國立高雄科技大學

智慧商務系

資料結構期末專題報告

氣泡排序及 DFS 應用

-以電腦購物平台為例

C109193207 黃建銘

指導教授：謝文川

中華民國 112 年 1 月

目錄

第壹章、緒論

第一節 研究動機及目的

本人在暑假有自己做有關賣電腦的購物平台，是以原價屋作為原型下去做開發的一個網站，裡面有著可以依照自己桌機的需求，去做組合套餐的商品，也有能快速篩選並排序符合自己需求的筆電商品。而上過資料結構課程之後，我發現筆電商品畫面可以使用資料結構的搜尋與排序法去做篩選商品，因此此次專題以氣泡排序法與 DFS 搜尋法重構筆電篩選畫面。

第二節 研究流程

此次重構系統主針對電腦購物平台的筆電篩選畫面，首先會先規劃搜尋與排序導入，然後開始編寫程式，最後測試是否與先前的結果一致。

圖 1. 研究流程圖

第三節 計畫時程

本報告依循研究流順序來著手進行，圖 2. 計畫時程甘特圖所示：於 112 年 1 月 14 日開始搜集資料。1 月 15 號至 1 月 17 號完成編寫程式，並於 1 月 17 號完成測試。

計畫	2023			
時程	1/14	1/15	1/16	1/17
規劃架構				
編寫程式				
測試				

圖 2. 研究流程圖

第貳章、實作與展示

第一節 規劃架構

右手邊下拉式選單主要是用氣泡排序法去做到排序動作，而左手邊選取表單可以根據所選取的標籤利用 DFS 搜尋法搜尋相關筆電商品。

氣泡排序法

DFS 搜尋法

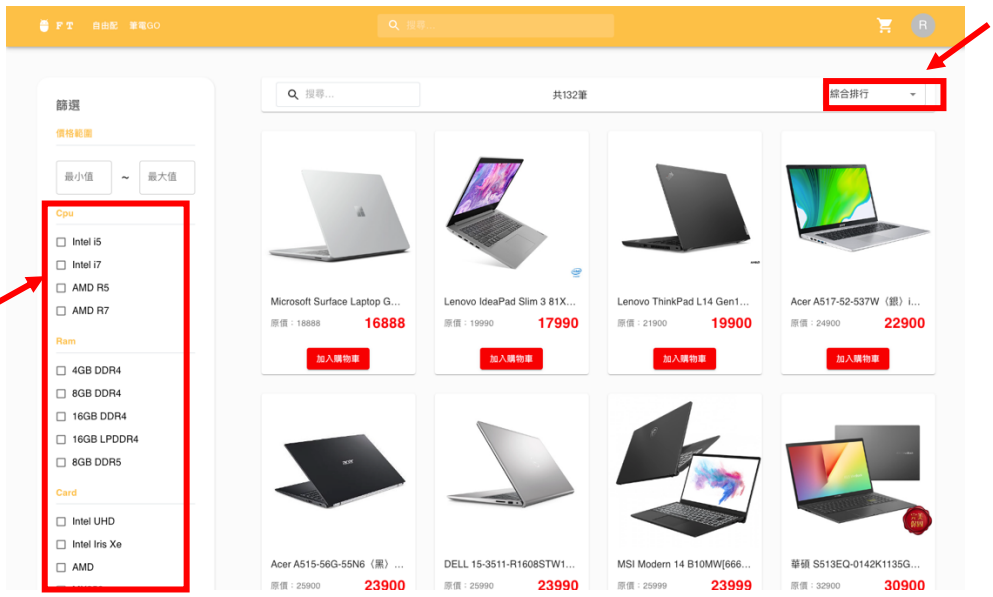

圖 3 規劃架構示意圖

第二節 氣泡排序法

根據所選取的值去做氣泡排序函式

```
const options = ['綜合排行','價格低到高','價格高到低'];
const [value, setValue] = useState(options[0]);

switch (value) {
  case '價格低到高':
    bubbleSort(laptop,true,'price')

    break;
  case '價格高到低':
    bubbleSort(laptop,false,'price')

    break;
  default:
    bubbleSort(laptop,true,'proid')

    break;
}
```

圖 4 示意圖

由圖四裡 ascending 的值判斷氣泡排序降冪還升冪

```
const bubbleSort=(arr,ascending,name)=>{
  var len = arr.length;
  //透過ascending判斷降冪還升冪
  if (ascending) {
    for (var i = 0; i < len; i++) {
        for (var j = 0; j < len-i-1; j++) {
            if (parseInt(arr[j][name]) > parseInt(arr[j+1][name])) {
                var temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
  }
  else{
    for (var i = 0; i < len; i++) {
      for (var j = 0; j < len-i-1; j++) {
          if (parseInt(arr[j][name]) < parseInt(arr[j+1][name])) {
              var temp = arr[j];
              arr[j] = arr[j+1];
              arr[j+1] = temp;
          }
      }
    }
  }
  return arr;
}
```

圖 4 Bubble sort 示意圖

和先前使用方法比較

```
const options = ['綜合排行','價格低到高','價格高到低'];
const [value, setValue] = useState(options[0]);

switch (value) {
  case '價格低到高':

    laptop.sort((a, b) => {return a.price - b.price;});
    break;
  case '價格高到低':

    laptop.sort((a, b) => {return b.price - a.price;});
    break;
  default:

    laptop.sort((a, b) => {return a.proid - b.proid;});
    break;
}
```

```
const options = ['綜合排行','價格低到高','價格高到低'];
const [value, setValue] = useState(options[0]);

switch (value) {
  case '價格低到高':
    bubbleSort(laptop,true,'price')

    break;
  case '價格高到低':
    bubbleSort(laptop,false,'price')

    break;
  default:
    bubbleSort(laptop,true,'proid')

    break;
}
```

圖 5 方法比較示意圖

第三節 DFS 搜尋法

根據所選取的標籤一一去做 DFS 搜尋，搜尋的資料全部放到 new_laptop 陣列裡，最後篩選出陣列不重複的資料回傳。

```
let new_laptop=[]
option.map((aname)=>{

    aname.allname.map((sname)=>{
     if (sname.status==true) {
        new_laptop.push(...DFS(original_laptop,aname.classname,sname.name))
     }
    })
    })
    var result = new_laptop.filter(function(item, index, array){
        return array.indexOf(item) === index;
    });
    setlaptop(()=>result.length==0?original_laptop:result)
```

圖 6 示意圖

graph 是所有資料，start 是勾選標籤類別，name 是標籤名稱，當 graph 的每一筆的 start 有 name 的字串，就加進 stackdata 陣列，最後回傳 stackdata。

```
const DFS=(graph, start,name)=> {
  var visited = new Set();
  var stack = [start];
  let stackdata=[]
  while (stack.length) {
    var vertex = stack.pop();
    if (!visited.has(vertex)) {
      visited.add(vertex);
      Object.entries( graph).map(function(key,neighbor) {
        if (key[1][start].indexOf(name)>=0) {
          stackdata.push(key[1])

        }
      });
    }
  }
  return stackdata
}
export default DFS
```

圖 7 DFS 示意圖

先前使用方法比較

```
let new_laptop=[]
option.map((aname)=>{

    aname.allname.map((sname)=>{
     if (sname.status==true) {
        new_laptop.push(...DFS(original_laptop,aname.classname,sname.name))
     }
    })
    })
    var result = new_laptop.filter(function(item, index, array){
        return array.indexOf(item) === index;
    });
    setlaptop(()=>result.length==0?original_laptop:result)
```

```
let result=[]
 option.map((aname)=>{
     aname.allname.map((sname)=>{
      if (sname.status==true) {
         result.push(...original_laptop.filter((item, index, array)=>(
             item[aname.classname].indexOf(sname.name)>=0&&array.indexOf(item) === index)))
      }
     })
     })

     console.log(result);
     setlaptop((laptop)=>result.length==0?original_laptop:result)
```

圖 8 方法比較示意圖

第參章、結論

在此次重構完下拉式選單和核取表單並完成測試以後，發現重構前與重構後的執行速度其實是差不多的，表示蠻成功的。而在攥寫程式的過程當中，也在思考如何讓整個畫面執行速度快，比如在氣泡排序的函式裡，就會思考說 ascending 這個判斷要放到 for 迴圈裡，還是在一開始先判斷再去繞 for 迴圈等等問題。而現在有個問題，就是在搜尋時無法精確的搜尋符合每個條件的資料，只能符合其中一項，希望未來能改善這一問題。