# CẤU TRÚC HEAP

## I. GIỚI THIỆU

### 1. Khái niệm

Heap là một trong những cấu trúc dữ liệu đặc biệt quan trọng, nó giúp ta có thể giải được nhiều bài toán trong thời gian cho phép. Độ phức tạp thông thường khi làm việc với heap là *O(logN)*. Heap là một cây cân bằng thỏa mãn *tính chất heap*: nếu B là nút con của A thì khóa(A) ≥ khóa(B).

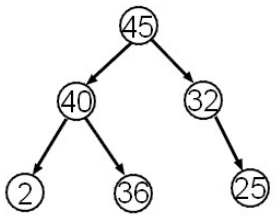
***Hệ quả***: khóa lớn nhất luôn nằm ở nút gốc, do đó một đống như vậy thường được gọi là heap-max.

Nếu mọi phép so sánh bị đảo ngược khiến cho khóa nhỏ nhất luôn nằm ở nút gốc thì đống đó gọi là heap-min. Không có hạn chế nào về số lượng nút con của mỗi nút trong đống nhưng thông thường mỗi nút có không quá hai nút con.

Cấu trúc heap là một cách thực hiện kiểu dữ liệu trừu tượng mang tên hàng đợi ưu tiên và có vai trò quan trọng trong nhiều thuật toán cho đồ thị chẳng hạn như thuật toán Dijkstra, hay thuật toán sắp xếp heapsort.

### 2. Các thao tác thường gặp trên heap

Mặc dù được mô tả như cây nhưng heap có thể được biểu diễn bằng mảng. Nút con của nút i là 2*i và 2*i+1. Do Heap là cây cân bằng nên độ cao của 1 nút luôn <= lgN.




| 45 | 40 | 32 | 2 | 36 | 25 |
|---|---|---|---|---|---|

*Mô hình biểu diễn heap bằng cây nhị phân và bằng mảng*

* **Khai báo Heap:**

```
Const
                                                    maxn=10000;
Var
                                                  nHeap:LongInt;
     Heap : array[0..maxn] of LongInt;
```

* **Upheap**: Nếu một nút lớn hơn nút cha của nó thì di chuyển nó lên trên

```
Procedure UpHeap(I : LongInt);
```

```
begin
    if (i = 1) or (Heap[i] < Heap[i div 2]) then exit; // Nếu i là nút gốc hoặc nhỏ hơn nút cha thì không làm việc
    swap(Heap[i] , Heap[i div 2]); // Đổi chỗ 2 phần tử trong Heap;
    UpHeap(i div 2); // Tiếp tục di chuyển lên trên
end;
```

*** Downheap**: Nếu một nút nhỏ hơn nút con thì đẩy nó xuống dưới

```
Procedure UpHeap(i : LongInt);
Begin
     if (i = 1) or (Heap[i] < Heap[i div 2]) then exit; // Nếu i là nút gốc hoặc nhỏ hơn nút cha thì không làm việc
     swap(Heap[i] , Heap[i div 2]); // Đổi chỗ 2 phần tử trong Heap;
     UpHeap(i div 2); // Tiếp tục di chuyển lên trên
end;
```

*** Push:** Đưa một phần tử mới vào Heap bằng cách thêm một nút vào cuối Heap và tiến hành UpHeap từ đây

```
Procedure Push(x : LongInt);
begin
     Inc(nHeap); // Tăng số phần tử của Heap
     Heap[nHeap] := x; // Thêm x vào Heap
     UpHeap(nHeap);
end;
```

*** Pop:** Lấy ra phần tử ở vị trí v trong Heap bằng cách gán Heap[v] := Heap[nHeap] rồi tiến hành chỉnh lại Heap

```
Function Pop(v : LongInt) : LongInt;
begin
     Pop := Heap[v]; // Lấy phần tử ở vị trí v ra khỏi Heap
     Heap[v] := Heap[nHeap]; // Đưa phần tử ở cuối Heap vào vị trí v
     Dec(nHeap); // Giảm số phần tử của Heap đi 1
     {Chỉnh lại Heap}
     UpHeap(v);
     DownHeap(v);
end;
```

## II. ỨNG DỤNG CỦA HEAP

### 1. Ứng dụng của heap trong giải thuật heapsort

### Tư tưởng thuật toán

*Đổi chỗ (Swap)*: Sau khi mảng a[1..n] đã là đống, lấy phần tử a[1] trên đỉnh của đống ra khỏi đống đặt vào vị trí cuối cùng n, và chuyển phần tử thứ cuối cùng a[n] lên đỉnh đống thì phần tử a[n] đã được đứng đúng vị trí.

*Vun lại*: Phần còn lại của mảng a[1..n-1] chỉ khác cấu trúc đống ở phần tử a[1]. Vun lại mảng này thành đống với n-1 phần tử.

*Lặp*: Tiếp tục với mảng a[1..n-1]. Quá trình dừng lại khi đống chỉ còn lại một phần tử.

Thuật toán Heapsort có độ phức tạp O(nlogn).

**2. Ứng dụng heap trong tổ chức hàng đợi ưu tiên**

Sau đây ta xét một số bài tập để thấy ứng dụng của heap trong tổ chức hàng đợi ưu tiên

**Bài 1:**

*Một người nông dân muốn cắt 1 thanh gỗ có độ dài L của mình thành N miếng, mỗi miếng có độ dài là 1 số nguyên dương A[i] (A[1] + A[2] + … A[N] = L). Tuy nhiên, để cắt một miếng gỗ có độ dài là X thành 2 phần thì ông ta sẽ mất X tiền. Ông nông dân này không giỏi tính toán lắm, vì vậy bạn được yêu cầu lập trình giúp ông ta cho biết cần để dành ít nhất bao nhiêu tiền thì mới có thể cắt được tấm gỗ như mong muốn .*

*Input:*

*- Dòng 1: một số nguyên dương T là số bộ test .*

*- T nhóm dòng tiếp theo mô tả các bộ test, mỗi nhóm dòng gồm 2 dòng: Dòng 1 là số nguyên dương N (1 ≤ N ≤ 20000); Dòng 2: N số nguyên dương A[1], …, A[N] (1 ≤ A[i] ≤ 50000)*

*Output: Kết quả mỗi test ghi ra trên 1 dòng, ghi ra 1 số nguyên dương duy nhất là chi phí tối thiểu cần để cắt tấm gỗ.*

Đầu tiên cho tất cả các phần độ dài trên vào heap min, thực hiện n-1 lần vòng lặp như sau: Lấy 2 phần tử đầu tiên (tức là 2 phần tử nhỏ nhất trong heap hiện tại ra), sau đó thêm 1 phần tử mới có giá trị bằng tổng 2 phần tử vừa lấy ra cho vào heap. Như thế sau n-1 lần, heap chỉ còn 1 phần tử duy nhất và giá trị của phần tử này chính là kết quả cần tìm. Chú ý: kết quả vượt quá longint nên phải để heap là int64.

**Bài 2:**

*Cho 2 dãy số nguyên A và B. Với mọi số A[i] thuộc A và B[j] thuộc B người ta tính tổng nó. Tất cả các tổng này sau khi được sắp xếp không giảm sẽ tạo thành dãy C. Nhiệm vụ của bạn là: Cho 2 dãy A, B. Tìm K số đầu tiên trong dãy C*

*Input:*

*- Dòng đầu tiên gồm 3 số: M, N, K*

*- M dòng tiếp theo gồm M số mô tả dãy A*

*- N dòng tiếp theo gồm N số mô tả dãy B*

*Output: Gồm K dòng tương ứng là K phần tử đầu tiên trong dãy C*

*Giới hạn:*

- *1 ≤ M, N, K ≤ 50000*
- *1 ≤ Ai, Bi ≤ 109*

Với $m,n \leq 50000$ thì không thể tạo một mảng $50000^2$ rồi sắp xếp lại được, vì vậy chúng ta cần những thuật toán tinh tế hơn, 1 trong những thuật toán đơn giản mà hiệu quả nhất là dùng heap.

Đầu tiên sắp xếp 2 mảng A và B không giảm, dĩ nhiên phần tử đầu tiên chính là A[1] + B[1], vấn đề là phần tử thứ 2 là A[2] + B[1] hay là A[1] + B[2], ta xử lý như sau:

Trước hết ta tạo một heap min gồm các phần tử A[1]+B[1], A[1]+B[2], …, A[1]+B[n], mỗi khi lấy ra phần tử ở gốc (tức phần tử có giá trị nhỏ nhất trong heap hiện tại) giả sử phần tử đó là A[i]+B[j] ta lại đẩy vào heap phần tử mới là A[i+1]+B[j] (nếu i=m thì không đẩy thêm phần tử nào vào heap). Cứ thế cho đến khi ta lấy đủ K phần tử cần tìm.

Độ phức tạp thuật toán trong trường hợp lớn nhất là O(50000*log(50000)), có thể chạy trong thời gian 1s, bộ nhớ là n.

**Bài 3:**

*Ở vương quốc XYZ, phép thuật không còn là điều xa lạ với các cư dân ở đây. Hằng ngày mọi người đều ăn ở và sinh hoạt với các pháp sư ở đây trong yên bình. Các pháp sư "tốt" đã tập hợp với nhau và thành lập các "bạch" hội với mục đích giúp đỡ người dân. Nhưng cái gì có mặt "tốt" thì cũng có mặt "xấu". Các "hắc" pháp sư là những pháp sư luôn muốn sử dụng sức mạnh của mình để ức hiếp người dân với tiêu chí "ai mạnh thì thắng". Những pháp sư này cũng đã cùng nhau thành lập các "hắc" hội. Các cuộc giao tranh giữa các pháp sư "tốt" và các "hắc" pháp sư đang ngày một căng thẳng. Đỉnh điểm là cuộc giao tranh sắp tới, một cuộc tổng tiến công của cả 2 phe pháp sư.*

*Theo thống kê thì có tất cả N "bạch" hội được đánh số từ 1 tới N theo thứ tự bất kỳ và mỗi hội có M pháp sư. N hội này đã hội họp lại với nhau để bàn chiến thuật cho cuộc chiến sắp tới với các "hắc" hội. Vì đây sẽ là cuộc chiến lâu dài nên chiến thuật được đưa ra là sẽ đưa từng tốp N pháp sư trong đó không có 2 pháp sư nào thuộc cùng một hội ra đánh thay phiên nhau. Chi tiết chiến thuật thì các tốp có sức mạnh yếu sẽ được đưa ra trước. Sức mạnh của một tốp pháp sư được đánh giá bằng tổng chỉ số sức mạnh của các pháp sư được chọn. Khi không cầm cự được nữa thì tốp đó sẽ quay về để tốp khác lên đánh. Một pháp sư có thể được chọn trong 2 đợt liên tiếp. Ngoài ra 2 tốp pháp sư liên tiếp nhau có thể mạnh hoặc yếu như nhau.*

*Yêu cầu: Tính sức mạnh của tốp pháp sư ở lượt thứ K.*

*Input:*

*- Dòng đầu chứa 3 số nguyên dương N, M, K.*

*- N dòng tiếp theo, dòng thứ i sẽ chứa M số nguyên dương là chỉ số sức mạnh của các pháp sư từ yếu tới mạnh trong "bạch" hội thứ i. Pháp sư u yếu hơn pháp sư v nếu chỉ số sức mạnh của pháp sư u nhỏ hơn pháp sư v.*

*Output: Chứa 1 số là tổng chỉ số sức mạnh của tốp pháp sư ở lượt thứ K.*

*Ràng buộc:*

- *K <= MN.*
- *Sức mạnh của các pháp sư <= 109.*
- *20% số test sẽ có N = 5 và M <= 20.*
- *20% số test tiếp theo sẽ có N = 2 và M <= 106.*
- *20% số test tiếp theo sẽ thỏa mãn: N = 3, M <= 105, tổng chỉ số sức mạnh trong mỗi hội <= 107.*
- *40% số test tiếp theo có N = 10, M <= 105 và K <= 106.*

Với $N=5, M \le 20$: Sinh $M^N$ số bằng đệ quy, sort lại rồi lấy số thứ $K$.

Với $N=2, M \le 10^6$: Chặt nhị phân kết quả rồi đếm trên 2 dãy với độ phức tạp O(N).

Với $N=3, M \le 10^5$:Chú ý rằng trong 1 hội thì tổng không vượt quá $10^7$, vậy nên số phần tử phân biệt trong 1 hội không vượt quá $\sqrt{10^7}$. Ta co dãy số lại rồi thực hiện thuật toán như trên.

Với $N=10, M \le 10^5, K \le 10^6$: Ta đưa về bài toán 2 trên N dãy: thực hiện thuật toán ở bài 2 cho 2 dãy đầu, được K số, tiếp tục áp dụng dãy K số đó với dãy thứ 3, được K số tiếp theo,… Làm lần lượt như vậy ta sẽ được K số bé nhất, đó là đáp án.

**Bài 4:**

*Cho một dãy số a1, a2, ..., an được sinh ngẫu nhiên như sau:*

*a1 = seed*

*ai = (ai-1*mul+add) % 65536*

*Với mul, add, seed là các số cho trước.*

*Cho một số k ≤ n. Dãy đã cho có n-k+1 dãy con độ dài k. Hãy tính tổng tất cả các phần tử trung vị (phần tử nhỏ thứ (k+1)/2) của n-k+1 dãy con này.*

*Input:*

*- Dòng đầu tiên chứa số test (không quá 30). Mỗi dòng tiếp theo chứa 5 số nguyên seed, mul, add, N, K. (0 ≤ seed, mul, add ≤ 65535, 1 ≤ N ≤ 250000, 1 ≤ K ≤ 5000, K ≤ N)*

*Output: Với mỗi test in ra số hiệu test (theo mẫu) cùng với tổng các trung vị tìm được.*

Giả sử ta có một dãy số tăng dần 1 → n, dùng 2 heap để quản lý dãy số này: 1 heap max quản lý các số từ 1 → n/2 (max), 1 heap min để quản lý n/2 + 1 → n (min). Gọi số phần tử heap max là Nmax, heap min là Nmin. Ta sẽ duy trì số lượng phần tử sao cho Nmax = Nmin hoặc Nmax-1 = Nmin (tức là heap min chỉ được hơn heap max tối đa 1 phần tử).

Từ 2 heap này có thể suy ra phần tử trung vị là max[1].

Cách duy trì 2 heap:

Khi push 1 phần tử giá trị x, có thể xác định được x được push vào heap nào (max hay min) bằng cách so sánh với phần tử trung vị hiện tại (max[1]). Ta push vào heap đó. Sẽ xảy ra 3 trường hợp sau:

- Nếu Nmax < Nmin thì ta pop min[1] và push sang max.

- Nếu Nmax = Nmin+2, ta pop max[1] và push sang min.

- Nếu Nmax = Nmin, bỏ qua.

Khi pop 1 phần tử ra khỏi dãy số cũng tương tự.

Thuật toán: Xây dựng 2 heap max và min với K phần tử đầu. Giả sử đoạn tiếp theo xét là từ i -> i+k-1, ta pop a[i-1] và push a[i+k-1] theo cách duy trì trên.

**Bài 5:**

*Sau khi thực thi quy hoạch của Bộ Giao thông, sơ đồ giao thông của thành phố H gồm n tuyến đường ngang và n tuyến đường dọc cắt nhau tạo thành một lưới ô vuông với n x n nút giao thông. Các nút giao thông được gán tọa độ theo hàng từ 1 đến n, từ trên xuống dưới và theo cột từ 1 đến n, từ trái sang phải. Ban chỉ đạo an toàn giao thông quyết định điều n cảnh sát giao thông đến các nút giao thông làm nhiệm vụ. Ban đầu mỗi cảnh sát được phân công đứng trên một nút của một tuyến đường ngang khác nhau. Đến giờ cao điểm, xuất hiện ùn tắc tại các tuyến đường dọc không có cảnh sát giao thông. Để sớm giải quyết tình trạng này, Ban chỉ đạo an toàn giao thông quyết định điều động một số cảnh sát giao thông ở một số nút, từ nút hiện tại sang một nút khác cùng hàng ngang để đảm bảo mỗi tuyến đường dọc đều có mặt của cảnh sát giao thông.*

*Yêu cầu: Biết rằng cảnh sát ở hàng ngang thứ i cần ti đơn vị thời gian để di chuyển qua 1 cạnh của lưới ô vuông (i = 1, 2, ..., n), hãy giúp Ban chỉ đạo an toàn giao thông tìm cách điều động các cảnh sát thỏa mãn yêu cầu đặt ra sao cho việc điều động được hoàn thành tại thời điểm sớm nhất. Giả thiết là các cảnh sát được điều động đồng thời thực hiện việc di chuyển đến vị trí mới tại thời điểm 0.*

*Ràng buộc: 50% số tests ứng với 50% số điểm của bài có n ≤ 100.*

*Input:*

*- Dòng thứ nhất chứa một số nguyên dương n (n ≤ 10000).*

*- Dòng thứ i trong số n dòng tiếp theo chứa hai số nguyên dương $c_i$, $t_i$ ($t_i$ ≤ 10000) tương ứng là tọa độ cột và thời gian để di chuyển qua 1 cạnh của lưới ô vuông của cảnh sát đứng trên tuyến đường ngang thứ i (i = 1, 2, ..., n).*

*Hai số trên cùng một dòng được ghi cách nhau ít nhất một dấu cách.*

*Output: Ghi ra một số nguyên duy nhất là thời điểm sớm nhất tìm được.*

Chặt nhị phân kết quả, giả sử là x. Với mỗi cảnh sát ta xác định được lo[i] và hi[i] là vị trí mà cảnh sát i có thể di chuyển được trong thời gian x. Bài toán quy về xếp các cảnh sát vào vị trí ans[i] sao cho $lo[i] \le ans[i] \le hi[i]$ và $ans[i] \ne ans[j] với i \ne j$.

Sắp xếp các cảnh sát theo tăng dần theo lo, với một cột j, ta sẽ tìm cảnh sát i thỏa mãn $lo[i] \le j \le hi[i]$ để xếp vào và $hi[i]$ là nhỏ nhất trong các i thỏa mãn điều kiện. Để thực hiện thao tác này, ta dùng 1 heap min để chứa các hi[i].

Code:

```
i := 1;
for j := 1 to n do
 begin
      while (i <= n) and (lo[i] <= j) do
            begin
                  push(hi[i]);
                  inc(i);
            end;
      u := pop;
      ans[u] := j;
 end;
```

### 3. Ứng dụng của heap trong lý thuyết đồ thị

Ứng dụng của heap trong lý thuyết đồ thị là giúp cải tiến tốc độ thực hiện các thuật toán. Chẳng hạn:

#### a) Trong thuật toán Dijkstra

Trường hợp đồ thị có nhiều đỉnh, ít cạnh, thuật toán cần n lần cố định nhãn và mỗi lần tìm đỉnh để cố định nhãn sẽ mất một đoạn chương trình với độ phức tạp O(n). Để tăng tốc độ, người ta thường sử dụng cấu trúc dữ liệu Heap để lưu các đỉnh chưa cố định nhãn. Heap ở đây là một cây nhị phân hoàn chỉnh thỏa mãn: Nếu u là đỉnh lưu ở nút cha và v là đỉnh ở nút con thì d[u] ≤ d[v] (đỉnh r lưu ở gốc Heap là đỉnh có d[r] nhỏ nhất).

Tại mỗi bước lặp của thuật toán Dijkstra có hai thao tác: Tìm đỉnh cố định nhãn và sửa nhãn

- Thao tác tìm đỉnh cố định nhãn sẽ lấy đỉnh lưu ở gốc Heap, cố định nhãn, đưa phần tử cuối Heap và thế chỗ và thực hiện việc vun đống.

- Thao tác sửa nhãn, sẽ duyệt danh sách kề của đỉnh vừa cố định nhãn và sửa nhãn những đỉnh tự do kề với đỉnh này, mỗi lần sửa nhãn một đỉnh nào đó, ta xác định đỉnh này nằm ở đâu trong Heap và thực hiện việc chuyển đỉnh đó lên phía gốc Heap nếu cần bảo toàn cấu trúc Heap.

Thuật toán Dijkstra có độ phức tạp là $O(n^2)$, việc kết hợp với cấu trúc dữ liệu Heap giúp thuật toán được cải tiến và có độ phức tạp là O(max(n,m).logn).

**b) Trong thuật toán Kruskal tìm cây khung nhỏ nhất**

Một trong những vấn đề quan trọng là làm thế nào để xét được các cạnh từ cạnh có trọng số nhỏ tới cạnh có trọng số lớn. Ta có thể thực hiện bằng cách sắp xếp danh sách cạnh theo thứ tự không giảm của trọng số, sau đó duyệt từ đầu tới cuối danh sách cạnh, nên sử dụng các thuật toán sắp xếp hiệu quả để đạt được tốc độ nhanh trong trường hợp số cạnh lớn. Trong trường hợp tổng quát, thuật toán Heapsort là hiệu quả nhất bởi nó cho phép chọn lần lượt các cạnh từ cạnh trọng số nhỏ nhất tới cạnh trọng số lớn nhất ra khỏi Heap và có thể xử lý (bỏ qua hay thêm vào cây) luôn.

Thuật toán Kruskal có độ phức tạp là O(mlogn), nếu đồ thị có cây khung thì $m \geq n-1$ thì chi phí thời gian chủ yếu sẽ nằm ở thao tác sắp xếp danh sách cạnh bởi độ phức tạp của HeapSort là O(mlogm), do đó độ phức tạp tính toán của thuật toán là O(mlogm) trong trường hợp xấu nhất.

**c) Trong thuật toán Prim tìm cây khung nhỏ nhất**

Xét về độ phức tạp tính toán, thuật toán Prim có độ phức tạp là $O(n^2)$. Tương tự thuật toán Dijkstra, nếu kết hợp thuật toán Prim với cấu trúc Heap sẽ được một thuật toán với độ phức tạp O((m+n)logn).

## III. KẾT LUẬN

Có thể thấy cấu trúc dữ liệu heap là một cấu trúc dữ liệu quan trọng và có tính ứng dụng cao, kết hợp các thuật toán với cấu trúc Heap giúp cải tiến một cách đáng kể thời gian thực hiện thuật toán. Ngoài ra, có rất cấu trúc dữ liệu heap còn có nhiều dạng biến thể như B-heap, Binary heap, Fibonacci heap….