Упражнение 7 (алгоритми върху масиви)

Ще считаме, че $a[n]$ и $b[m]$ са масиви от цели числа със съответните размери. Алгоритмите са написани на $C++$ стандарт 11.

Задача 1.1. Множества са зададени чрез масиви от числа. Да се намери сечението им.

Версия 1 - чрез предварително сортиране:

```
void inters1()
{
	sort(a, a + n); sort(b, b + m);

	for (int i = 0, j = 0; i < n && j < m;)
		if (a[i] == b[j]) cout<<a[(j++, i++)]<<" ";
		else if (a[i] > b[j]) j++; else i++;
}
```

Тук sort е със сложност $O(k \log k)$, което определя сложността на алгоритъма - $O(n \log n + m \log m)$, поради двете сортирания в началото.

В случая, когато става въпрос за цели числа, може да се сведе и до $O(n + m)$ чрез бърз сортиращ алгоритъм за цели числа. Това, разбира се, ще е за сметка на повече допълнителна памет.

Версия 2 - чрез Hash-структура:

```
void inters2()
{
	unordered_set<int> s;
	for (int i = 0; i < n; i++) s.insert(a[i]);
	for (int i = 0; i < m; i++)
		if (s.find(b[i]) != s.end()) cout<<b[i]<< " ";
}
```

Тук очакваната сложност е $O(n + m)$, тъй като find и insert са със средна сложност $O(1)$. Разбира се, допълнителната памет е поне $O(n + m)$.

Задача 1.2. Множества A и B са зададени чрез масиви от числа. Да се намери $B \setminus A$.

Версия 1 - чрез предварително сортиране:

```
void comple1()
{
	sort(a, a + n); sort(b, b + m);

	int i = 0, j = 0;
	while (i < n && j < m)
		if (a[i] == b[j]) (j++, i++);
		else if (a[i] < b[j]) i++; else cout<<b[j++]<<" ";
	if (j < m) while (j < m) cout<<b[j++]<<" ";
}
```

Версия 2 - чрез Hash-структура:

```
void comple2()
{
	unordered_set<int> s;
	for (int i = 0; i < n; i++) s.insert(a[i]);
	for (int i = 0; i < m; i++)
		if (s.find(b[i]) == s.end()) cout<<b[i]<<" ";
}
```

Тук сложностите са аналогични на задача 1 - $O(n \log n + m \log m)$ за първата версия и очаквана $O(n + m)$ за втората.

Задача 1.3. Множества A и B са зададени чрез масиви от числа. Да се намери $A \cup B$.

Версия 1 - чрез предварително сортиране:

```
void uni1()
{
	sort(a, a + n);
	sort(b, b + m);

	int i = 0, j = 0;
	while (i < n && j < m)
		if (a[i] == b[j]) cout<<a[(j++, i++)]<<" ";
		else if (a[i] > b[j]) cout<<b[j++]<<" ";
			else cout<<a[i++]<<" ";

	if (i == n) while (j < n) cout<<b[j++]<<" ";
		else while (i < n) cout<<a[i++]<<" ";
}
```

Версия 2 - чрез Hash-структура:

```
void uni2()
{
	unordered_set<int> s;
	for (int i = 0; i < n; i++)
	{
		cout<<a[i]<<" ";
		s.insert(a[i]);
	}
	for (int i = 0; i < m; i++)
		if (s.find(b[i]) == s.end()) cout<<b[i]<<" ";
}
```

И тук сложностите са аналогични на Задача 1 - $O(n \log n + m \log m)$ за първата версия и очаквана $O(n + m)$ за втората.

Задача 2. 3SUM - в масив от различни цели числа има ли 3, чиято сума е 0?

Версия 1 - чрез предварително сортиране:

```
void sum3()
{
	sort(a, a + n);

	for (int i = 0; i < n - 2; i++)
		for (int j = i + 1, k = n - 1; j<k;)
			if (a[i] + a[j] + a[k] == 0)
				cout<<a[i]<<" "<<a[j++]<<" "<<a[k--]<<endl;
			else if (a[i] + a[j] + a[k] > 0) k--; else j++;
}
```

Тази версия извежда всички възможни решения. Сложността ѝ е $O(n^2)$. Вътрешният цикъл включва два броча, които „вървят“ един срещу друг.

Пример:

След сортиране примерният масив е:

$$a[10] = \{-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28\};$$

Ще бележим индексираните елементи с различни цветове: i, j, k

За $i = 0$, започваме с $j = 1$, $k = n - 1$:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Сумата е $-15 < 0$, така че увеличаваме j

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Пак сумата е по-малка от 0, така че:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

След известно време стигаме до:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Тук сумата е 0, така че извеждаме -36, 8, 28, увеличаваме j и намаляме k:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Тук 27 стана лилав, защото е едновременно и син и червен... (това са неформализми с цел да илюстрират алгоритъма) и минаваме на следващата стъпка за i:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

След време стигаме до:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Извеждаме -7, 1, 6 и продължаваме:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Тук j и k се разминаха, така че минаваме нататък за i:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Други решения освен тези няма. Последната стъпка на циклите изобщо е:

$$-36, -7, -6, -5, -4, 1, 6, 7, 8, 27, 28$$

Можем да сложим и още едно условие за основния цикъл:

```
for (int i = 0; i < n - 2 && a[i] <= 0; i++)
```

Идеята е, че ако $a[i] > 0$, то от i нататък всички елементи са положителни, така че няма как сумата на 3 от тях да е 0.

Версия 2 - с Hash-структура:

```
bool sum3_1()
{
	unordered_set<int> s;
	for (int i = 0; i < n; i++) s.insert(a[i]);
	for (int i = 0; i < n - 1; i++) for (int j = i + 1; j < n; j++)
		if (s.find(-a[i] - a[j]) != s.end() && a[i] != -a[i] -
a[j] && a[j] != -a[i] - a[j])
	{
		cout<<a[i]<<" "<<a[j]<<" "<<-a[i] - a[j]<<endl;
		return true;
	}
	return false;
}
```

В тази версия извеждаме само едно решение и връщаме стойност $true$ или $false$.

Тук за всеки 2 елемента $a[i]$ и $a[j]$ проверяваме дали $-a[i] - a[j]$ се съдържа в структурата (времето отново е $O(n^2)$). Разбира се, това число трябва да е различно и от двата елемента.

Ако имаме -1 и 2, то $1 - 2 = -1$ се съдържа в структурата, но съвпада с първото число, така че не е решение.

Ако допуснем, че елементите се повтарят, може да използваме unordered_map, където за всеки ключ ще пазим колко пъти се съдържа в масива.

Сега, ако -1 и 2 са текущите елементи, но -1 се съдържа повече от веднъж, то -1, 2, -1 е коректно решение.

Задача 3. Да се намерят всички най-често срещани елементи на масив

Версия 1 - с предварително сортиране:

```
void all_mods()
{
	sort(a, a + n);

	stack<pair<int, int> > st;
	st.push(pair<int, int>(a[0], 0));

	int s;

	for (int i = 1; i<n; i++)
	{
		s = 1;
		while (a[i] == a[i - 1] && i++ < n) s++;
		if (s >= st.top().second)
			st.push(pair<int, int>(a[i - 1], s));
	}

	s = st.top().second;
	while (st.top().second == s)
	{
		cout << st.top().first << " " << st.top().second << endl;
		st.pop();
	}
}
```

Пазим най-често срещаните елементи и честотата им в стек. След като са сортирани, честотата може да се намери лесно, тъй като еднаквите елементи са един след друг в масива.

Вкарваме елемент в стека, само ако върхът му е с не по-голяма честота от текущо разглеждания елемент.

Извеждаме елементите с най-голяма честота от стека. Те започват от върха му.

Цикълът, които извежда тези елементи спира, тъй като дъното на стека е с честота 0.
Този елемент не носи информация за масива, а просто е сложен като разделител.

Разбира се, не е задължително да се използва стек. Може например да се използва допълнителен масив.

Версия 2 - с Hash-структура:

```
void all_mods2()
{
	unordered_map<int, int> m;

	for (int i = 0; i < n; i++)
		if (m.find(a[i]) == m.end()) m[a[i]] = 1; else m[a[i]]++;

	pair<int, int> * st = new pair<int, int>[n];
	st[0] = pair<int, int>(a[0], 0);

	int p = 0;

	for (int i = 0; i < n; i++)
		if (m[a[i]] >= st[p].second)
			st[++p] = pair<int, int>(a[i], m[a[i]]);

	int s = st[p--].second;

	while (st[p].second == s)
		if (st[p - 1].first != st[p--].first)
			cout<<st[p + 1].first<<" "<<st[p + 1].second<<endl;
}
```

За всеки елемент хешираме колко пъти се среща. После следва идея подобна на идеята за стека, само че резлизирана с масив.

Задача 4. (Задача от контролното) Даден е масив с n положителни цели числа. Има ли сред тях две, разликата на които се дели на n?

```
bool zad_ot_kontr()
{
	bool p[n] = { false };

	for (int i = 0; i < n; i++)
		if (p[a[i] % n] == true) return true;
		else p[a[i] % n] = true;

	return false;
}
```

За всеки индекс i от булевия масив пазим дали има елемент, които дава остатък i при делене на n. За да се дели разликата на 2 елемента на n е необходимо, те да дават еднакви остатъци при делене на n.

Задача 5. Уникални елементи с Hash-структура.

```
bool elem_uniq()
{
      unordered_set<int> s;
      for (int i = 0; i<n; i++)
            if (s.find(a[i]) == s.end())s.insert(a[i]);
            else return false;
      return true;
}
```

Очакваната сложност на този алгоритъм е $\Theta(n)$. Защо тогава долната граница на тази задача е $\Omega(n \log n)$?

Отговорът е, че за Hash-структурата съществуват примери, при които тя ще се изроди до свързан списък (всички елементи ще се хешират на едно и също място), докато $\Omega(n \log n)$ е долна граница, която ни трябва за абсолютно сигурен отговор дали елементите са различни.

Такива примери е трудно (дори практически невъзможно) да се дадат и затова подобна резлизация е достатъчно добра.