CHAPTER 03

CHIẾN LƯỢC THAM LAM – CHIA ĐỂ TRỊ

Design by Minh An
Email: anvanminh.haui@gmail.com

Chiến lược tham lam

1. **Bài toán: COIN CHANGING**

Giả sử rằng ta đang có các đồng tiền mệnh giá 100 đồng, 25 đồng, 10 đồng, 5 đồng và 1 đồng. Cũng giả sử rằng số đồng tiền là vô hạn.

Cho trước một số tiền bất kỳ n. Hãy cho biết số lượng đồng tiền ít nhất cần thiết để có tổng mệnh giá bằng n.

vd: n=34 → SOLUTION: S(0, 1, 0, 1, 4)

Design by Minh An

Coin changing

- **Phương pháp:**

Tại mỗi bước lặp, lấy các đồng tiền có mệnh giá cao nhất với số lượng nhiều nhất có thể

INPUT: C[], m, n
OUTPUT: S[]

Design by Minh An

Coin changing

- **Phương pháp:**

- Giả sử C được sắp giảm (nếu chưa, xin sắp C).
- Chuẩn bị mảng S[] và khởi gán các phần tử của S bằng 0.
- Duyệt C từ trái qua phải. Với mỗi C[i]:
  - + S[i] = số đồng tiền có mệnh giá C[i] nhiều nhất có thể lấy mà tổng giá trị không vượt quá n
  - + Tính lại n = số tiền còn lại.
- Nếu duyệt hết C mà n > 0: → No solution
- Ngược lại: → return S[].

Design by Minh An

Coin changing

- **Thuật toán**

```
bool CASHIERS_ALGORITHM(int *C, int m, long n, int * S)
{
    Khởi tạo S[]: S[i]=0 ∀i = 0..m;
    i=0;
    while (n>0 && i<m)
    {
            S[i] = Số_đồng_C[i]_nhiều_nhất_có_thể_lấy;
            n = Số_tiền_còn_lại ;
            i++;
    }
    if(n>0)     return false;
    else        return true;
}
```

Design by Minh An

Coin changing

- **Nhận xét**

Xét trường hợp có ba loại mệnh giá: 1, 7, 10
Cho số tiền n = 15.

Giải thuật tham lam cho kết quả: 6
Nghiệm tối ưu toàn cục: 3

Xét trường hợp có ba loại mệnh giá: 3, 7, 10
Cho số tiền n = 12.

Giải thuật tham lam: không tìm thấy nghiệm
Nghiệm tối ưu: 4

Design by Minh An

Chiến lược tham lam

2. BÀI TOÁN TỐI ƯU

$$\min_{x \in D} f(x)$$

f: hàm mục tiêu - objective function
x: biến - variable
D: miền xác định - domain

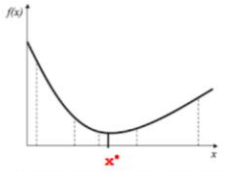
$x^* = \arg\min f(x)$
được gọi là nghiệm tối ưu - optimal solution

Chiến lược tham lam

2. BÀI TOÁN TỐI ƯU

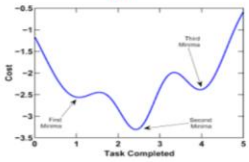
Nghiệm tối ưu toàn cục
Global optimum

$x^*: f(x) >= f(x^*) \ \forall x \in D$



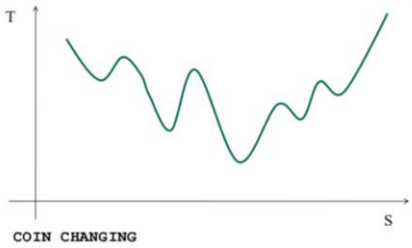
Nghiệm tối ưu địa phương
Local optimum

$x^*: f(x) >= f(x^*) \ \forall x \in U$
với U là lân cận của x



Chiến lược tham lam

2. BÀI TOÁN TỐI ƯU



COIN CHANGING

3. Giải thuật tham lam – Greedy Algorithm

- Là một giải thuật tìm lời giải tối ưu cho bài toán theo từng bước.
- Tại mỗi bước, ta lựa chọn một khả năng tốt nhất tại điểm đó (tối ưu cục bộ) mà không quan tâm tới tương lai.
- Ta hy vọng rằng tập hợp các lời giải tối ưu cục bộ chính là lời giải tối ưu cần tìm.

- Trong một vài trường hợp, các giải thuật tham lam tìm được nghiệm tối ưu toàn cục bằng cách lặp đi lặp lại việc lựa chọn các khả năng tối ưu cục bộ.

- Trong nhiều trường hợp phương pháp này có thể không cho lời giải tối ưu toàn cục hoặc thậm chí không đưa ra được lời giải.

3. Giải thuật tham lam – Greedy Algorithm

Giải thuật tham lam là một chiến thuật có thể áp dụng tốt cho các bài toán tối ưu có hai đặc điểm sau:

1. Greedy-choice property: một nghiệm tối ưu toàn cục có thể được xây dựng bằng cách lựa chọn các nghiệm tối ưu địa phương.

2. Optimal substructure: một nghiệm tối ưu của bài toán lại chứa một nghiệm tối ưu của các bài toán con của nó.

3. Giải thuật tham lam – Greedy Algorithm

Ưu điểm

- **Simplicity:** các giải thuật tham lam thường dễ dàng được mô tả và cài đặt hơn các giải thuật khác.
- **Efficiency:** các giải thuật tham lam thường có hiệu quả hơn các giải thuật khác.

Nhược điểm

- **Hard to design:** khi đã xác định được quy luật lựa chọn tham lam cho một bài toán, việc thiết kế giải thuật là dễ. Tuy nhiên, việc tìm ra quy luật tham lam là khó khăn.
- **Hard to verify:** khó để chứng minh một giải thuật tham lam là đúng.

3. Giải thuật tham lam – Greedy Algorithm

Bài tập 2: Một bình chứa chứa đầy nước với một lượng nước hữu hạn n. Cho m chiếc chai rỗng (dung tích các chai khác nhau) để chiết nước từ bình chứa vào đầy các chai. Hãy cho biết số lượng chai tối đa có thể được đổ đầy nước.

Input: dòng thứ nhất chứa hai số nguyên n và m; dòng thứ 2 chứa m số nguyên là dung tích của các chai.

Output: một số nguyên là số chai tối đa được đổ đầy nước.

INPUT	OUTPUT
10 5 8 5 4 3 2	3

3. Giải thuật tham lam – Greedy Algorithm

3. Giải thuật tham lam – Greedy Algorithm

Bài tập 3: INTERVAL SCHEDULING

- Có n công việc, công việc j bắt đầu tại thời điểm s_j và kết thúc tại thời điểm f_j.
- Hai công việc được gọi là tương hợp nếu thời gian thực hiện chúng không giao nhau.
- Tìm một tập cực đại các công việc mà chúng tương hợp với nhau.

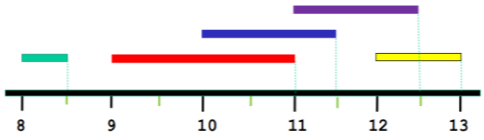
INPUT	OUTPUT
5 8 9 10 11 12 8.5 11 11.5 12.5 13	1 1 0 1 0

3. Giải thuật tham lam – Greedy Algorithm

Bài tập 3: INTERVAL SCHEDULING

INPUT	OUTPUT
5 8 9 10 11 12 8.5 11 11.5 12.5 13	1 1 0 1 0

Hai việc i và j không tương hợp nếu:
f(i) >= s(j) or f(j) >= s(i).



3. Giải thuật tham lam – Greedy Algorithm

Bài tập 3: INTERVAL SCHEDULING

S[], F[]: tập start time và finish time
n: số công việc
schedule[]: mảng kết quả
Duyệt danh sách công việc
    Chọn công việc thứ i sao cho F[i] → Min
    Thêm i vào schedule[].
    Xóa mọi công việc không tương hợp với i
return schedule

3. Giải thuật tham lam – Greedy Algorithm

Bài tập 3: INTERVAL SCHEDULING

```
Sắp xếp mảng đồng hành S, F theo chiều tăng dần của F
Khởi gán mảng schedule với các phần tử 0.
last_finish = 0
for i = 1 to n:
    if s(i) >= last_finish:
        Add i to schedule
        last_finish = f(i)
return schedule
```

Sử dụng cấu trúc dữ liệu phù hợp: queue

3. Giải thuật tham lam – Greedy Algorithm

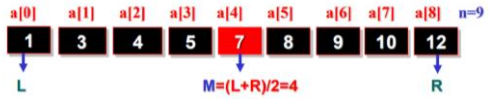
Bài tập 3: INTERVAL SCHEDULING

- [Earliest start time]
- [Earliest finish time]
- [Shortest interval]
- [Fewest conflicts]

Chiến lược chia để trị

1. TÌM KIẾM NHỊ PHÂN

Cho một dãy a gồm n phần tử đã được sắp tăng, cho một phần tử C. Cho biết C có xuất hiện trong a hay không?



C=10

```
a[M]=c :  Yes
a[M] < c:  L=M+1;
a[M] > c:  R=M-1;
L>R    :  No
```


Chiến lược chia để trị

1. TÌM KIẾM NHỊ PHÂN

```
a[M]=c :  Yes
a[M] < c:  L=M+1;
a[M] > c:  R=M-1;
L>R    :  No
```

```
int TKNP_DQ(int a[100], int c, int L, int R)
{
   int M=(L+R)/2;

   if(suy_biến)
          return <CÔNG_THỨC_SUY_BIẾN>;
   else
          return <CÔNG_THỨC_TỔNG_QUÁT>;
}
```


Chiến lược chia để trị

1. TÌM KIẾM NHỊ PHÂN

```
a[M]=c :  Yes
a[M] < c:  L=M+1;
a[M] > c:  R=M-1;
L>R    :  No
```

```
int TKNP_Lap(int a[100], int n, int c)
{
      int L=0, R=n-1, M;
      do
      {
         M = (L+R)/2;
         if (a[M]>c) R=M-1;
         if (a[M]<c) L=M+1;
      }
      while(a[M]!=c && L<R);
      if(a[M]==c)       return M;
      else              return -1;
}
```


Chiến lược chia để trị

1. TÌM KIẾM NHỊ PHÂN

Cho một dãy a gồm n phần tử đã được sắp tăng, cho một phần tử C. Cho biết C có xuất hiện trong a hay không?

Nếu C không xuất hiện trong a, hãy tìm vị trí để chèn C vào a mà không phá vỡ tính được sắp của a.

```
for i = 1 to n do
    if A [i] ≥ q then
        return index i
    return n + 1
```

```
Search(a, L, R, C)
   if L = R then
      return L (index)
   M= (L + R)/2
   if C < a [M] then
      return Search(a, L, M, C)
   else
      return Search(a, M, R, C)
```


Chiến lược chia để trị

2. CHIẾN THUẬT CHIA ĐỂ TRỊ

- A top-down technique for designing algorithms
- **Dividing** the problem into smaller subproblems
- Hoping that the solutions of the sub-problems are easier to find
- **Composing** the partial solutions into the solution of the original problem

Chiến lược chia để trị

2. CHIẾN THUẬT CHIA ĐỂ TRỊ

Divide-and-conquer paradigm consists of following major phases:

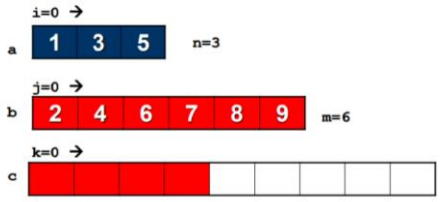
- **Breaking** the problem into several sub-problems that are similar to the original problem but smaller in size.
- **Solve** the sub-problem recursively (successively and independently), and then
- **Combine** these solutions to subproblems to create a solution to the original problem.

Chiến lược chia để trị

3. SẮP XẾP TRỘN

Bài toán trộn:

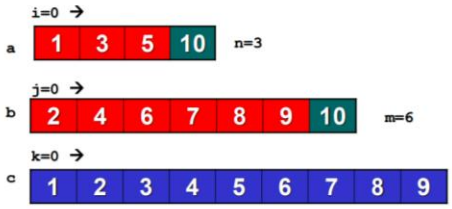
- Cho hai dãy các phần tử cùng kiểu đã được sắp tăng, hãy trộn hai dãy để thu được một dãy cũng được sắp tăng.



Chiến lược chia để trị

3. SẮP XẾP TRỘN

Bài toán trộn:



Chiến lược chia để trị

3. SẮP XẾP TRỘN

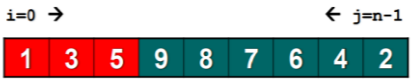
Bài toán trộn:

```
int c[100];
void Tron2(int a[50],int n, int b[50], int m)
{
   int Max=a[n-1];
   if(Max<b[m-1]) Max=b[m-1];
   a[n]=b[m]=Max+1;
   //------------------------
   int i=0, j=0;
   for(int k=0; k<n+m; k++)
   if(a[i]<b[j])
      {c[k]=a[i]; i++;}
   else
      {c[k]=b[j]; j++;}
}
```


Chiến lược chia để trị

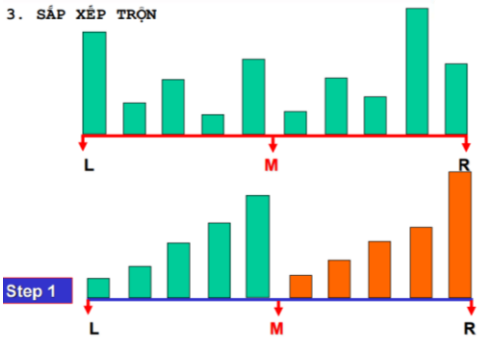
3. SẮP XẾP TRỘN

Bài toán trộn:



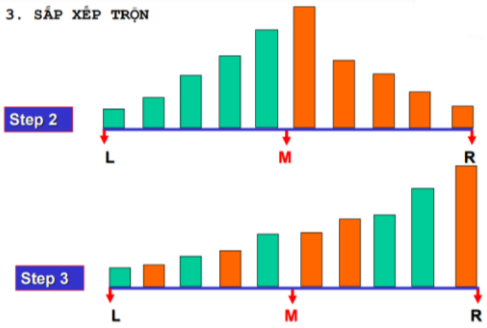
Chiến lược chia để trị

3. SẮP XẾP TRỘN



Chiến lược chia để trị

3. SẮP XẾP TRỘN



Chiến lược chia để trị

3. SẮP XẾP TRỘN

```
void MergeSort(float a[], int l, int r)
{
   if(r>l)
   {
      int m=(l+r)/2;
      MergeSort(a,l,m); MergeSort(a,m+1, r);
      - Sao chép nửa đầu của a sang b
      - Sao chép nửa còn lại sang b theo thứ tự ngược lại
      - Trộn hai nửa.
   }
}
```


Chiến lược chia để trị

3. SẮP XẾP TRỘN

```
void MergeSort(float a[], int l, int r)
{
   if(r>l)
   {
      int m=(l+r)/2;
      MergeSort(a,l,m); MergeSort(a,m+1, r);
      //Sao chep nua dau cua a sang b
      for(int i=m; i>=l; i--) b[i]=a[i];
      //Sao chep nua con lai cua a sang b theo thu tu nguoc lai
      for(int j=m+1; j<=r;j++) b[r+m+1-j]=a[j];
      //i chay tu dau mang b, j chay tu cuoi mang b va tron
      i=l; j=r;
      for(int k=l; k<=r; k++)
      if(b[i]<b[j])      {a[k]=b[i];i++;}
      else               {a[k]=b[j];j--;}
   }
}
```


Chiến lược chia để trị

4. TÌM MAX

Cho một dãy số thực gồm n phần tử, hãy tìm phần tử lớn nhất của dãy.