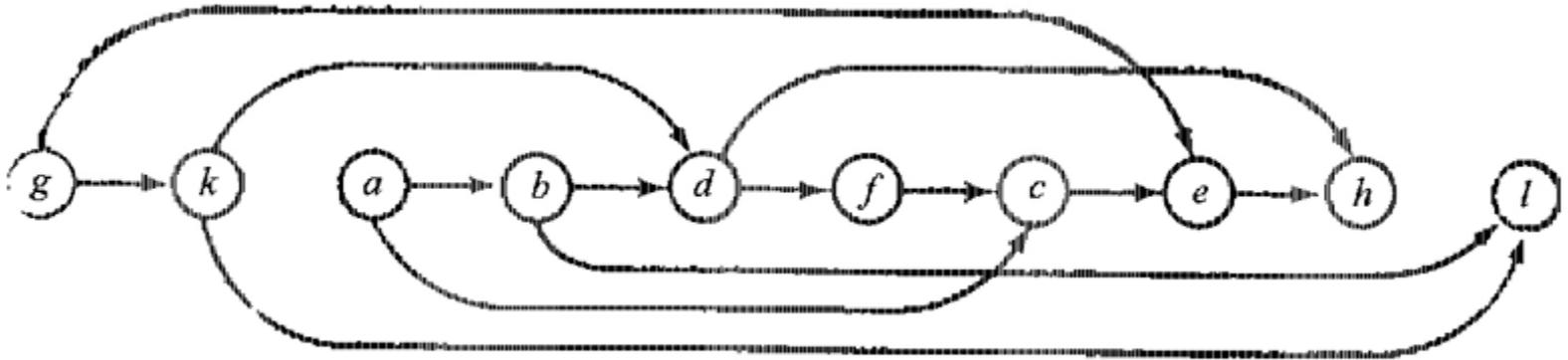
Набор отношений (*) можно представить в виде следующего ациклического графа (см. рисунок):



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.

Топологическая сортировка узлов ациклического ориентированного графа

- ◇ Поскольку рассматриваемый граф – ациклический, **существует** хотя бы один узел графа, у которого нет предшествующих узлов. Каждый такой узел будем называть *ведущим* узлом. **Шаг алгоритма:** Выберем один из ведущих узлов и поместим его в начало линейного списка отсортированных узлов, удалив его из исходного графа.
- ◇ Поскольку граф, у которого удалили один из ведущих узлов, останется ациклическим, в нем будет хотя бы один ведущий узел. Следовательно, можно повторить шаг алгоритма.
- ◇ Легко видеть, что каждый узел графа рано или поздно станет ведущим и попадет в формируемый линейный список, и алгоритм завершится через несколько шагов.

Топологическая сортировка узлов ациклического ориентированного графа

- ◇ Структуры данных для представления узлов:
 - ◇ Каждый узел исходного графа представляется с помощью дескриптора узла, который имеет вид:

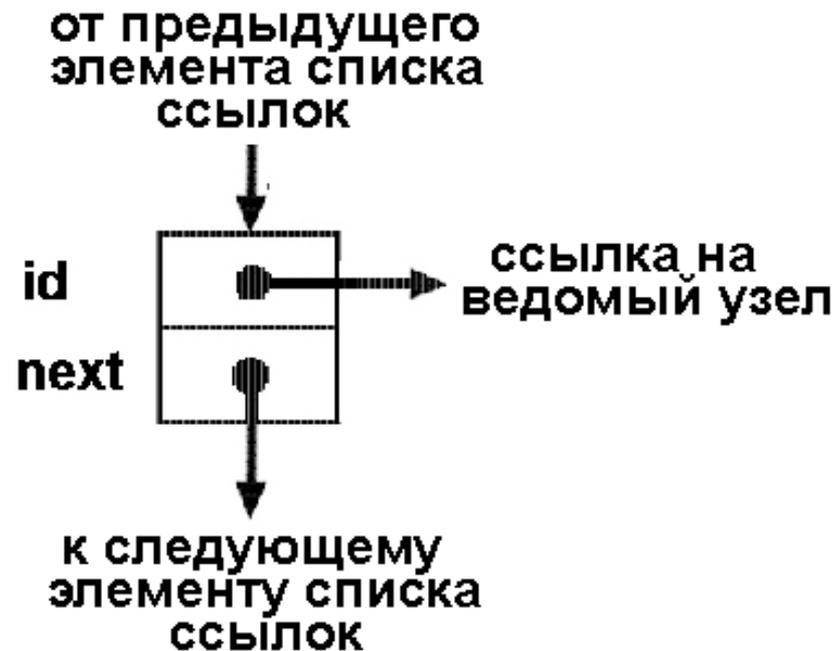


Дескриптор узла.

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

Топологическая сортировка узлов ациклического ориентированного графа

- ◆ 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.



Топологическая сортировка узлов ациклического ориентированного графа

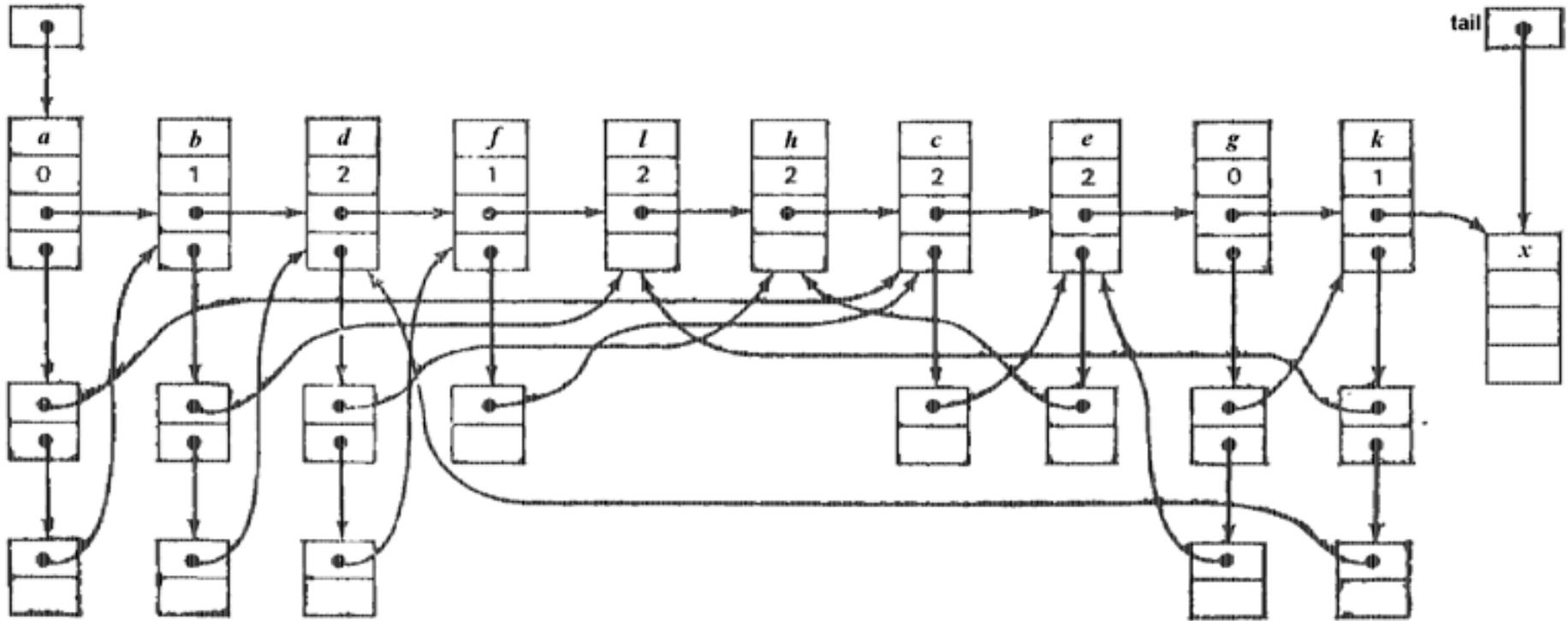
♦ *1 фаза алгоритма: ввод исходного графа*

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 счетчик предшественников y увеличивается на 1 (начальные значения всех счетчиков равны 0).

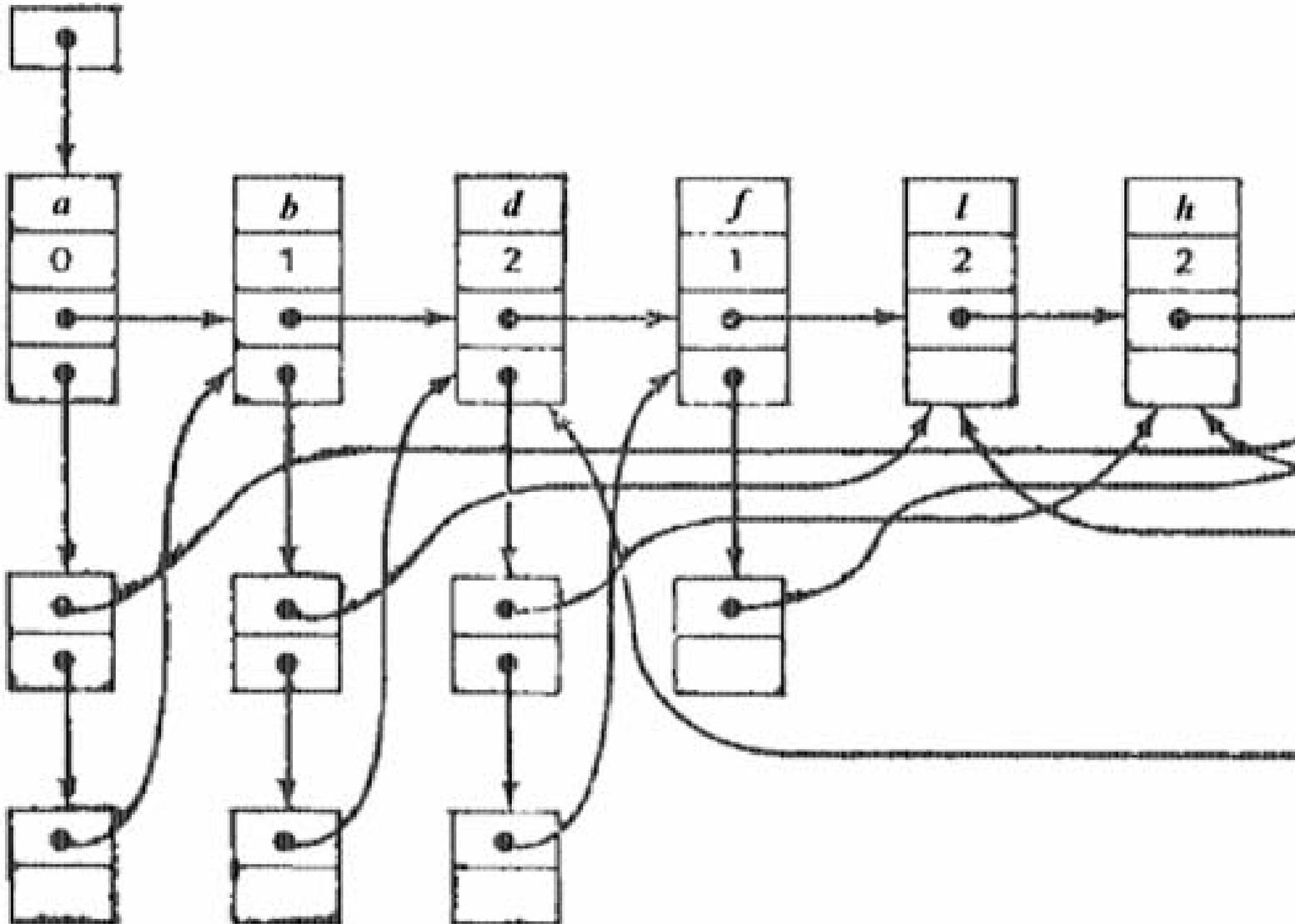
По окончании фазы ввода будет сформирована структура, показанная на следующем слайде (для множества пар ключей (*)).

Топологическая сортировка узлов ациклического ориентированного графа

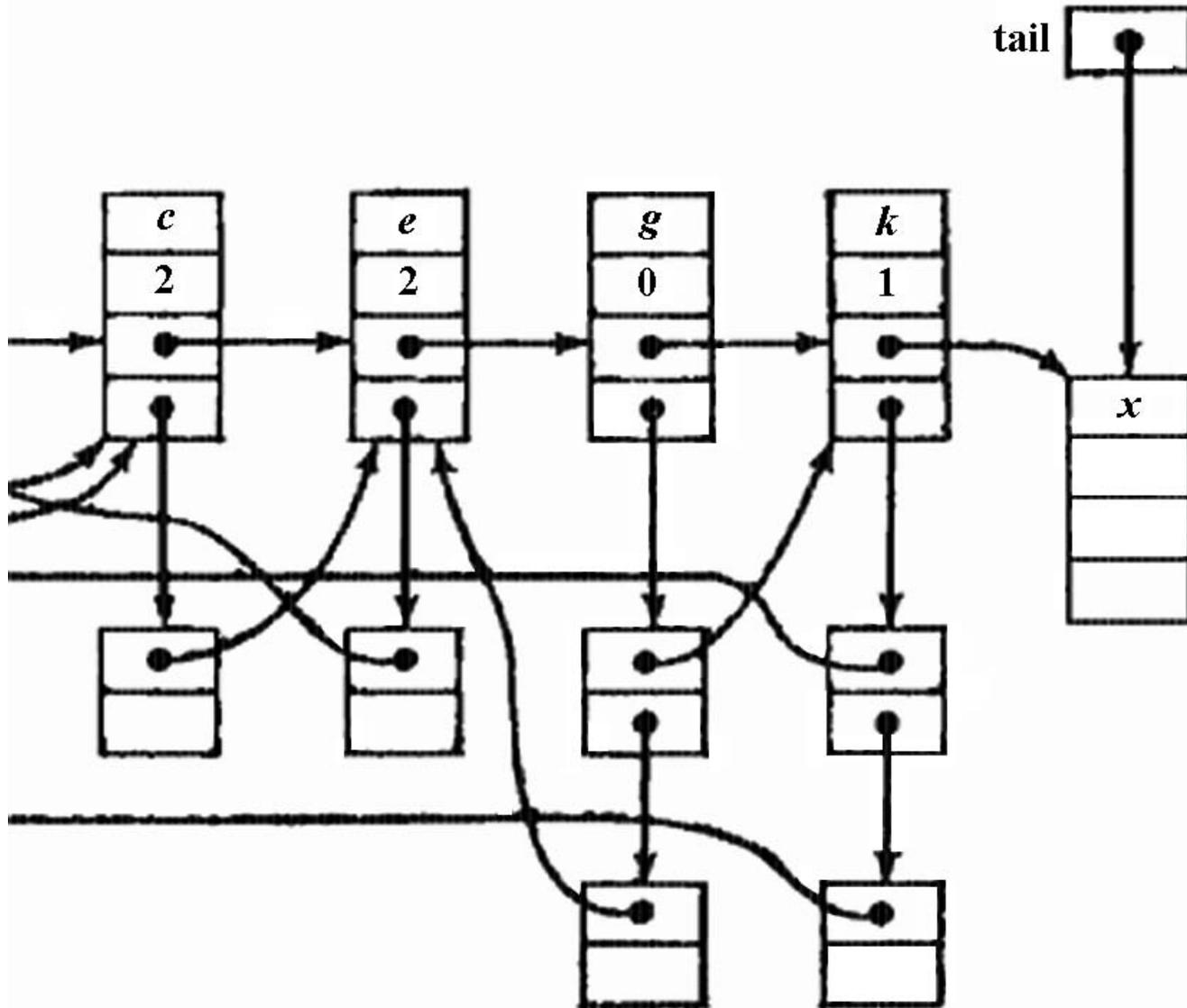


Структура данных, сформированная фазой ввода

Топологическая сортировка узлов ациклического ориентированного графа



Топологическая сортировка узлов ациклического ориентированного графа



Топологическая сортировка узлов ациклического ориентированного графа

◇ *2 фаза алгоритма: сортировка*

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

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

Описание алгоритма топологической сортировки на языке 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;                /*счетчик ведущих узлов*/
```

Описание алгоритма топологической сортировки на языке Си

```
/* поиск по ключу w */
```

```
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 = (leader *) malloc (sizeof (leader));  
        /*старый tail становится новым элементом списка */  
        lnum++;  
        h->count = 0;  
        h->trail = NULL;  
        h->next = tail;  
    }  
    return h;  
}
```

Описание алгоритма топологической сортировки на языке Си

```
void init_list() {
    /* инициализация списка «ведущих» */
    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);
        /* коррекция списка */
        t = (trailer *) malloc (sizeof (trailer));
        t->id = q;
        t->next = p->trail;
        p->trail = t;
        q->count += 1;
    }
}
```

Описание алгоритма топологической сортировки на языке Си

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

Описание алгоритма топологической сортировки на языке Си

```
/* Фаза сортировки и вывода результатов из нового списка */
```

```
<...>
```

```
    q = head;
    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;
                q = p;
            }
            t = t->next;
            /* здесь можно удалить ведомый элемент */
        }
    }
    /* lnum == 0 */
}
```

Описание алгоритма топологической сортировки на языке Си

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