TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

————-oOo————-

BÀI TẬP SỐ 04.d
THIẾT KẾ GIẢI THUẬT
PHƯƠNG PHÁP THAM LAM

Giảng viên hướng dẫn: ThS. Huỳnh Thị Thanh Thương

Nhóm sinh viên:

1. Phan Thanh Hải 18520705

TP. HỒ CHÍ MINH, 13/02/2020

Mục lục

Bài tập 1

BÀI TOÁN NGƯỜI GIAO HÀNG (TRAVELING SALESMAN PROBLEM)

Có một người giao hàng cần đi giao hàng tại n thành phố. Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, khoảng cách từ một thành phố đến các thành phố khác là xác định được. Giả thiết rằng mỗi thành phố đều có đường đi đến các thành phố còn lại. Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, có thể là cước phí di chuyển hoặc thời gian di chuyển. Ta gọi chung là độ dài. Hãy tìm một chu trình (một đường đi khép kín thỏa mãn điều kiện trên) sao cho tổng độ dài các cạnh là nhỏ nhất. Hay còn nói là tìm một phương án có giá nhỏ nhất.

a. Phát biểu bài toán

INPUT

Số lượng các thành phố và khoảng cách từ mỗi thành phố này đến mỗi thành phố kia.

OUTPUT

Độ dài của chu trình mà người giao hàng đi qua tất cả các thành phố (mỗi thành phố chỉ đi qua đúng 1 lần) sao cho đạt giá trị nhỏ nhất.

$$x_{ij} = \begin{cases} 1 & \text{đường đi từ thành phố } i \text{ đến thành phố } j \\ 0 & \text{ngược lại} \end{cases}$$

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n;$$

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Chu trình đi qua hết các thành phố, mỗi thành phố chỉ đi qua đúng 1 lần, là nhỏ nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi đường đi từ thành phố này đến các thành phố kia, ta sẽ lựa chọn đường đi ngắn nhất.

Tính chính xác: Giải thuật này trong một số trường hợp có thể đem lại kết quả không tối ưu.

Bước 1: Chọn thành phố đầu tiên là điểm bắt đầu.

Bước 2: Trong những thành phố còn lại chưa đi qua, chọn thành phố có đường đi ngắn nhất từ thành phố bắt đầu.

Bước 3: Cập nhật đã đi qua thành phố đó và lấy thành phố đó là thành phố bắt đầu để đi tiếp.

Bước 4: Lặp lại bước 2 cho đến khi đã đi hết các thành phố (đi qua n thành phố).

c. Thuật giải

Bài toán có thể biểu diễn bởi một đồ thị vô hướng có trọng số $G = (V, E)$, trong đó mỗi thành phố được biểu diễn bởi một đỉnh, cạnh nối hai đỉnh biểu diễn cho đường đi giữa hai thành phố và trọng số của cạnh là khoảng cách giữa hai thành phố.

// Input: Một ma trận kề *cost* lưu các giá trị là khoảng cách từ thành phố này đến thành phố kia.

// Output: Độ dài của chu trình mà người giao hàng đi qua tất cả các thành phố (mỗi thành phố chỉ đi qua đúng 1 lần) sao cho đạt giá trị nhỏ nhất.

TSP($cost[1..n, 1..n]$)

```
1.  sum = 0;
2.  start = 1;
3.  KiemTra[start] = 1;
4.  for i = 1 to n do
5.      temp = 0;
6.      min = ∞;
7.      for j = 1 to n − 1 do
8.          if cost[start, j] > 0 and KiemTra[j] = 0
9.              if cost[start, j] < min
10.                 min = cost[start, j];
11.                 temp = j;
12.     sum = sum + min;
13.     start = temp;
14.     KiemTra[start] = 1;
15. sum = sum + cost[start, 1];
16. return sum;
```

d. VD minh họa

INPUT	OUTPUT
3 0 1 2 1 0 3 2 3 0	6
4 0 10 15 20 10 0 35 25 15 35 0 30 20 25 30 0	80
5 0 20 42 25 30 20 0 30 34 15 42 30 0 10 10 25 34 10 0 25 30 15 10 25 0	80

e. Độ phức tạp của thuật toán

Quá trình chọn đường đi ngắn nhất giữa 2 thành phố có thời gian thực hiện là $O(n)$.

Quá trình tạo thành chu trình từ n đường đi ngắn nhất sẽ có thời gian thực hiện là $O(n^2)$.

Vậy độ phức tạp của hàm thời gian thực hiện chương trình là $O(n^2)$.

Bài tập 2.1

0/1 KNAPSACK PROBLEM

Cho n đồ vật và một cái ba lô có thể đựng trọng lượng tối đa M, mỗi đồ vật i có trọng lượng w_i và giá trị là p_i.

Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá M và tổng giá trị là lớn nhất.

Mỗi đồ vật hoặc là lấy đi hoặc là bỏ lại.

a. Phát biểu bài toán

INPUT

Tập hợp n đồ vật bao gồm thông tin về giá trị và trọng lượng của chúng.

Trọng lượng tối đa của ba lô.

OUTPUT

Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

$\max \sum_{i=1}^{n} p_i x_i;$

thỏa mãn $\sum_{i=1}^{n} w_i x_i \leq M$ và $x_i \in \{0, 1\}$.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Tổng giá trị của các đồ vật cho vào ba lô là lớn nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi lần chọn một đồ vật cho vào ba lô, ta sẽ chọn đồ vật có giá trị $\frac{p_i}{w_i}$ lớn nhất.

Tính chính xác: Giải thuật này trong một số trường hợp có thể đem lại kết quả không tối ưu.

Bước 1: Chọn đồ vật có giá trị $\frac{p_i}{w_i}$ lớn nhất trong tập hiện hành và kiểm tra xem nếu đồ vật có trọng lượng nhỏ hơn trọng lượng hiện tại của ba lô thì mới đưa vào ba lô.

Bước 2: Cập nhật lại trọng lượng hiện tại của ba lô (lấy trọng lượng tối đa ban đầu trừ đi cho trọng lượng của đồ vật mới cho vào, nếu có).

Bước 3: Bỏ qua đồ vật đã chọn trên, lặp lại bước 1 đối với các đồ vật còn lại cho đến khi không thể bỏ thêm đồ vật vào ba lô nữa.

c. Thuật giải

// Input:

Giá trị p và trọng lượng w của n đồ vật tương ứng.

Trọng lượng tối đa của ba lô là M.

// Output: Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

FractionalKnapsack$(p[1..n], w[1..n], x[1..n], M)$

1. Sắp xếp các đồ vật theo thứ tự không tăng của tỷ số price/weight ($p[1]/w[1] \geq p[2]/w[2] \geq ... \geq p[n]/w[n]$);
2. **for** $i = 1$ **to** n **do**
3. $\quad x[i] = 0$;
4. **for** $i = 1$ **to** n **do**
5. $\quad$ **if** $w[i] \leq M$
6. $\quad\quad x[i] = 1$;
7. $\quad\quad M = M - w[i]$;
8. **return** x;

Để tính tổng giá trị của các đồ vật cho vào túi thì ta có thể sử dụng hàm:

TinhTong$(p[1..n], x[1..n])$

1. $sum = 0$;
2. **for** $i = 1$ **to** n **do**
3. $\quad sum = sum + p[i]x[i]$;
4. **return** sum;

d. VD minh họa

INPUT	OUTPUT
3 1 4 2 5 3 1 4	3
4 10 5 40 4 30 6 50 3 10	90
5 33 15 24 20 36 17 37 8 12 31 80	130

e. Độ phức tạp của thuật toán

Quá trình sắp xếp các đồ vật, giả sử ta sử dụng thuật toán sắp xếp tốt nhất hiện giờ thì thời gian thực hiện là $O(n \log n)$.

Quá trình tính toán từng giá trị cho $x[i]$ có thời gian thực hiện là $O(1)$. Quá trình tính toán này thực hiện n lần nên thời gian thực hiện là $O(n)$.

Vậy độ phức tạp của hàm thời gian thực hiện chương trình là $O(n \log n + n) = O(n \log n)$.

Bài tập 2.2

FRACTIONAL KNAPSACK PROBLEM

Cho n đồ vật và một cái ba lô có thể đựng trọng lượng tối đa M, mỗi đồ vật i có trọng lượng w_i và giá trị là p_i.

Đồ vật có thể được tháo rời/bẻ ra làm nhiều phần. Một phần x_i ($0 \leq x_i \leq 1$) của đồ vật i có giá trị là $p_i x_i$ (VD: $\frac{1}{2}$ đồ vật i có giá trị $\frac{1}{2}$ p_i).

Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá M và tổng giá trị là lớn nhất. Có thể lấy đi 1 phần của đồ vật.

a. Phát biểu bài toán

INPUT

Tập hợp n đồ vật bao gồm thông tin về giá trị và trọng lượng của chúng.

Trọng lượng tối đa của ba lô.

OUTPUT

Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

$\max \sum_{i=1}^{n} p_i x_i;$

thỏa mãn $\sum_{i=1}^{n} w_i x_i \leq M$ và $0 \leq x_i \leq 1$, x_i là số thực.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Tổng giá trị của các đồ vật cho vào ba lô là lớn nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi lần chọn một đồ vật cho vào ba lô, ta sẽ chọn đồ vật có giá trị $\frac{p_i}{w_i}$ lớn nhất.

Bước 1: Chọn đồ vật có giá trị $\frac{p_i}{w_i}$ lớn nhất trong tập hiện hành và kiểm tra xem nếu đồ vật có trọng lượng nhỏ hơn trọng lượng hiện tại của ba lô thì mới đưa vào ba lô.

Bước 2: Cập nhật lại trọng lượng hiện tại của ba lô (lấy trọng lượng tối đa ban đầu trừ đi cho trọng lượng của đồ vật mới cho vào).

Bước 3: Bỏ qua đồ vật đã chọn trên, lặp lại bước 1 đối với các đồ vật còn lại cho đến khi không thể bỏ thêm toàn bộ nguyên phần của đồ vật vào ba lô nữa.

Bước 4: Nếu trọng lượng hiện tại của ba lô vẫn còn và còn đồ vật để bỏ vào, ta tiến hành tháo rời đồ vật có giá trị $\frac{p_i}{w_i}$ lớn nhất trong tập hiện hành ra và lấy đúng một ít phần trong số đó cho vào ba lô mà không vượt quá trọng lượng hiện tại của ba lô.

c. Thuật giải

// Input:

Giá trị p và trọng lượng w của n đồ vật tương ứng.

Trọng lượng tối đa của ba lô là M.

// Output: Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

FractionalKnapsack$(p[1..n], w[1..n], x[1..n], M)$

1. Sắp xếp các đồ vật theo thứ tự không tăng của tỷ số price/weight ($p[1]/w[1] \leq p[2]/w[2] \leq ... \leq p[n]/w[n]$);
2. **for** $i = 1$ **to** n **do**
3. $\quad x[i] = 0$;
4. **for** $i = 1$ **to** n **do**
5. $\quad$ **if** $w[i] > M$
6. $\quad\quad$ Kết thúc vòng lặp;
7. $\quad$ **else**
8. $\quad\quad x[i] = 1$;
9. $\quad\quad M = M - w[i]$;
10. **if** $i \leq n$
11. $\quad x[i] = M/w[i]$;
12. **return** x;

Để tính tổng giá trị của các đồ vật cho vào túi thì ta có thể sử dụng hàm:

TinhTong$(p[1..n], x[1..n])$

1. $sum = 0$;
2. **for** $i = 1$ **to** n **do**
3. $\quad sum = sum + p[i]x[i]$;
4. **return** sum;

d. VD minh họa

INPUT	OUTPUT
3 25 18 24 15 15 10 20	31.5
1 20 20 15	15
4 50 25 60 100 80 35 100 40 15	37.5

e. Độ phức tạp của thuật toán

Quá trình sắp xếp các đồ vật, giả sử ta sử dụng thuật toán sắp xếp tốt nhất hiện giờ thì thời gian thực hiện là $O(n \log n)$.

Quá trình tính toán từng giá trị cho $x[i]$ có thời gian thực hiện là $O(1)$. Quá trình tính toán này thực hiện n lần nên thời gian thực hiện là $O(n)$.

Cộng thêm quá trình tại **bước 4** của giải thuật thì độ phức tạp của hàm thời gian thực hiện chương trình là $O(n \log n)$.

Bài tập 2.3

UNBOUNDED KNAPSACK PROBLEM

Cho một cái ba lô có thể đựng trọng lượng M với n loại đồ vật, mỗi đồ vật loại i có trọng lượng w_i và giá trị là p_i. Chọn một cách lựa chọn các đồ vật cho vào túi sao cho trọng lượng không quá M và tổng giá trị là lớn nhất.

Có thể chọn nhiều đồ vật cùng loại. Giả sử mỗi loại đồ vật không giới hạn về số lượng.

a. Phát biểu bài toán

INPUT

Tập hợp n đồ vật bao gồm thông tin về giá trị và trọng lượng của chúng.

Trọng lượng tối đa của ba lô.

OUTPUT

Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

$\max \sum_{i=1}^{n} p_i x_i$;

thỏa mãn $\sum_{i=1}^{n} w_i x_i \leq M$ và $x_i \geq 0$, x_i là một số nguyên.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Tổng giá trị của các đồ vật cho vào ba lô là lớn nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi lần chọn một đồ vật cho vào ba lô, ta sẽ chọn những đồ vật cùng loại có giá trị $\frac{p_i}{w_i}$ lớn nhất.

Tính chính xác: Giải thuật này trong một số trường hợp có thể đem lại kết quả không tối ưu.

Bước 1: Chọn những đồ vật cùng loại có giá trị $\frac{p_i}{w_i}$ lớn nhất trong tập hiện hành và kiểm tra xem nếu tổng trọng của những đồ vật cùng loại đó nhỏ hơn trọng lượng hiện tại của ba lô thì mới đưa vào ba lô.

Bước 2: Cập nhật lại trọng lượng hiện tại của ba lô (lấy trọng lượng tối đa ban đầu trừ đi cho tổng trọng lượng của những đồ vật cùng loại mới cho vào, nếu có).

Bước 3: Bỏ qua những đồ vật cùng loại đã chọn trên, lặp lại bước 1 đối với những đồ vật cùng loại còn lại cho đến khi không thể bỏ thêm đồ vật vào ba lô nữa.

c. Thuật giải

// Input:

Giá trị p và trọng lượng w của n đồ vật tương ứng.

Trọng lượng tối đa của ba lô là M.

// Output: Tổng giá trị của tất cả đồ vật cho vào ba lô sao cho trọng lượng không quá M.

UnboundedKnapsack$(p[1..n], w[1..n], x[1..n], M)$

1. Sắp xếp các đồ vật theo thứ tự không tăng của tỷ số price/weight ($p[1]/w[1] \leq p[2]/w[2] \leq ... \leq p[n]/w[n]$);
2. **for** $i = 1$ **to** n **do**
3. $\quad x[i] = 0$;
4. **for** $i = 1$ **to** n **do**
5. $\quad$ **while** $w[i] \leq M$ **do**
6. $\quad\quad x[i] = x[i] + 1$;
7. $\quad\quad M = M - w[i]$;
8. **return** x;

Để tính tổng giá trị của các đồ vật cho vào túi thì ta có thể sử dụng hàm:

TinhTong$(p[1..n], x[1..n])$

1. $sum = 0$;
2. **for** $i = 1$ **to** n **do**
3. $\quad sum = sum + p[i]x[i]$;
4. **return** sum;

d. VD minh họa

INPUT	OUTPUT
3 1 4 2 5 3 1 4	12
2 1 1 30 50 100	100
4 10 1 40 3 50 4 80 5 8	120

e. Độ phức tạp của thuật toán

Quá trình sắp xếp các đồ vật, giả sử ta sử dụng thuật toán sắp xếp tốt nhất hiện giờ thì thời gian thực hiện là $O(n \log n)$.

Quá trình tính toán từng giá trị cho $x[i]$ có thời gian thực hiện là $O(1)$. Quá trình tính toán này thực hiện n lần nên thời gian thực hiện là $O(n)$.

Vậy độ phức tạp của hàm thời gian thực hiện chương trình là $O(n \log n + n) = O(n \log n)$.

Bài tập 3

BÀI TOÁN TRẢ TIỀN CỦA ATM

Trong máy ATM có chuẩn bị sẵn các loại tiền 50K, 100K, 200K, 500K. Giả sử số lượng không hạn chế. Khi có một khách hàng cần rút n đồng, với n chia hết cho 50K. Tìm phương án để máy ATM trả ra n đồng với số lượng tờ là thấp nhất.

a. Phát biểu bài toán

INPUT

Số tiền cần rút từ máy ATM.

Các loại tiền có thể rút từ máy ATM.

OUTPUT

Số lượng tờ tiền cần rút là thấp nhất.

$\min \sum_{j=1}^{4} x_j;$

thỏa mãn $\sum_{j=1}^{4} w_j x_j = n$

với x_j là số lượng của tờ tiền mệnh giá thứ j và w_j là giá trị của tờ tiền mệnh giá thứ j.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Số lượng tờ tiền cần rút là thấp nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi tờ tiền rút ra có giá trị lớn nhất trong mức cho phép.

Bước 1: Chọn mệnh giá tiền lớn nhất thỏa mãn $loai\_tien[k] \leq n$.

Bước 2: Rút 1 tờ có mệnh giá ở bước 1.

Bước 3: Cập nhật số tiền còn lại cần rút.

Bước 4: Lặp lại bước 1 cho đến khi đã rút hết n đồng.

c. Thuật giải

Ta dùng mảng để lưu trữ các mệnh giá tiền có thể đổi: $loai\_tienp[1..4] = 500, 200, 100, 50$ (chú ý là các giá trị trong mảng phải là những giá trị giảm dần).

// Input:

Số tiền n cần rút và một mảng $loai\_tien$ thể hiện các loại tiền có thể rút từ máy ATM.

// Output:

Số lượng tờ tiền cần rút là thấp nhất.

TraTienATM($loai\_tien[1..4], n$)

1. $S = \varnothing$;
2. $k = 0$;
3. **while** $n \neq 0$ **do**
4. // Chọn mệnh giá tiền lớn nhất thỏa mãn $loai\_tien[k] \leq n$
5. **while** $n < loai\_tien[k]$ **do**
6. $k = k + 1$;
7. $S = S \cup loai\_tien[k]$;
8. $n = n - loai\_tien[k]$;
9. **return** S;

d. VD minh họa

INPUT	OUTPUT
100	1
3250	8
105300	212

e. Độ phức tạp của thuật toán

Độ phức tạp của hàm thời gian thực hiện chương trình, trong trường hợp trung bình là O(n).

Bài tập 4

BÀI TOÁN HÀN CÁC ĐIỂM TRÊN TẤM KIM LOẠI

Bài tập này ý tưởng thuật giải và nội dung khá giống với **bài tập 1. Bài toán người giao hàng** nên không trình bày lại.

Bài tập 5

BÀI TOÁN CHỌN HOẠT ĐỘNG (INTERVAL SCHEDULING)

Cho một tập các hoạt động $S = 1, 2, ..., n$. Một hoạt động i có thời điểm bắt đầu là s_i và thời điểm chấm dứt là $f_i, s_i < f_i$. Nếu hoạt động i được chọn thì i tiến hành trong thời gian $[s_i, f_i)$.

Hai hoạt động i và j là "tương thích nhau" (compatible) nếu $[s_i, f_i)$ và $[s_j, f_j)$ không chạm nhau ($f_i \leq s_j$ hoặc $f_j \leq s_i$).

Yêu cầu: Tìm tập hợp lớn nhất các hoạt động tương thích nhau?

a. Phát biểu bài toán

INPUT

Tập các hoạt động gồm có thông tin về thời gian bắt đầu và thời gian kết thúc.

OUTPUT

Số hoạt động tương thích được chọn là nhiều nhất và thông tin của các hoạt động đó.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Số hoạt động tương thích được chọn là nhiều nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi bước chọn hoạt động, ta sẽ chọn hoạt động có thời gian kết thúc là sớm nhất.

Bước 1: Sắp xếp các hoạt động theo thứ tự không giảm của thời gian kết thúc ($A[1].finish \leq A[2].finish \leq ... \leq A[n].finish$)

Bước 2: Chọn hoạt động có thời gian kết thúc là sớm nhất trong tập hiện hành.

Bước 3: Bỏ qua những hoạt động không tương thích với hoạt động trên.

Bước 4: Lặp lại bước 2 cho đến khi không còn hoạt động nào nữa để chọn.

c. Thuật giải

// Input: Một tập các hoạt động được lưu trên mảng một chiều A, mỗi một hoạt động sẽ có thời gian bắt đầu *start* và thời gian kết thúc $finish$.

// Output: Số hoạt động tương thích được chọn là nhiều nhất và thông tin của các hoạt động đó.

IntervalScheduling($A[1..n]$)

1. Sắp xếp các hoạt động theo thứ tự không giảm của thời gian kết thúc ($A[1].finish \leq A[2].finish \leq ... \leq A[n].finish$);
2. $S = \varnothing$;
3. $prev\_finish = 0$;
4. **for** $i = 1$ **to** n **do**
5. **if** $A[i].start \geq prev\_finish$;
6. $S = S \cup A[i]$;
7. $prev\_finish = A[i].finish$;
8. **return** S;

d. VD minh họa

INPUT	OUTPUT
3 1 2 5 8 8 10	1 2 5 8 8 10
6 5 9 1 2 3 4 0 6 5 7 8 9	1 2 3 4 5 7 8 9
6 1 3 2 4 0 2 5 6 7 8 6 10	0 2 2 4 5 6 7 8

e. Độ phức tạp của thuật toán

Quá trình sắp xếp các hoạt động, giả sử ta sử dụng thuật toán sắp xếp tốt nhất hiện giờ thì thời gian thực hiện là $O(n \log n)$.

Quá trình chọn hoạt động đưa vào solution có thời gian thực hiện là $O(n)$.

Vậy độ phức tạp của hàm thời gian thực hiện chương trình là $O(n \log n + n) = O(n \log n)$.

Bài tập 6

OPTIMAL STORAGE ON TAPES

Cho n file $f_1, f_2, ..., f_n$ với độ dài (thời gian) file tương ứng là $l_1, l_2, ..., l_n$. Tìm thứ tự tốt nhất để lưu n file này trên băng từ sao cho thời gian đọc hết n file là ít nhất biết:

1. Khi đọc một file phải tua lại từ đầu.
2. Thời gian đọc một file bằng độ dài của tất cả các file đọc trước cùng với độ dài của file cần đọc.

a. Phát biểu bài toán

INPUT

Danh sách các file và độ dài file tương ứng.

OUTPUT

Trình tự lưu file sao cho thời gian đọc hết tất cả các file là ít nhất.

$$\min \sum_{j=1}^{n} \sum_{k=1}^{j} l_{i_k};$$

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Thời gian đọc hết n là ít nhất.

Tiêu chuẩn tối ưu cục bộ: Chọn file đầu tiên sẽ có độ dài tương ứng nhỏ nhất, file thứ hai sẽ có độ dài tương ứng nhỏ thứ hai...

Quá trình: Sắp xếp các file theo thự tự không giảm của độ dài file ($l_1 \leq l_2 \leq ... \leq l_n$).

Kết quả đã được tổng hợp một cách hiển nhiên nên không cần làm gì thêm.

c. Thuật giải

// Input: Một mảng lưu trữ l độ dài của n file.

// Output: Trình tự lưu n file được thể hiện bằng thứ tự của độ dài file tương ứng sao cho thời gian đọc hết n file là ít nhất.

OptimalStorage($l[1..n]$)

1. Sắp xếp các file theo thứ tự không giảm của độ dài file ($l_1 \leq l_2 \leq ... \leq l_n$);
2. **return** l;

Để tính tổng thời gian đọc hết n file thì ta có thể sử dụng hàm:

RetrievalTime($l[1..n]$)

1. $sum = 0$;
2. $count = n$;
3. **for** $i = 1$ **to** n **do**
4. $\quad sum = sum + l[i] * count$;
5. $\quad count = count - 1$;
6. **return** sum;

d. VD minh họa

INPUT	OUTPUT
3 5 10 3	3 5 10 29
3 2 5 4	2 4 5 19
13 12 5 8 32 7 5 18 26 4 3 11 10 6	3 4 5 5 6 7 8 10 11 12 18 26 32 659

e. Độ phức tạp của thuật toán

Quá trình sắp xếp các file, giả sử ta sử dụng thuật toán sắp xếp tốt nhất hiện giờ thì thời gian thực hiện là $O(n \log n)$.

Sau khi sắp xếp thì kết quả đã được tổng hợp một cách hiển nhiên. Do đó độ phức tạp của hàm thời gian thực hiện chương tình là $O(n \log n)$.

Bài tập 7

PICK K NUMBERS

Lấy k số từ tập n số sao cho tổng k số lấy được là lớn nhất.

a. Phát biểu bài toán

INPUT

Tập hợp các số nguyên.

Số lượng k phần tử cần lấy ra từ tập hợp trên.

OUTPUT

Tổng k số lấy từ tập hợp các số nguyên sao cho tổng đạt giá trị lớn nhất.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Tổng k số lấy từ tập n số đạt giá trị lớn nhất.

Tiêu chuẩn tối ưu cục bộ: Mỗi lần lấy 1 số từ tập n số, ta sẽ lấy số có giá trị lớn nhất.

Bước 1: Chọn số lớn nhất có trong tập số hiện hành.

Bước 2: Đưa số đó vào solution và đồng thời xóa số đó trong tập số ban đầu.

Bước 3: Lặp lại bước 1 cho đến khi đã lấy được đủ k số.

c. Thuật giải

// Input: Một mảng *array* gồm n các số nguyên và số nguyên k.

// Output: Tổng k số lấy từ mảng trên sao cho tổng đạt giá trị lớn nhất.

PickNumbers$(arr[1..n], k)$

1. $S = \varnothing$;
2. **for** $i = 1$ **to** k **do**
3. $\quad pos =$ **ViTriMax**(arr);
4. $\quad S = S \cup arr[pos]$;
5. $\quad$ **XoaPhanTu**(arr, pos);
6. **return** S;

Hàm **ViTriMax** sẽ trả về kết quả là vị trí của phần tử có giá trị lớn nhất trong mảng.

Hàm **XoaPhanTu** sẽ thực hiện việc xóa phần tử tại vị trí *pos* có trong mảng.

Để tìm tổng k số ta chỉ cần tính tổng các số trong solution.

TinhTong($S[1..k]$)

1. $sum = 0$;
2. **for** $i = 1$ **to** k **do**
3. $\quad sum = sum + S[i]$;
4. **return** sum;

d. VD minh họa

INPUT	OUTPUT
3 5 3 8 2	13
4 2 4 5 6 4	17
6 7 12 4 9 8 5 1	12

e. Độ phức tạp của thuật toán

Quá trình tìm kiếm phần tử có giá trị lớn nhất trong mảng, cập nhật solution và xóa phần tử trong mảng tất cả đều có thời gian thực hiện là O(n).

Công việc trên được thực hiện hết k lần cho nên độ phức tạp của hàm thời gian thực hiện chương trình là O(kn) (với k là hằng số).

Bài tập 8

REMOVE K DIGITS

Ta có một số nguyên có n chữ số. Hãy tìm cách xóa đi k chữ số trong số nguyên trên sao cho kết quả thu được là nhỏ nhất có thể.

a. Phát biểu bài toán

INPUT

Một số nguyên dương được biểu diễn bởi một xâu ký tự và số chữ số cần xóa.

OUTPUT

Số ban đầu sau khi đã xóa đi k chữ số sao cho số thu được đạt giá trị nhỏ nhất có thể.

b. Ý tưởng giải

Tiêu chuẩn tối ưu toàn cục: Xóa đi k chữ số trong số nguyên được kết quả là số nhỏ nhất có thể.

Tiêu chuẩn tối ưu cục bộ: Mỗi lần xóa 1 chữ số, ta sẽ xóa chữ số đầu tiên có giá trị lớn hơn hoặc bằng so với số kế tiếp ($number[i] \geq number[i+1]$).

Bước 1: Chọn chữ số số đầu tiên có giá trị lớn hơn hoặc bằng so với số kế tiếp ($number[i] \geq number[i+1]$).

Bước 2: Xóa chữ số đó trong số nguyên.

Bước 3: Lặp lại bước 1 cho đến khi đã xóa được đủ k số.

c. Thuật giải

// Input:

Một số nguyên dương $number$ có n chữ số được biểu diễn bởi một xâu ký tự gồm n ký tự.

Một số nguyên k ($0 \leq k \leq n$).

// Output: Số ban đầu sau khi xóa k ký tự đạt giá trị nhỏ nhất.

RemoveDigits($number[1..n], k$)

1. **for** $i = 1$ **to** k **do**
2. $pos = 1$;
3. **while** $number[pos] < number[pos+1]$ **and** $pos < n$ **do**
4. $pos = pos + 1$;
5. **XoaKyTu**($number, pos$);
6. **return** $number$;

Hàm **XoaPhanTu** sẽ thực hiện việc xóa ký tự tại vị trí pos có trong xâu.

d. VD minh họa

INPUT	OUTPUT
1542218 3	1218
10400 1	400
10 2	0

e. Độ phức tạp của thuật toán

Quá trình tìm kiếm chữ số đầu tiên có giá trị lớn hơn hoặc bằng so với chữ số kế tiếp và xóa chữ số đó trong số nguyên ban đầu, tất cả đều có thời gian thực hiện là $O(n)$.

Công việc trên được thực hiện hết k lần cho nên độ phức tạp của hàm thời gian thực hiện chương trình là $O(kn)$ (với k là hằng số).