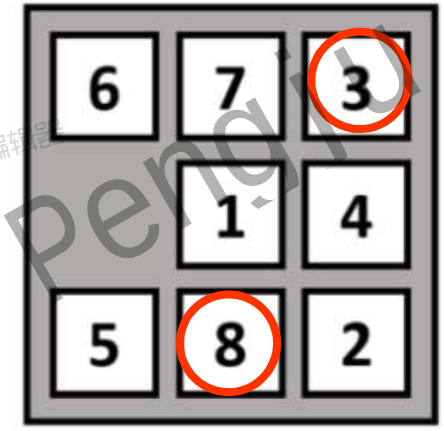
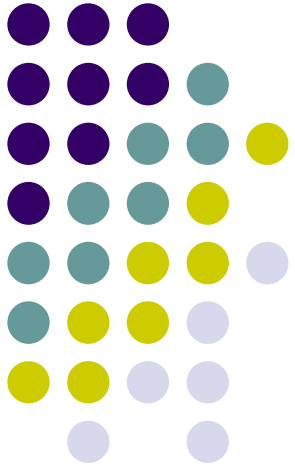
Introduction to AI

Chapter03 Solving Problems by Informed Searching(3.5~3.6)

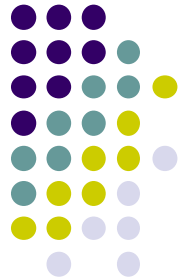
Pengju Ren@IAIR



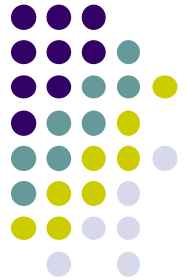


Outline

- **Best-first search**
  - Greedy search
  - A* search
  - Optimality of A*
- **Memory Bounded Search**
  - Iterative deepening A*
  - Recursive best-first search
  - Simplified memory-bounded A*
- **Heuristic**
  - Performance
  - Generating heuristics



Best first Search



- Informed search, a,k,a. heuristic search.
- Idea: use an evaluation function *f(n)* for each node estimate of "desirability "

  Expand the most desirable unexpanded node
- Implementation: Order the nodes in the fringe in decreasing order of desirability
- The evaluation function is called heuristic, denoted as *h(n)*

  It estimates of cost from node *n* to the closest goal
- Special cases:
  - Greedy search. *f(n)=h(n)*
  - A* search *f(n)=g(n)+h(n)* *g(n):* path cost

    A* ≈ greedy + Uniform-cost = *h(n) + g(n)*

Greedy Search

- $h_{SLD}(n)$ = straight-line distance from n to DongSi
- Greedy search expands the node that appears to be closest to goal

Greedy Search

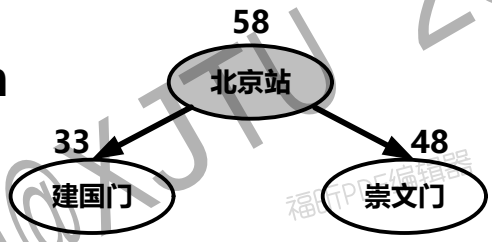
到达东四的直线距离:
-东四 0
-朝阳门 15
-张自忠路 20
-南锣鼓巷 30
-建国门 33
-灯市口 35
-永安里 35
-东单 45
-王府井 47
-崇文门 48
-天安门东 50
-北京站 58
-前门 62
-东四十条 75
-东直门 75
-东大桥-77

- $h_{SLD}(n)$ = straight-line distance from n to DongSi
- Greedy search expands the node that appears to be closest to goal

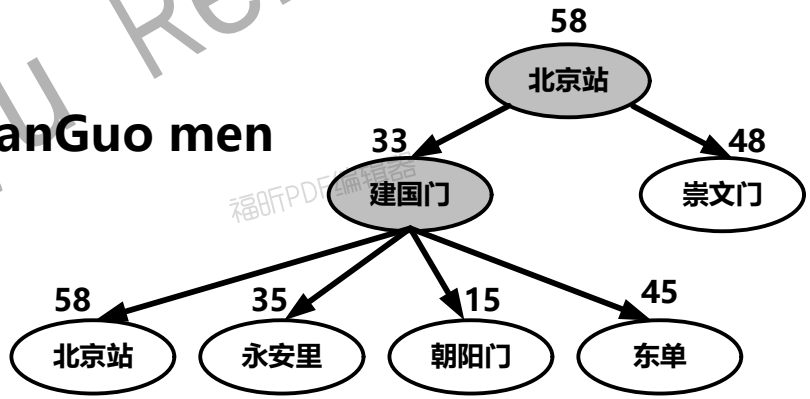
Greedy Search

(a) The initial state

(b) After expanding Beijing Station



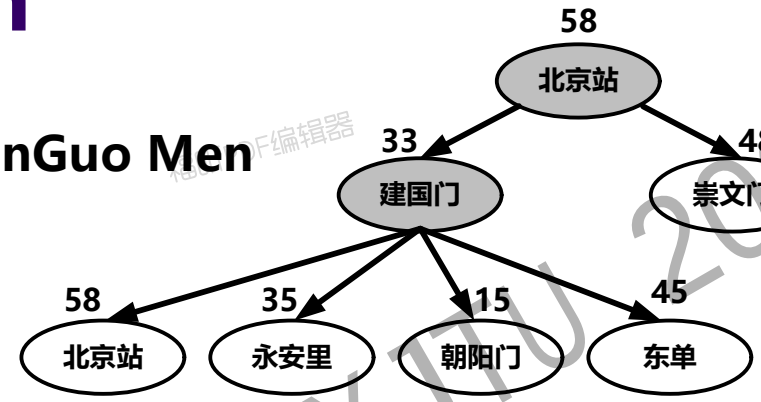
(c) After expanding JianGuo men



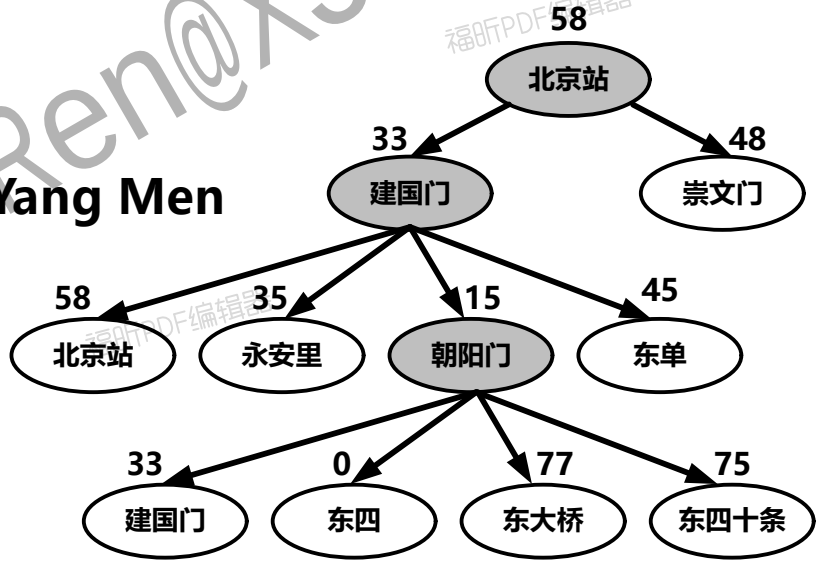
- $h_{SLD}(n)$ = straight-line distance from n to DongSi
- Greedy search expands the node that appears to be closest to goal

Greedy Search

(c) After expanding JianGuo Men



(d) After expanding ChaoYang Men

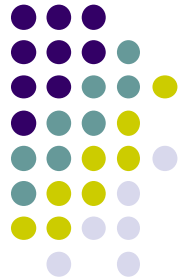


- $h_{SLD}(n)$ = straight-line distance from n to DongSi
- Greedy search expands the node that appears to be closest to goal

Property of Greedy Search

- **Completeness**: No.
  - *TREE-SEARCH* may get stuck in loops and never reach any goal even in finite state spaces.
    GRAPH-SEARCH is complete in finite spaces, but not complete in infinite ones.
- **Optimality:** No.
- **Time complexity:** $O(b^m)$, but a good heuristic can give dramatic improvement.
- **Space complexity:** $O(b^m)$, since it keeps all nodes in memory.

A* Search



- Idea: Avoid expanding paths that are already expensive.
- Evaluation function: $f(n) = g(n) + h(n)$
  - $g(n)$: cost so far to reach n.
  - $h(n)$: estimated cost to goal from n.
  - $f(n)$: estimated total cost from the starting node to goal through n.
- A*search combines the advantages of the uniform-cost search and the greedy search.

A* Search

到达东四的直线距离:
-东四 0
-朝阳门 15
-张自忠路 20
-南锣鼓巷 30
-建国门 33
-灯市口 35
-永安里 35
-东单 45
-王府井 47
-崇文门 48
-天安门东 50
-北京站 58
-前门 62
-东四十条 75
-东直门 75
-东大桥-77

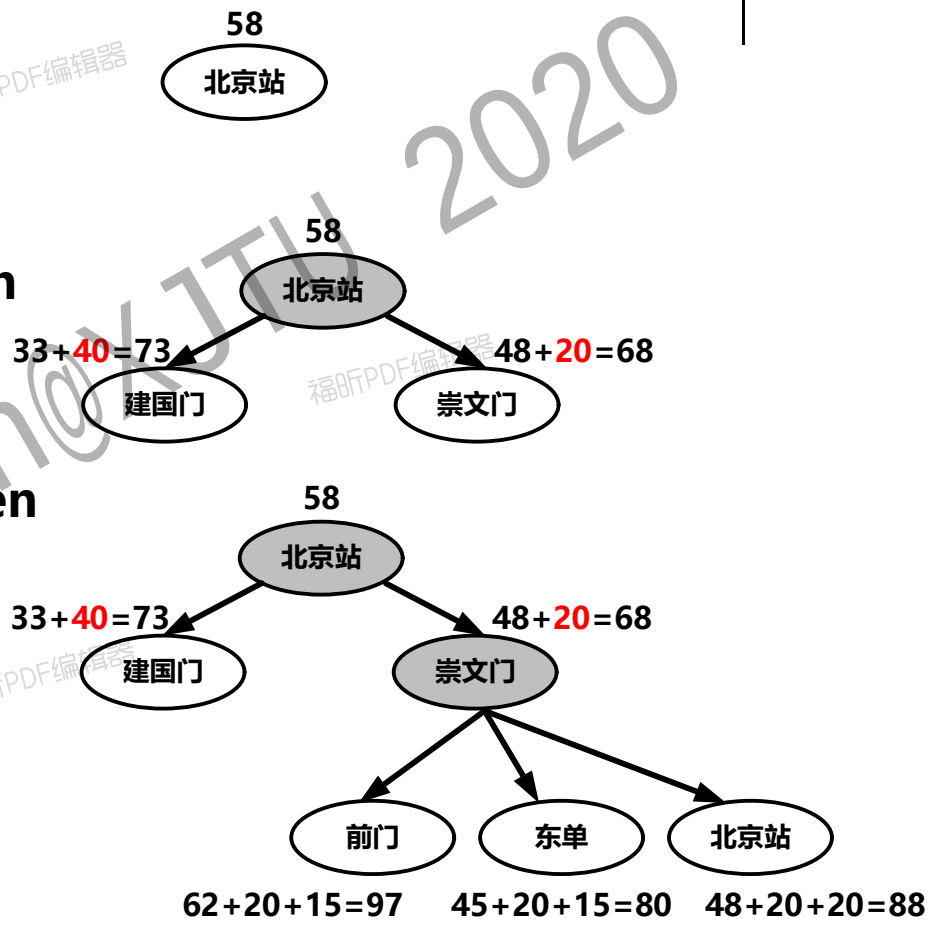
- $h_{SLD}(n)$ = straight-line distance from n to DongSi
- Greedy search expands the node that appears to be closest to goal

A* Search

(a) The initial state

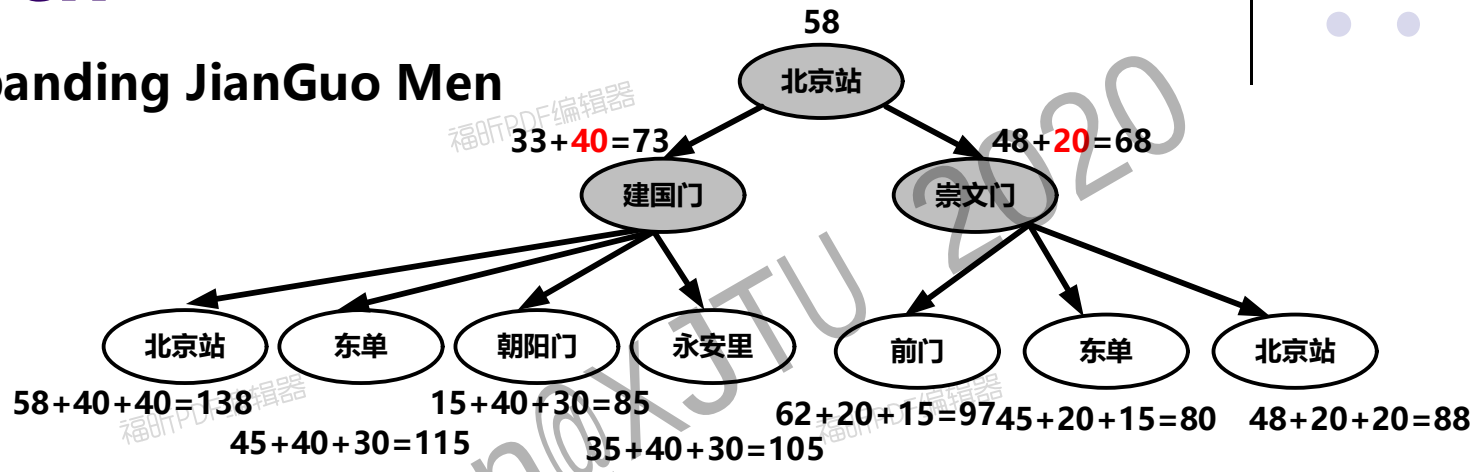
(b)After expanding Beijing Station

(c)After expanding Chongwen Men

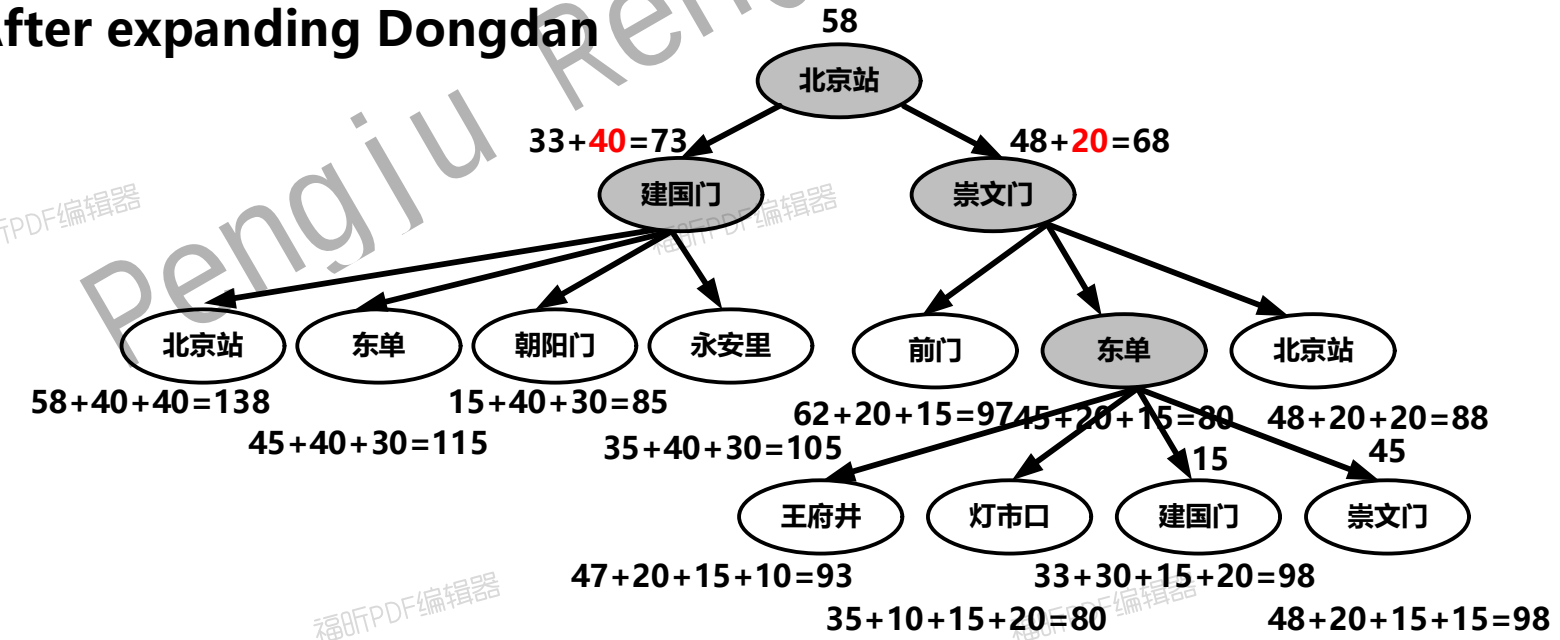


A* Search

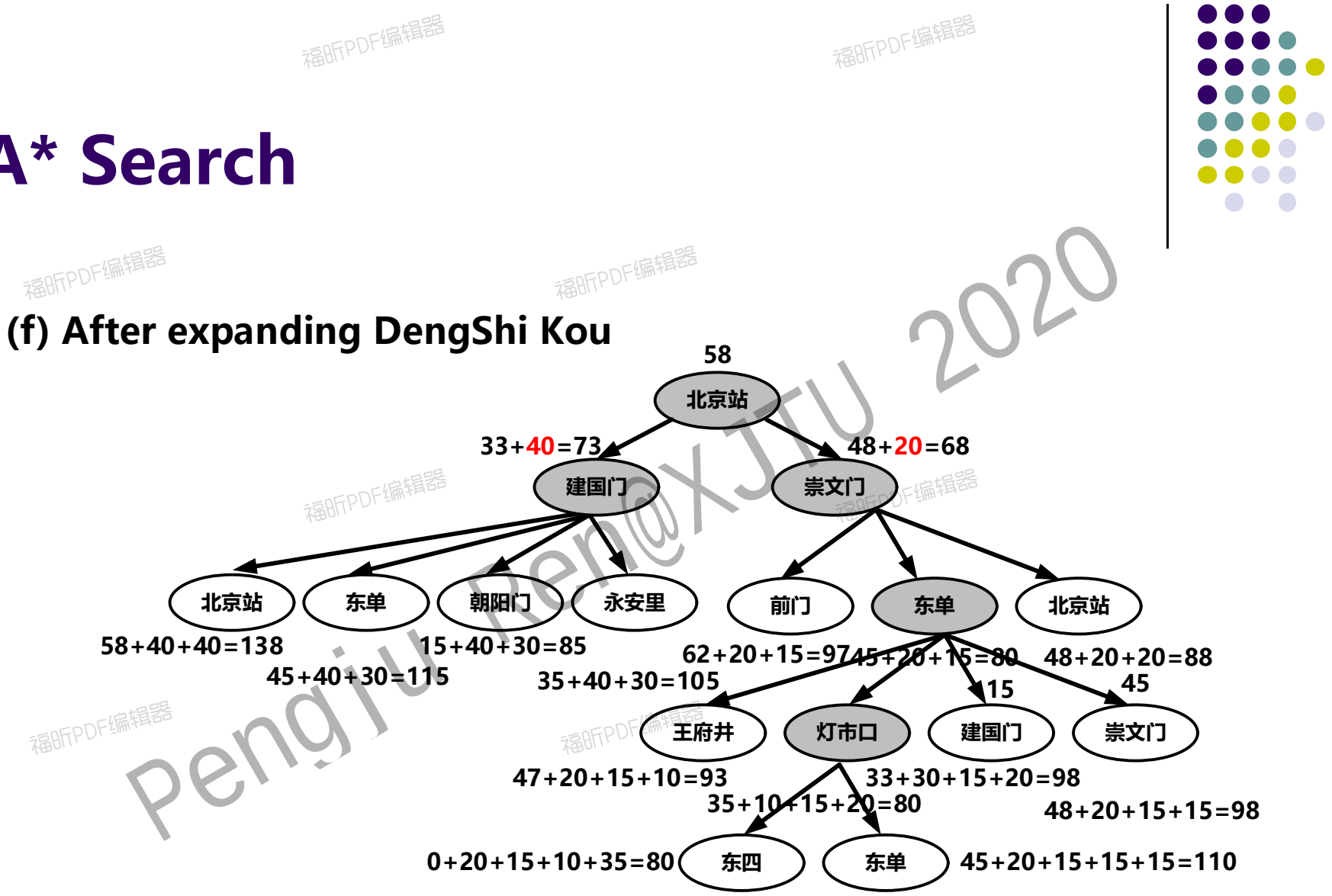
(d) After expanding JianGuo Men



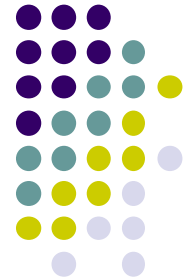
(e) After expanding Dongdan



A* Search



Properties of A* Search



Completeness: Yes. Unless infinite nodes with $f \leq f(goal)$.
Optimality: Depends on whether h is

Admissible

- Never overestimates the actual cost.
- $\forall n, h(n) \leq h^*(n)$, where h^* is the actual cost.
- e.g., $h_{SLD}(n) \leq h^*(n)$.

Consistent

- A.k.a. monotonicity.
- $\forall$ successor n' of any n generated by any action a, $h(n) \leq c(n, a, n') + h(n')$, where c is the step cost.

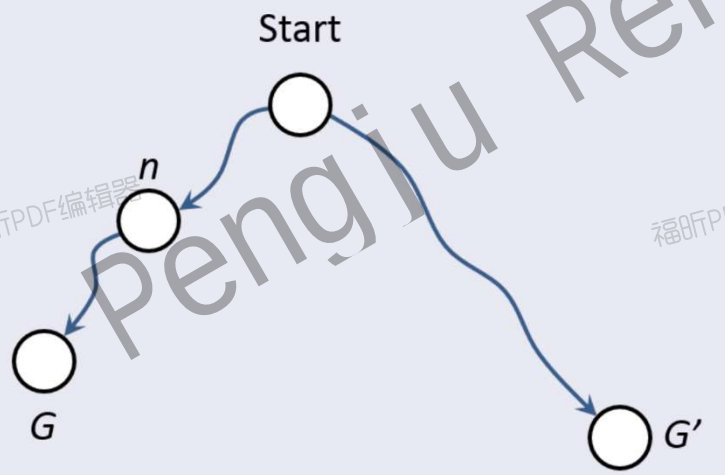
Time complexity: $O(b^{\varepsilon d})$ for constant step costs, where $\varepsilon = (h^* - h)/h^*$ (relative error) and d is the solution depth. Effective branch factor is b^{ε}.
Space complexity: $O(b^d)$, since it keeps all nodes in memory.

Optimality of A*

- A* is optimal on trees if h is admissible.
- A* is optimal on graphs if h is admissible and consistent.

Proof of A*'s optimality on trees.

Suppose some suboptimal goal G' has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G.



$$
\begin{aligned}
f(G') &= g(G') + h(G') \\
&= g(G') \\
&> g(G) \\
&= g(n) + h^*(n) \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

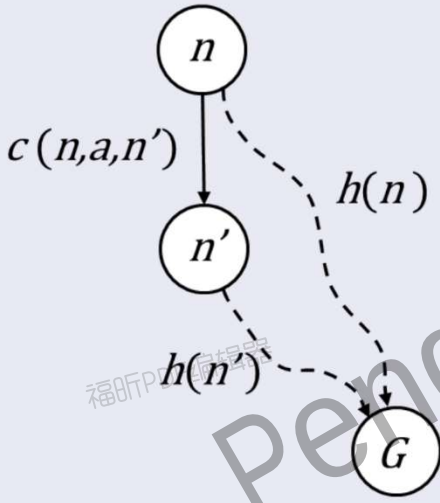
□

Optimality of A* on Graphs

- **Lemma**: If $h(n)$ is **consistent**, the values of f along any path in A* are nondecreasing.
- Gradually adds f-contours of nodes.
- Contour i has all nodes with $f = fi$, where $fi < fi+1$.

Optimality of A* on Graphs

Lemma: if $h(n)$ is consistent, the values of f along any path are nondecreasing.



Consistent heuristic: $h(n) \leq c(n, a, n') + h(n')$

Therefore,

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

□

- **Now we see that consistency is actually triangle inequality.**

Iterative Deepening A* (IDA*)

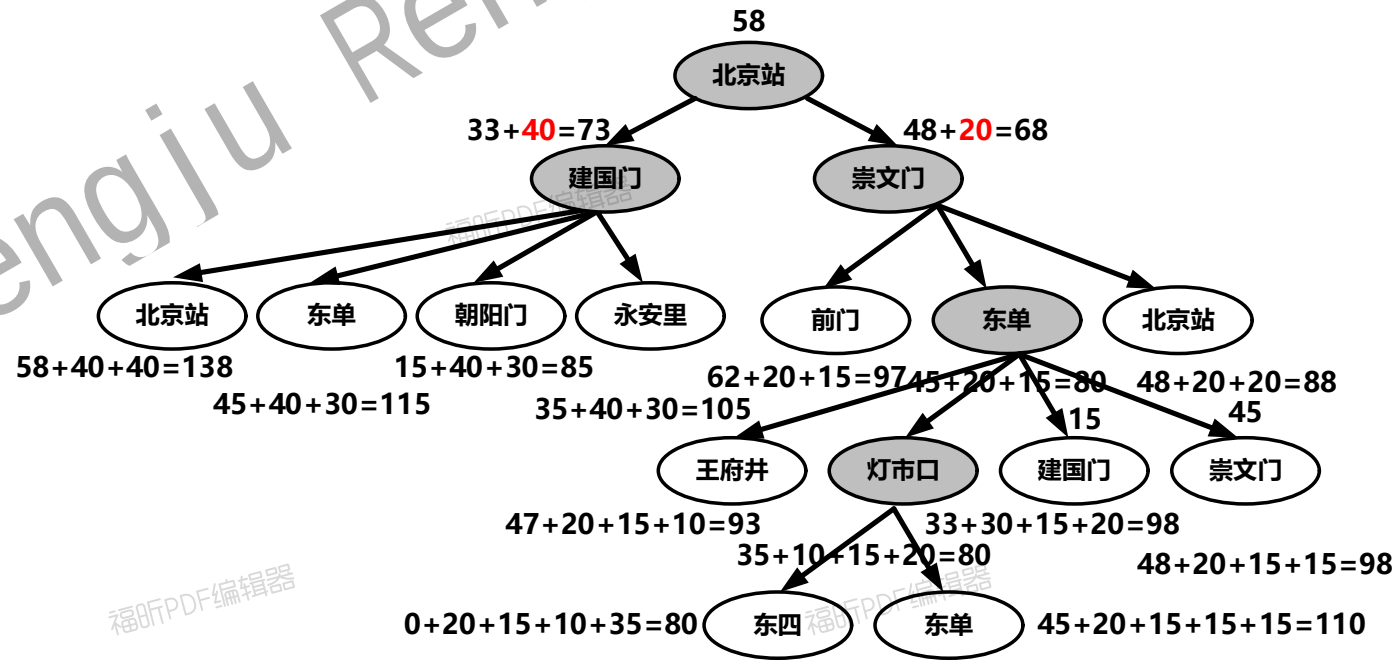
- Time complexity is not A*'s biggest drawback.
- A* usually runs out of memory before it reaches goals.
- Iterative deepening A* (IDA*):
  - Use $f(g + h)$ as cutoff instead of the depth.
  - Initial cutoff: $f(s0) = h(s0)$
  - Perform DFS on nodes where $f(n) <$ cutoff.
  - Reset cutoff to smallest f of non-expanded nodes.

IDA*(*problem*)

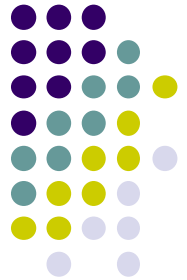
```
1  currentCutoff = f(s0)
2  repeat
3      result = f-LIMITED-SEARCH(problem, currentCutoff)
4      if result ≠ cutoff
5          return result
6      currentCutoff = smallest f-value of non-expanded nodes.
```

IDA* Traversal on Beijing Subway

- 1*st* iteration: *currentCutoff* = 58 (北京站)
  北京站 →*建国门* → *崇文门*
- 2*nd* iteration: *currentCutoff* = 68 (崇文门)
  北京站 →*建国门*→崇文门→*前门* → *东单*
- 3*rd* iteration: *currentCutoff* = 73 (建国门)
  北京站 →*建国门*→*东单*→*朝阳门*→*永安里*→崇文门→*前门*
- 4*th* iteration: *currentCutoff* = 80 (东单)
  北京站 →*建国门*→*东单*→*朝阳门*→*永安里*→崇文门→*前门* → *东单*→*王府井*→*灯市口*



Properties of IDA*



- **Completeness** and **Optimality** same as A*.
- **Time complexity**: $O(b^{\varepsilon d})$
- **Space complexity**: $O(bd)$
- Practical for problems with unit step costs.
- What happens if all f-values are different (real-values)?
  The number of iterations can equal the number of nodes whose f-value is less than the cost of an optimal path!

Recursive Best-First Search (RBFS)

- IDA* is problematic when g are real-valued.
- RBFS is a simple recursive algorithm that mimics standard best-first search using only linear space.

RECURSIVE-BEST-FIRST-SEARCH(*problem*)

return RBFS(*problem*, MAKE-NODE(*problem.initial_state*), ∞)

Recursive Best-First Search (RBFS)

- DFS where each node on the current path remembers the best *f*-value of any alternative path from its ancestors.
  - Maintains all nodes on current path plus all their siblings (*ancestor*(*n*)).
- When expanding node *n*
  - $\forall n' \in children(n)$, compute $f(n')$.
  - if an ancestor *n*'' has a lower *f*-value than all *n*'s children, then
    - Assign *f*-value of the cheapest child to *n*.
    - Backtrack to *n*''.
  - Otherwise, proceed as normal.

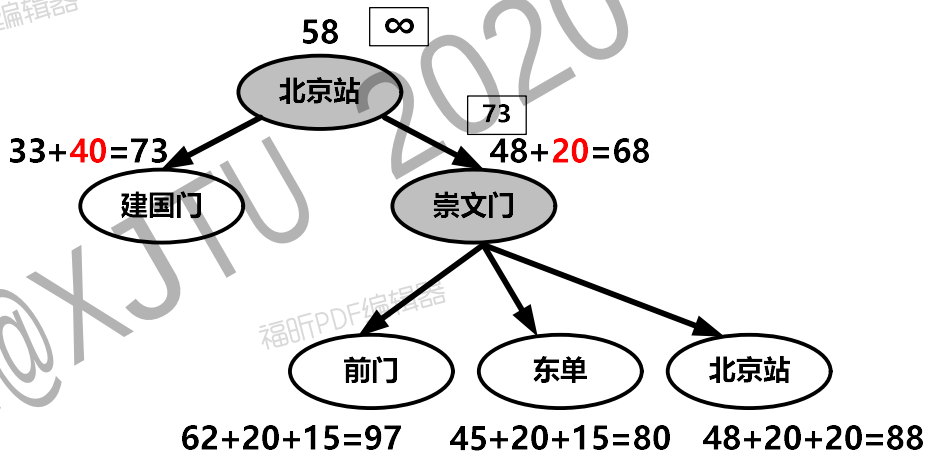
Recursive Best-First Search (RBFS)

RBFS(*problem*, *node*, *f_limit*)

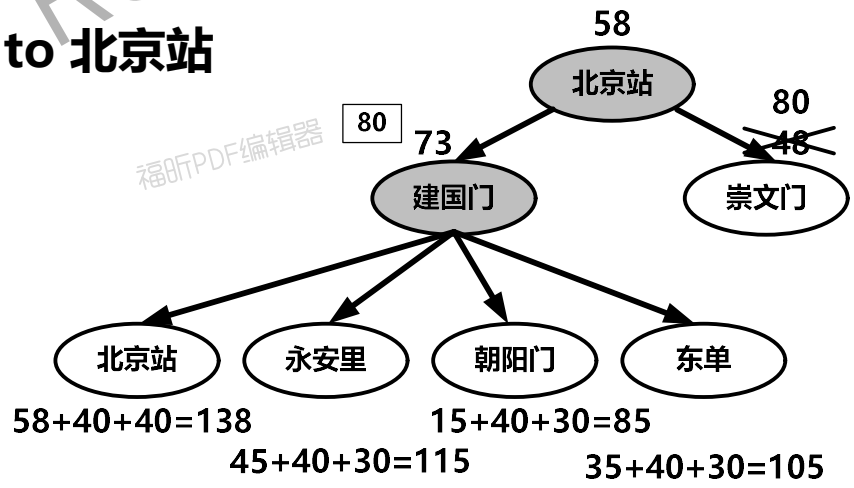
```
1  if problem.GOAL-TEST(node.state) return SOLUTION(node)
2  successors = φ
3  for each action in problem.ACTIONS(node.state) repeat
4        add CHILD-NODE(problem, node, action) into successors
5  if successors is empty return failure, ∞
6  for each s in successors repeat
7        s.f = max(s.g + s.h, node.f)
8  repeat
9        best = the lowest f-value node in successors
10       if best.f > f_limit return failure, best.f
11       alternative = the second-lowest f-value among successors
12       result, best.f = RBFS(problem, best, min(f_limit, alternative))
13       if result ≠ failure return result
```

RBFS on Beijing Subway

(a) After expanding 建国门&崇文门

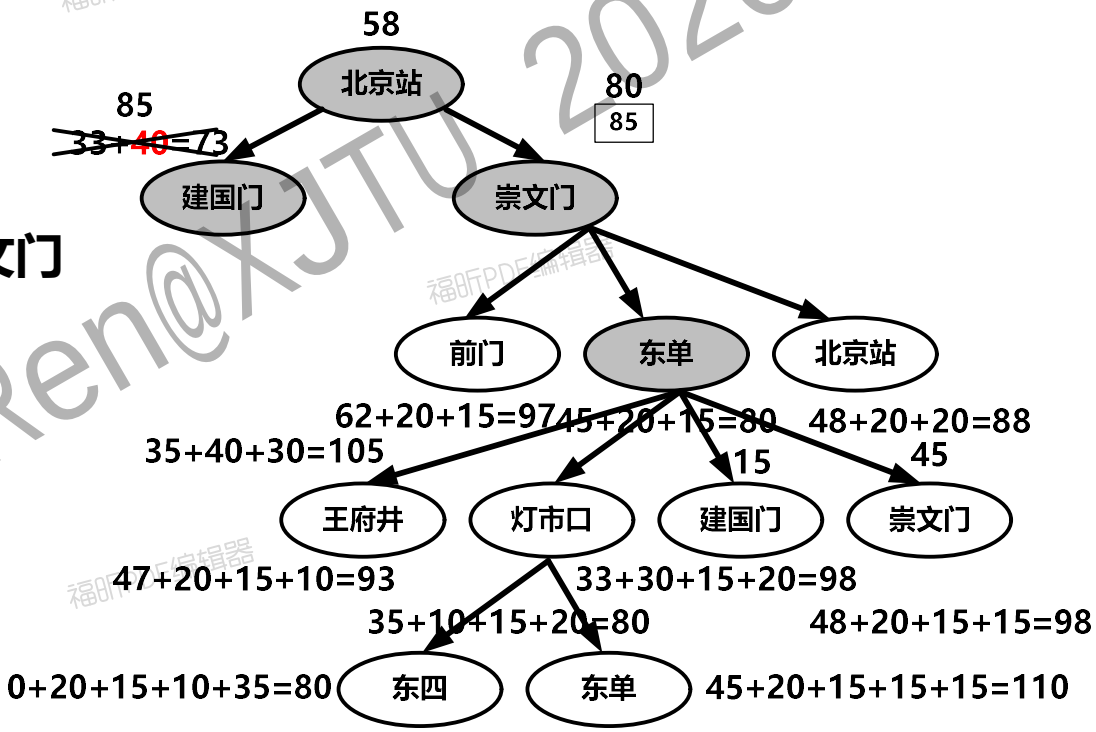


(b) After unwinding back to 北京站 and expanding 建国门



alternative=the second-lowest f-value among successors

RBFS on Beijing Subway



(c) After switching back to 崇文门 and expanding 东单

alternative=the second-lowest f-value among successors

RBFS Traversal on Beijing Subway

- *f_limit* = ∞, expanding 北京站
  北京站 → *建国门* → *崇文门*
- *f_limit* = 73 (建国门), expanding 崇文门
  北京站 → *建国门* →*前门* → *东单*
- Cutoff occurs. Record *f*(东单) as 80. *f_limit* = 80 (东单), expanding 建国门
  北京站 → *东单* → *永安里* → *朝阳门*
- Cutoff occurs. Record *f*(建国门) as 85. *f_imit* = 85 (朝阳门), expanding 崇文门 (again)
  北京站 → *建国门* → *前门* → *东单*
- *f_limit* = 85 (朝阳门), expanding 东单
  王府井→灯市口 →建国门 → *崇文门*
- . . .

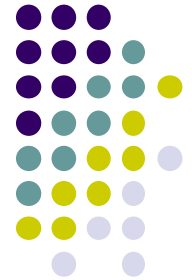
Properties of RBFS

- Completeness and optimality same as A*.
- Time complexity: Depends on accuracy of *h* and on how often best path changes.
- Space complexity: *O*(*bd*)
  Each time RBFS *changes its mind* corresponds to one iteration of IDA*.
- RBFS may need to re-expand forgotten nodes to re-create best-path.

Memory-Bounded Search

- In a sense, both IDA* and RBFS use too little memory.
  - Between iterations, IDA* maintains only one number, the current *f*-limit (*currentCutoff*).
  - RBFS maintains more, but uses only linear space: if more space were available, it would not benefit from it.
- It seems reasonable to use all the memory available — the more, the better.
- We'd like a memory-bounded version of A*.

Simplified Memory-Bounded A* (SMA*)



- Idea: Run A* as normal until memory is full. Then replace something in memory with newly generated nodes.
- SMA*:
  - When memory is full, drop the worst leaf — node with highest *f*-value.
  - Like RBFS, SMA* backs up *f*-value of this forgotten node to it's parent, so we know when to go back to it.
  - If all descends of a node *n* are forgotten, we don't know which way to go from *n*, but we know if it 's worth re-exploring *n*.

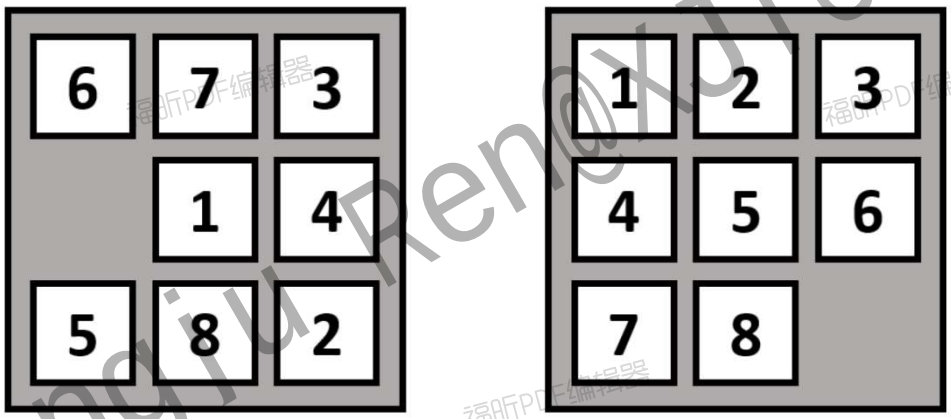
Simplified Memory-Bounded A* (SMA*)

- **Problem**: What if many nodes have the same f-value?
- **Solution**: delete the **oldest** and expand the **newest**.

- SMA* works as long as there is **enough memory for the complete optimal path.**
- If not, SMA* needs to switch **continuously between** candidate paths.
- Causes a similar problem to **thrashing** in disk paging systems.

Admissible Heuristics for 8-Puzzle

- $h1$ = the number of misplaced tiles.
- $h2$ = the sum of Manhattan distances of the tiles from their goal positions.



$$h_1(s_0) = 6$$

$$h_2(s_0) = 2 + 3 + 0 + 2 + 2 + 3 + 3 + 0 = 15$$

Performance of Heuristic

Definition

For two admissible heuristics h_1 and h_2, h_2 dominates h_1 **iff** $\forall n, h_2(n) \geq h_1(n)$.

Theorem: A* using h_2 never expands more nodes than using h_1.

- Every node with $f(n) < C^*$ is expanded.
- Every node with $h(n) < C^* - g(n)$ is expanded.
- $|\{n|\ h_2(n) < C^* - g(n)\}| \leq |\{n|\ h_1(n) < C^* - g(n)\}|$

□

- **Given any admissible heuristics *ha* and *hb*, *h* = max(*ha*, *hb*) is also admissible and dominates *ha* and *hb*.**

Effective Branching Factor

- One way to characterize the quality of a heuristic is effective branching factor b^*.
  - Total number of nodes generated by A*: N
  - Solution depth: d

$$N + 1 = 1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$

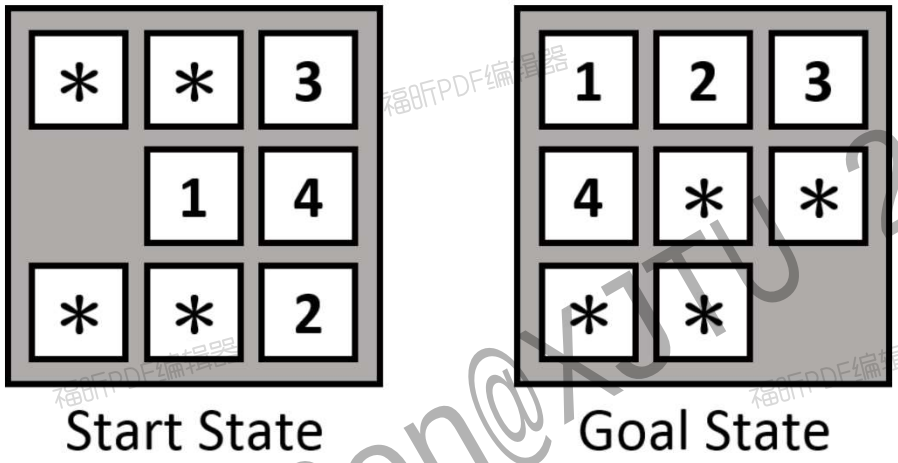
- A well-designed heuristic would have a value of b^* close to 1.

Depth	Nodes generated			Effective branching factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24

Generating Heuristic from Relaxed Problems

- Admissible heuristics can be derived from exact solution cost to a relaxed version of the problem.
- In 8-puzzle, *h*1 is derived from that a tile can move to anywhere in one step.
- In 8-puzzle, *h*2 is derived from that a tile can move to any adjacent square in one step.
- Key: The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the original problem.

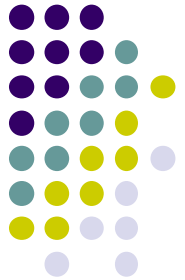
Generating Heuristic from Sub-problems



Start State Goal State

- **Admissible** heuristic can also be derived from a **subproblem**.
- **Pattern databases** store exact solution costs for every possible subproblem instances.
  - For example, every possible position of 1-2-3-4 and the blank.
- Can we use the costs of 1-2-3-4 and 5-6-7-8?
  - Simple addition breaks the **admissibility**.
  - How about count only those moves involving 1-2-3-4?
  - Then the addition is still **admissible**.
  - This is the idea behind **disjoint pattern databases**.

Generating Heuristic from Experience



- Convert a state into the feature domain.
- Feature $f1(n)$: "number of misplaced tiles".
- Feature $f2(n)$: "number of pairs of adjacent tiles that are not adjacent in the goal state".
- Both $f1(goal) = 0$ and $f2(goal) = 0$.
- $h(n) = c1\,f1(n) + c2\,f2(n)$ with $c1 > 0$, $c2 > 0$ (why?).
- We could take randomly generated 8-puzzle and gather statistics to decide constants.
- No guarantee to be admissible or consistent.

Summary

- Heuristic functions estimate costs of shortest paths.
- Good heuristics can dramatically reduce search cost.
- Greedy best-first search expands lowest *h*.
  - In general not complete nor optimal.
- A* search expands lowest *g* + *h*.
  - Optimal when *h* is admissible (and consistent).
- Memory limitation is an important issue to heuristic search. Search with forgetting and re-expanding are the keys, but still suffers from different conditions.
- A more efficient heuristic can be generated from several admissible heuristics.
- Admissible heuristics can be derived from relaxed problems, subproblems, and experience.