中国科学院大学
University of Chinese Academy of Sciences

CS101

Algorithmic Thinking

Greedy, Stable Match

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

Outline

- What is algorithmic thinking
- Divide-and-conquer paradigm
- Other interesting paradigms
  - Dynamic Programming
  - Randomization
  - Greedy***
- P vs. NP

These slides acknowledge sources for additional data not cited in the textbook

3. Paradigms other than divide-and-conquer

- Dynamic programming
  - What if subproblems have overlapping elements
- Randomization
  - Avoid being trapped in a bad situation
- Greedy***
  - Try the obviously best from the possible next steps
  - Look at the stable match problem in detail

3.3 The greedy strategy***

- At a step in an algorithm, make the best choice from all possible choices
- Suitable optimization problems have two features
  - After making a choice, the subproblem left is similar to the original problem
  - Solution is optimal, if choice + sub-solution is optimal

GreedyAlgorithm(Problem)

choice = GreedyChoice

subSolution = GreedyAlgorithm(subProblem)

Return combine(choice, subSolution)

KnapsackProblem

- Input: N items in a room where item i has value v[i] and weight w[i]; the burglar has a knapsack which can hold at most weight W
- Output: put items in the knapsack to maximize total value, with the constraint total weight < W

Initially, Knapsack is empty

Room = {1, 2, …, N}

Knapsack = ∅

totalWeight = 0; totalValue = 0;

GreedyBurglar(KnapsackProblem)

choice = Choose the most valuable item k left, subject to weight constraint W

totalValue += v[k]; totalWeight += w[k];

subSolution = GreedyBurglar(subKnapsackProblem)

Return combine(choice, subSolution)

subKnapsackProblem

The subproblem is the same as the original problem except that item k is already in knapsack. Thus, after first recursion,

Room = {1, 2, …, N} – {k}; Knapsack = {k};

totalWeight = v[k]; totalValue = v[k]

Can lead to optimal solution

- Knapsack Problem
- Example: N=5, W=7
- Initially, all items in Room

Item k	v[k]	w[k]
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10

Optimal solution: item 3 in knapsack

Knapsack = {3},
totalValue = 5
totalWeight = 6

KnapsackProblem

- Input: N items in a room where item i has value v[i] and weight w[i]; the burglar has a knapsack which can hold at most weight W
- Output: put items in the knapsack to maximize total value, with the constraint total weight < W

Initially, Knapsack is empty
Room = {1, 2, …, N}
Knapsack = ∅
totalWeight = 0; totalValue = 0;

GreedyBurglar(KnapsackProblem)
choice = Choose the most valuable item k left, subject to weight constraint W
totalValue += v[k]; totalWeight += w[k];

subSolution = GreedyBurglar(subKnapsackProblem)

Return combine(choice, subSolution)

subKnapsackProblem
The subproblem is the same as the original problem except that item k is already in knapsack. Thus, after first recursion,
Room = {1, 2, …, N} – {k}; Knapsack = {k};
totalWeight = v[k]; totalValue = v[k]

Does not always lead to optimal solution

- Knapsack Problem
- Example: N=5, W=15

Item k	v[k]	w[k]
1	3	3
2	3	4
3	3	7
4	3	9
5	5	10

GreedyBurglar generates
Knapsack = {5, 1},
totalValue = 8
totalWeight = 13
Optimal solution
Knapsack = {3, 4, 7},
totalValue = 9
totalWeight = 14

KnapsackProblem

- Input: N items in a room where item i has value v[i] and weight w[i]; the burglar has a knapsack which can hold at most weight W
- Output: put items in the knapsack to maximize total value, with the constraint total weight < W

Initially, Knapsack is empty
Room = {1, 2, …, N}
Knapsack = ∅
totalWeight = 0; totalValue = 0;

GreedyBurglar(KnapsackProblem)
choice = Choose the most valuable item k left, subject to weight constraint W
totalValue += v[k]; totalWeight += w[k];

subSolution = GreedyBurglar(subKnapsackProblem)

Return combine(choice, subSolution)

subKnapsackProblem
The subproblem is the same as the original problem except that item k is already in knapsack. Thus, after first recursion,
Room = {1, 2, …, N} – {k}; Knapsack = {k};
totalWeight = v[k]; totalValue = v[k]

Does not always lead to optimal solution

- The above are called 0-1 Knapsack Problem

- How about fractional knapsack problem?
  - A fraction of an item can be put in the knapsack
  - There is an greedy algorithm leading to optimal solution

0-1 KnapsackProblem

- Input: N items in a room where item i has value v[i] and weight w[i]; the burglar has a knapsack which can hold at most weight W
- Output: put items in the knapsack to maximize total value, with the constraint total weight < W

Initially, Knapsack is empty

Room = {1, 2, …, N}
Knapsack = ∅
totalWeight = 0; totalValue = 0;

GreedyBurglar(KnapsackProblem)

choice = Choose the most valuable item k left, subject to weight constraint W
totalValue += v[k]; totalWeight += w[k];

subSolution = GreedyBurglar(subKnapsackProblem)

Return combine(choice, subSolution)

subKnapsackProblem

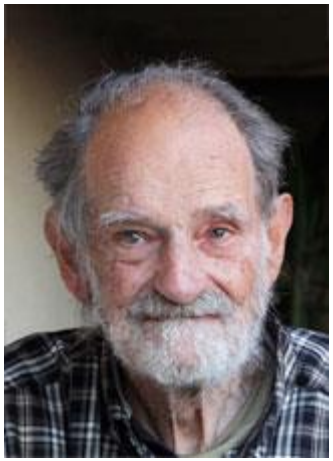
The subproblem is the same as the original problem except that item k is already in knapsack. Thus, after first recursion,

Room = {1, 2, …, N} – {k}; Knapsack = {k};
totalWeight = v[k]; totalValue = v[k]

Gale-Shapley algorithm for stable matching

2012 Nobel Prize in Economics

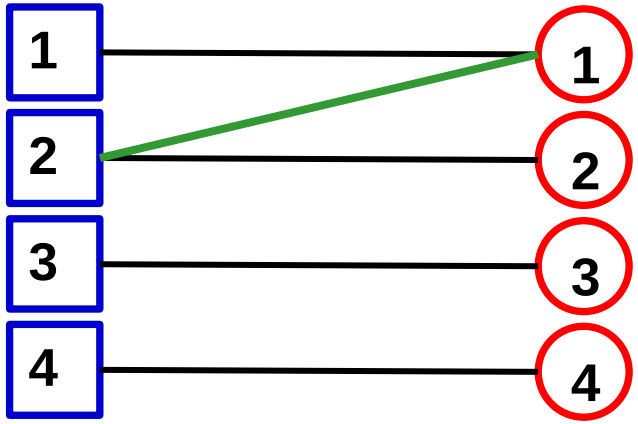
Alvin E. Roth



Lloyd S. Shapley

The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 2012 was awarded jointly to Alvin E. Roth and Lloyd S. Shapley "for the theory of stable allocations and the practice of market design"

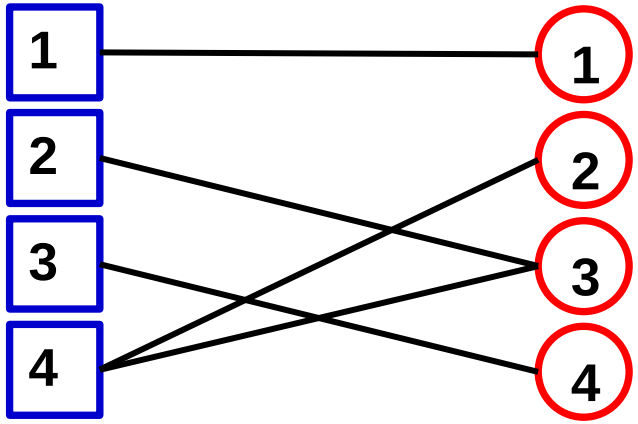
Stable Marriage Problem



1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

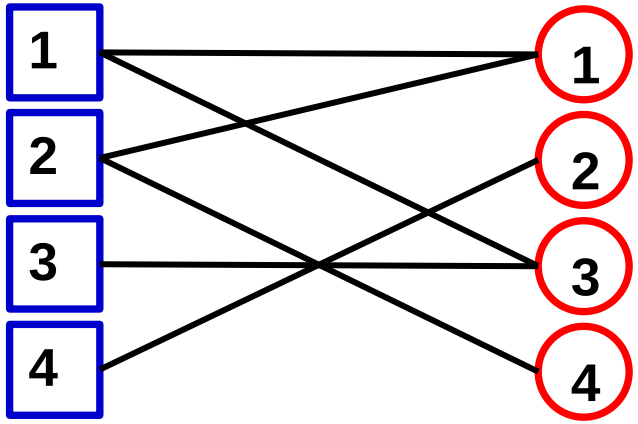
Gale-Shapley Algorithm (1962)



1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

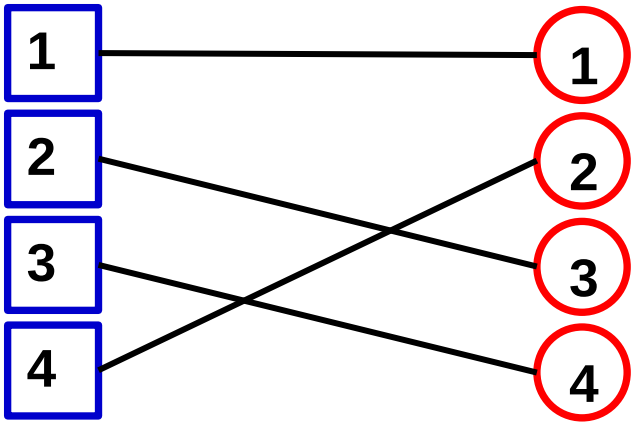
Gale-Shapley Algorithm (1962)

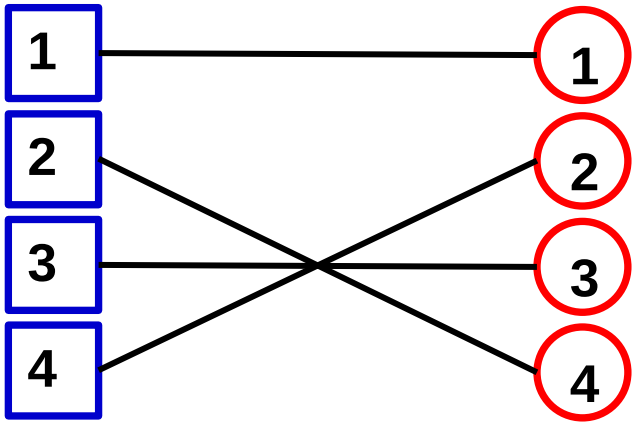


1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale-Shapley Algorithm (1962)





1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Stable Marriage Problem

- Stable marriage may not be unique
- Gale-Shapley algorithm
  - Also call “men propose” algorithm
  - Can find a stable marriage
  - Good for men
    - Please think: why?
- Applications
- 2012 Nobel Prize in Economics