Binarni Radix sort

DN1

Binarni Radix sort

- V sklopu 1. domače naloge implementirajte algoritem Binarni Radix sort za urejanje poljubnih 8-bitnih (tj. podatkovni tip unsigned char) pozitivnih celih števil (v razponu [0, 255]), kot konzolno aplikacijo v programskem jeziku C++.
- **Binarni Radix sort**

    1. Od polja vhodnih števil *A* vzamite od vsakega števila *k*-ti bit (*k* je na začetku 0). Tako dobite polje bitov *D*.
    2. Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).
    3. Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).
    4. Indeks k inkrementirate in se vrnete na prvi korak. Postopek ponovite še 7-krat saj sortiramo 8-bitna števila.

Binarni Radix sort

- Vzemimo 8-bitna števila 14, 2, 5 in 12 iz vhodnega polja A in jih uredimo z binarnim Radix sort algoritmom:

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	5	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0

i:	0	1	2	3
A:	14	5	2	12

1. korak

- Od polja vhodnih števil *A* vzemite od vsakega števila *k*-ti bit (*k* je na začetku 0). Tako dobite polje bitov *D*.

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	5	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0

i:	0	1	2	3
A:	14	5	2	12
j:	0	1	2	3
D:	0	1	0	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	5	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0

i:	0	1	2	3
A:	14	5	2	12
j:	0	1	2	3
D:	0	1	0	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	5	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0

i:	0	1	2	3
A:	14	5	2	12
j:	0	2	3	1
D:	0	0	0	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	5	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0

i:	0	1	2	3
A:	14	5	2	12
j:	0	2	3	1
D:	0	0	0	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

Binarni zapis števil	A: \ k:	7	6	5	4	3	2	1	0
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1

i:	0	2	3	1
A:	14	2	12	5
j:	0	2	3	1
D:	0	0	0	1

4. korak

- Indeks k inkrementirate in se vrnete na prvi korak. Postopek ponovite še 7-krat saj sortiramo 8-bitna števila.
- k=1

1. korak

- Od polja vhodnih števil *A* vzemite od vsakega števila *k*-ti bit (*k* je sedaj 1). Tako dobite polje bitov *D*.

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1

i:	0	1	2	3
A:	14	2	12	5
j:	0	1	2	3
D:	1	1	0	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1

i:	0	1	2	3
A:	14	2	12	5
j:	0	1	2	3
D:	1	1	0	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1

i:	0	1	2	3
A:	14	2	12	5
j:	2	3	0	1
D:	0	0	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**14**	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1

i:	0	1	2	3
A:	14	2	12	5
j:	2	3	0	1
D:	0	0	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

Binarni zapis števil	A: \ k:	7	6	5	4	3	2	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0

i:	2	3	0	1
A:	12	5	14	2
j:	2	3	0	1
D:	0	0	1	1

4. korak

- Indeks k inkrementirate in se vrnete na prvi korak.
- k=2

1. korak

- Od polja vhodnih števil *A* vzemite od vsakega števila *k*-ti bit (*k* je sedaj 2). Tako dobite polje bitov *D*.

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**12**	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0

i:	0	1	2	3
A:	12	5	14	2
j:	0	1	2	3
D:	1	1	1	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**12**	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0

i:	0	1	2	3
A:	12	5	14	2
j:	0	1	2	3
D:	1	1	1	0

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**12**	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0

i:	0	1	2	3
A:	12	5	14	2
j:	3	0	1	2
D:	0	1	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**12**	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0
	2	0	0	0	0	0	0	1	0

i:	0	1	2	3
A:	12	5	14	2
j:	3	0	1	2
D:	0	1	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

Binarni zapis števil	A: \ k:	7	6	5	4	3	2	1	0
	2	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0

i:	3	0	1	2
A:	2	12	5	14
j:	3	0	1	2
D:	0	1	1	1

4. korak

- Indeks k inkrementirate in se vrnete na prvi korak.
- k=3

1. korak

- Od polja vhodnih števil *A* vzemite od vsakega števila *k*-ti bit (*k* je sedaj 3). Tako dobite polje bitov *D*.

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**2**	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0

i:	0	1	2	3
A:	2	12	5	14
j:	0	1	2	3
D:	0	1	0	1

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**2**	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0

i:	0	1	2	3
A:	2	12	5	14
j:	0	1	2	3
D:	0	1	0	1

2. korak

- Bite (polje *D*) sortirajte s stabilnim algoritmom za sortiranje (najboljše counting sort).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**2**	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0

i:	0	1	2	3
A:	2	12	5	14
j:	0	2	1	3
D:	0	0	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

	k: / A:	7	6	5	4	3	2	1	0
Binarni zapis števil	**2**	0	0	0	0	0	0	1	0
	12	0	0	0	0	1	1	0	0
	5	0	0	0	0	0	1	0	1
	14	0	0	0	0	1	1	1	0

i:	0	1	2	3
A:	2	12	5	14
j:	0	2	1	3
D:	0	0	1	1

3. korak

- Glede na indekse sortiranih bitov popravite vrstni red števil v *A* (tako velja *i == j*, za *A[i]* in *D[j]*).

Binarni zapis števil	A: \ k:	7	6	5	4	3	2	1	0
	2	0	0	0	0	0	0	1	0
	5	0	0	0	0	0	1	0	1
	12	0	0	0	0	1	1	0	0
	14	0	0	0	0	1	1	1	0

i:	0	2	1	3
A:	2	5	12	14
j:	0	2	1	3
D:	0	0	1	1

4. korak

- Indeks k inkrementirate in se vrnete na prvi korak.
- k=4 ->7
  - Ni sprememb v vrstnem redu

	A: \ k:	7	6	5	4	3	2	1	0
Binarni zapis števil	**2**	0	0	0	0	0	0	1	0
	5	0	0	0	0	0	1	0	1
	12	0	0	0	0	1	1	0	0
	14	0	0	0	0	1	1	1	0

i:	0	1	2	3
A:	2	5	12	14

Pomoč pri implementaciji

- Uporaba tipa: unsigned char
- Dostop do k-tega bita števila A[i]:
  - bool bit = (A[i] >> k) & 1;
- Izpis bitov števila A[i]:
  - for(int k=0;k<8;k++)

    std::cout<<((A[i] >> k) & 1) <<"\n";
- Namig:
  - V vsakem prehodu skozi korake 1-3 je možno urediti števila v polju A tudi brez uporabe polja D, saj so vse vrednosti v polju D neposredno dostopne iz polja A.

Counting sort za stabilno urejanje polja A po k-tem bitu:

- 1. korak: za vsak A[i] velja C[(A[i] >> k) & 1]++
- 2. korak (prefix sum): C[1] += C[0]
- 3. korak: B[--C[(A[i] >> k) & 1]] = A[i]
- 4. Imamo sortirano polje A po k-tem bitu v polju B. Zamenjamo kazalca od polj: std::swap(A, B)