



ASA 8

● BACKTRACKING, DIJKSTRA, A-STAR

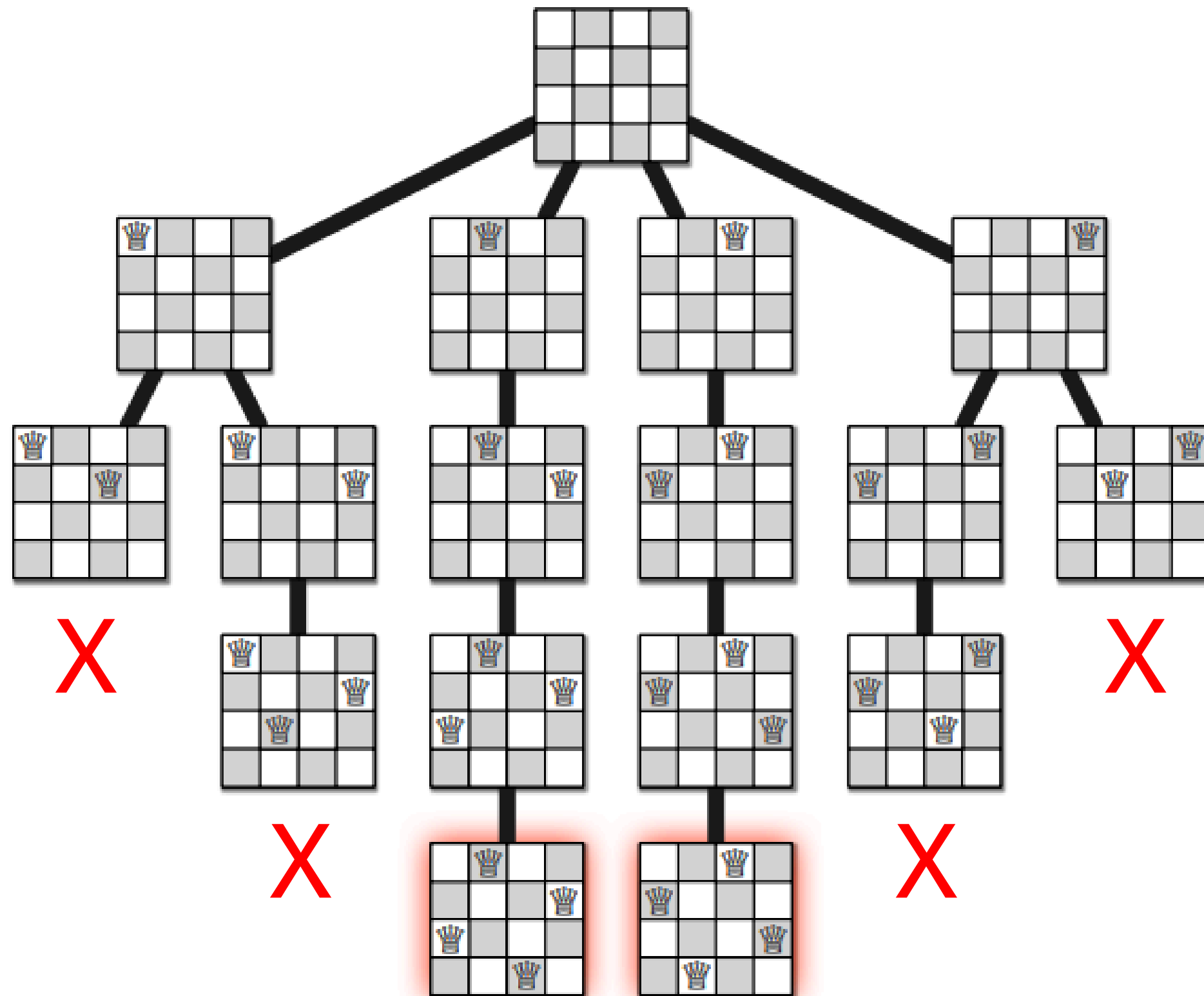


BACKTRACKING

Backtracking adalah teknik algoritma yang digunakan untuk memecahkan masalah secara rekursif dengan mencoba membangun solusi secara bertahap, dan jika ditemukan bahwa solusi tersebut gagal pada suatu titik, maka algoritma akan backtrack (mundur) dan mencoba alternatif lain.

Note: Alur dari penjabaran pohon pada backtracking seperti DFS, dikarenakan menggunakan rekursif. DFS dan backtracking juga bersifat sama, yaitu menjelajahi satu path terlebih dahulu, jika mencapai end-state, jelajahi path lainnya.

CONTOH: N-QUEEN




Karena kita tahu bahwa pada satu baris hanya bisa ditempati oleh 1 Queen, maka cukup lakukan percabangan untuk setiap baris. Setiap meletakkan ratu, cek jika posisi valid lanjutkan percabangan, jika tidak berhenti (end state)

KODE N-QUEEN



```
def is_safe(board, row, col):  
    # Cek kolom di atas  
    for i in range(row):  
        if board[i] == col:  
            return False  
    # Cek diagonal kiri atas  
    if abs(board[i] - col) == abs(i - row):  
        return False  
    return True
```

KODE N-QUEEN



```
def backtrack(row):
    if row == n:
        # Konversi solusi dalam bentuk papan
        solution = []
        for i in range(n):
            line = ['.'] * n
            line[board[i]] = 'Q'
            solution.append(''.join(line))
        hasil.append(solution)
        return

    for col in range(n):
        if is_safe(board, row, col):
            board[row] = col # Tempatkan ratu
            backtrack(row + 1)
            board[row] = -1 # Backtrack
```

DJIKSTRA

Algoritma Dijkstra adalah algoritma **greedy** yang digunakan untuk mencari jarak terpendek dari sebuah simpul sumber (source) ke semua simpul lain dalam graf berbobot tak-negatif.

Algoritma Dijkstra tidak cocok digunakan untuk graph yang memiliki bobot negatif.

CARA KERJA DIJKSTRA

1. Inisialisasi jarak (dist) node sumber dengan 0, dan semua node sisanya dengan tak hingga, juga inisialisasi priority queue (minimum binary heap) dengan (0, node_awal).
2. Pilih simpul yang belum dikunjungi dengan jarak terkecil.
3. Perbarui jarak ke semua tetangga simpul itu jika ditemukan jarak yang lebih pendek.
4. Tandai simpul sebagai sudah dikunjungi.
5. Ulangi sampai semua simpul sudah dikunjungi atau jarak terkecil berikutnya adalah ∞ .

KODE DIJKSTRA

```
import heapq

def dijkstra(graph, start):
    # graph: dict {node: [(neighbor, weight), ...]}
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    queue = [(0, start)]

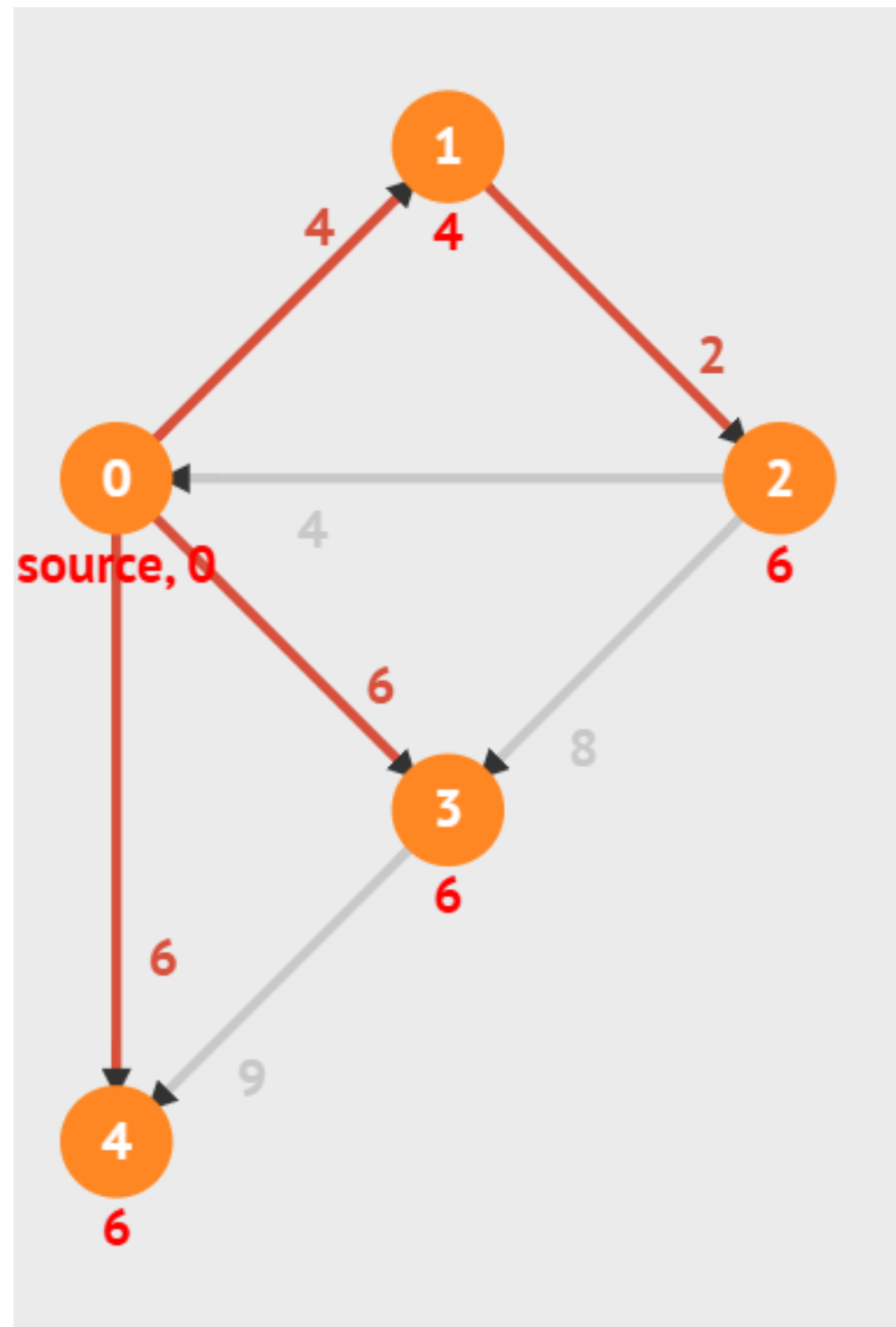
    while queue:
        current_distance, current_node = heapq.heappop(queue)

        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node]:
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(queue, (distance, neighbor))

    return distances
```


ILUSTRASI DIJKSTRA



Untuk ilustrasi step-by-step
dari Dijkstra dapat dilihat
melalui website
<https://visualgo.net/en/sssp>

A-STAR

A* adalah algoritma pencarian jalur terpendek yang sangat efisien. Algoritma ini merupakan kombinasi dari algoritma Dijkstra dan penambahan heuristic. Dengan menggunakan fungsi evaluasi $f(n)$, A* dapat mencari jalur terpendek dari sumber ke tujuan secara lebih cepat dibandingkan dengan Dijkstra, terutama jika kita hanya ingin menemukan jalur ke satu target (karena Dijkstra mencari jarak terpendek menuju semua node, sedangkan A* fokus pada 1 node tujuan).

A-STAR

Algoritma A* menggunakan fungsi evaluasi $f(n)$ untuk memprioritaskan node yang akan dieksplorasi. Fungsi ini menggabungkan dua komponen, yaitu $f(n) = g(n) + h(n)$ dimana $g(n)$ adalah biaya (jarak) yang sudah ditempuh dari sumber menuju node n , sedangkan $h(n)$ adalah heuristic atau perkiraan biaya (jarak) yang tersisa dari node n menuju tujuan (goal).

FUNGSI HEURISTIK

Heuristic adalah perkiraan terhadap biaya perjalanan dari node saat ini ke tujuan. Pilihan heuristic yang tepat sangat mempengaruhi efisiensi A*. Contoh heuristic yang umum:

- Euclidean distance (jarak garis lurus)
- Manhattan distance (jarak berbasis grid, hanya horizontal dan vertikal)

CARA KERJA A-STAR

1. Inisialisasi jarak (dist) node sumber dengan 0, dan semua node sisanya dengan tak hingga, juga inisialisasi priority queue (minimum binary heap) dengan $(0, \text{node_awal})$.
2. Pilih node dengan nilai $f(n)$ terkecil untuk dieksplorasi.
3. Untuk setiap tetangga dari node yang sedang dieksplorasi, hitung nilai $f(n)$ berdasarkan $g(n)$ dan $h(n)$.
4. Jika jalur baru lebih pendek dari yang sebelumnya, perbarui nilai $g(n)$ dan $h(n)$.
5. Ulangi sampai node tujuan ditemukan atau tidak ada jalur yang tersisa.

CONTOH KODE A-STAR

```
import heapq
import math


# Fungsi untuk menghitung jarak Euclidean (heuristik) antara dua titik (lintang, bujur)
def euclidean_distance(p1, p2):
    lat1, lon1 = p1
    lat2, lon2 = p2
    return math.sqrt((lat1 - lat2) ** 2 + (lon1 - lon2) ** 2)

# Fungsi A* untuk mencari jalur terpendek
def a_star(start, goal, graph):
    open_set = [(0 + euclidean_distance(start, goal), 0, start)] # (f_score, g_score, node)
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current_g, current = heapq.heappop(open_set) # Ambil node dengan f_score terkecil

        if current == goal:
            # Jika kita sudah sampai goal, kembalikan jalur yang ditempuh
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1] # Balikkan jalur agar dari start ke goal
```

CONTOH KODE A-STAR



```
# Mengeksplorasi tetangga dari node current (indentasi setara dengan if current == goal)
for neighbor, weight in graph[current]:
    tentative_g = current_g + weight # Menggunakan berat (biaya perjalanan antar kota)

    if neighbor not in g_score or tentative_g < g_score[neighbor]:
        came_from[neighbor] = current
        g_score[neighbor] = tentative_g
        f_score = tentative_g + euclidean_distance(neighbor, goal)
        heapq.heappush(open_set, (f_score, tentative_g, neighbor))

return None # Tidak ada jalur ditemukan
```

CONTOH KODE A-STAR

```
# Koordinat (latitude, longitude) untuk beberapa kota
city_coords = {
    'A': (1.0, 1.0),
    'B': (1.0, 2.0),
    'C': (2.0, 2.0),
    'D': (3.0, 2.0),
    'E': (2.0, 1.0),
}

# Graph berbentuk dictionary yang menunjukkan kota dan tetangganya dengan biaya perjalanan (dalam km)
graph = {
    'A': [('B', 1), ('E', 1.5)],
    'B': [('A', 1), ('C', 1.5)],
    'C': [('B', 1.5), ('D', 2)],
    'D': [('C', 2)],
    'E': [('A', 1.5), ('C', 1)],
}

# Koordinat start dan goal
start = 'A'
goal = 'D'

# Menjalankan A* untuk mencari jalur terpendek dari A ke D
path = a_star(start, goal, graph)
print("Jalur terpendek dari {} ke {}".format(start, goal), path)
```


THANK YOU



● FOR YOUR NICE ATTENTION

link canva: [https://www.canva.com/design/DAGiKI7GW8/wXE9kZO88PUo-iWzQUyCAQ/edit?
utm_content=DAGiKI7GW8&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton](https://www.canva.com/design/DAGiKI7GW8/wXE9kZO88PUo-iWzQUyCAQ/edit?utm_content=DAGiKI7GW8&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)