Задачите на една панда

Книга 2

Динамично програмиране

Дейкстра

Крускал

Merge Sort

Bamboo

Записки по
Дизайн и анализ на алгоритми
практикум

Здравей, скъпи читателю! За разлика от другите сериозни предмети във ФМИ със сериозни записки, курсът „Дизайн и анализ на алгоритми-практикум“ ще ти бъде обясняван от мен – Панда.

Панда ще те запознае с основните сортиращи алгоритми, ще премине през двоично и троично търсене, ще се катери по графи с помощта на алгоритми и ще решава оптимизационни задачи, защото въпросът как да си осигури максимално количество бамбук с минимални усилия е от изключителна важност.

*Панда предлага да започваме с материала! Какво чакаш, скролвай надолу ^^*

Неща, които Панда смята, че са полезни

1. Вградени константи

`INT_MAX`	най-голямата стойност на int
`INT_MIN`	най-малката стойност на int
`LLONG_MAX`	най-голямата стойност на long long
`LLONG_MIN`	най-малката стойност long long

2. Вградени функции

Функции, които връщат най-малък и най-голям елемент в масив. Те приемат за аргументи началото и края на масива и връщат адреса на екстремума.

`*min_element(arr, arr + n)`	най-малък елемент
`*max_element(arr, arr + n)`	най-голям елемент

Внимание: Функциите работят с **линейна** сложност. Единствено си съкращавате кода с тях.

3. Функцията *`fill_n(arr, n, a)`*

Приема като аргументи началото на масив, някаква дължина и константа. Тя присвоява на първите n елемента от масива стойност тази константа.

Внимание: Функцията работи с **линейна** сложност. Единствено си съкращавате кода с нея.

4. `pair<> и tuple<>`

Контейнерите `pair<>` и `tuple<>` са доста удобни при представянето на претеглени, (не)ориентирани графи.

pair<> - двойка tuple<> - n-торка

Неформално: За tuple можем да си мислим като масивче, чиито елементи могат да са от различни типове.

Пример:

```
pair<int, string> examplePair;
examplePair = make_pair(19, “date”);
examplePair = {19, “date”};
tuple<int, int, string> exampleTuple;
exampleTuple = make_tuple(19, 9, ”birthday”);
exampleTuple = {19, 9, “birthday”};
```

Достъпваме елементите от тип pair по следния начин:

```
cout<< examplePair.first<<’\n’; // отпечатва 19
cout<< examplePair.second<<’\n’; // отпечатва date
```

Достъпваме елементи от тип tuple по следния начин:

```
cout<<get<0>(exampleTuple)<<’\n’; // отпечатва 19
cout<<get<1>(exampleTuple)<<’\n’; //отпечатва 9
cout<<get<2>(exampleTuple)<<’\n’; // отпечатва birthday
```

Тоест извиквате функция get<> и като аргумент й подавате индекса на елемента, който искате да достъпите.

За контейнерите pair и tuple има предефинирана операция <, която ги сравнява лексикографски.

Пример:

```
pair<int, int> pointA, pointB, pointC;
pointA = make_pair(9, 0);
pointB = make_pair(6, 6);
pointC = make_pair(9,10);
pointB < pointA // защото 6 < 9
pointA < pointC //тъй като 9==9, сравнява 0 < 10
```

5. Побитови операции

&	**0**	**1**
0	0	0
1	0	1

^	**0**	**1**
0	0	1
1	1	0

|	**0**	**1**
0	0	1
1	1	1

NOT(~) – Ако на променлива приложим побитово „не“, то резултатът е число, което има единица на всяка позиция, където е било 0 и обратното- единиците са станали нули. Накратко отрицанието обръща битовете.

Пример:

~					
	1	0	0	1	1
	0	1	1	0	0

Shift left(<<) – Операцията премества битовете на дадено число наляво. *Всички единици, които излизат извън рамките на числото биват игнорирани*. Действа като **умножение** на число със степен на двойката.(степента е броят на отместванията)

Пример: $25 << 2 = 100$

$11001_2 = 25_{10}$, $1100100_2 = 100_{10}$ и наистина $25.4 = 100$

Shift right(>>) - Операцията премества битовете на дадено число надясно. *Всички единици, които излизат извън рамките на числото биват игнорирани*. Действа като **деление** на число със степен на двойката.(степента е броят на отместванията)

Пример: $24 >> 2 \ = \ 6$

$11000_2 = 24_{10}$, $110_2 = 6_{10}$ и наистина $24 : 4 = 6$

Задачи за побитови операции:

Задача 1: Да се провери дали едно число е четно с помощта на побитови операции.

Решение: Прилагаме побитово „и“ на числото с 1. Ако получим 1 е нечетно, в противен случай – четно.

Примери:

&				
	1	*0*	*0*	*1*
	0	*0*	*0*	*0*
	0	*0*	*0*	***1***

нечетно

&				
	1	*0*	*0*	*0*
	0	*0*	*0*	*1*
	0	*0*	*0*	***0***

четно

Задача 2: Да се намери броя на единиците в двоично число.

Решение: псевдо код:

```
while(x)
{
    br++; // брояч за единиците
    x= x&(x-1);
}
```

разсъждение: След като x-1 е с 1 по-малко от x => на мястото на най-дясната единица от x ще има 0.

Схема:

	1	*1*	*0*	*0*	*1*	=x
&	*1*	*1*	*0*	*0*	*0*	=x-1
	1	*1*	*0*	*0*	*0*	=резултат = r
&	*1*	*0*	*1*	*1*	*1*	=r-1
	1	*0*	*0*	*0*	*0*	=резултат = r1
&	*0*	*1*	*1*	*1*	*1*	=r1-1
	0	*0*	*0*	*0*	*0*	=край на цикъла

В зелено са маркирани единиците, които броячът брои при всяко завъртане на цикъла.

Задача 3: Да се провери дали дадено число x е степен на двойката

Решение: *Прилагаме побитово „и“ на x с x-1.*

Псевдо код: `x&(x-1)==0`

6. Класът/Типът vector<>

Класът/типът вектор е вграден динамичен масив, който е доста удобен за работа.

`vector<тип> име;`

Вкарване на елементи: push_back();

```
vector<int> exampleVector;
int n,value;
cin>>n;
for(int i = 0; i < n; i++) {
         cin>>value;
       exampleVector.push_back(value);
}
```

Премахване на елементи: Можете да махате елементи **само от края** на вектора: pop_back();

```
for(int i = 0; i<n; i++){
      exampleVector.pop_back();
}
```

7. Вход/Изход

Тъй като cin/cout са доста бавнички два начина, по които можете да оптимизирате времето за четене/отпечатване на данни са:

Вариант 1: Използвате cin/cout със спряна синхронизация на **cin -> scanf и cout ->printf**

Код за спиране на синхронизацията:

```
ios_base::sync_with_studio(false);
cin.tie(nullptr);
```

Внимание: Ако изберете да работите с този вход/изход на данни, когато отпечатвате нови редове използвайте ‘\n‘ вместо endl

Вариант 2: Използвате само `scanf` и `printf`

`scanf` - https://www.cplusplus.com/reference/cstdio/scanf/

`printf` - https://www.cplusplus.com/reference/cstdio/printf/

Предимството на `scanf/printf` е по-малкото код и доста бързо ръчно дебъгване, доколкото предимството на `cin/cout` е това, че **не** трябва да се грижите да описвате типа на променливите, които ще отпечатвате/вкарвате.