Лекция 7

Фундаментальные компьютерные алгоритмы (продолжение)

План лекции

Методы быстрой сортировки

- Быстрая сортировка с разделением и перестановками

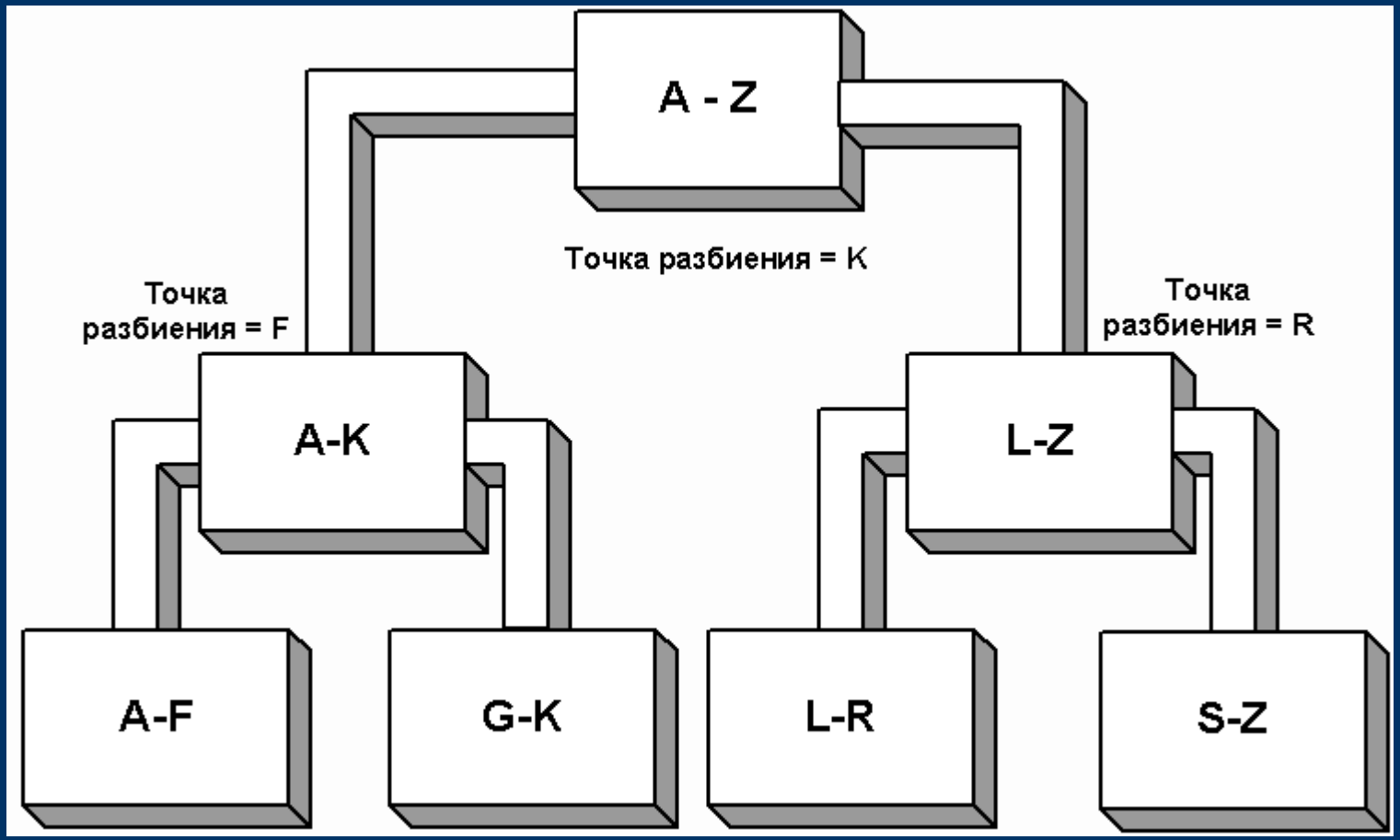
Поиск

- Последовательный
- Бинарный
- Поиск с хэшированием

Структуры данных

связные списки

«Быстрая» сортировка



«Быстрая» сортировка

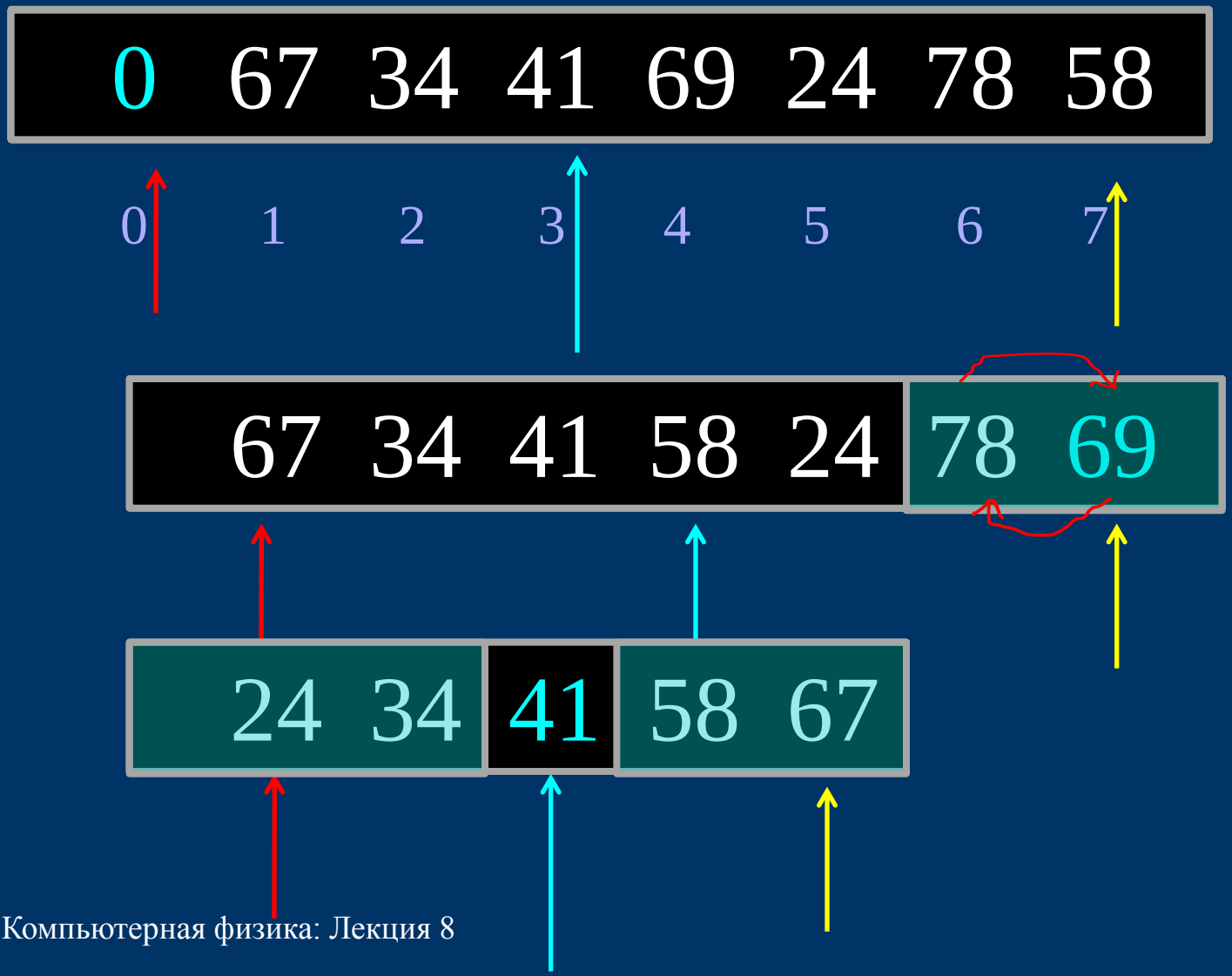
Идея алгоритма:

1. В массиве выбирается некоторый элемент, который называется опорным. Например, средний по положению.

2. Разделение массива: перестановка элементов, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него.

3. Для упорядочивания подмассивов, лежащих слева и справа от опорного элемента можно использовать рекурсию. Базой рекурсии являются наборы из одного или двух элементов. Первый возвращается без изменения, во втором, при необходимости, элементы переставляются.

«Быстрая» сортировка – иллюстрация алгоритма



«Быстрая» сортировка - шаги алгоритма

•Пусть low – наименьший индекс массива, а high – наибольший. Тогда опорный элемент имеет индекс m = (low+high)/2.

•Присваиваем l = low и r = high.

•Индекс l последовательно увеличивается до m до тех пор, пока l-й элемент не превысит опорный.

•Индекс r последовательно уменьшается до m до тех пор, пока r-й элемент не окажется меньше либо равен опорному.

•Если l < r — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r, которые были достигнуты. Если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r-й или l-й элемент соответственно.

•Если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент.

Функция «Быстрой» сортировки с рекурсией

```
void qSort(int a[], int low, int high)
{
   int l, r, x, t;
   l = low;
   r = high;
   x = a[(high+low)/2];
   do{
      while (a[l] < x) l++;
      while (x < a[r]) r--;
      if ( l <= r )
      {
         if( l < r )
         {
            t = a[l];
            a[l] = a[r];
            a[r] = t;
         }
         l++;
         r--;
      }
   } while (l <= r);
   if (low < r) qSort(a, low, r);
   if (l < high) qSort(a, l, high);
}
```

Поиск

	Ключ	Ключ	
№	ФИО	ID	Возраст, лет
1	Иванов	11	45
2	Петров	5	34
3	Агеев	9	23
4	Моторин	14	41

Агеев ? 9 ?

Последовательный поиск на неупорядоченном массиве

```
#define N 4
int id[N], i;
```

Ищем номер элемента массива id со значением v

11
5
9
14

```
i = 0;
while(id[i]!=v && i<N) i++;
if (id[i]==v) printf("\n i =%d",i);
else printf("\n Не найдено");
```

«Быстрый» последовательный поиск

```
#define N 4
int id[N+1], i;
```

Ищем номер элемента массива id со значением v

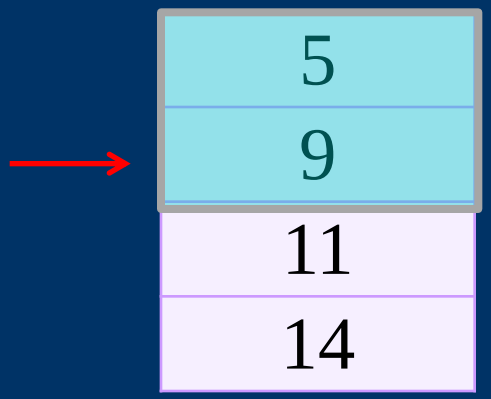
11
5
9
14
v

```
i = 0;
while(id[i]!=v) i++;
if (i < N) printf("\n i =%d", i);
else printf("\n Не найдено");
```

Бинарный поиск в упорядоченном массиве

Ищем номер элемента массива id
со значением v, иначе -1

```
#define N 4
int id[N], i, j, m;
```



Пусть v=10

$M \sim Log_2 N$

```
int bseek(int id[], int N, int v)
{
  int i = 0, j = N;
  int m;
  m = (i+j)/2;
  while (id[m]!=v)
  {
     if (i >= j) return -1;
     if (id[m] < v) i=m+1;
     else j=m-1;
     m=(i+j)/2;
  }
  return m;
}
```

Индексный файл

Ключ

№	ФИО	ID	Возраст, лет
1	Иванов	11	45
2	Петров	5	34
3	Агеев	9	23
4	Моторин	14	41

ID	№ записи
5	2
9	3
11	1
14	4

Бинарный поиск

Ассоциативный поиск

	ID	№ записи
1	5	2
2	9	3
3	11	1
4	14	4

Соответствие данных и адреса – ассоциативный доступ

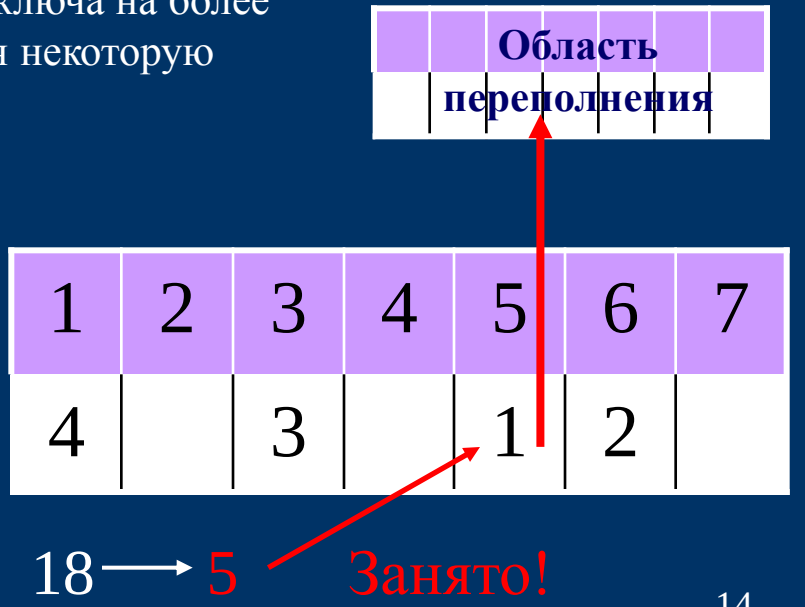
Адрес (номер элемента =ID)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
№ записи					2				3		1			4

Хэширование

Адрес (номер элемента =ID)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
№ записи					2				3		1			4

Типично: разреженность в хранении данных

Решение: отобразить разреженное множество значений ключа на более плотное множество (хэширование), например, используя некоторую функцию – хэш-функция.



Пример: Новый адрес = (старый адрес)%7 + 1

5 → 6 9 → 3

11 → 5 14 → 1

Поиск последовательности элементов

Пример:

«янной»

«нахождение постоянной подстроки длины *m* в тексте (строке) длины *n*»

```
int subseek(char* str, char* sub)
{
    int i, j, ret = -1;
    for (i=0; i<strlen(str)-strlen(sub); i++)
    {
        for (j=0; j<strlen(sub); j++) if(str[i+j]!= sub[j]) break;
        return i;
    }
    return -1;
}
```

Алгоритм Рабина-Карпа

Временные проблемы: «аааааааб»

«ааб»

Алгоритм Р-К: Сравниваются хэш-функции строк (числа)

```
int R-K(char* str, char* sub)
{
    int hstr, hsub, len;
    len = strlen(sub);
    hsub = hash(sub, len);
    for (i=0; i<strlen(str)-len; i++)
    {
        hstr = hash(str+i, len);
        if(hstr == hsub) return i;
    }
    return -1;
}
```

Кольцевой хэш

Время: Затраты на вычисление хэш-функции

«ааааааб»
«аааб»

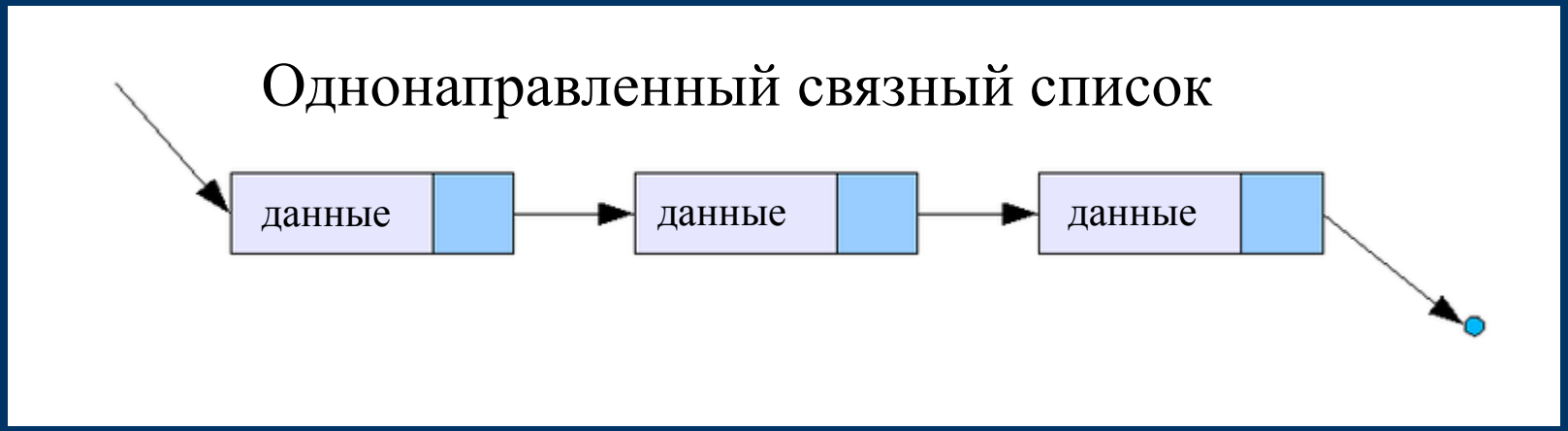
Выход: Вычисление очередного хэш-значения на основе предыдущего

Специальная хэш-функция или кольцевой хэш, например: hash = сумма всех символов (кодов) подстроки

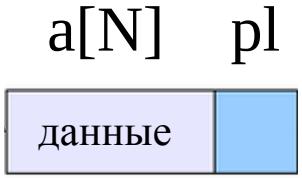
Тогда: hash(str+i+1, len) = hash(str+i,len)+str[i+len]-str[i];

Связные списки

Свя́зный спи́сок — структура данных, состоящая из узлов (элементов), каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка



Пример связного списка

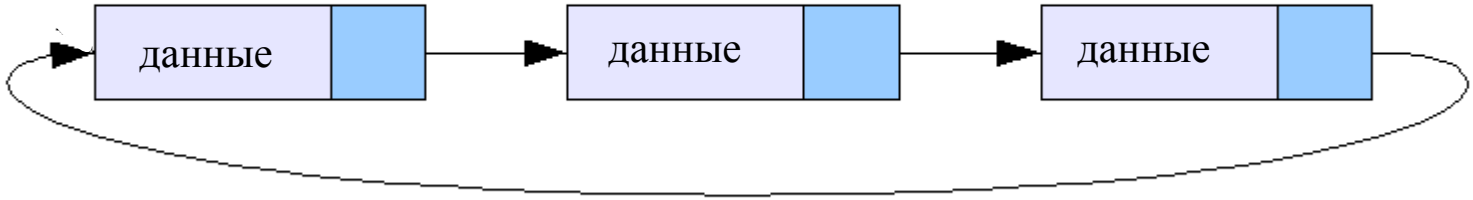



```
#define N 100
struct list
{
   int a[N];
   struct list* pl;
} s1;
```

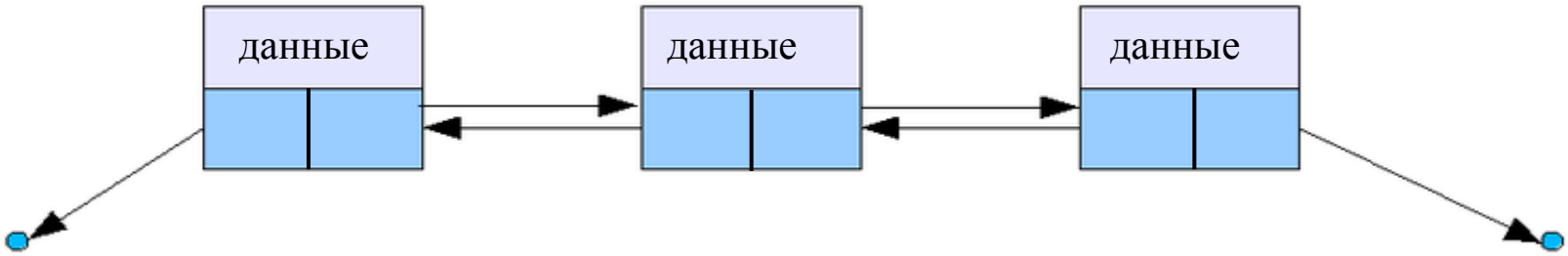
```
int main()
{
    struct list s1;
    …
```

Связные списки

Кольцевой однонаправленный связный список



Двунаправленный связный список



Литература по лекционному материалу

1. Н. Вирт. Алгоритмы и структуры данных. — М.: ДМК Пресс, 2014, 272 с.
2. Д. Кнут. Искусство программирования, том 3. Сортировка и поиск. — М. и др.: «Вильямс», 2014, 822 с.
3. Б. Керниган, Д. Ритчи. Язык программирования С. — М.: Вильямс, 2015, 288 с.