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

APPLIED ALGORITHMS

GREEDY ALGORITHMS

ONE LOVE. ONE FUTURE.

CONTENT

- Basis of Greedy algorithms
- Coin exchange
- Knapsack
- Disjoint segments

Basis of Greedy algorithms

- Notations
  - $\boldsymbol{S}$: solution under construction
  - $\boldsymbol{C}$: set of candidates
  - $\boldsymbol{select}(\boldsymbol{C})$: select a potential candidate for inserting to the solution
  - $\boldsymbol{solution}(\boldsymbol{S})$: return TRUE if S is a complete accepted solution, and return FALSE, otherwise
  - $\boldsymbol{feasible}(\boldsymbol{S})$: return TRUE if S does not violate given constraints, and return FALSE, otherwise

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

Basis of Greedy algorithms

- Travelling Salesman Problem (TSP)
  - Find the shortest closed tour starting from point 1, visiting 2, . . ., n (each point is visited exactly once) and coming back to 1.
  - Nearest neighbor selection
    - At each step, find the nearest point to the current point (the last point of the solution under construction)

```
GreedyTSP() {
  S = [1]; cur = 1;
  C = {2, 3, . . ., n};
  while C ≠ ∅  do {
    x = selectNearest(C, cur);
    C = C \ {x};
    S = S::x; // append S with x
    cur = x;
  }
  return S;
}
```

Basis of Greedy algorithms

- In general, greedy algorithms cannot ensure to find optimal solutions in all cases
- In some cases, there exist greedy algorithms and can find optimal solutions
  - Coin exchange with denominations 1, 2, 5, 10
  - Disjoint segments
  - Kruskal, Prim algorithms for finding minimum spanning tree of a given undirected weighted graph
  - Huffman code

Coin exchange problem

- Given coins of denominations 1, 2, 5, 10. Given a positive integer *Y*, how to get an amount of money *Y* from the coins such that the number of coins used is minimal.

```
Greedy(Y) {
  D = [1, 2, 5, 10];
  res = [];
  while Y > 0 do {
    x = select max item from D such that x ≤ Y;
    Y = Y - x;
    res = res::x; // append res with x
  }
  return res;
}
```

Knapsack Problem

- There are n items S ={1, 2, 3, . . ., n}. Item j has weight W_j and value C_j (j = 1, 2, . . . n). Given a bin with capacity B. Select a subset S of items from the given items such that the sum of weights of items of S is not greater than B and the sum of values of the items is maximal.

Knapsack Problem: Greedy 1

- Sort the items in a non-increasing of values
- Explore the items from left to right in the sorted list, select the item if it can be inserted into the bin without violating the capacity constraint

```
Greedy1(B, W, C) {
  L = sort items in a non-increasing of values;
  res = {};
  for j in L do {
    if W_j <= B then {
      res = res ∪ {j}; B = B – W_j;
    }
  }
  return res;
}
```

Knapsack Problem: Greedy 1

- Counter example
- Number of items n = 3
- Capacity of the bin B = 19

Items	1	2	3
C_i	20	16	8
W_i	14	6	10

- Solution returned by Greedy1: S_1 = {1} with total values 20
- Optimal solution S^* = {2, 3} with total values 24

Knapsack Problem: Greedy 2

- Sort the items in a non-decreasing of weights
- Explore the items from left to right in the sorted list, select the item if it can be inserted into the bin without violating the capacity constraint

```
Greedy2(B, W, C) {
  L = sort items in a non-decreasing of weights;
  res = {};
  for j in L do {
    if Wj <= B then {
      res = res ∪ {j}; B = B – Wj;
    }
  }
  return res;
}
```

Knapsack Problem: Greedy 2

- Counter example
- Number of items n = 3
- Capacity of the bin B = 11

Items	1	2	3
C_j	10	16	28
W_j	5	6	10

- Solution returned by the Greedy2: S_2 = {1, 2} with total values 26
- Optimal solution S^* = {3} with total values 28

Knapsack Problem: Greedy 3

- Sort the items in a non-increasing of weights:

$$\frac{C_1}{W_1} \geq \frac{C_2}{W_2} \geq \ldots \frac{C_n}{W_n}$$

- Explore the items from left to right in the sorted list, select the item if it can be inserted into the bin without violating the capacity constraint

```
Greedy3(B, W, C) {
  L = sort items in a non-increasing of C_i/W_i;
  res = {};
  for j in L do {
    if W_j <= B then {
      res = res ∪ {j}; B = B – W_j;
    }
  }
  return res;
}
```

Knapsack Problem: Greedy 3

- Counter example
- Number of items n = 2
- Capacity of the bin $B \geq 2$

Item	1	2
C_i	10	$10B - 1$
W_i	1	B

- Clearly: $\frac{C_1}{W_1} = \frac{10}{1} \geq \frac{10B-1}{B} = \frac{C_2}{W_2}$
- Solution returned by the Greedy3: S_3 = {1} with value 10
- Optimal solution S^* = {2} with value $10B$ - 1

Knapsack Problem: Greedy 4

- Let S_j be the solution obtained by Greedy*j*, (j = 1, 2, 3). Let S_4 be the best solution among S_1, S_2, S_3:
- Then we have $\sum_{i \in S_4} Ci \geq \frac{1}{2} OPT$ (in which *OPT* is the total values of the optimal solution)

Knapsack Problem: Greedy 4

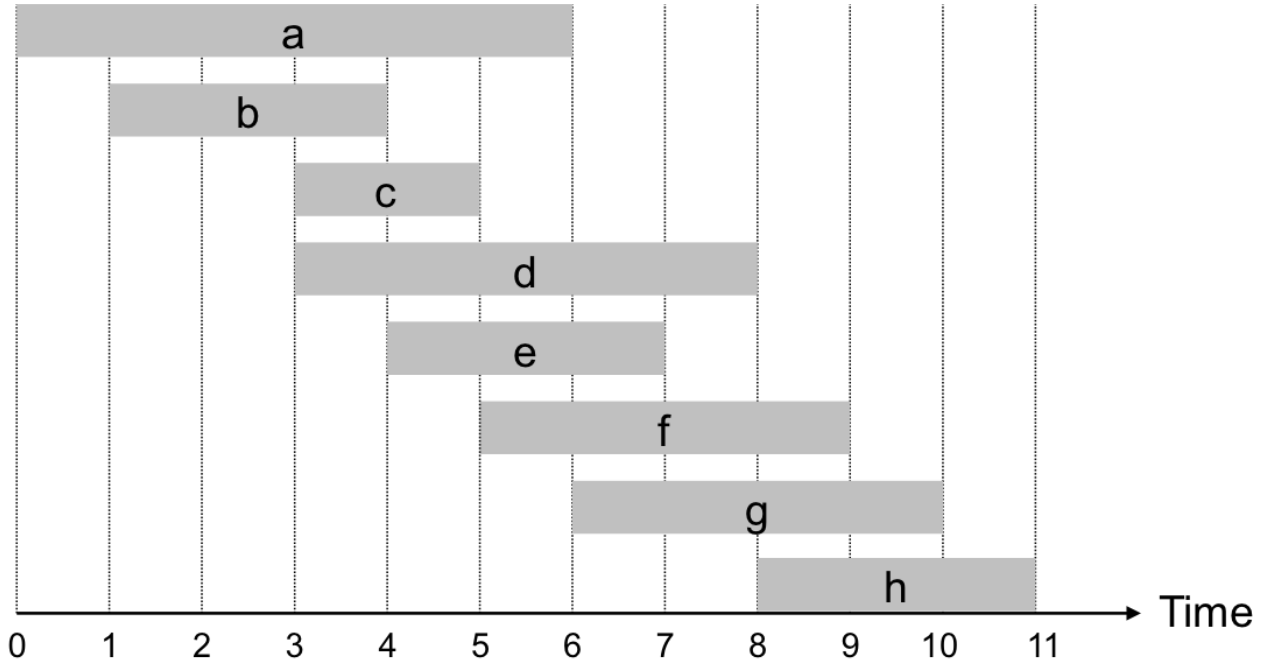
- Counter example
- Number of items n = 4
- Capacity of the bin B = 11

Items	1	2	3	4
C_i	9	10	18	27
W_i	4	5	6	10
C_i/W_i	2.25	2	3	2.7
Greedyi	27	19	27	7

- Solution returned by the Greedy4 has value 27
- Optimal solution S^* = {2, 3} with value 28

Disjoint segments

- There are *n* jobs 1, 2, . . ., n. Job *j* starts at time-point S_j and finishes at time-point F_j
- Two jobs *i* and *j* are compatible if $[S_i, F_i]$ and $[S_j, F_j]$ are not overlap.
- **Goal:** Find a subset of the given jobs such that all pair of two jobs of the are pair-wise compatible.



Greedy algorithm ideas

- Greedy 1: Sort the jobs in non-decreasing order of start time S_i.
- Greedy 2: Sort the jobs in non-decreasing order of finish time F_i.
- Greedy 3: Sort the jobs in non-decreasing order of duration $F_i - S_i$.
- Greedy 4: Sort the jobs in non-decreasing order of conflict (conflict of a job *j* is the number of jobs that are not compatible with *j*)

Greedy algorithm ideas

- Counter examples

Greedy 1

Greedy 3

Greedy 4

Disjoint segments: Greedy 2 is correct

- Algorithm Greedy 2 ensures to find an optimal solution
- Độ phức tạp O(nlog n)

```
Greedy2([S₁, F₁], . . ., [Sn, Fn]) {
  L = sort segments in a non-decreasing of Fj;
  res = {};
  for j in L do {
    if [Sj, Fj] is compatible with segments in res then {
      res = res ∪ {j};
    }
  }
  return res;
}
```

THANK YOU !

hust.edu.vn fb.com/dhbkhn