HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

DATA STRUCTURES AND ALGORITHMS

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

DATA STRUCTURES AND ALGORITHMS

Week 5: Greedy algorithms, Divide and Conquer algorithms

ONE LOVE. ONE FUTURE.

CONTENT

A. Greedy algorithms

B. Divide and conquer algorithms

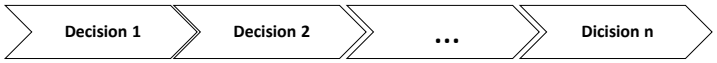
CONTENT

A. Greedy algorithms

B. Divide and conquer algorithms

INTRODUCTION

- Greedy algorithm is a simple and highly abstract approach to solving optimization problems, especially combinatorial optimization.
- The process of finding a solution using the greedy algorithm is divided into several stages



GENERAL ALGORITHM

- Notation:
  - S: Solution to find
  - C: A set of candidates
  - $select(C)$: A function to select the best candidate
  - $solution(S)$: A function returns TRUE if S is a validate solution, otherwise returns FALSE
  - $feasible(S)$: A function returns TRUE if S does not violate contrains, otherwise return FALSE

```
Greedy() {
  S = ∅;
  while C ≠ ∅ and not solution(S){
    x = select(C);
    C = C \ {x};
    if feasible(S ∪ {x}) {
      S = S ∪ {x};
    }
  }
  return S;
}
```


GENERAL ALGORITHM

Init, Empty solution S

As long as the candidate set is non-empty and S is not a valid solution

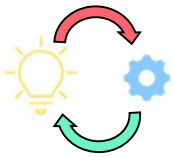
Select x as the most potential candidate and eliminate x from the candidate set C

If x is added to the solution being constructed if the constraint is not violated

```
Greedy() {
  S = ∅;
  while C ≠ ∅ and not solution(S){
    x = select(C);
    C = C \ {x};
    if feasible(S ∪ {x}) {
      S = S ∪ {x};
    }
  }
  return S;
}
```

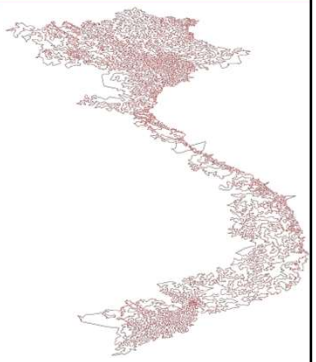
CHARACTERISTICS

- A simple algorithm➔ Easy to propose and implement
- In some input data sets and problems, the greedy algorithm gives the optimal solution. However, in most cases, especially real problems with high complexity, this algorithm does not guarantee the optimal solution.
- The algorithm is quite effective in real problems with many constraints and high complexity.



EXAMPLE: DELIVERY PERSON

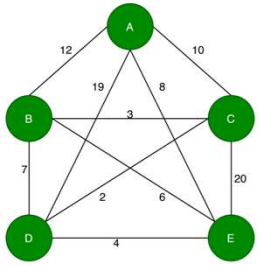
- The Traveling Salesman Problem (TSP) is a classic optimization problem in the field of combinatorial optimization.
- The problem belongs to the NP-Hard class (There is no algorithm that runs in polynomial time that can be solved efficiently)
- Statement: The salesperson needs to find a route through a set of cities such that the trip has the lowest total travel cost and each city is visited only once before returning to the starting city.



Nguồn: https://www.math.uwaterloo.ca

EXAMPLE: DELIVERY PERSON

- **Design:**
  - $\boldsymbol{S}$: The solution to find
  - $\boldsymbol{C}$: A set of candidate
  - $\boldsymbol{select(C)}$: A function to select the best candidate
  - $\boldsymbol{solution(S)}$: A function returns TRUE if S is a validate solution, otherwise returns FALSE
  - $\boldsymbol{feasible(S)}$: A function returns TRUE if S does not violate contrains, otherwise return FALSE



EXAMPLE: DELIVERY PERSON

- **Problem:** Design a greedy algorithm for TSP problem

Subsets of edges of a graph

All edges of the graph

The shortest edge

Edges in S form a cycle, each node appears one time

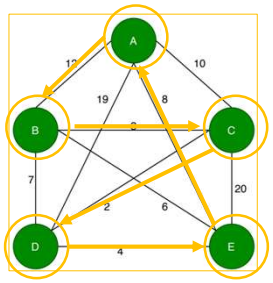
Edges in S can form a closed cycle, each node appear one time

- Design:
  - $\boldsymbol{S}$: Solution to find
  - $\boldsymbol{C}$: A set of candidates
  - $\boldsymbol{select(C)}$: A function to select the best candidate
  - $\boldsymbol{solution(S)}$: A function returns TRUE if S is a validate solution, otherwise returns FALSE
  - $\boldsymbol{feasible(S)}$: A function returns TRUE if S does not violate contrains, otherwise return FALSE

EXAMPLE: DELIVERY PERSON

- **Problem:** Design a greedy algorithm for TSP problem
  - **Algorithm**

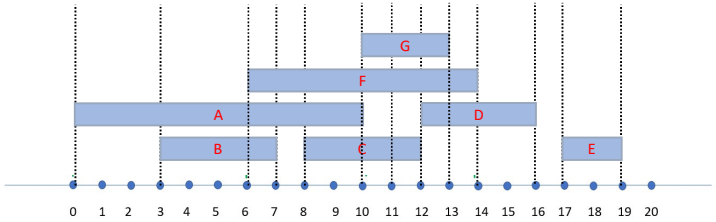
```
Greedy() {
  S = ∅;
  while C ≠ ∅ and not
    solution(S){
    x = select(C);
    C = C \ {x};
    if feasible(S ∪ {x}) {
      S = S ∪ {x};
    }
  }
  return S;
}
```



Start from B: B→C→D→E→A→B

EXAMPLE: MAXIMUM NON-INTERSECTING SUBSET PROBLEM

- **Problem:** Given a set of straight segments $X = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ where $a_i < bi, \forall i \in \{1, \ldots, n-1\}$ are coordinates of the two ends of i[th] segment in a line, for all $i \in \{1, \ldots, n\}$. Find the subset of non-intersecting pairwise segments that has the largest number of elements.



Optimal solution: $S = \{B, C, D, E\}$

EXAMPLE: MAXIMUM NON-INTERSECTING SUBSET PROBLEM

- **Design: Greedy algorithm**

- Design:
  - $\boldsymbol{S}$: Solution to find
  - $\boldsymbol{C}$: A set of candidates
  - $\boldsymbol{select(C)}$: A function to select the best candidate
  - $\boldsymbol{solution(S)}$: A function returns TRUE if S is a validate solution, otherwise returns FALSE
  - $\boldsymbol{feasible(S)}$: A function returns TRUE if S does not violate contrains, otherwise return FALSE

A subsets of straight segments

All straight segments

The shortest segment

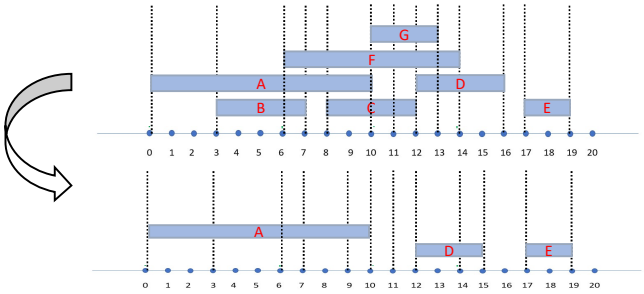
Non-intersecting segments

Non-intersecting segments in S

EXAMPLE: MAXIMUM NON-INTERSECTING SUBSET PROBLEM

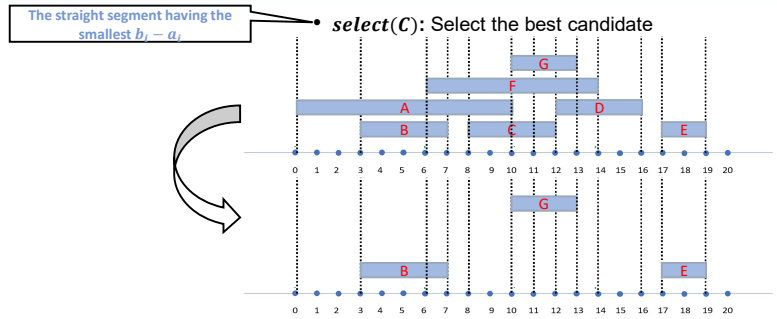
The straight segment having the smallest a_i

- $\boldsymbol{select(C)}$: A function to select the best candidate



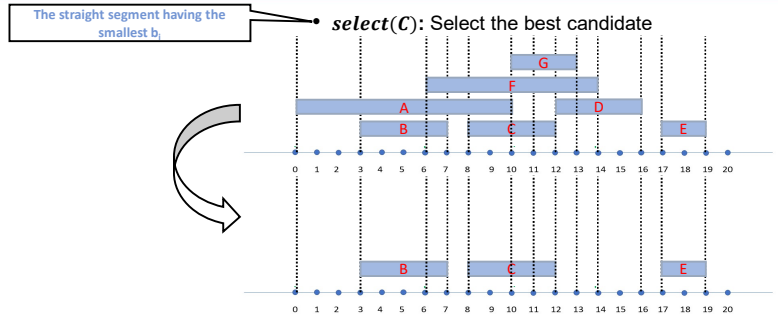
This solution has 3 segments ➔ Not optimal. (A better one: 4 segments ($S = \{B, C, D, E\}$))

EXAMPLE: MAXIMUM NON-INTERSECTING SUBSET PROBLEM



This solution has 3 segments ➔ Not optimal. (A better one: 4 segments ($S = \{B, C, D, E\}$))

EXAMPLE: MAXIMUM NON-INTERSECTING SUBSET PROBLEM



Can find the optimal solution (with 4 segments)

Homework: Prove that this algorithm returns the optimal solution.

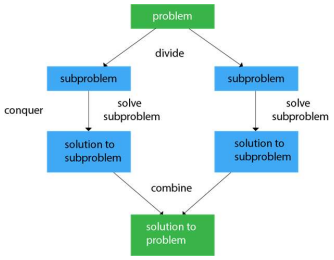
EXERCISES

- **Problem**: There are n jobs 1, 2, ..., n. Task i has a completion deadline of $d[i]$ and a profit when put into operation of $p[i]$ (i=1,...,n). Know that only at most 1 job can be done at a time and the time it takes to complete each job is 1 unit. Find a way to choose tasks to put into practice so that the total profit is maximized and each task must be completed before or on time..
- Submit code to the online system

CONTENT

INTRODUCTION

- Divide and Conquer algorithm) is a powerful technique in Computer Science and Mathematics.
- The idea: is to simplify the problem by dividing the complex problem into simpler sub-problems (DIA), solving the sub-problems separately (TRUE) and finally synthesizing the sub-problem results. to obtain the solution to the original problem.



Source: https://www.javatpoint.com/divide-and-conquer-introduction

ALGORITHM DIAGRAM

- Using the Algorithm: Use Backtracking Recursion to demonstrate the general scheme of a divide-and-conquer algorithm
- Easy to understand and clearly express the idea of the algorithm

```
DC(P_n) {
    if n ≤ n_0 {
        solve_problem(p_n);
    } else {
        SP = divide(P_n);
        foreach p ∈ SP {
            s(p) = DC(p)
        }
        S = aggregate(s(p_1), ... s(p_k))
    }
    return S;
}
```


ALGORITHM DIAGRAM

```
DC(P_n) {
    if n ≤ n_0 {
        solve_problem(p_n);
    } else {
        SP = divide(P_n);
        foreach p ∈ SP {
            s(p) = DC(p)
        }
        S = aggregate(s(p_1), ... s(p_k))
    }
    return S;
}
```

n_0 is the small size of the problem, which we can easily show its solution.
This is the basic step in Recursion technique.

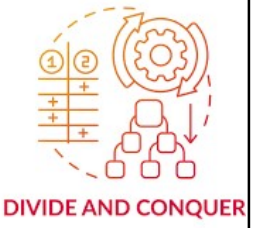
Divide the parent problem into multiple child problems

Solve each subproblem

Combine the results of the sub-problems to build a solution to the parent problem (the original problem).

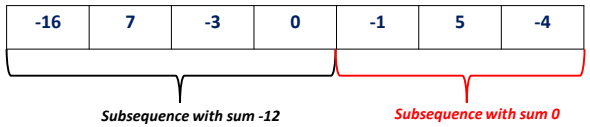
CHARACTERISTICS

- The idea of the "Dividing" algorithm is very natural and similar to the human problem-solving approach
- Subproblems are independent, meaning that solving one subproblem does not affect the solution of other congruent subproblems.
- Dividing the problem ensures that the solution is not lost
- Some outstanding applications
  - Parallel computing
  - Search



EXAMPLE: SUBSEQUENCE WITH THE LARGEST SUM

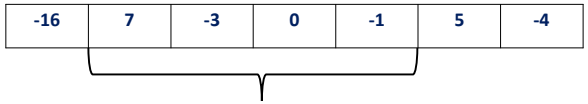
- **Problem**: Given a sequence of n integers $(a_1, a_2, \dots, a_n)$, find a subsequence consisting of consecutive elements of the sequence such that the sum of the selected elements is maximized.
- **Example**: Given a sequence of 7 intergers:



  - *Optimal solution: (subsequence with the largest sum of 8): 7, -3, 0, -1, 5*
- The exhaustive algorithm has complexity $O(n^2)$, can we do better with the divide and conquer algorithm?

EXAMPLE: SUBSEQUENCE WITH THE LARGEST SUM

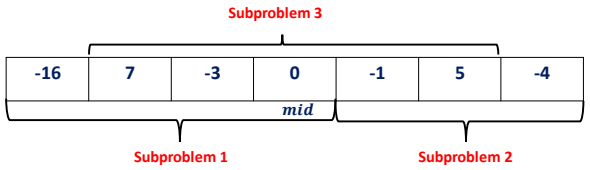
- **Subproblem**: A subproblem of the problem of finding the maximum subsequence of the sequence $(a_1, \dots, a_n)$ is a problem to find the maximum of the subsequence $a_i, a_{i+1}, \dots, a_{i+j}, i \geq 1,\ i + j \leq n$.



Subproblem: Find the maximum subsequence of (7, -3, 0, -1)

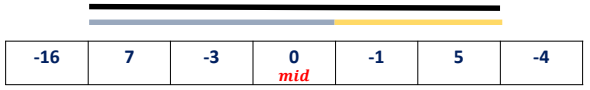
EXAMPLE: SUBSEQUENCE WITH THE LARGEST SUM

- **Divide:** Divide the sequence of n elements in half at the data point $mid = round(\frac{n+1}{2})$
  - Subproblem 1: Same as the parent problem, find the maximum sub-sequence of sequence 1, ...mid
  - Subproblem 2: Like the parent problem, find the maximum subsequence of the sequence mid+1, ...n
  - Subproblem 3: Find the maximum subsequence containing element mid and the first and last elements are unknown



EXAMPLE: SUBSEQUENCE WITH THE LARGEST SUM

- Solving subproblems: Subproblems 1 and 2 can be solved by recursion. Subproblem 3 can be solved directly due to the constraint that it must contain element mid. The solution to problem 3 is the union of the maximum sub-segment ending at mid and the maximum sub-segment starting at mid + 1. To solve these 2 small problems, we just need to browse from mid back to 1 and from mid + 1 approaches n, with complexity $O(n)$)



EXAMPLE: SUBSEQUENCE WITH THE LARGEST SUM

- **Determine subproblems**: A subproblem of the problem of finding the maximum subsequence of the sequence $(a_1, \dots, a_n)$ is a problem to find the maximum of the subsequence $a_i, a_{i+1}, \dots, a_{i+j}, i \geq 1,\ i + j \leq n$.
- **Divide:** Divide the sequence of n elements in half at the point $mid = round(\frac{n+1}{2})$
  - Sub-problem 1: Same as the parent problem, find the maximum sub-sequence of sequence 1, ...mid
  - Sub-problem 2: Like the parent problem, find the maximum subsequence of the sequence mid+1, ...n
  - Subproblem 3: Find the maximum subsequence containing element mid and the first and last elements are unknown
- **Solving subproblems:** Using recursion with time complexity of $O(n)$
- **Combine:** The solution is the best result among the three subproblems
- **Total time complexity**: $O(\log(n))$

EXAMPLE: CALCULATE a^n

- **Problem**: Calculate a^n with positive integers a and n
- **Analyze**: The direct method, which performs n multiplication, has computational complexity $O(n)$. Can we do better with the Divide and Conquer Algorithm?
- **Design a divide and conquer algorithm:**
  - **Subproblem:** Calculate $a^k, k < n$
  - **Divide:** Divide into 2 subproblems but only need to solve 1
    - If n is even, then $a^n = \left(a^{\frac{n}{2}}\right) \times \left(a^{\frac{n}{2}}\right)$
    - If n is odd, then $a^n = a \times a^{n-1}$
  - **Method:** Recursion
  - **Combine:** A multiplication
  - **Time complexity**: $O(\log(n))$

EXAMPLE: CALCULATE a^n

- Code

```
int Pow(int x, int n) {
    if (n == 0) return 1;
    if (n % 2 != 0) return x * pow(x, n - 1);
    int res = Pow(x, n/2);
    return res * res;
}
```


EXERCISES

- **Problem**: Sort an array of n integers using a divide and conquer algorithm. Hint: Merge sort by Dividing the array into two subarrays.
- Homework: Submit code to the online system

THANK YOU !