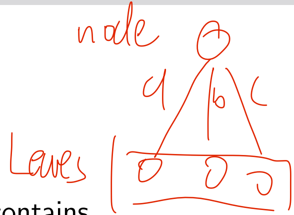
Fundamentals of Artificial Intelligence
Exercise 2a: Uninformed Search

Sebastian Mair

Technical University of Munich

November 3th, 2023

Recap: Solving Problems by Searching



Search problem: defined by

- state space
- initial state
- actions
- transition model
- goal test
- action cost

Search tree: contains

- root (initial state)
- branches (actions)
- nodes (reached states)
- leaves (unexpanded nodes)

Recap: Solving Problems by Searching

The search algorithms use:

- **frontier**: all leaf nodes available for expansion. **FIFO**, **priority queue**, or **LIFO**.
- **reached** set/lookup table: all states that have been visited.

Recap: Performance Measures

4 Criteria for measuring problem-solving performance:

- **Completeness**: Is it guaranteed that the algorithm finds a solution if one exists?
- **Optimality**: Does the strategy find the optimal solution (minimum costs)?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

Recap: Classifying Search Algorithms

Tree-like Search vs. Graph-Search

- **Tree-like search**: Revisiting states is possible
- **Graph search**: Remember visited states in the *reached* set/lookup table. No revisiting of states → usually better time complexity

Uninformed vs. Informed Search

- **Uninformed Search**: Searching "blindly", i.e., no additional information apart from the provided problem definition (e.g., BFS, UCS, DFS, ...)
- **Informed Search**: Estimate how promising a state is to reach the goal using heuristic functions (e.g., GBFS, A*, ...)

Recap: Breadth-First Search

function Breadth-First-Search (problem) **returns** a solution or failure

```
node ← Node(State=problem.Initial-State, Path-Cost=0)
if problem.Is-Goal(node.State) then return Solution(node)
frontier ← a FIFO queue with node as the only element
reached ← {node.State}
loop do
  if Is-Empty(frontier) then return failure
  node ← Pop(frontier) // chooses a shallowest node in frontier
  for each child in Expand(problem, node) do
    s ← child.State
    if problem.Is-Goal(s) then return Solution(child)
    if s is not in reached then
      add s to reached
      frontier ← Add(child,frontier)
```

Recap: Uniform-Cost Search: $f(n) = g(n) = \texttt{n.PATH-COST}$

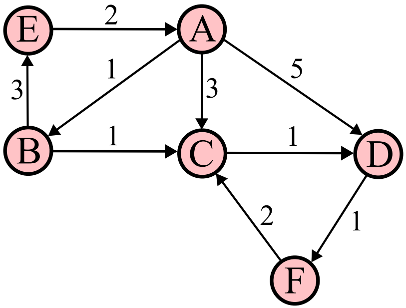
function Best-First-Search (problem, f) **returns** a solution or failure

$node \leftarrow$ `Node(State=`*problem*`.Initial-State,Path-Cost=0)`
$frontier \leftarrow$ a priority queue ordered by f, with *node* as the only element
$reached \leftarrow$ a lookup table, with one entry ($node$.`State` $\rightarrow$ $node$)
loop do
  if `Is-Empty`(*frontier*) **then return** failure
  $node \leftarrow$ `Pop`(*frontier*) // chooses the node n with minimum $f(n)$ in *frontier*
  if *problem*.`Is-Goal`(*node*.`State`) **then return** `Solution`(*node*)
  for each *child* **in** `Expand`(*problem*,*node*) **do**
    $s \leftarrow$ *child*.`State`
    if s is not in *reached* **or** *child.Path-Cost* $<$ *reached*[s].`Path-Cost` **then**
      reached[s] $\leftarrow$ *child*
      $frontier \leftarrow$ `Add`(*child*,*frontier*)

Problem 2.1: Search Algorithms Basics

- Start: A, goal: F
- Action costs are shown on the arcs
- If there is no clear preference, we visit the state next that is first alphabetically.

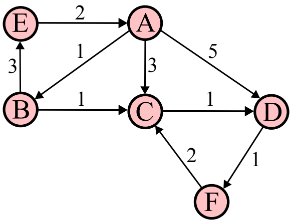
Tasks: perform Breadth-First, Depth-First and Uniform-Cost search.



Problem 2.1: Search Algorithms Basics

Breadth-First: BFS FIFO

Node	Frontier	Reached
A	B C D	∅
B	C D E	A
C	D E	A B
D	E F	A B C
E	F	A B C D



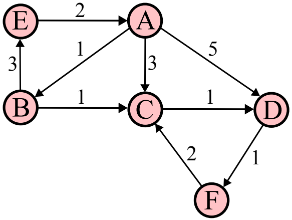
What is the result?
Today's tweedback code: **zjqb** (twbk.de/zjqb)

Problem 2.1: Search Algorithms Basics

Breadth-First: BFS FIFO

Node	Frontier	Reached
-	A(-)	A(-) B(A) C(A) D(A) E(B)
A(-)	B(A) C(A) D(A)	
B(A)	C(A) D(A) E(B)	
C(A)	D(A) E(B)	
D(A)	E(B)	

⟹ goal is landed

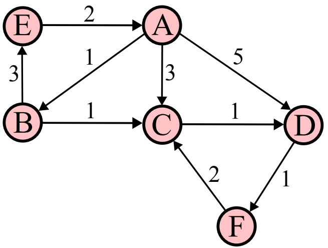


What is the result?
Today's tweedback code: **zjqb**
(twbk.de/zjqb)

Problem 2.1: Search Algorithms Basics

Depth-First: LIFO

Node	Frontier
–	A(–)
A(–)	B(A) C(A) ~~D(A)~~
D(A)	B(A) C(A) ~~F(D)~~
F(D)	⇒ goal

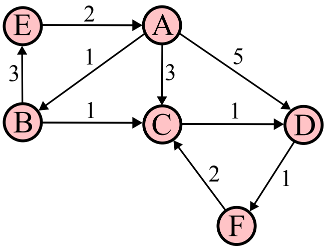


What is the result?
Today's tweedback code: **zjqb**
(twbk.de/zjqb)

Problem 2.1: Search Algorithms Basics

Depth-First:

Node	Frontier
–	A(–)
A(–)	D(A) C(A) B(A)
B(A)	D(A) C(A) E(B) ~~C(B)~~
C(B)	D(A) C(A) E(B) D(C)
D(C)	D(A) C(A) E(B) F(D)
F(D)	=> goal



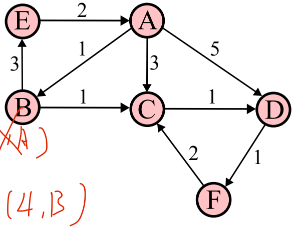
What is the result, if we reverse the order of visiting child nodes and check for cycles of length 2?

Today's tweedback code: **zjqb**
(twbk.de/zjqb)

Problem 2.1: Search Algorithms Basics

Uniform-Cost:

Node	Children	Frontier	Reached
–	–	A(0)	A(0)
A(0)	B(1,A) C(3,A) D(5,A)	~~B(1,A)~~ C(3,A) D(5,A)	B(1,A) ~~C(3,A)~~ D(~~5,A~~) C(2,B) E(4,B)
B(1,A)	C(2,B) E(4,B)	~~C(2,B)~~ E(4,B) D(5,A)	D(3,C)
C(2,B)	D(3,C)	~~D(3,C)~~ E(4,B)	F(4,D)
D(3,C)	F(4,D)	~~E(4,B)~~ F(4,D)	
E(4,B)	A(6,E)	F(4,D) A(6,E)	
F(4,D)			



What is the result?
Today's tweedback code: **zjqb**
(twbk.de/zjqb)

Problem 2.2: Application of Search Algorithms: Transport

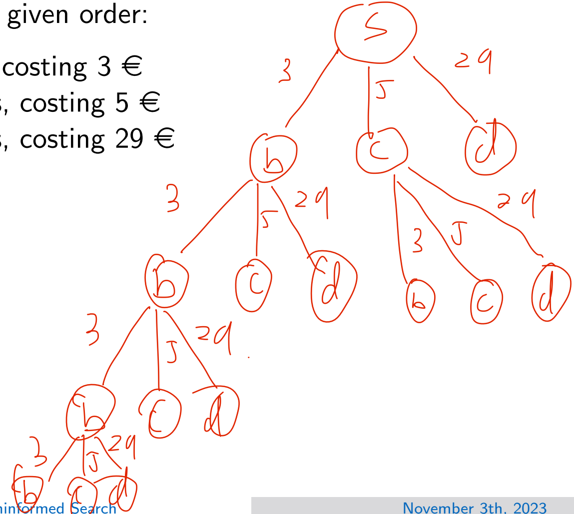
I have bought 120 bricks at the hardware store, which I need to transport. I have the following means of transport, which I consider in the given order:

1. **bus (*b*)** – I can carry up to 30 bricks, costing 3 €
2. **car (*c*)** – it can carry up to 100 bricks, costing 5 €
3. **delivery (*d*)** – it delivers all the bricks, costing 29 €

Problem 2.2: Application of Search Algorithms: Transport

I have bought 120 bricks at the hardware store, which I need to transport. I have the following means of transport, which I consider in the given order:

1. **bus** (*b*) – I can carry up to 30 bricks, costing 3 €
2. **car** (*c*) – it can carry up to 100 bricks, costing 5 €
3. **delivery** (*d*) – it delivers all the bricks, costing 29 €



Let us define the search problem:

- state space
- initial state
- actions
- transition model
- goal test
- path cost

Problem 2.2.2:

Uniform

Assuming I want to minimize the cost, which search method should I use and what is the resulting solution?

120 bricks, following actions:

1. **bus (*b*)** – 30 bricks, 3 €
2. **car (*c*)** – 100 bricks, 5 €
3. **delivery (*d*)** – all bricks, 29 €

Today's tweedback code: **zjqb** (twbk.de/zjqb)

Node	Frontier	Reached
–	s(0,-)	s(0,-)
s(0,-)	b(~~3~~,s) c(5,s) d(29,s)	b(3,s) c(5,s) d(29,s)
b(3,s)	c(5,s) b(6,b) c(8	

Problem 2.2.1:

BFC

Assuming I want to make as few trips as possible, which search method should I use and what is the resulting solution?

120 bricks, following actions:

1. **bus** (b) – 30 bricks, 3 €
2. **car** (c) – 100 bricks, 5 €
3. **delivery** (d) – all bricks, 29 €

Today's tweedback code: **zjqb** (twbk.de/zjqb)

Node	Frontier	Reached set
–	s(–)	s(–)
s(–)	b(s) c(s) d(s)	b(s) c(s) d(s)
	goal	d

s → d

Problem 2.2.3:

Perform depth-first search.

120 bricks, following actions:

1. **bus (*b*)** – 30 bricks, 3 €
2. **car (*c*)** – 100 bricks, 5 €
3. **delivery (*d*)** – all bricks, 29 €

Today's tweedback code: **zjqb** (twbk.de/zjqb)

Problem 2.2.4:

Perform depth-first search again, but now assume I explore actions in the order: delivery, car, bus.

120 bricks, following actions:

1. **bus (*b*)** – 30 bricks, 3 €
2. **car (*c*)** – 100 bricks, 5 €
3. **delivery (*d*)** – all bricks, 29 €

Today's tweedback code: **zjqb** (twbk.de/zjqb)