Data Structures and Algorithms

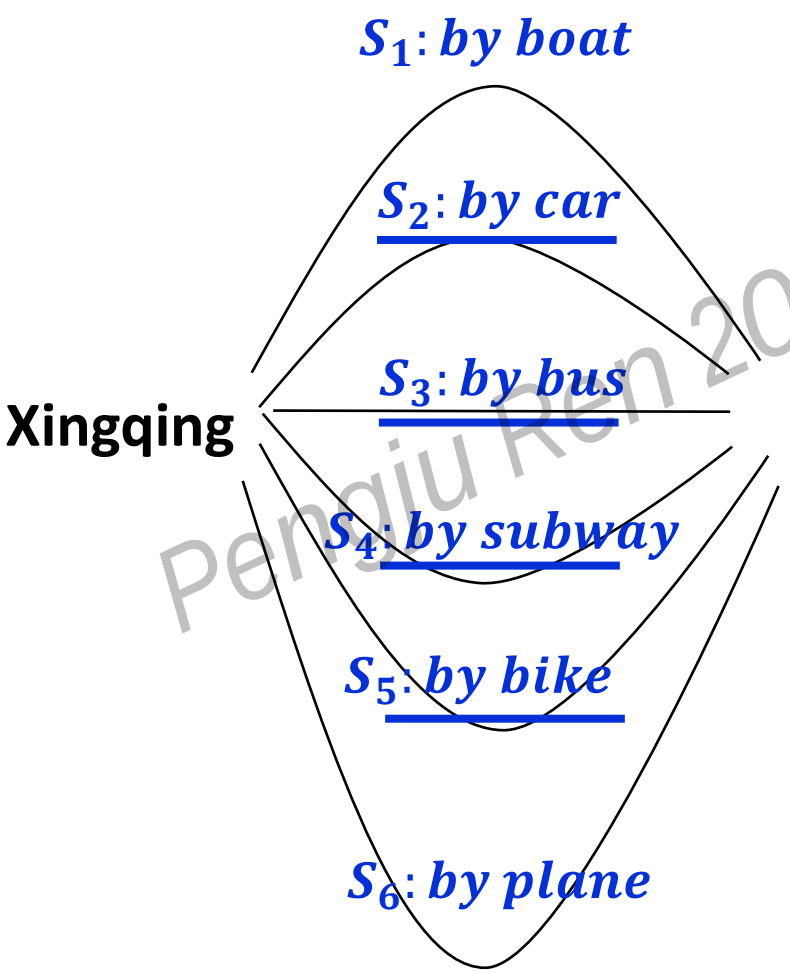
Lecture 09 – Optimization Problems (Greedy, Branch and Bound)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

Optimization Problems

P: from *Xingqing* Campus to *ChuangXin Gang* Campus



S_2, S_3, S_4, S_5 are feasible solution

Fastest: S_2 is the optimal

Cheapest: S_3 is the optimal

Cost function might be different

- *Dynamic Programing*
- ***Greedy***
- ***Branch and Bound***

Knapsack Problem

Object	1	2	3	4	5	6	7
Profits	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1
P/W	5	1.66	3	1	6	4.5	3

Selection = { 1 0 1 0 1 1 1 }

Total Weight = 15

***Greedy policy*: The higher value density the better**

Remain = 15-1=14, Value=6

Remain = 14-2=12, Value=16

Remain = 12-4=8, Value=34

Remain = 8-1=7, Value=37

Remain = 7-5=2, Value=52

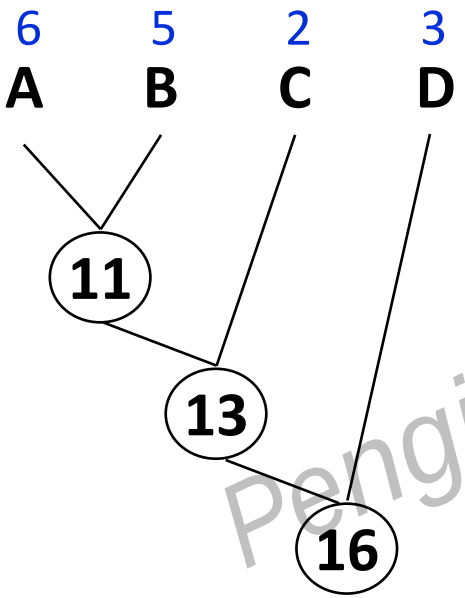
Remain = 2

Constrain: $\sum S_i W_i \leq Weight$

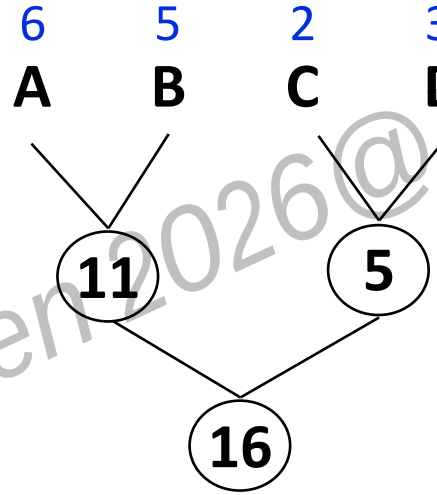
Objective: $max \sum S_i P_i$

Optimal Merge Pattern

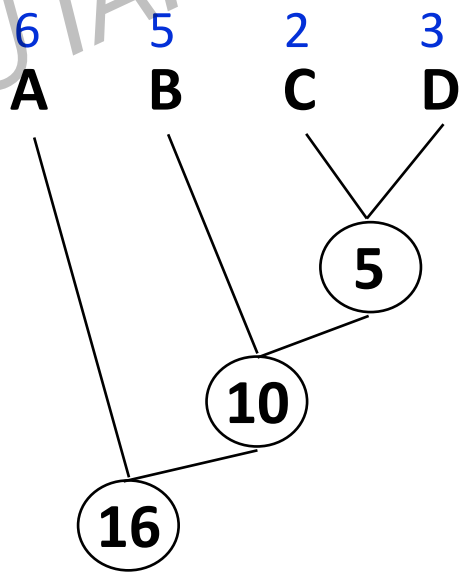
List	A	B	C	D
Sizes	6	5	2	3



$11 + 13 + 16 = 40$



$11 + 5 + 16 = 32$



$5 + 10 + 16 = 31$

Greedy policy ： ***Merge smaller nodes first***

Greedy Method

Greedy algorithms make the *locally optimal choice at each step (best first Search)*, hoping that this leads to a global optimum

Input: An instance of an optimization problem that typically exhibits optimal substructure and the greedy choice property.

Output: A feasible solution that is usually globally optimal built from locally optimal choices.

State representation: The set of choices already made, together with the remaining subproblem.

State transition rule: At each step, choose the option that looks best *currently*, then *recursively solve the remaining subproblem*.

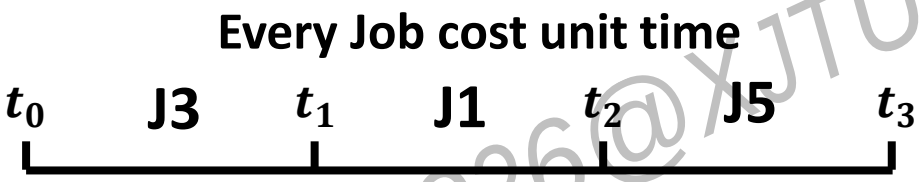
Greedy v.s Dynamic Programming

	Greedy	Dynamic Programming
Decision style	**One choice per step**, does not depend on future	Considers all subproblems, uses state transition equations
Overlapping subproblems	Usually each subproblem appears once	Relies on overlapping subproblems to avoid recomputation
Global optimal guarantee	**No**. Yes only when the greedy choice property holds	Always guaranteed (if optimal substructure holds)
Time complexity	Low	High
Space complexity	Low	High (stores subproblem solutions)
Uses memoization	No	Yes (table or array)

Greedy can be seen as a special case of DP: when the DP state transition *depends only on the current local optimum* and *each subproblem appears only once*, DP degenerates into greedy.

Job Sequencing with Deadlines

Jobs	J1	J2	J3	J4	J5
Profits	20	10	15	1	5
deadline	2	1	2	3	3



Greedy policy ***：the higher profit the better***

J1@ slot1 or slot2 ? ***least constraining value*** **-> @slot 2**

J3@ slot1

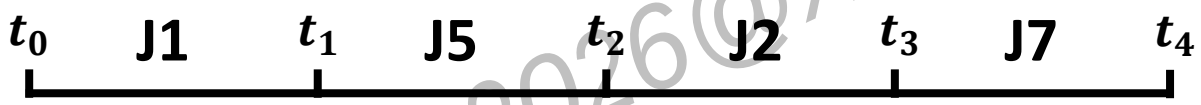
J2@ ∅

J5@ slot3

Quiz：Job Sequencing with Deadlines

Jobs	J1	J2	J3	J4	J5	J6	J7
Profits	20	35	15	12	25	5	30
deadline	2	3	3	1	4	2	4

Every Job cost unit time



Greedy policy ***：the higher profit the better***

Huffman Coding

Message : BCCABBDDAECCBBAEDDCC

Size(Message) = 20

ASCII – 8bit

A: 65 01000001

B: 66 01000010

C: 67 01000011

D: 68 01000100

E: 69 01000101

$20 \times 8bit = 160bit$

Character	Code
A	000
B	001
C	010
D	011
E	100

$Message\ after\ coding: 20 \times 3bit = 60bit$

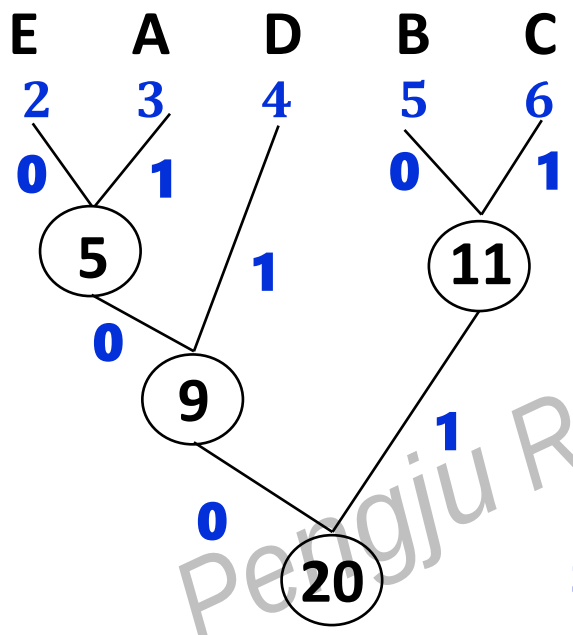
Codebook is also NEEDED!

$Codebook: 5 \times 8bit + 5 \times 3bit = 55bit$

$Total\ is\ 115bit$

Huffman Coding

Message : BCCABBDDAECCBBAEDDCC



Character	Count/Freq	Code
A	3	001
B	5	10
C	6	11
D	4	01
E	2	000

$$Message\ after\ coding:$$

$$3 \times 3bit + 5 \times 2bit + 6 \times 2bit + 4 \times 2bit + 2 \times 3bit = 45bit$$

$$Code\ book:\ 5 \times 8bit + 2 \times 3bit + 2 \times 3bit = 52bit$$

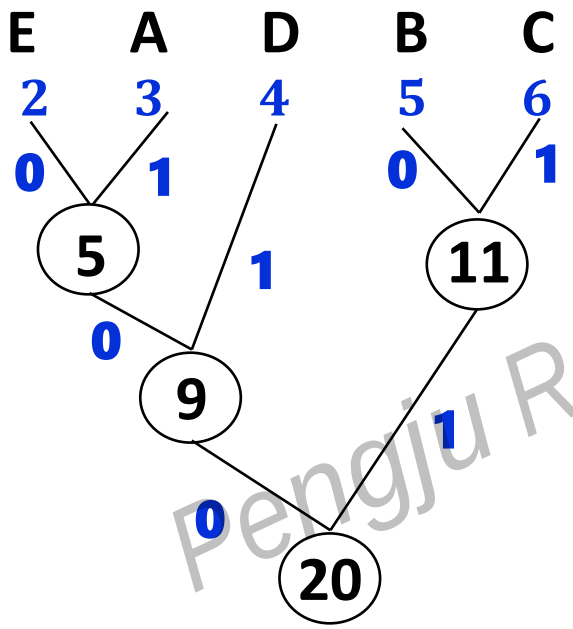
$$Total\ is\ 97bit$$

Greedy policy : The higher the frequency of a character, the shorter its encoding.

Huffman Decoding

Codes: 10111100110100101001000111110100010000101111 1

B C C A B B D D A E C C B B A E D D C C

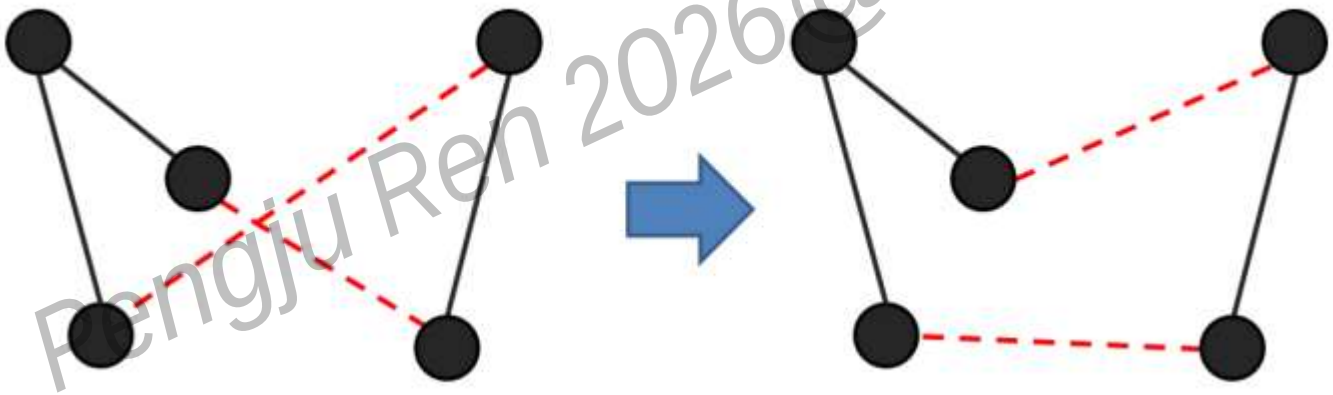


Traveling Salesman Problem (TSP)

You needs to start from a starting point, visit several cities (each exactly once), and finally return to the starting point. The distances between every pair of cities are marked on the map. You want to find a tour with the *shortest total distance*.

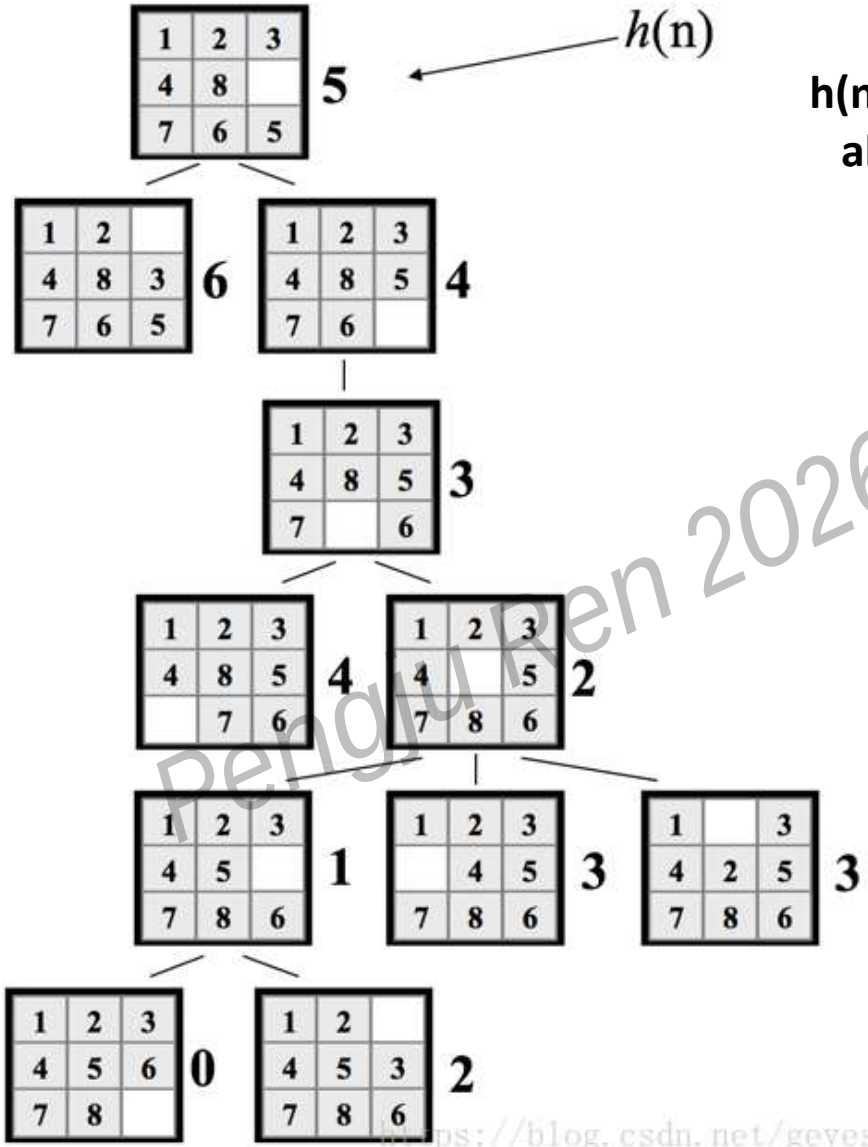
Traveling Salesman Problem (TSP)

- Begin with any complete tour (random).
- Check if any pairwise exchange (*swap-2*) shorten the tour.
- We can check *swap-k* of course, but it takes $O(n^k)$ time.



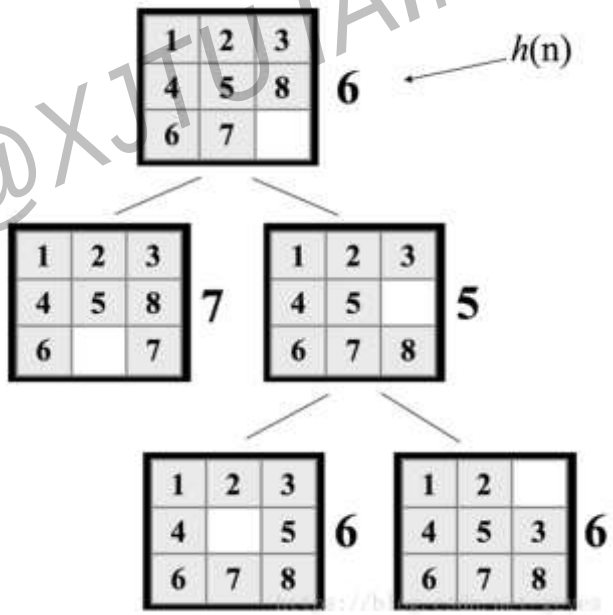
- Empirically, *swap-2* and *swap-3* makes a good TSP searcher.

Example: 8-Puzzle



Find a solution

h(n) records the Manhattan distance of all numbers to their correct location

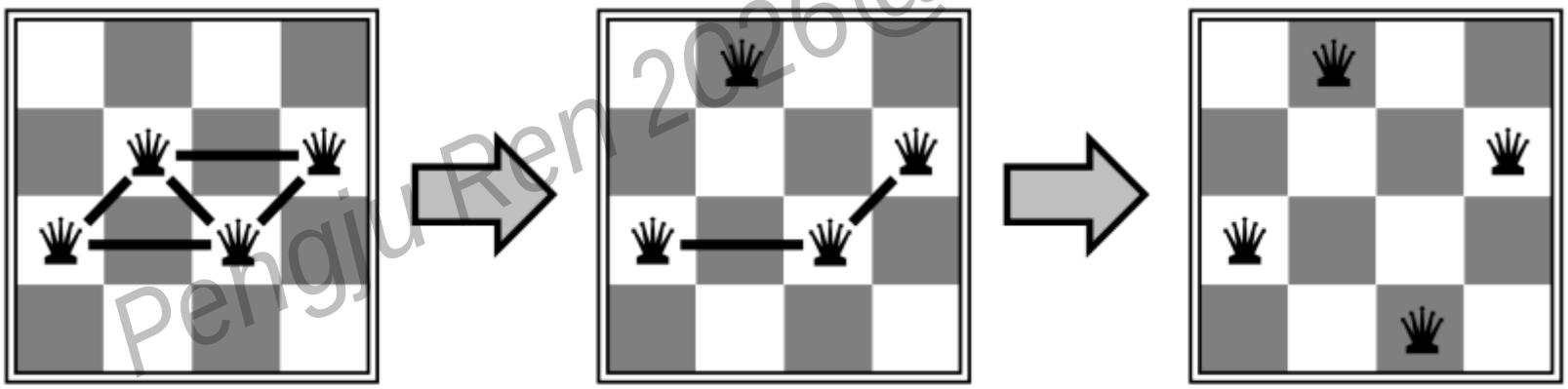


Failed

Example: n-Queens

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal.
- Move a queen to reduce number of conflicts.
- Almost always solve *n*-queen puzzle immediately even for

.



h(n) records the number of conflicts between all queens

Pros and Cons of Greedy Method

Advantages

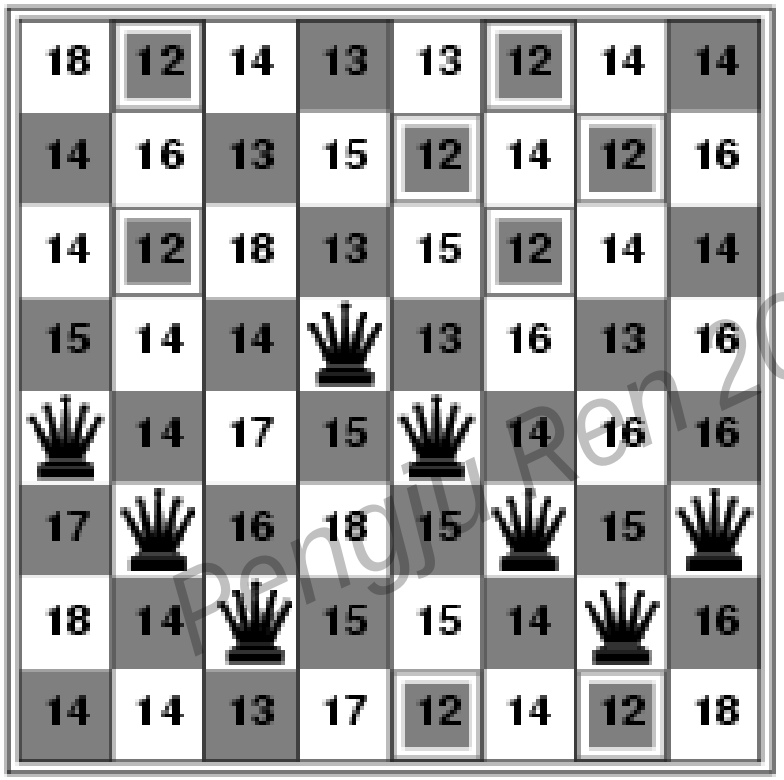
- Simple and intuitive to implement
- High efficiency
- Low memory footprint
- Yields global optima on problems where it applies
- Can serve as an approximation algorithm

Disadvantages

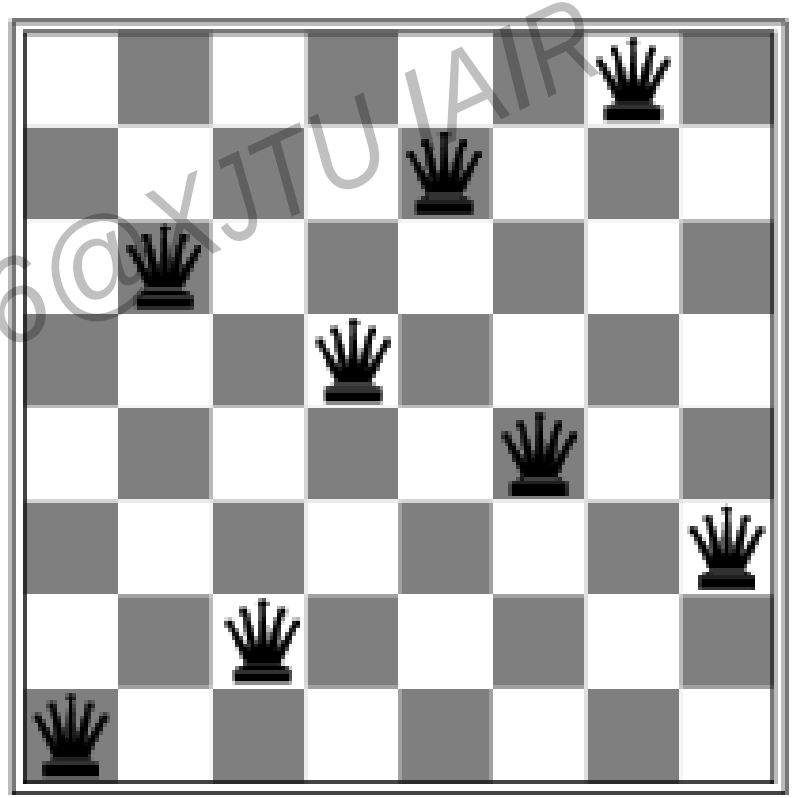
- Limited scope of application
- Cannot backtrack or correct mistakes
- Sensitive to problem constraints and Not suitable for all optimization objectives

Example: n-Queens

h(n) records the number of conflicts between all queens



h = 17



h =1

Sometimes, there is no guarantee that a feasible solution can be found.

Example: Coin Change

Problem Description

Given several coin denominations (unlimited supply of each), we want to make a certain amount using the fewest coins.

The greedy strategy: *at each step, take the largest coin that does not exceed the remaining amount, and repeat.*

Case1: Coin denominations: 1, 5, 10 and **Target amount**: 16

- **Greedy solution**: 10 + 5 + 1 = **3 coins**
- **Optimal solution = Greedy Solution**

Case2: Coin denominations: 1, 3, 4 and **Target amount**: 6

- **Greedy solution**: 4 + 1 + 1 = **3 coins**
- **True optimal solution**: Take two 3's: 3 + 3 = 6 → **2 coins**

There is no guarantee that an optimal solution can be found.

How to Prove the Greedy Solution is Optimal?

We usually need to show that the problem satisfies the following two properties:

- **Greedy Choice Property**: The first (local) choice made by the greedy strategy can be part of some global optimal solution.
- **Optimal Substructure**: After making the greedy choice, the optimal solution to the remaining subproblem, combined with the greedy choice, yields an optimal solution to the original problem.

Common proof techniques are **exchange argument** and **induction**.

Branch and bound

- Branch and bound is a systematic method for searching a state space tree. It uses an ***upper bound (for maximization)*** or a ***lower bound (for minimization)*** to prune branches that cannot lead to an optimal solution.
- Unlike backtracking, which is typically used to find all solutions or any feasible solution, branch and bound focuses on optimization and uses *bounding functions for pruning*.
- **Core challenge**:
  How to design a *bounding function* that is both tight and easy to compute? How to choose branching strategies and search order (DFS, BFS, priority queue) to quickly find the optimal solution and prune effectively?

Basic Flow of Branch and Bound (minimization problem)

Initialization: Create a root node, ***Compute a upper bound*** for the root node, insert the root node into the *live node set*.

Node Selection: Extract a node from the *live node set* (typically the one with the smallest lower bound), If the live node set is empty, terminate and output best.

Branching (expansion): ***Expand*** the current node by generating all legal child nodes, For each child node: compute its lower bound.

Bound comparison (pruning): If the lower bound of the current node ≥ upper bound, this node cannot yield a better solution; ***discard*** **it** (prune) and go back to step 2.

Branch and Bound (Knapsack)

Total Weight = 10

Object	1	2	3	4
Profits	40	42	25	12
Weight	4	7	5	3
P/W	10	6	5	4

$$up\ bound = \sum p_i + \left(W - \sum w_i\right) * \max\left(\frac{p_j}{W_j}\right)$$

$$\mathbf{i} \in Selection, \mathbf{j}\ ! \in Selection$$

- If Object 1 is selected: $up\ bound = 40 + (10 - 4) * 6 = 76$
- If Object 1,3 are selected: $up\ bound = (40 + 25) + (10 - 4 - 5) * 6 = 71$
- Final decision (maximize Profits): Obj 1 and Obj 3 are Selected.

Branch and Bound (Job Sequencing with ddl)

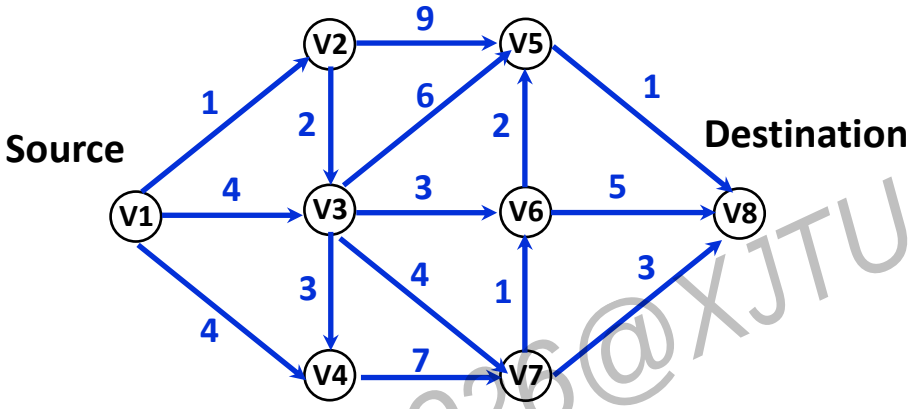
Jobs	J1	J2	J3	J4
Penalty	5	10	6	3
deadline	1	3	2	1
Time cost	1	2	1	1

$$\boldsymbol{Lower\ bound} = \sum \boldsymbol{p_i} \qquad \boldsymbol{i} \in \boldsymbol{Selection}$$

$$\boldsymbol{penalty} = \sum \boldsymbol{p_j} \qquad \boldsymbol{j} \in \boldsymbol{Selection\ till\ current\ Job}$$

- **Final decision (minimize penalty) is J2 & J3 are selected, and penalty is 8**

Branch and Bound (Single Source Shortest Path)



$$Lower\ bound(t) = \delta(s,t) + \min_{(t,u)\in E} w(t,u)$$

t: *is the current node*

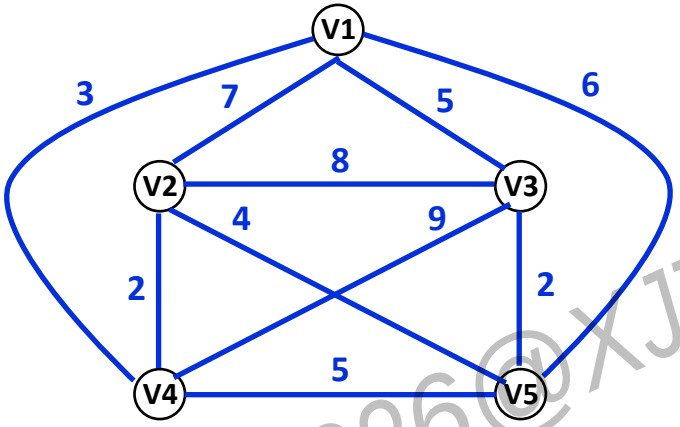
$\delta(s,t)$: *the shortest path from* s *to* t, *or* ∞ *if not reachable*

$w(t,u)$: *the weight of a directed edge from* t *to* u

A* uses a *heuristic function* to estimate the cost from *t* to *d*

- Final decision (minimize cost) is *v1-v2-v3-v6-v5-v8*, and cost is 9

Branch and Bound (TSP on undirect Graph)



Lower bound:

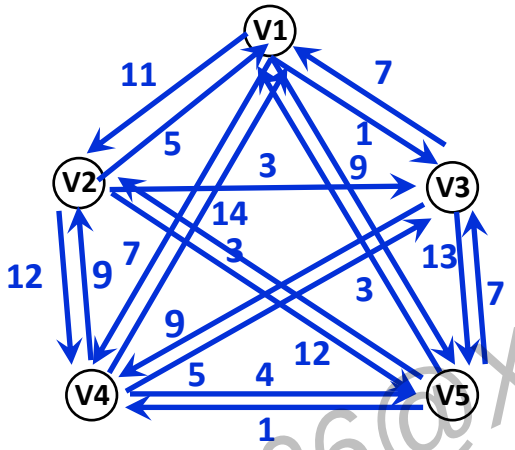
for each vertex u_i ,let w_i^1 and w_i^2 denote the costs of the two cheapest edges *connect with* u_i, so:

$$W_i = w_i^1 + w_i^2 , \quad \sum_{i \in V} W_i \leq 2W(\text{TSP*})$$

Therefore $h = \frac{1}{2}\sum_{i \in V} W_i$ could be the lower bound

Using the result of the greedy search as the Upper Bound can improve search efficiency.

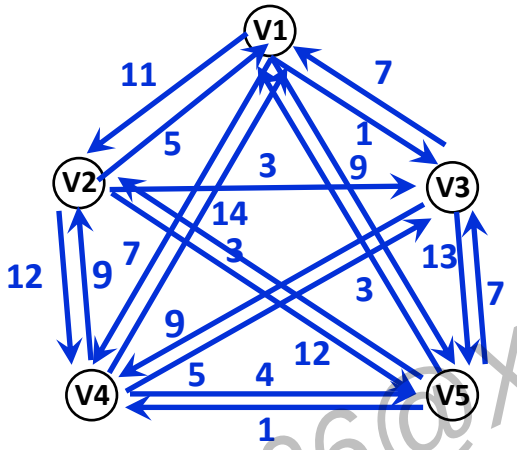
Branch and Bound (TSP on Direct Graph)



Lower bound:

- **Row reduction:** For each vertex find its ***cheapest exit edge*** u_i^{out}, and subtract this value from all outgoing edges to form a row reduced matrix
- **Col reduction:** Then for each vertex in the row reduced matrix find its ***cheapest incident edge*** v_j^{in}, and subtract this value from all incoming edges, to form the reduced matrix
- $\sum u_i^{out} + \sum v_j^{in}$ is the lower bound

Branch and Bound (TSP on Direct Graph)



Row reduction

	V1	V2	V3	V4	V5	min_row
V1	∞	11	1	7	9	1
V2	5	∞	3	12	3	3
V3	7	1	∞	9	13	1
V4	14	9	5	∞	4	4
V5	3	12	7	1	∞	1

Initial Matrix

	V1	V2	V3	V4	V5
V1	∞	10	0	6	8
V2	2	∞	0	9	0
V3	6	0	∞	8	12
V4	10	5	1	∞	0
V5	2	11	6	0	∞
min_col	2	0	0	0	0

Col reduction

	V1	V2	V3	V4	V5
V1	∞	10	0	6	8
V2	0	∞	0	9	0
V3	4	0	∞	8	12
V4	8	5	1	∞	0
V5	0	11	6	0	∞

Reduced Matrix

$$lb = \sum u_i^{out} + \sum v_j^{in} = 10 + 2 = 12$$

Branch and Bound (TSP on Direct Graph)

Lemma1: let $G = (V, E)$ be a complete directed graph, where each arc $(u, v) \in E$ has a cost $c(u, v) \geq 0$ .For an arbitrary vertex $i \in V$ and any constant $a \in \mathbb{R}$, define a new cost function c' as follows:

- **Case 1 (subtract constant from all incoming edges to i):**
  For every $u \neq i$, $c'(u, i) = c(u, i) - a$;
- **Case 2 (subtract constant from all outgoing edges from i):**
  For every $v \neq i$, $c'(i, v) = c(i, v) - a$;

Then for any two Hamiltonian cycles H_1, H_2 in the graph,

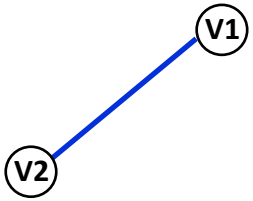
$$\boldsymbol{W(H_1) > W(H_2) \implies W'(H_1) > W'(H_2)},$$

where W and W' denote the total cycle cost under c and c' respectively.

Lemma2: Given a complete directed graph G, for any vertex i:

- subtract the same constant a from the weights of all incoming edges to i; or
- subtract the same constant a from the weights of all outgoing edges from i, to obtain a new graph G'. **TSP*(G)=TSP*(G')**

Branch and Bound (TSP on Direct Graph)



Reduced Matrix of parent

	V1	V2	V3	V4	V5
V1	∞	10	0	6	8
V2	0	∞	0	9	0
V3	4	0	∞	8	12
V4	8	5	1	∞	0
V5	0	11	6	0	∞

Initial Matrix of child

	V1	V3	V4	V5
V2	∞	0	9	0
V3	4	∞	8	12
V4	8	1	∞	0
V5	0	6	0	∞

Row reduction

min_row: 0, 4, 0, 0

Reduced Matrix of child

	V1	V3	V4	V5
V2	∞	0	9	0
V3	0	∞	4	8
V4	8	1	∞	0
V5	0	6	0	∞

min_col: 0 0 0 0

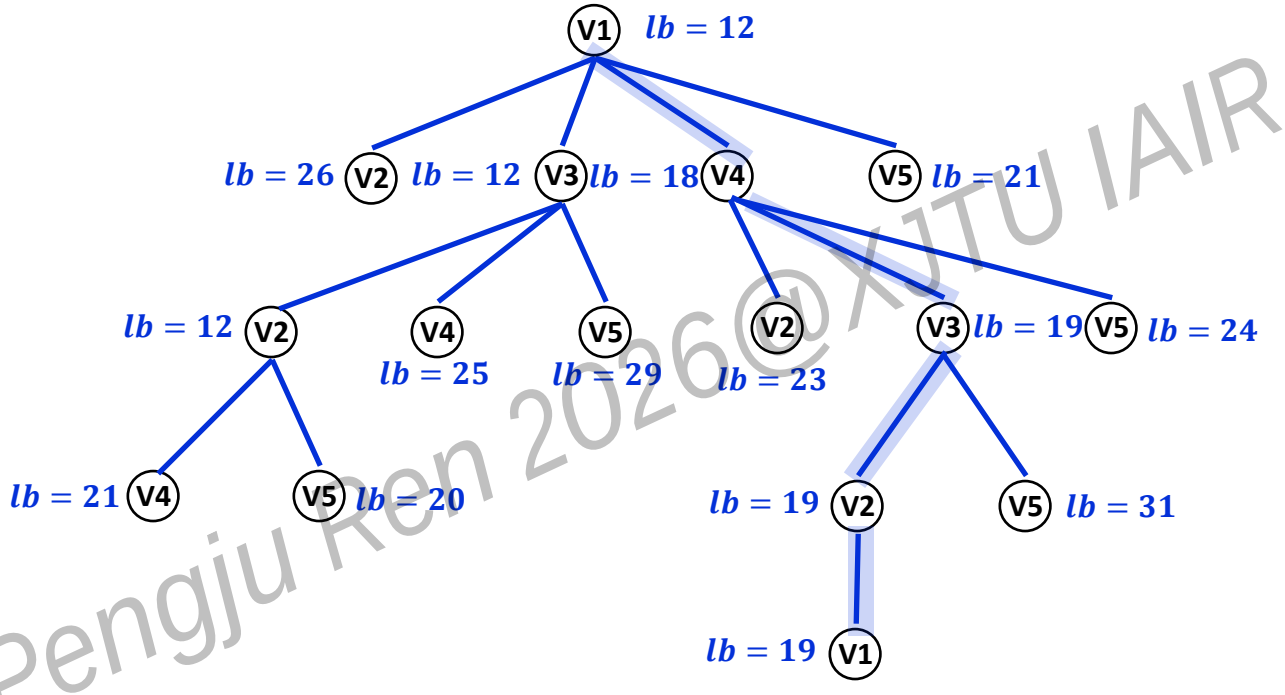
Global base cost already determined

Extra cost of the chosen edge not yet in parent's LB

Minimum additional cost of the remaining subproblem

$$lb = lb(parent) + w(vi - vj) + child.reduced = 12 + 10 + 4 = 26$$

Branch and Bound (TSP on Direct Graph)



Three strategies for UB/LB used for Efficient Pruning

■ Quickly construct an initial feasible solution

- **Method**: Before the search begins, use a **heuristic/greedy algorithm** to rapidly generate a feasible one.
- **Effect**: The "threshold" for pruning exists before the search even starts. If the root node already satisfies LB ≥ UB(or UB ≥ LB), stop immediately; otherwise, any node whose lower(or upper) bound reaches this threshold is discarded.

■ Activate pruning during the search

- **Method**: If no feasible solution can be pre-constructed, start with UB = +∞ (or LB=0), leaving the pruning condition "dormant." During the search, as soon as you encounter any feasible solution, use it to reset UB (or LB) and activate pruning.
- **Effect**: The search behaves as a "seeking mode" in the first phase, then turns into a "pruning mode" afterward.

■ LB/UB-guided search when no feasible solution is available

- **Scenario**: Feasible solutions are very hard to find (e.g., highly constrained problems with great depth), and UB (or LB) remains infinity for a long time.
- **Strategy**: Pruning here relies entirely on **the lower (or upper) bound itself**. The algorithm essentially degrades to **pure best-first search**.

Branch Bound v.s Greedy

	Greedy	Branch and Bound
Search style	Constructive, no explicit search tree	Explicitly builds a search tree, systematic search
Branching	No branching, follows a single path	Multi-branching, keeps ***live nodes***
Pruning	No pruning	Uses ***bounding functions to prune branches*** that cannot lead to an optimal solution
Backtracking	No backtracking	Can backtrack (via other nodes in the queue)
Optimality guarantee	Only when greedy property holds	Always guaranteed (if bounds are admissible)
Complexity	Low	High (exponential)
Applicable size	Can handle large problems	Only moderate size (n ≤ 30-40)

- **If branch and bound keeps only one node (i.e., always expands the current best node and discards the others), it degenerates into a greedy algorithm.**
- **Both can be used for optimization, but greedy is "*one-shot*" while branch and bound is "*multi-path exploration*".**

Greedy, DP and Branch Bound

- **All leverage optimal substructure**:
  - Greedy: local optimum.
  - DP: global optimum contains optimal subproblem solutions.
  - Branch and bound: uses bounds to estimate subproblem optimal values.
- **Search strategy relations**:
  - Greedy is **single-path constructive search**.
  - DP is **bottom-up state space search**.
  - Branch and bound is **top-down search with pruning**.
- **Recurrence vs pruning**:
  - Greedy directly constructs a solution via “one-shot” choices.
  - DP avoids recomputation via recurrence.
  - Branch and bound avoids invalid branches via pruning.

Diff between Branch bound and Backtracking

- **Goal**: Backtracking is usually for finding all solutions or any feasible solution; branch and bound is for finding the optimal solution.
- **Pruning basis**: Backtracking uses ***feasibility pruning***; branch and bound uses ***bound pruning*** (upper/lower bounds).
- **Search order**: Backtracking often uses DFS; branch and bound often uses BFS or Best-First Search.

Summary

- **Greedy algorithm makes the locally optimal choice at each step and never backtracks. It is extremely fast but offers no guarantee of global optimality.**
- **Branch bound solves optimization problems through systematic search and bounding, it always guarantees optimality.**
- **Greedy, Branch bound, backtracking and Dynamic Programming**
  All these four methods exploit problem structure to avoid exhaustive search. Backtracking targets feasibility problems and is a special case of branch and bound where the cost is merely feasible/infeasible. dynamic programming relies on optimal substructure and overlapping subproblems, building the global optimum bottom-up through recursion with memorization or tabulation.