Алгоритми за сортиране и търсене

Лекция 2 по СДА, Софтуерно Инженерство
Зимен семестър 2019-2020г

План на днешната лекция

- Организационни въпроси.
- Алгоритми за сортиране
- Алгоритми за търсене

Резултати от входяща анкета

Времето на провеждане на лекциите удобно ли ви е и ще можете ли да ги посещавате?

Отговор	Средно	Общо
Да	88%	103
Не	12%	14
Total responses to question	100%	117/117

Работите ли?

Отговор	Средно	Общо
Да	18%	21
Не	82%	96
Total responses to question	100%	117/117

Разполагате ли с лаптоп, който да може да носите за лекциите по структури от данни и алгоритми?

Отговор	Средно	Общо
Да	100%	117
Не		0
Total responses to question	100%	117/117

Подготвени ли сте за курса?
За да отговорите положително трябва като минимум да знаете какво е масив, как да използвате операторите for и if, как да получавате и записвате данни от конзола или файл, как да дефинирате клас/структура, какво е указател и как се работи с него.

Отговор	Средно	Общо
Да	95%	111
Не	5%	6
Total responses to question	100%	117/117

Известен ли ви е сайта https://www.hackerrank.com/ и решавали ли сте задачи в него?

Отговор	Средно	Общо
Да	29%	34
Не	71%	83
Total responses to question	100%	117/117

Занимавали ли сте се със състезателно програмиране?

Отговор	Средно	Общо
Да	7%	8
Не	93%	109
Total responses to question	100%	117/117

Мислите ли, че представеният курс ще ви е полезен за професионалната ви реализация?

Отговор	Средно	Общо
Да	100%	117
Не		0
Total responses to question	100%	117/117

Ще можете ли да отделяте в допълнение към 3-те часа лекции и 2-та часа упражнения още 3-4 часа на седмица за самостоятелна подготовка вкъщи?

Отговор	Средно	Общо
Да	98%	115
Не	2%	2
Total responses to question	100%	117/117

Интересувате ли се от летен стаж в голяма технологична компания?

Отговор	Средно	Общо
Да	89%	104
Не	11%	13
Total responses to question	100%	117/117

Самостоятелното писане на програмен код е от изключителна важност за развитие на уменията за програмиране и решаване на алгоритмични проблеми.

С участието си в този курс се съгласявате да не участвате в схеми за преписване или измама. С подобни действия в дългосрочен план ще ощетите хората, на които евентуално краткосрочно "помагате". Ако искате да помогнете на някой обяснявайте му извън лекции, как се решават задачите, но не му ги решавайте. Потвърждавате ли своето съгласие?

Отговор	Средно	Общо
Да	100%	117
Не		0
Total responses to question	100%	117/117

Обратна връзка

8 попълнени форми - благодаря на всички за отделеното време.

Избрани коментари:

- “Повече примери”
- “Групи за напреднали и студенти, които имат повече нужда от помощ”

Алгоритми за сортиране

Какво ще научим?

1. Какво е сортиране и за какво ни е необходимо?
2. “Бавни” алгоритми за сортиране (bubble, selection, insertion)
3. “Бързи” алгоритми за сортиране (merge, quick)
4. Специфични алгоритми за сортиране(count sort)

Какво е сортиране и за какво ни е необходимо?

- Основен клас алгоритми много често необходим за решаване на реални проблеми
- Добър пример за демонстриране на това какво е алгоритъм, как се изчислява сложност на алгоритъм, как да бъдем по-критични към това с каква сложност решаваме проблем.

Дефиниция на сортиране

Проблем: Да се напише процедура, която подрежда в нарастващ ред обекти подадени като вход.

Вход: масив с числа

Изход: масив със същите числа наредени в нарастващ ред.

Сортиране с метод на мехурчето

Основна идея:
Започвайки последователно от началният елемент до крайният сравняваме всеки елемент със следващия като ги разменяме ако не са подредени. По този начин на всяка стъпка изкарваме най-големият в края на правилното за него място. Повтаряйки този процедура толкова пъти колкото е големината на масива постигаме правилно подреждане на целият масив.

Визуален пример

https://visualgo.net/en/sorting

Сортиране с метод на мехурчето

```
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}
```

Сортиране с пряка селекция

Намаля броя на разменянията в сравнение с метода на мехурчето!

Основна идея: Търсим най-големият елемент в масива и директно го поставяме на последно място. След това следващият по големина и отново го поставяме на място и т.н. Докато всички се подредят.

Визуален пример

https://visualgo.net/en/sorting

Сортиране с пряка селекция

```
for (j = 0; j < n-1; j++){
	int iMin = j;
	for (i = j+1; i < n; i++){
		if (a[i] < a[iMin]){
			iMin = i;
		}
	}

	if (iMin != j){
		swap(a[j], a[iMin]);
	}
}
```

Сортиране с вмъкване

Основна идея: сортиране постепенно на все по-голяма част от масива, като обхождайки несортираната част всеки един елемент го поставяме в сортираната част с намиране на правилното за него място на което сортираният масив остава сортиран.

Визуален пример

https://visualgo.net/en/sorting

Сортиране с вмъкване

```
void insertionSort(int arr[]) {
   for (int i = 1; i < arr.length; i++) {
       int key = arr[i];
       j = i-1;
       while (int j >= 0 && arr[j] > key) {
           arr[j+1] = arr[j];
           j = j-1;
       }
       arr[j+1] = key;
   }
}
```

Сложност на сортиращите алгоритми

Бавни сортировки:

Bubble, Selection, Insertion - O(n^2)

Бързи сортировки:

Merge, Quick - O(n*log(n))

Сортиране чрез сливане(merge sort)

Основна идея: Ако имаме два сортирани масива то със линейна сложност може да ги влеем в един масив. Тогава ако разделим масива който искаме да сортираме на по-малки масиви и на всяка стъпка сливаме два по-малки масива в един голям то за log(N) стъпки ще слеем всички масиви до един масив, като всяка от стъпките е била линейна.

Визуален пример

https://visualgo.net/en/sorting

Сортиране със сливане реализация

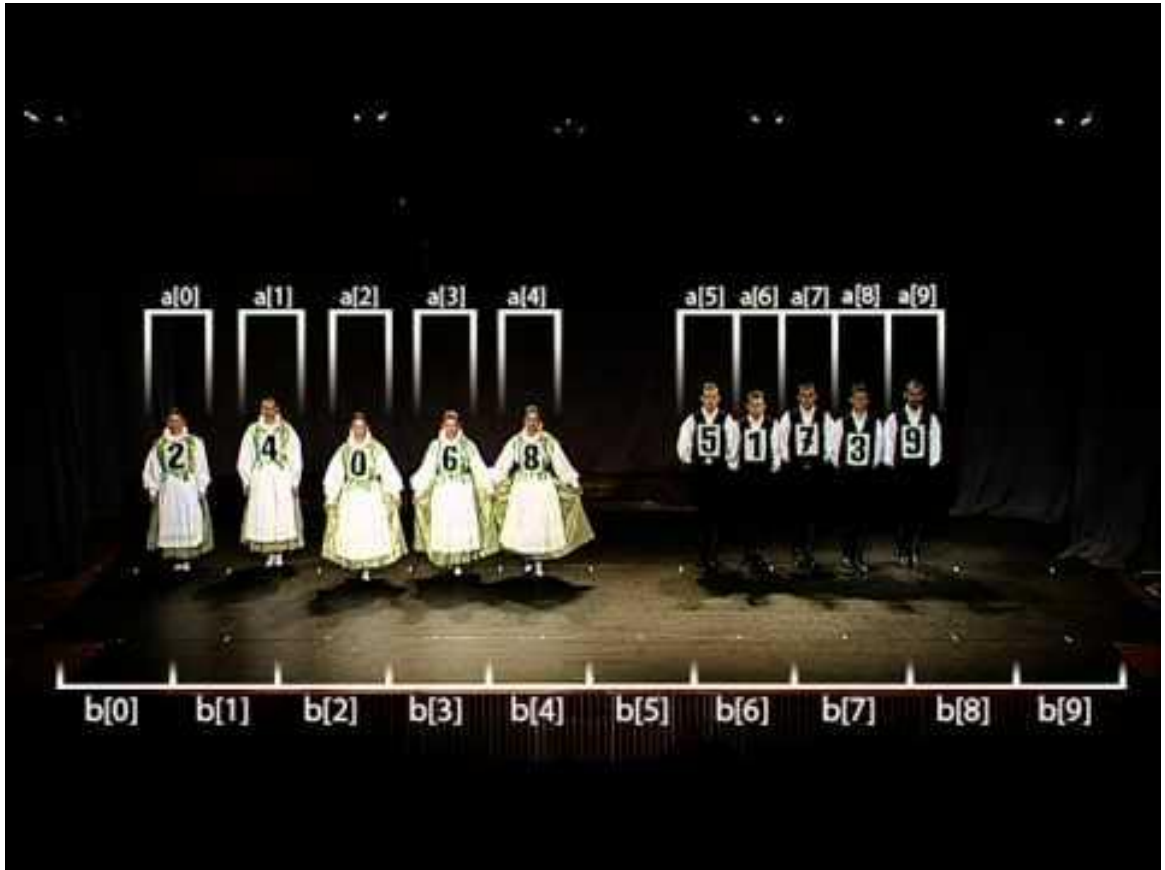
```
void mergesort(int arr[], int l, int r)
   {
      if (l < r) //гранично условие на рекурсията
      {
         m= (l+r)/2;
         mergesort(arr, l, m);
         mergesort(arr , m+1, r);
         merge(arr, l, m, r); //функция която слива два масива
      }
   }
```

Сортиране със сливане реализация(2)

```
void merge(int arr[], int start, int middle, int end){
    // Създаваме масивите arr1 и arr2, които съдържат частите, които ще копитаме. По този начин си освобождаваме основният масив за презаписване.

    i = 0, j = 0, k = start;
    while( i < arr1.length; j < arr2.length){
        // по малкото число от arr[i] и arr2[j] го записваме в  arr[k]
        // увеличаваме брояча на масива от който копирахме
        k++; // увеличаваме брояча за основният масив
    }
    // допълваме с всички останали необходени елементи от arr1 и arr2
}
```

Сортиране със сливане(Забавно видео)



Merge sort complexity

O(nlog(n)) - изчислителна сложност в най-лошият случай

O(n) - сложност по памет

Бързо сортиране (quick sort)

Основна идея: Ако вземем едно произволно число от масива, то с линейна сложност можем да прехвърлим всички по-малки числа от масива да са в ляво на числото, а всички по-големи в дясно.

При бързото сортиране избираме число от масива прехвърляме по-малите отляво, по-големите от дясно и след това изпълняваме същата процедура за лявата и дясната половина.

По тази процедура получаваме сложност в средният случай O(n*log(n))

Визуален пример

https://visualgo.net/en/sorting

Бързо сортиране реализация

```
void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            int pi = partition(arr, low, high);

            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }
```

Бързо сортиране реализация(2)

```
int partition(int arr[], int low, int high){
        int pivot = arr[high];
        int i = low; // index of smaller element
        for (int j=low; j<high; j++){
            if (arr[j] <= pivot){
                swap(arr, i, j)
                i++;
            }
        }

        swap(arr, i, high)
        return i;
    }
```

Сложност

Сложност в средният случай O(n.log(n)) ,но….

Сложност в най-лошият случай O(N^2) - когато масива е сортиран наобратно!

Рандомизирано Бързо Сортиране

Справя се с проблема, че точно определена редица прави сложността O (n^2), като използва произволно избиране на елемент за разделяне.

```
int random_partition(int arr[], int low, int high){
        swap(arr, high, random(arr.length));
        partition(arr,low, high);
}
```

Merge sort or quick sort

Merge sort изисква O(n) допълнителна памет, докато quicksort не изисква допълнителна памет.

Merge sort е със сложност в най-лошият случай O(nlog(n)), Quicksort -O(n^2)

Quicksort - по-бърз от mergesort за малки масиви, но mergesort е по-добър за големи масиви.

Сортиране с броене (Counting sort)

Можем ли да сортираме със сложност по-малка от O(n*log(n))?

Отговор: Да, но с добавяне на допълнителни ограничения.

При сортирането с броене сложността се определя от броя на различните елементи, които може да има в масива.

Сортиране с броене

Основна идея: Понеже имаме ограничен брой различни стойности в масива то може да преброим по колко пъти се среща всяка една от тези стойности(с едно обхождане на масива) и след това със второ обхождане да наредим стойностите по техният ред.

Стабилност на сортирането - ако имаме два елемента които са равни в първоначалният масив, то във финалният те се срещат във същият ред като първоначалният масив.

Визуален пример

https://visualgo.net/en/sorting

```
void conting_sort(char arr[],n) {
        char arr_copy[] = new char[arr.length];
        for (int i = 0; i<arr.length; ++i) {
            arr_copy[i] = arr[i];
        }
        int count[] = new int[256];
        for (int i=0; i<n; ++i) {
            count[arr[i]] = count[arr[i]]+1;
        }
        for (int i=1; i<=255; ++i) {
            count[i] += count[i-1];
        }
        // To make it stable we are operating in reverse order.
        for (int i = n-1; i>=0; i--) {
            arr[count[arr_copy[i]]-1] = [arr_copy[i];
            count[arr_copy[i]] = count[arr_copy[i]] - 1;
        }
}
```

Сложност на сортиране с броене

$O(n+k)$

Radix Sort (Допълнителен материал)

Основна идея - да използваме подход подобен на сортиране с броене, но да може да го прехвърлим и за големи числа.

При radix sort вместо да броим цели числа ще броим само цифри, като ще сортираме масива подред за всички позиции на цифри(единици,десетици, стотици, хиляди, десетохиляди, и т.н). Понеже сортирането с броене запазва подредбата веднъж сортирани числата по последна цифра, те си остават сортирани и при последващо сортиране по десетици и т.н. до последното сортиране.

https://visualgo.net/en/sorting

Обобщение и следващи стъпки

- Разгледахме основните алгоритми за търсене и сортиране
- От другият път започва изучаването на структури от данни
- Другият път(22.10.2019 от 17ч) ще се проведе контролно 1 върху темите сортиране и търсене