Phân Tích Thuật Toán

Bùi Thị Thanh Phương

Ngày 4 tháng 4 năm 2023

1 Bài 1.

1. Viết chương trình để biểu diễn một số thập phân N sang dạng biểu diễn nhị phân có độ phức tạp thuật toán là $O(log_2 N)$.

- Input: Số thập phân N
- Output: Dạng biểu diễn nhị phân của N.

2. Giả sử các em đã xây dựng thuật toán trên. Hãy viết một chương trình để đếm số phép gán và số phép so sánh mà chương trình trên đã dùng để biểu diễn một số thập phân N.

- Input: binary(N), $N = 100, 200, 300, 400, ..., 1000$.
- Output: Gan(N), Sosanh(N). Vẽ $log_2 N, Gan(N)$, và Sosanh(N) trên cùng một đồ thị để so sánh.

Hàm decimalToBinary(N) biểu diễn một số thập phân N sang dạng biểu diễn nhị phân có độ phức tạp thuật toán là

Bài làm

1. Hàm **decimalToBinary(N)** biểu diễn một số thập phân N sang dạng biểu diễn nhị phân có độ phức tạp thuật toán là $O(log_2 N)$.

```
def decimalToBinary(N):
    s = str(N%2)
    if N > 1:
        return decimalToBinary(N//2) + s
    return s


bin = decimalToBinary(100)
print("decimal {} to binary number is {}".format(100, bin))
```

Ta được kết quả như sau:

2. Hàm **toBinary(N)** đếm số phép gán và số phép so sánh mà chương trình trên đã dùng để biểu diễn một số thập phân N.

Khởi chạy chương trình trên với $N = 1, ..., 1000$ ta có bảng số phép gán và số phép so sánh tương ứng với từng giá trị N như sau

Và đồ thị của $log2N, Gan(N), Sosanh(N)$ có dạng như sau

Từ đồ thị trên ta có nhận xét như sau

```
decimal 100 to binary number is 1100100
```

```python
def toBinary(N):
    assign = compare = 0

    compare += 1
    if N <= 1:
        return assign, compare

    s = str(N%2)
    assign += 1
    while (N != 0):
        compare += 1
        N = N//2
        s = str(N%2) + s
        assign += 2
    compare += 1
    return [assign, compare]
```

- Đường $Gan(N)$ và $Sosanh(N)$ có dạng tương đối giống với đường $O(log2N)$.
- Cả 2 đường $Gan(N)$ và $Sosanh(N)$ đều có giá trị lớn hơn so với đường $O(log2N)$, trong đó$Gan(N)$dao động nhiều hơn và có giá trị lớn hơn.
- $Gan(N)$ và $Sosanh(N)$ đều là tập hợp các giá trị rời rạc và có những khoảng giá trị bằng nhau nên $Gan(N)$ và $Sosanh(N)$có dạng đường gấp khúc.

Bài toán 2 (Bonus). Cho một mảng A gồm n số tự nhiên nằm $[1...k]$. Hãy thiết kế thuật toán để kiểm tra xem có bao nhiêu phần tử của A nằm trong $[a, b]$ có độ phức tạp là $O(N + k)$ và chứng minh thuật toán của các em đưa ra có độ phức tạp như trên.

- Input: Xét hai trường hợp

  $-N = 10, 20, ..., 10000(kcØnh, k = 100)$.

  $-k = 10, 20, ...(NcØnh, N = 20000)$.

  Tạo ngẫu nhiên mảng A với $[a, b]$tương ứng.
- Output: Đếm số phép gán, số phép so sánh và so sánh với $O(N + k)$.

Bài Làm
Thuật toán kiểm tra xem có bao nhiêu phần tử của A nằm trong [a, b] có độ phức tạp là $O(N + k)$.

- Trong trường hợp tối ưu nhất $(igea)$, ta có tổng số phép so sánh là
  $\sum_{i=1}^{n-1} 1 + 1 + \sum_{j=1}^{k}(1 + 1 + 1) + 1 = N + 1 + 3k + 1 = N + 3k + 2 = O(N + k)$
  và tổng số phép gán là
  $2 + \sum_{i=0}^{N-1}(1 + 1) + \sum_{j=0}^{k}(1 + 1) = 2 + 2N + 2k = O(N + k)$
- Trong trường hợp kém tối ưu nhất $(i < a)$, ta có tổng số phép so sánh là
  $\sum_{i=0}^{N-1} 1 + 1 + \sum_{J=0}^{K} 1 = 2 + 2N + k = O(N + k)$
  Và tổng số phép gán là $2 + \sum_{i=0}^{N-1}(1 + 1) + \sum_{j=0}^{k}(1) = 2 + 2N + k = O(N + k)$

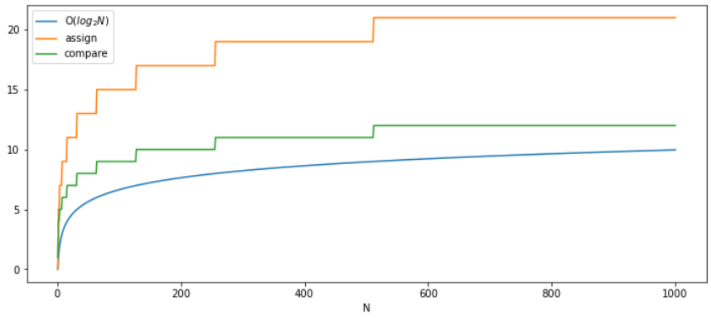
  Như vậy độ phức tạp của thuật toán trên là $O(N + k)$. Hơn nữa, ta cố định lần lượt các giá trị N và k ta có được các biểu đồ như sau

```
X = np.arange(1,1000+1,1)
temp = np.array([toBinary(x) for x in X])
output = pd.DataFrame(data={"N": X, "Assign": temp[:,0], "Compare": temp[:,1]}).set_index("N")
output
```

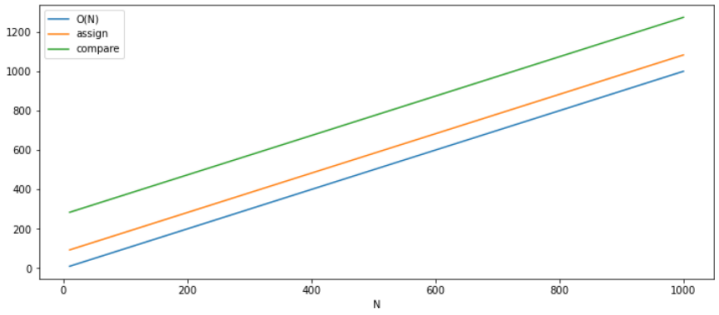
```
fig, ax = plt.subplots(figsize=(12,5))
ax.plot(X, np.log2(X))
ax.plot(X, temp[:,0])
ax.plot(X, temp[:,1])
ax.set(xlabel="N")
ax.legend(["O($log_2N$)", "assign", "compare"])
plt.savefig("ex1.jpg")
plt.show()
```

N	Assign	Compare
1	0	1
2	5	4
3	5	4
4	7	5
5	7	5
...	...	...
996	21	12
997	21	12
998	21	12
999	21	12
1000	21	12

1000 rows × 2 columns



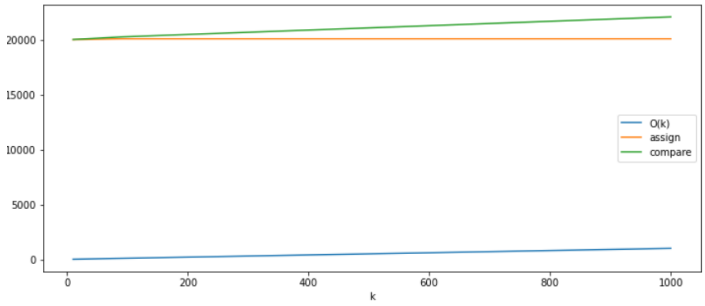
```
def check_Array(A, N, k, a, b):
       count_elements_in_A = np.zeros(k+1)
       count = 0

       for i in range (0, k-1):
           count_elements_in_A[A[i]] +=1
       for i in range(k):

           if i >=a:
               if i <=b:
                   count += count_elements_in_A[i]
   return count
```



- Trường hợp cố định $k = 100$, ta có biểu đồ

  Trong trường hợp này, ta nhận thấy rằng các đường assign và compare có phương song song với đường $O(N)$, mặc dù số phép so sánh và số phép gán đều có chênh lệch cao hơn so với đường $O(N)$.

- Trường hợp cố định $N = 20000$, ta có biểu đồ



  Trong trường hợp này, ta nhận thấy rằng đường assign bị lệch phương so với 2 đường còn lại là compare và $O(k)$ và chênh lệch của 2 đường compare và assign so với đường $O(k)$ rất lớn.