The Lazy Programmer

Big-O Blue Online Lecture 07: Heap

Tóm Tắt Đề Bài

Tóm Tắt Đề Bài

Dữ kiện:

- Có **N** công việc cần được hoàn thành được biểu diễn bởi ba thông số:
  - $b[i]$ là đơn vị thời gian để hoàn thành xong công việc thứ i.
  - $d[i]$ là mốc thời gian phải hoàn thành (deadline) công việc thứ i.
  - $a[i]$ là đơn vị giảm thiểu thời gian nếu ta trả 1 đô cho cho công việc thứ i. Một cách khác, ta có thể trả x đô để thời gian hoàn thành công việc thứ i giảm xuống thành $b[i] - x * a[i]$

Yêu cầu:

- Hãy đưa ra số tiền ít nhất cần được chi trả sao cho tất cả các công việc hoàn thành đúng deadline.

Giải Thích Ví Dụ

Giải Thích Ví Dụ

Input	Output	Giải thích ví dụ
N = 2 a = 20, b = 50, d = 100 a = 10, b = 100, d = 50	5.00	• Hoàn thành công việc thứ 2 trước: Do deadline là 50, nên ta cần bỏ ra 5 đô để giảm thiểu thời gian hoàn thành xuống 100 - 10 * 5 = 50. • Hoàn thành công việc thứ 1: Do deadline là 100, mốc thời gian hiện tại là 50 và thời gian để hoàn thành là 50 -> Vừa kịp deadline nên ta sẽ không cần phải chi tiền. • Tổng tiền đã chi: 5.00

Giải Thích Ví Dụ

Input	Output	Một ví dụ khác
N = 2 a = 20, b = 50, d = 100 a = 10, b = 100, d = 50	5.00	• Hoàn thành công việc thứ 1 trước: Do deadline là 100 mà thời gian hoàn thành là 50, ta không cần phải chi tiền. • Hoàn thành công việc thứ 2: Do tại thời điểm 50 cũng chính là thời điểm deadline, ta cần trả 10 đô để hoàn thành công việc **ngay lập tức**. • Tổng tiền đã chi: 10.00

Hướng Dẫn Giải

Ý Tưởng

- Một cách thực tế, ta cần **sắp xếp** tất cả các công việc theo **deadline.**

current	a	b	d
0	20	30	30
	10	40	50
	5	50	70

Ý Tưởng

- Đối với những công việc thời gian hoàn thành **không vượt quá** deadline, ta sẽ không trả tiền cho công việc này.

current	a	b	d
0	20	30	30
	10	40	50
	5	50	70

Ý Tưởng

- Đối với những công việc thời gian hoàn thành **vượt quá** deadline, ta sẽ phải trả tiền cho công việc này.
- Ở trường hợp này ta cần trả tiền như thế nào cho tối ưu ?

current	a	b	d
0	20	30	30
30	10	40	50
	5	50	70

Ý Tưởng

- Ta nhận thấy rằng có thể tận dụng lại thời gian giảm thiểu ở những công việc trước công việc thứ i đang xét.
- Để giảm thiểu xuống 20 đơn vị thời gian, nếu ta trả cho công việc thứ 1 chỉ mất 1 đô trong khi nếu trả cho công việc thứ 2, ta cần phải trả mất 2 đô.

current	a	b	d
0	20	30	30
30	10	40	50
	5	50	70

Ý Tưởng

- Như vậy, ta cần biết **giá trị lớn nhất** của các đơn vị ***a*** trước công việc thứ ***i*** để ta có thể tận dụng nó nhằm giảm thiểu thời gian hoàn thành.
- Tuy nhiên, có phải lúc nào việc lấy **giá trị lớn nhất** theo ***a*** cũng đúng ?

current	a	b	d
0	20	30	30
10	10	40	50
50	5	50	70

Ý Tưởng

- Theo nhận xét trên, ta vẫn tiếp tục lấy ***a* = 20** để trả tiền, tuy nhiên, lúc này ta chỉ còn lại đúng 10 đơn vị thời gian để trả (tương đương với **0.5 đô**) do ta đã dùng 1 đô trả tiền trước đó để giảm thiểu thời gian hoàn thành xuống 30 - 20 = 10.
- Khi đó, ta cần xét tiếp đến ***a*** lớn thứ nhì để trả tiếp (**a = 10**). Cần trả thêm **2 đô** để hoàn thành kịp deadline.

current	a	b	d
0	20	30	30
0	10	40	50
20	5	50	70

Ý Tưởng

- Như vậy, ta cần biết **giá trị lớn nhất** của các đơn vị ***a*** và **hợp lệ** trước công việc thứ ***i*** để ta có thể tận dụng nó nhằm giảm thiểu thời gian hoàn thành.
- "**Hợp lệ**" có nghĩa là ta có thể tận dụng lại được ***a*** mà tổng thời gian giảm thiểu không vượt quá thời gian hoàn thành ***b***.
- Với mỗi công việc thứ ***i***, ta tận dụng lại thời gian có thể giảm thiểu cho đến khi nó không còn **hợp lệ** nữa, sau đó ta lại tiếp tục xét với công việc có ***a*** lớn thứ nhì,..v.v

Ý Tưởng

- Ta cần tổ chức một cấu trúc dữ liệu để lấy nhanh ***max_a*** và **hợp lệ**
  - Mang tính truy xuất thường xuyên: lấy ra một phần tử khi trễ deadline và đẩy vào một phần tử khi có được công việc mới (cùng mục đích để tận dụng ***a***).
  - Sử dụng cấu trúc dữ liệu **Priority Queue** (Hàng đợi ưu tiên) để giải quyết
- Mỗi phần tử trong Priority Queue của chúng ta sẽ lưu những gì ?
  - ***a[i]*** : đơn vị giảm thiểu thời gian khi được trả 1 đô (**key so sánh** trên cây Heap)
  - ***b[i]*** : đơn vị thời gian thời gian còn lại có thể tận dụng (mang tính **hợp lệ**)

Ý Tưởng

- Ta sẽ tận dụng các công việc thông qua PriorityQueue như thế nào ?
  - Điều kiện để tận dụng là mốc thời gian hoàn thành > deadLine của công việc
  - Đặt ***dif*** là hiệu số giữa thời gian hoàn thành và deadLine.
  - Đặt ***w*** là phần tử đầu tiên của PriorityQueue.
  - Ta có hai trường hợp xảy ra:
    - Trường hợp 1: ***dif > w.b***
      - Tận dụng hết thời gian ***w.b*** và cập nhật ***dif -= w.b***
      - Cộng dồn thương ***w.b/w.a*** vào kết quả cuối cùng.
      - Xét tiếp tục với phần tử đầu tiên của Priority Queue
    - Trường hợp 2: ***dif <= w.b***
      - Cộng dồn ***dif/w.a*** vào kết quả cuối cùng
      - Đẩy lại phần tử ***(w.a, w.b - dif)*** vào PriorityQueue

Thuật Toán

Độ phức tạp: **O(NlogN)**

- Bước 1: Đọc **N** và các ***(a[i], b[i], d[i])***
- Bước 2: **Sort** dữ liệu tăng dần theo ***d[i]***
- Bước 3: Khởi tạo Priority Queue lưu cặp giá trị ***(a[i], b[i])*** ưu tiên theo ***a[i]*** và biến **curTime = 0** là mốc thời gian hiện tại, **answer** là lượng tiền cần trả cuối cùng.
- Bước 4: Duyệt lần lượt các công việc theo danh sách đã **sort**:
  - Push***(a[i], b[i])*** vào Priority Queue.
  - Cập nhật ***curTime += b[i]*** và xét nếu ***curTime > d[i]***:
    - Lấy ra công việc ***(a, b)*** có ***a*** lớn nhất thông qua Priority Queue. **(*)**
    - Tận dụng công việc bằng cách tìm ***x*** sao cho ***curTime - x * a <= d[i]*** :
      - Nếu ***d[i] - curTime >= b:***
        - Cập nhật ***curTime -= b, x = b/a*** và ***answer += x***
        - Quay lại tiếp tục bước (*)
      - Ngược lại,
        - Cập nhật ***curTime = d[i], b -= (d[i] - curTime), x = (d[i] - curTime)/a***
        - Push***(a, b)*** vào lại Priority Queue và chuyển sang công việc kế tiếp.
        - ***answer += x***
- Bước 5: Đưa ra ***answer***

Mã Giả

18

Pseudo Code

```
class Task:
     constructor(a, b, d):
          this.a = a
          this.b = b
          this.d = d

     operator < (other):
          return this.d < other.d

class Node:
     constructor(a, b):
          this.a = a
          this.b = b

     operator < (other):
          return this.a < other.a
```

Pseudo Code

```
read(N, tasks)
sort(tasks)
PriorityQueue = [], curTime = 0, answer = 0
for i := 0 to N - 1 do:
     PriorityQueue.push(Node(tasks[i].a, tasks[i].b)))
     curTime += tasks[i].b
     while (curTime > tasks[i].d):
          w = PriorityQueue.pop()
          if (w.b <= curTime - tasks[i].d):
               answer += w.b / w.a
               curTime -= w.b
          else:
               answer += (curTime - tasks[i].d) / w.a
               PriorityQueue(Node(w.a, w.b - (curTime - tasks[i].d)))
               curTime = tasks[i].d

print(answer)
```

Thank you