ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
VNUHCM - UIT

ĐỒ ÁN MÔN HỌC

Xây dựng hệ thống minh họa thuật toán cho môn học: Cấu trúc dữ liệu và Giải thuật

Thành phố Hồ Chí Minh – 12/2025

Môn học CS106- Trí tuệ nhân tạo

GVHD: PGS Nguyễn Đình HIển

Lớp LT.K2025.2

Nhóm 13

Bùi Tấn Thiện Tâm - 25410127

Lê Ngọc Thái - 25410130

Kiêm Hoàng Tuấn - 25410153

Tóm tắt

Đồ án xây dựng một hệ thống minh họa thuật toán dành cho môn Cấu trúc dữ liệu và Giải thuật (DSA). Hệ thống cho phép người học nhập dữ liệu, chạy thuật toán theo từng bước (step-by-step), tua timeline, quan sát trạng thái cấu trúc dữ liệu (mảng, cây, đồ thị), đồng thời sinh giải thích và câu hỏi kiểm tra hiểu bài dựa trên vết chạy . Kiến trúc tách biệt phần engine (chạy thuật toán và sinh trạng thái) với phần giao diện (render/animation) giúp dễ mở rộng thêm thuật toán mới. Thành phần AI trong phạm vi đồ án ưu tiên tính "giải thích được" thông qua hệ luật IF-THEN và các chiến lược sinh câu hỏi dựa trên trace.

Chương I. Trình bày nội dung vấn đề cần giải quyết

1.1 Bối cảnh và động cơ

Trong DSA, nhiều thuật toán có cơ chế thay đổi trạng thái theo thời gian: hoán vị phần tử khi sắp xếp, mở rộng biên khi duyệt đồ thị, xoay cây khi cân bằng, hoặc cập nhật bảng khi quy hoạch động. Nếu chỉ đọc mã nguồn hoặc slide tĩnh, người học khó hình dung được "vì sao" từng thao tác được thực hiện.

Algorithm visualization (minh họa thuật toán) giúp chuyển đổi luồng thực thi thành hình ảnh/hoạt hình để người học quan sát và tương tác, từ đó rút ngắn khoảng cách giữa lý thuyết và triển khai.

1.2 Mục tiêu tổng quát

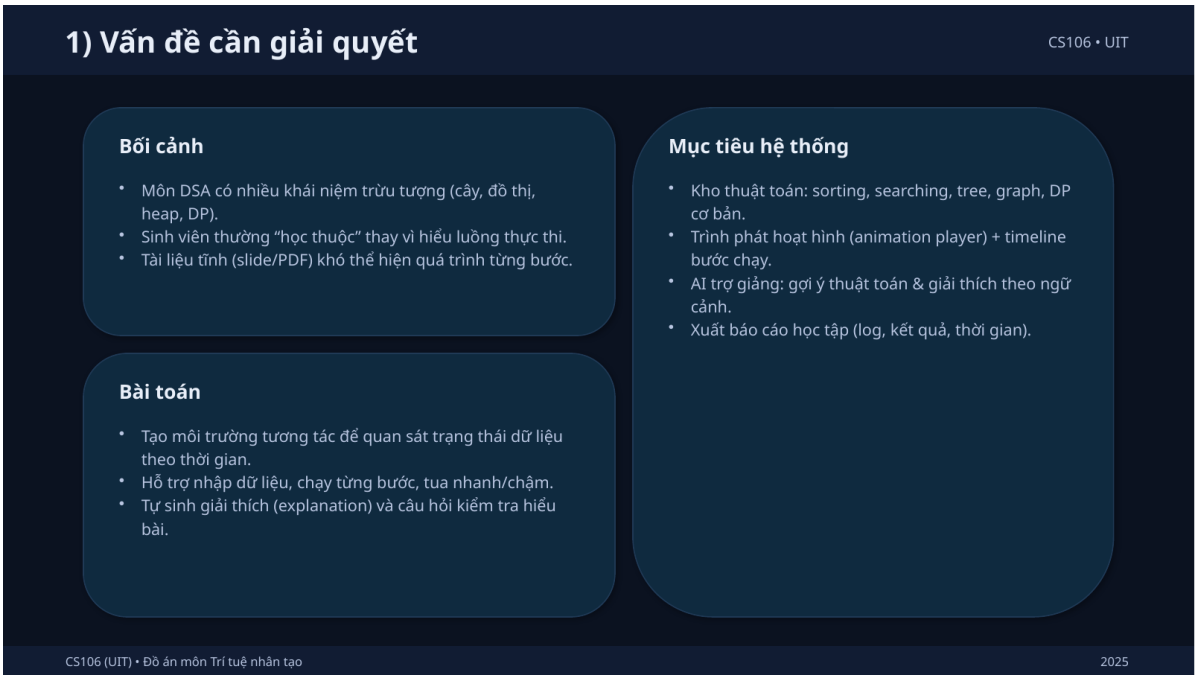
- Minh họa trực quan các thuật toán DSA theo từng bước, có thể tạm dừng/tua lại.
- Hiển thị rõ trạng thái dữ liệu, biến quan trọng và thông điệp giải thích cho mỗi bước.
- Hỗ trợ người học tự kiểm tra qua quiz được sinh từ trace (vết chạy).
- Thiết kế dễ mở rộng: thêm thuật toán mới theo chuẩn plugin mà không sửa lõi UI.

1.3 Phạm vi và đối tượng sử dụng

Đối tượng sử dụng chính là sinh viên đang học môn Cấu trúc dữ liệu và Giải thuật. Ngoài ra hệ thống có thể hỗ trợ trợ giảng/giảng viên minh họa nhanh trên lớp hoặc tạo bài tập tương tác.

1.4 Các ràng buộc thực tế

- Đồ án ở mức môn học: ưu tiên tính đúng đắn, rõ ràng, dễ trình diễn hơn tối ưu cực hạn.
- Mã nguồn (đã có sẵn) cần được mô tả theo cấu trúc tổng quát; chi tiết có thể khác theo code của nhóm.
- Kết quả thử nghiệm trình bày theo format chuẩn, có thể thay bằng số đo thực tế từ máy chạy dự án.



Hình 1.1. Tóm tắt bối cảnh và mục tiêu hệ thống (từ slide trình bày).

Chương II. Cơ sở lý thuyết và giải thuật (mã giả)

2.1 Khái niệm minh họa thuật toán (Algorithm Visualization)

Minh họa thuật toán là quá trình biểu diễn tiến trình của thuật toán bằng hình ảnh/hoạt hình, cho phép quan sát các bước trung gian thay vì chỉ xem input-output. Một hệ thống minh họa tốt thường cung cấp: (i) mô hình trạng thái (state model), (ii) cơ chế phát trạng thái theo bước, (iii) bộ render chuyển state thành khung hình, và (iv) điều khiển tương tác (step, play, tốc độ).

2.2 Mô hình hóa thuật toán thành chuỗi trạng thái (Trace/State)

Ý tưởng cốt lõi của đồ án là tách thuật toán khỏi giao diện bằng cách chuẩn hóa mỗi bước thành một đối tượng trạng thái (state). State nên đủ thông tin để tái tạo hình vẽ tại bước đó: dữ liệu chính (mảng/đồ thị/cây), các con trỏ/nhãn quan trọng (i, j, pivot, queue...), và thông điệp giải thích.

Thành phần	Ví dụ	Vai trò trong minh họa
data	mảng A, danh sách kề	Nội dung chính cần vẽ
highlights	i, j, pivot, visited	Tô màu/nhấn mạnh vị trí quan trọng
aux	stack/queue, dist, parent	Giúp hiểu cấu trúc phụ trợ
message	“swap A[j] và A[j+1]”	Giải thích tại từng bước
meta	step_id, timestamp	Timeline, tái lập vết chạy

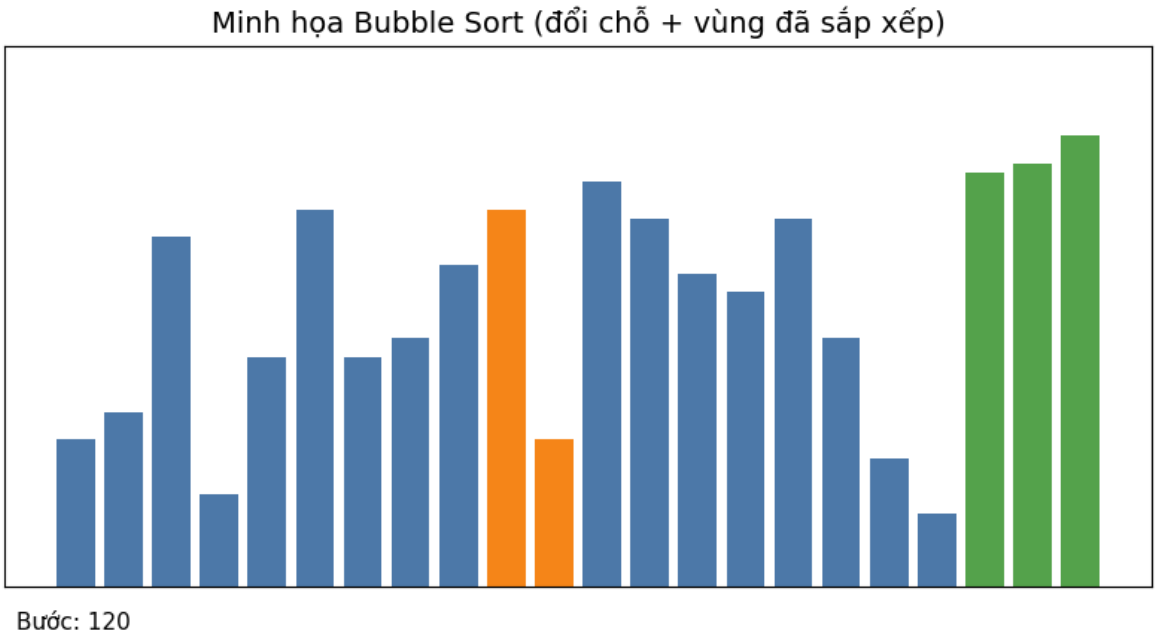
2.3 Các thuật toán minh họa tiêu biểu

2.3.1 Bubble Sort

Bubble Sort lặp lại việc so sánh các cặp phần tử kề nhau và đổi chỗ nếu sai thứ tự. Vùng cuối mảng dần được sắp xếp sau mỗi vòng.

Mã giả Bubble Sort (có emit_state để minh họa)

```
BUBBLE_SORT(A):
  n <- length(A)
  for i <- 0 .. n-1:
    swapped <- false
    for j <- 0 .. n-i-2:
      emit_state(A, compare=(j, j+1), sorted_tail=i)
      if A[j] > A[j+1]:
        swap(A[j], A[j+1])
        swapped <- true
        emit_state(A, swap=(j, j+1), sorted_tail=i)
    if swapped == false:
      break
  return A
```



Hình 2.1. Animation Bubble Sort (GIF minh họa).

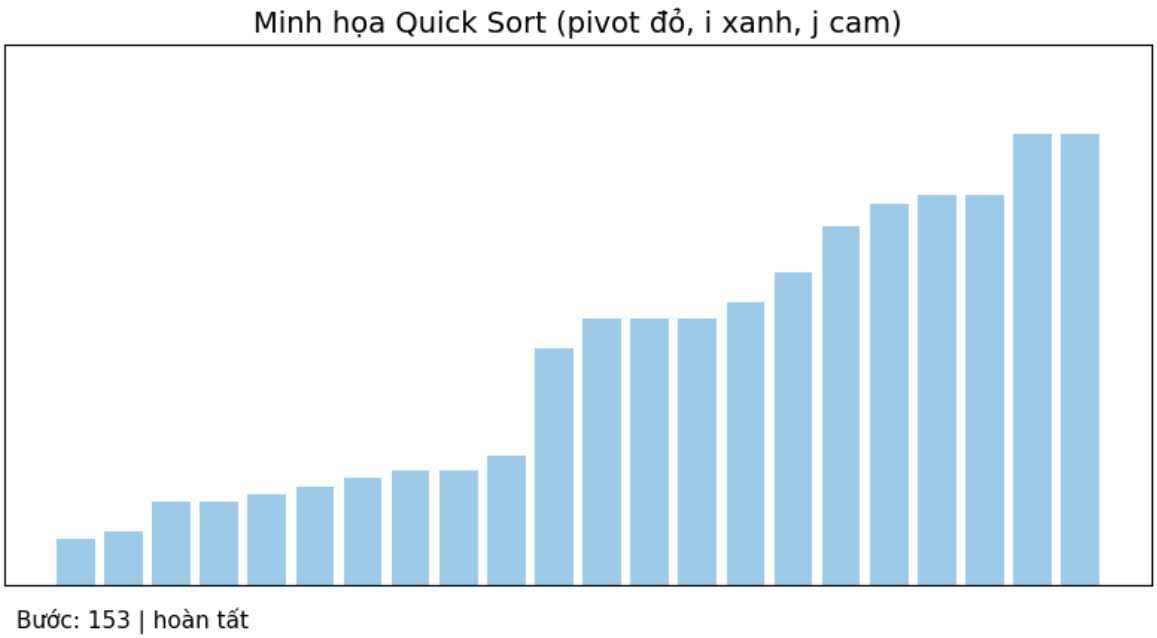
2.3.2 Quick Sort

Quick Sort chọn một pivot và phân hoạch mảng thành hai phần: <= pivot và > pivot, sau đó đệ quy.

Mã giả Quick Sort (phân hoạch)

```
QUICK_SORT(A, lo, hi):
  if lo >= hi: return
  p <- PARTITION(A, lo, hi)
  QUICK_SORT(A, lo, p-1)
  QUICK_SORT(A, p+1, hi)

PARTITION(A, lo, hi):
  pivot <- A[hi]
  i <- lo
  for j <- lo .. hi-1:
    emit_state(A, lo, hi, i, j, pivot_idx=hi)
    if A[j] <= pivot:
      swap(A[i], A[j]); i <- i+1
      emit_state(A, swap=(i-1, j), pivot_idx=hi)
  swap(A[i], A[hi])
  emit_state(A, pivot_placed=i)
  return i
```



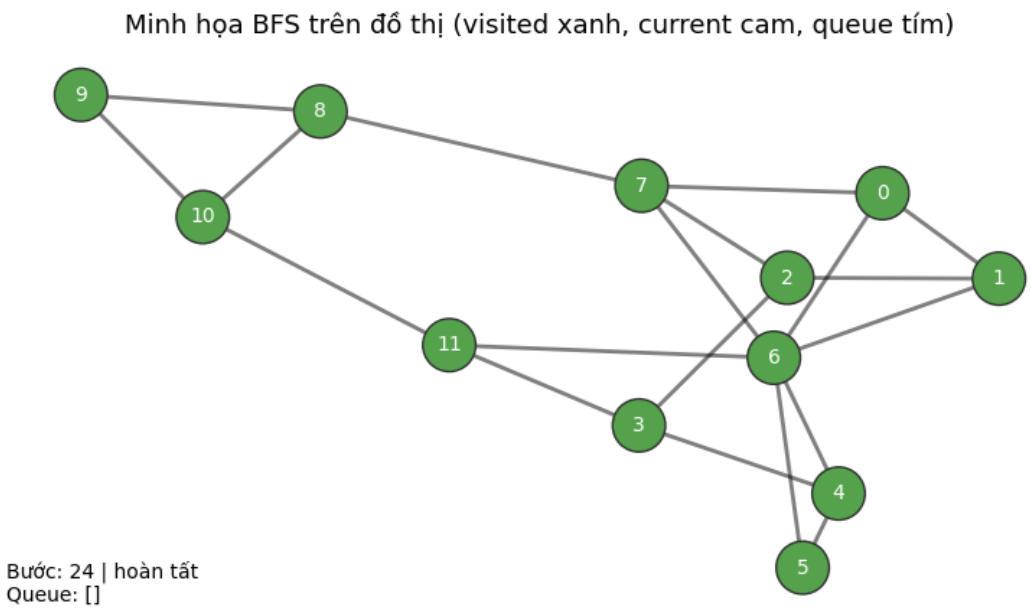
Hình 2.2. Animation Quick Sort (pivot, i, j).

2.3.3 Breadth-First Search (BFS)

BFS duyệt đồ thị theo từng lớp (level) bằng hàng đợi (queue). BFS cho đường đi ngắn nhất trên đồ thị không trọng số.

Mã giả BFS (có trace queue và visited)

```
BFS(G, s):
  for v in V(G): visited[v] <- false
  Q <- empty queue
  visited[s] <- true; enqueue(Q, s)
  while Q not empty:
    u <- dequeue(Q)
    emit_state(visited, Q, current=u)
    for each neighbor v of u:
      if visited[v] == false:
        visited[v] <- true
        parent[v] <- u
        enqueue(Q, v)
        emit_state(visited, Q, discovered=v)
  return parent
```



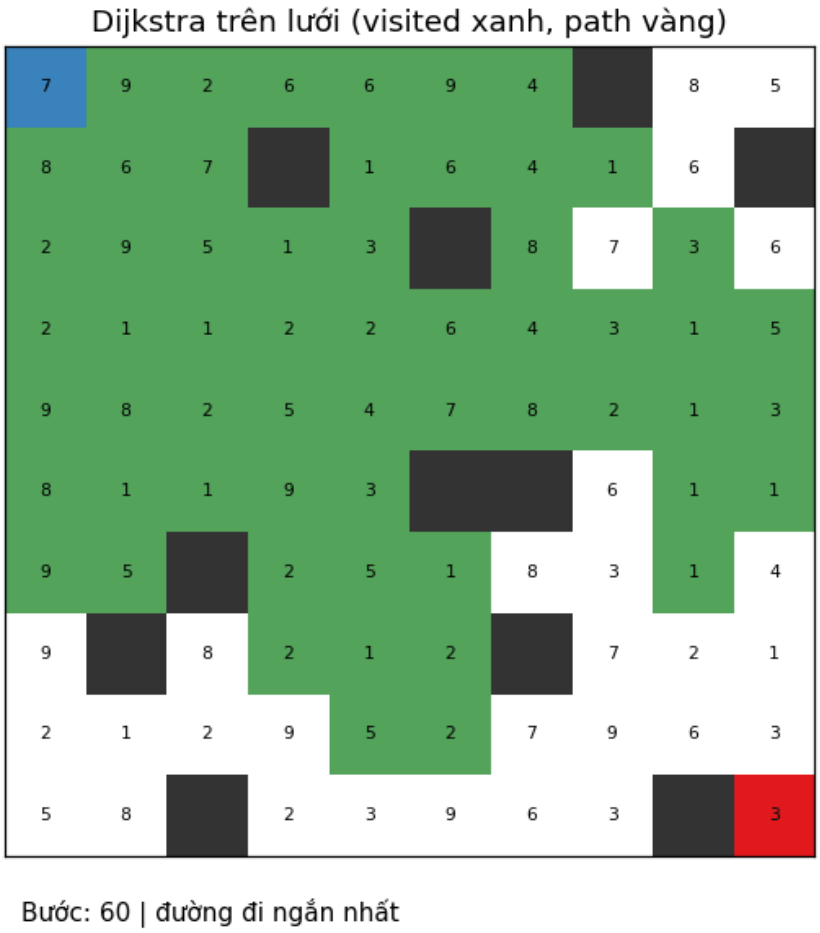
Hình 2.3. Animation BFS trên đồ thị (visited, current, queue).

2.3.4 Dijkstra

Dijkstra tìm đường đi ngắn nhất từ một nguồn đến các đỉnh khác trên đồ thị có trọng số không âm, sử dụng priority queue để chọn đỉnh có dist nhỏ nhất chưa chốt.

Mã giả Dijkstra (settle/relax)

```
DIJKSTRA(G, s):
  for v in V(G): dist[v] <- +inf; parent[v] <- NIL
  dist[s] <- 0
  PQ <- priority queue of (dist, v)
  push(PQ, (0, s))
  while PQ not empty:
    (d, u) <- pop_min(PQ)
    if u is settled: continue
    settle(u)
    emit_state(dist, parent, u, action="settle")
    for each edge (u, v, w):
      if dist[v] > dist[u] + w:
        dist[v] <- dist[u] + w
        parent[v] <- u
        push(PQ, (dist[v], v))
        emit_state(dist, parent, v, action="relax")
```



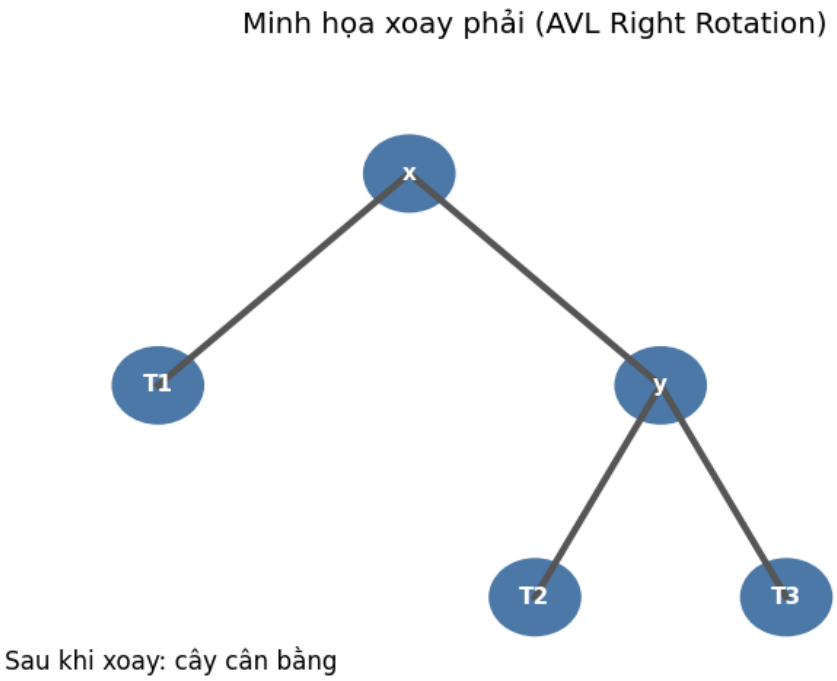
Hình 2.4. Animation Dijkstra trên lưới (visited và path).

2.3.5 AVL Tree Rotation

AVL tree là cây tìm kiếm nhị phân tự cân bằng, đảm bảo độ cao O(log n) thông qua phép xoay trái/phải. Minh họa xoay giúp người học hiểu cách tái cấu trúc cây khi mất cân bằng.

Mã giả xoay phải (Right Rotation)

```
RIGHT_ROTATE(y):
  x <- y.left
  T2 <- x.right
  // perform rotation
  x.right <- y
  y.left <- T2
  // update heights
  update_height(y)
  update_height(x)
  emit_state(tree, rotated=(y,x))
  return x
```

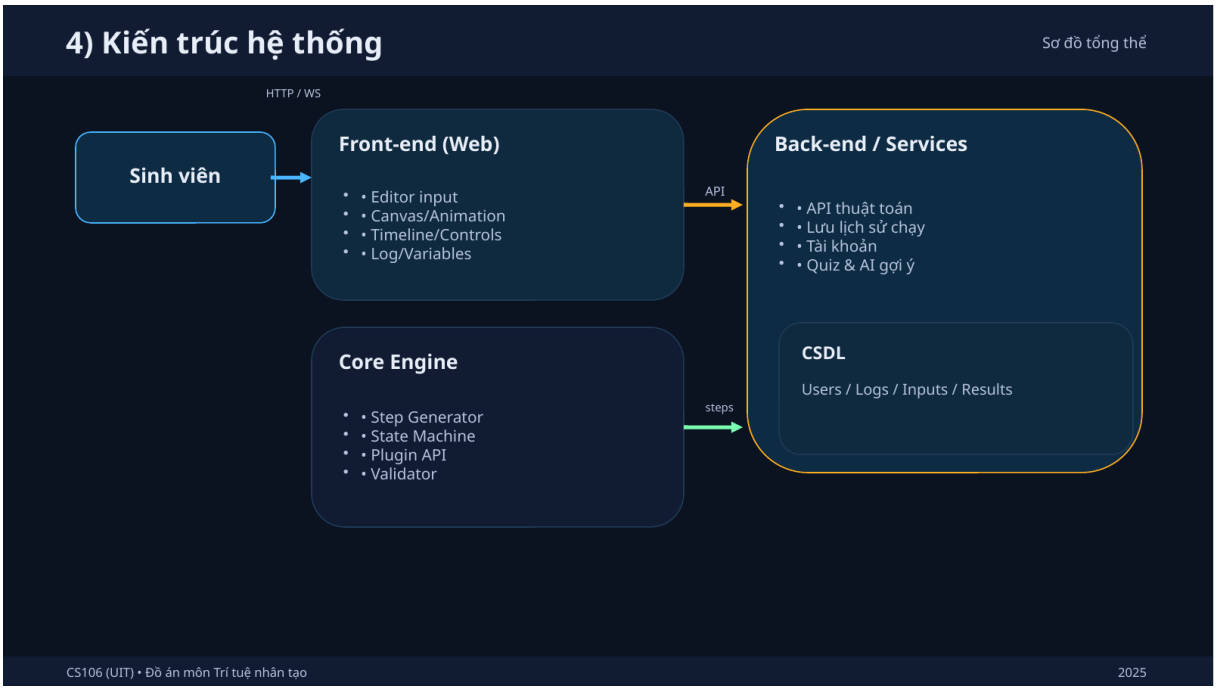


Hình 2.5. Animation xoay phải AVL (trước/sau).

Chương III. Vẽ sơ đồ kiến trúc hệ thống

3.1 Kiến trúc tổng thể

Kiến trúc hệ thống được thiết kế theo nguyên tắc Separation of Concerns: Engine chịu trách nhiệm chạy thuật toán và sinh chuỗi state; UI chỉ tập trung render/animation; Back-end cung cấp lưu trữ, tài khoản, sinh quiz và (tùy chọn) gợi ý AI.



Hình 3.1. Sơ đồ kiến trúc tổng thể của hệ thống.

3.2 Luồng dữ liệu (Data Flow)

Luồng xử lý chuẩn: Input -> Parse/Validate -> Engine tạo trace -> UI render -> Sinh giải thích/quiz -> Lưu log. Thiết kế này cho phép tái lập (replay) một lần chạy và phục vụ debug/đánh giá.

Hình 3.2. Luồng xử lý từ input đến animation/quiz.

Chương IV. Phương pháp thực hiện và thiết kế hệ thống

4.1 Phân tích yêu cầu

Các yêu cầu được chia thành hai nhóm: chức năng và phi chức năng.

Hình 4.1. Yêu cầu chức năng và phi chức năng.

4.2 Thiết kế giao diện người dùng

UI được thiết kế theo mô hình 3 vùng: (i) thanh chọn thuật toán + nhập dữ liệu, (ii) canvas minh họa, (iii) timeline/trace và khối giải thích/quiz.

Hình 4.2. Mockup bố cục giao diện.

4.3 Thiết kế engine theo plugin

Để dễ mở rộng, mỗi thuật toán được đóng gói thành một plugin có giao diện chuẩn: parseInput, generator, renderModel, quiz. Generator trả về chuỗi state (iterator) để UI có thể play/step.

Giao diện plugin đề xuất

```
interface AlgorithmPlugin {
  id: string
  name: string
  parseInput(raw): Input
  generator(input): Iterable<State>
  toRenderModel(state): RenderModel
  quiz(state): Question[]
}
```

4.4 Thiết kế dữ liệu và lưu trữ

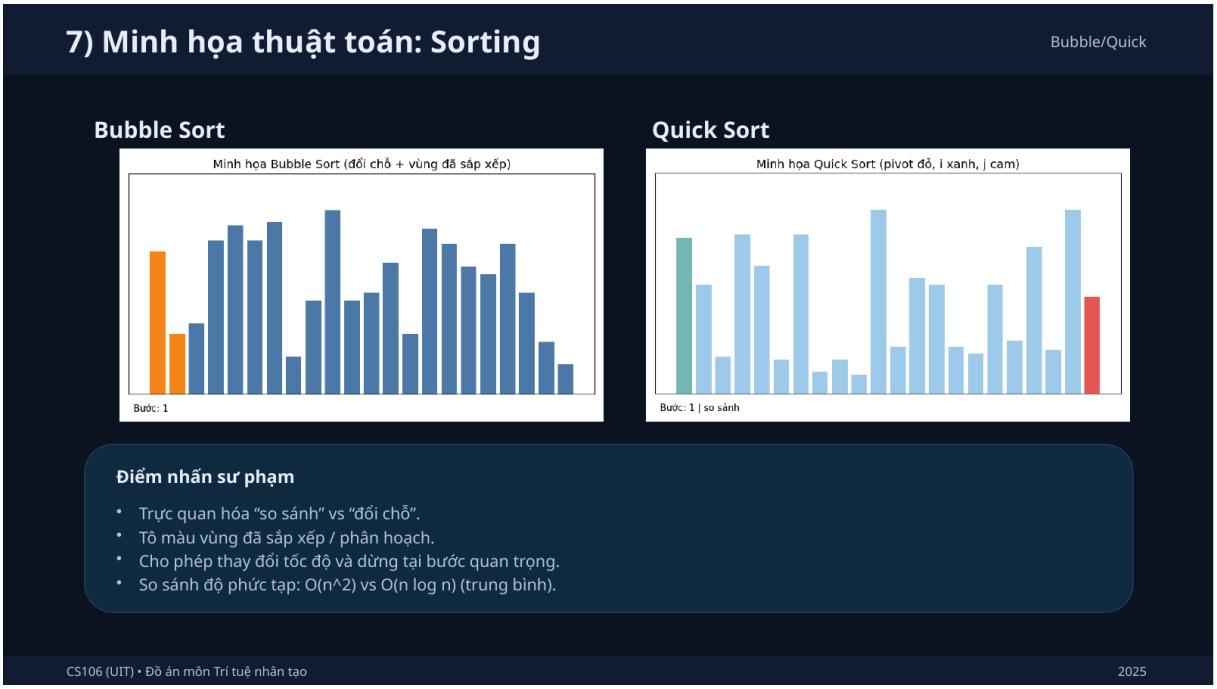
Tối thiểu, hệ thống lưu: tài khoản người dùng, lịch sử chạy (input + thuật toán + seed), kết quả quiz và log tương tác. Có thể triển khai bằng SQLite/PostgreSQL hoặc lưu file JSON.

Bảng/Collection	Thuộc tính chính	Ghi chú
Users	user_id, name, email, role	role: student/teacher
Runs	run_id, user_id, algo_id, input, created_at	lưu input & tham số
RunStates (optional)	run_id, step_id, state_json	lưu state để replay nhanh
Quizzes	quiz_id, run_id, question, answer, score	phục vụ đánh giá

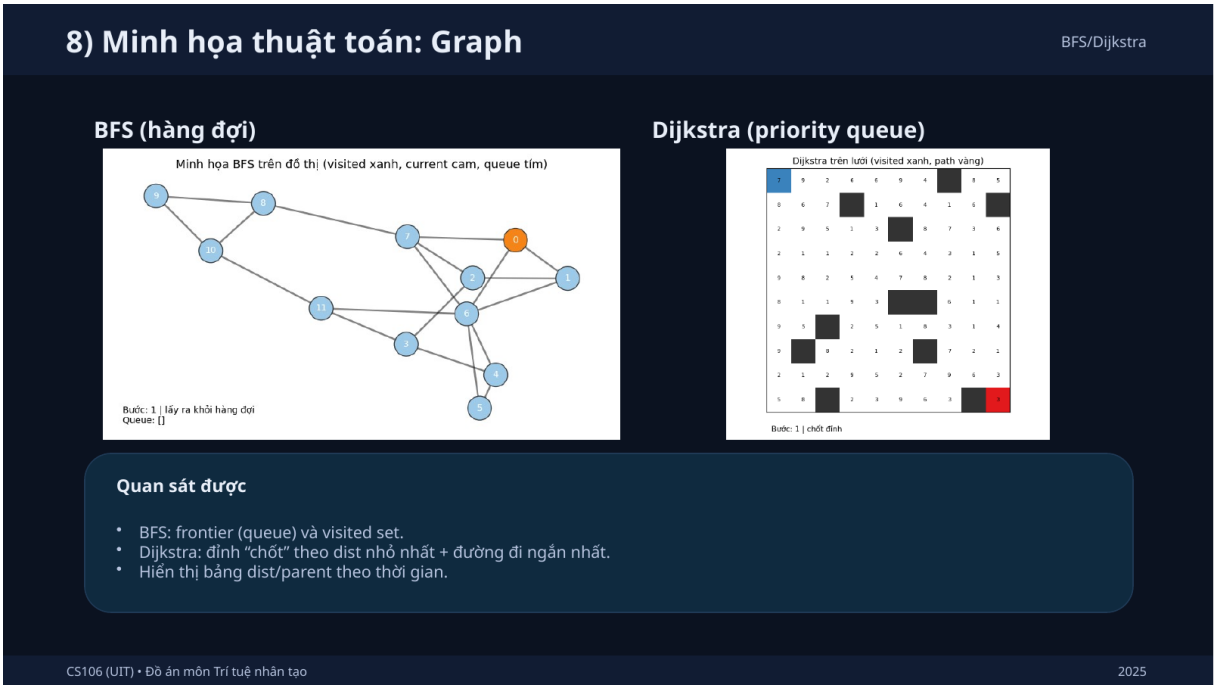
Chương V. Demo: hình ảnh + video

5.1 Hình ảnh minh họa

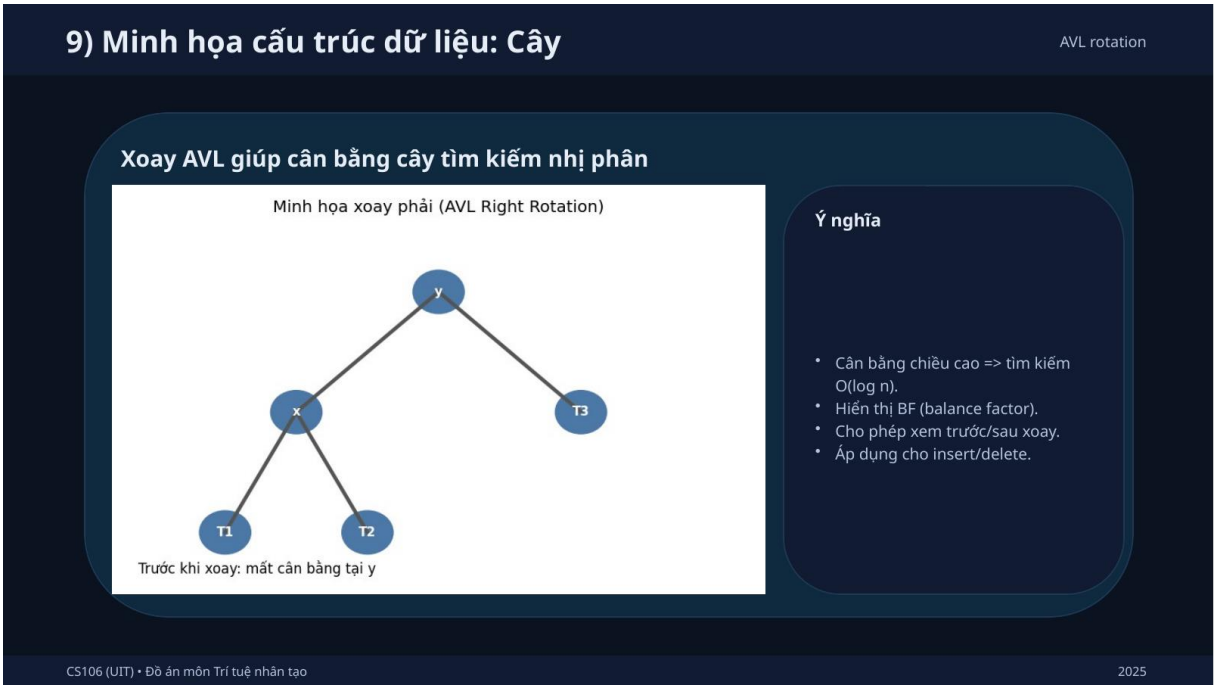
Các hình dưới đây minh họa một số thuật toán tiêu biểu. Trong demo thực tế, hệ thống cho phép chọn thuật toán, nhập input và chạy step-by-step.



Hình 5.1. Minh họa Sorting (Bubble/Quick).



Hình 5.2. Minh họa Graph (BFS/Dijkstra).



Hình 5.3. Minh họa AVL rotation.

5.2 Video demo

File video demo được cung cấp kèm theo đồ án: demo_dsa_visualizer.mp4 (khoảng 26 giây) gồm các đoạn: Bubble Sort, Quick Sort, BFS, Dijkstra, AVL rotation.

Gợi ý khi nộp: chèn link video vào slide demo và/hoặc đính kèm file trong thư mục nộp bài.

Hình 5.4. Slide demo (có thể nhúng video trong PowerPoint).

Chương VI. Kết quả thử nghiệm

6.1 Kế hoạch kiểm thử

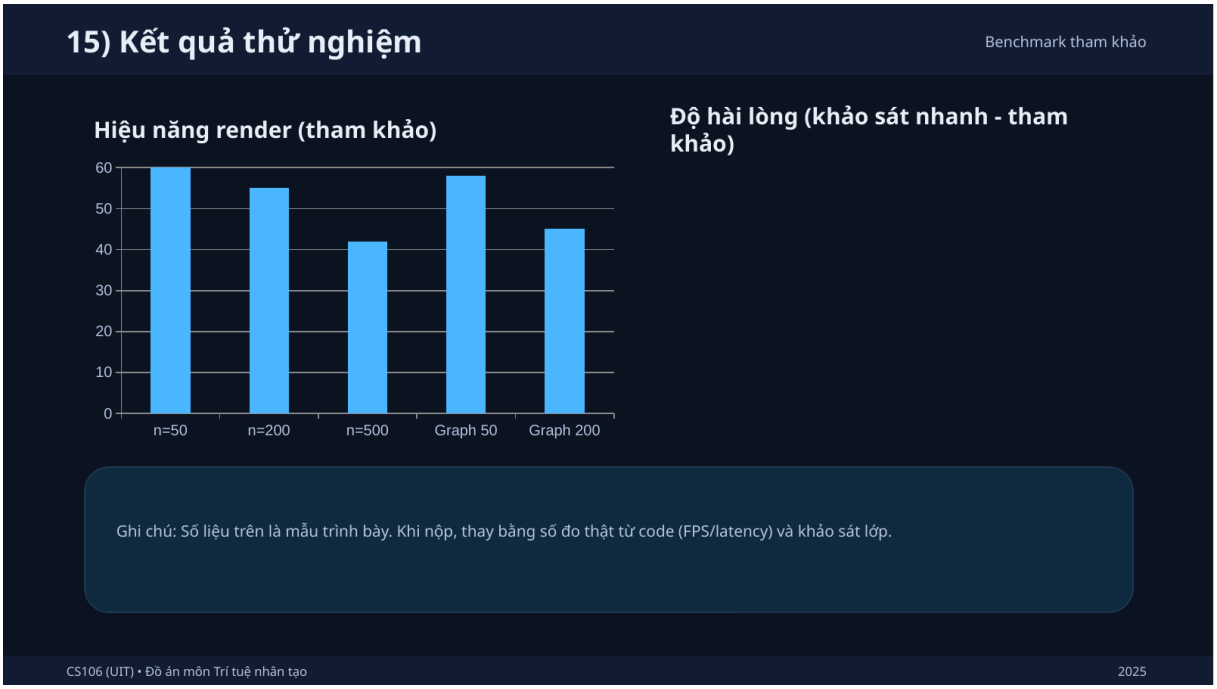
- Kiểm thử tính đúng (Correctness): so sánh output với thuật toán chuẩn, kiểm thử nhiều case biên.
- Kiểm thử UI: Step/Play/Reset, thay đổi tốc độ, tua timeline, hiển thị biến, reset state.
- Kiểm thử hiệu năng: đo FPS/latency theo kích thước input (n) và loại thuật toán.
- Kiểm thử khả dụng (Usability): khảo sát nhanh sinh viên/nhóm học thử.

6.2 Bộ dữ liệu kiểm thử đề xuất

Nhóm thuật toán	Input	Case kiểm thử
Sorting	mảng số nguyên	ngẫu nhiên, đã tăng dần, giảm dần, nhiều phần tử trùng, n nhỏ/lớn
Graph	danh sách cạnh	đồ thị liên thông/không liên thông, có chu trình, sparse/dense
Tree	dãy insert/delete	mất cân bằng nặng, nhiều xoay liên tiếp, khóa trùng
Searching	mảng + key	tồn tại/không tồn tại, biên trái/phải

6.3 Kết quả (mẫu trình bày)

Hình 6.1 minh họa format trình bày kết quả benchmark và khảo sát. Khi hoàn thiện báo cáo nộp, nhóm cần thay số liệu bằng kết quả đo thực tế (FPS/latency) trên máy chạy dự án và số liệu khảo sát.



Hình 6.1. Kết quả thử nghiệm (mẫu biểu đồ).

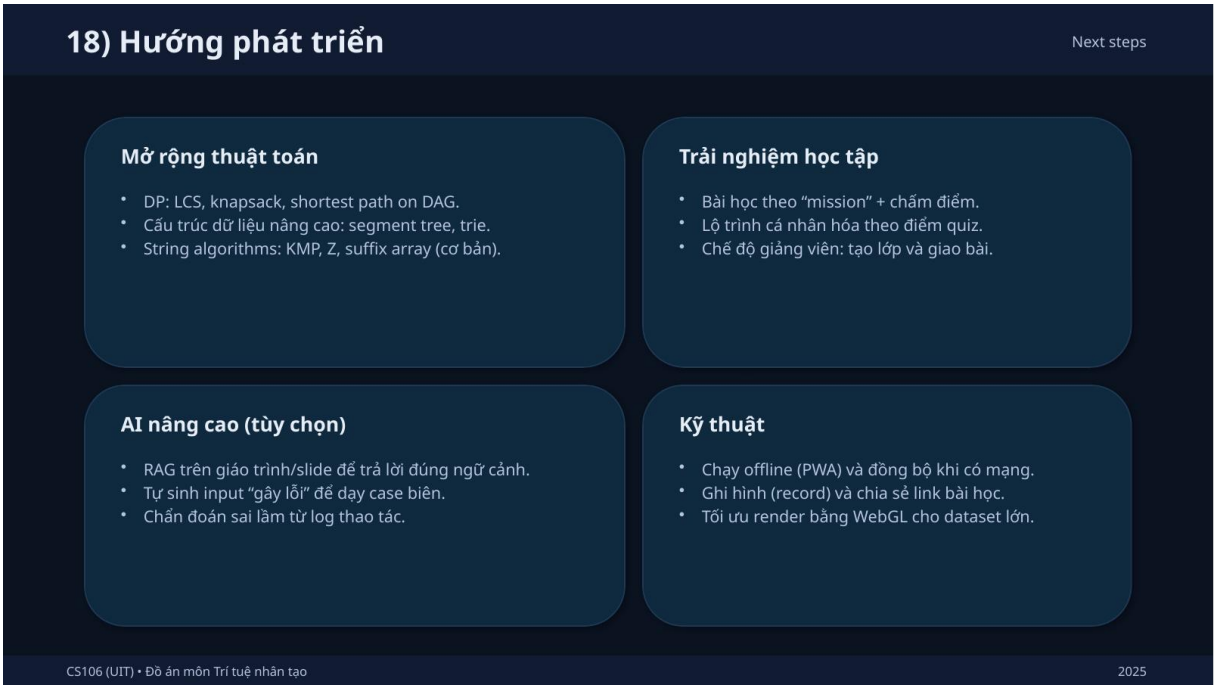
6.4 Thảo luận

Các thuật toán có nhiều bước (n lớn) sẽ tạo trace dài, do đó việc tối ưu render và giới hạn số state là quan trọng. Hướng tiếp cận thường dùng: (i) chỉ emit state ở sự kiện then chốt, (ii) nén trace, (iii) render incremental thay vì vẽ lại toàn bộ.

Chương VII. Hướng phát triển đồ án

Trong giới hạn thời gian môn học, hệ thống tập trung vào một số thuật toán cốt lõi. Các hướng phát triển:

- Mở rộng thư viện thuật toán (DP, string algorithms, cấu trúc dữ liệu nâng cao).
- Chế độ giảng viên: tạo lớp, giao bài, xem thống kê tiến độ học.
- Tối ưu hóa hiệu năng bằng WebGL hoặc kỹ thuật batching khi n lớn.
- Tích hợp AI nâng cao theo hướng RAG trên tài liệu môn học để trả lời đúng ngữ cảnh.
- Ghi hình lại buổi chạy (record) và chia sẻ liên kết cho người khác xem lại.



Hình 7.1. Hướng phát triển .

ChươngVIII. Hướng dẫn sử dụng chi tiết

8.1 Yêu cầu môi trường

- Hệ điều hành: Windows/macOS/Linux.
- Trình duyệt: Chrome/Edge/Firefox phiên bản mới.
- Nếu có back-end: Node.js hoặc Python/Java (tùy code của nhóm); CSDL tùy chọn (SQLite/PostgreSQL).

8.2 Cài đặt

Ví dụ lệnh cài đặt/chạy (cập nhật theo project thực tế)

```
# 1) Clone repository
git clone <repo-url>
cd <project>

# 2) Cài dependencies (ví dụ)
npm install

# 3) Chạy development
npm run dev

# 4) Build production
npm run build
npm run start
```

8.3 Hướng dẫn thao tác

- Bước 1: Chọn nhóm thuật toán (Sorting/Graph/Tree/...).
- Bước 2: Chọn thuật toán cụ thể (Bubble Sort, BFS, ...).
- Bước 3: Nhập dữ liệu đầu vào và tham số minh họa (tốc độ, chế độ tô màu).
- Bước 4: Nhấn Run để chạy tự động hoặc Step để chạy từng bước.
- Bước 5: Quan sát canvas và timeline; đọc giải thích từng bước ở panel log.
- Bước 6: Chuyển sang tab Quiz để làm bài và xem phản hồi.

8.4 Xuất và chia sẻ

- Xuất ảnh/screenshot của bước đang xem (PNG).
- Xuất log hoặc file JSON của trace để gửi cho bạn học/giảng viên.
- Ghi hình demo (screen record) để nộp kèm báo cáo.

Hình 8.1. Hướng dẫn sử dụng nhanh (từ slide).

Tài liệu tham khảo

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
2. Wikipedia: Algorithm visualization. https://en.wikipedia.org/wiki/Algorithm_visualization
3. Wikipedia: Bubble sort. https://en.wikipedia.org/wiki/Bubble_sort
4. Wikipedia: Quicksort. https://en.wikipedia.org/wiki/Quicksort
5. Wikipedia: Breadth-first search. https://en.wikipedia.org/wiki/Breadth-first_search
6. Wikipedia: Dijkstra's algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
7. Wikipedia: AVL tree. https://en.wikipedia.org/wiki/AVL_tree
8. Wikipedia: Separation of concerns. https://en.wikipedia.org/wiki/Separation_of_concerns

Chương IX. Bổ sung cơ sở lý thuyết và thuật toán (mở rộng)

9.1 Thuật toán sắp xếp: Merge Sort & Heap Sort

Ngoài Bubble/Quick, hệ thống có thể mở rộng thêm các thuật toán có giá trị sư phạm cao như Merge Sort (tư duy chia để trị + trộn) và Heap Sort (cấu trúc heap).

Mã giả Merge Sort (minh họa merge)

```
MERGE_SORT(A, l, r):
  if l >= r: return
  m <- floor((l+r)/2)
  MERGE_SORT(A, l, m)
  MERGE_SORT(A, m+1, r)
  MERGE(A, l, m, r)

MERGE(A, l, m, r):
  L <- A[l..m], R <- A[m+1..r]
  i <- 0, j <- 0, k <- l
  while i < |L| and j < |R|:
    emit_state(A, l, r, i, j, action="compare")
    if L[i] <= R[j]: A[k] <- L[i]; i++
    else: A[k] <- R[j]; j++
    k++
    emit_state(A, l, r, k, action="write")
  copy remaining elements
```

Mã giả Heap Sort (minh họa heap)

```
HEAP_SORT(A):
  BUILD_MAX_HEAP(A)
  for end <- n-1 .. 1:
    swap(A[0], A[end])
    emit_state(A, swap=(0,end), heap_size=end)
    SIFT_DOWN(A, 0, end)

SIFT_DOWN(A, i, heap_size):
  while has_child(i):
    j <- index of larger child
    emit_state(A, i, j, action="compare")
    if A[i] < A[j]:
      swap(A[i], A[j])
      emit_state(A, swap=(i,j), action="swap")
      i <- j
    else: break
```

9.2 Thuật toán tìm kiếm: Binary Search

Binary Search có thể minh họa trực quan bằng cách thu hẹp đoạn [l, r] và tô màu mid ở mỗi bước.

Mã giả Binary Search (minh họa l/r/mid)

```
BINARY_SEARCH(A, key):
  l <- 0; r <- n-1
  while l <= r:
    mid <- floor((l+r)/2)
    emit_state(A, l, r, mid, action="probe")
    if A[mid] == key: return mid
    else if A[mid] < key: l <- mid+1
    else: r <- mid-1
  return NOT_FOUND
```

9.3 Thuật toán đồ thị: DFS, Topological Sort, Kruskal

DFS (Depth-First Search) phù hợp để minh họa stack/đệ quy; Topological Sort minh họa thứ tự topo trên DAG; Kruskal minh họa Union-Find và quá trình chọn cạnh theo trọng số.

Mã giả DFS (đệ quy)

```
DFS(G, s):
  visited[s] <- true
  emit_state(visited, stack=[s], action="enter")
  for each neighbor v of s:
    if not visited[v]:
      parent[v] <- s
      DFS(G, v)
  emit_state(visited, stack_pop=s, action="exit")
```

Mã giả Topological Sort (Kahn)

```
TOPO_SORT(G):
  indeg[v] <- number of incoming edges
  Q <- all nodes with indeg=0
  order <- []
  while Q not empty:
    u <- pop(Q)
    order.append(u)
    emit_state(order, Q, u, action="select")
    for each edge (u, v):
      indeg[v]--
      if indeg[v]==0: push(Q, v)
```

```
  if |order| < |V|: report cycle
  return order
```

Mã giả Kruskal (Union-Find)

```
KRUSKAL(G):
  sort edges by weight
  make_set(v) for all v
  MST <- empty
  for each edge (u, v, w) in sorted:
    emit_state(MST, edge=(u,v,w), action="consider")
    if find(u) != find(v):
      union(u, v)
      MST.add(edge)
      emit_state(MST, action="take")
  return MST
```

9.4 Quy hoạch động: LCS (Longest Common Subsequence)

Quy hoạch động thường khó hình dung nếu chỉ xem bảng số. Minh họa DP bằng heatmap và đường truy vết (traceback) giúp người học hiểu quan hệ truy hồi.

Mã giả LCS (emit_state theo ô dp)

```
LCS(X, Y):
  n <- |X|, m <- |Y|
  dp[0..n][0..m] <- 0
  for i <- 1..n:
    for j <- 1..m:
      if X[i-1] == Y[j-1]:
        dp[i][j] <- dp[i-1][j-1] + 1
      else:
        dp[i][j] <- max(dp[i-1][j], dp[i][j-1])
      emit_state(dp, i, j, action="fill")
  return dp[n][m]
```

Chương 10. Phụ lục

Phụ lục A. Danh mục thuật toán và kiểu hiển thị

Nhóm	Thuật toán	Kiểu minh họa đề xuất
Sorting	Bubble/Selection/Insertion	Bars + highlight i/j + swap
Sorting	Merge/Quick/Heap	Partition/merge view + heap tree
Searching	Binary Search	Segment [l,r] + mid
Graph	BFS/DFS	Node color + queue/stack
Graph	Dijkstra/A*/Kruskal	dist table + chosen edges/path
Tree	BST/AVL/Heap	Node-link diagram + rotations
DP	LCS/Knapsack	Table heatmap + traceback

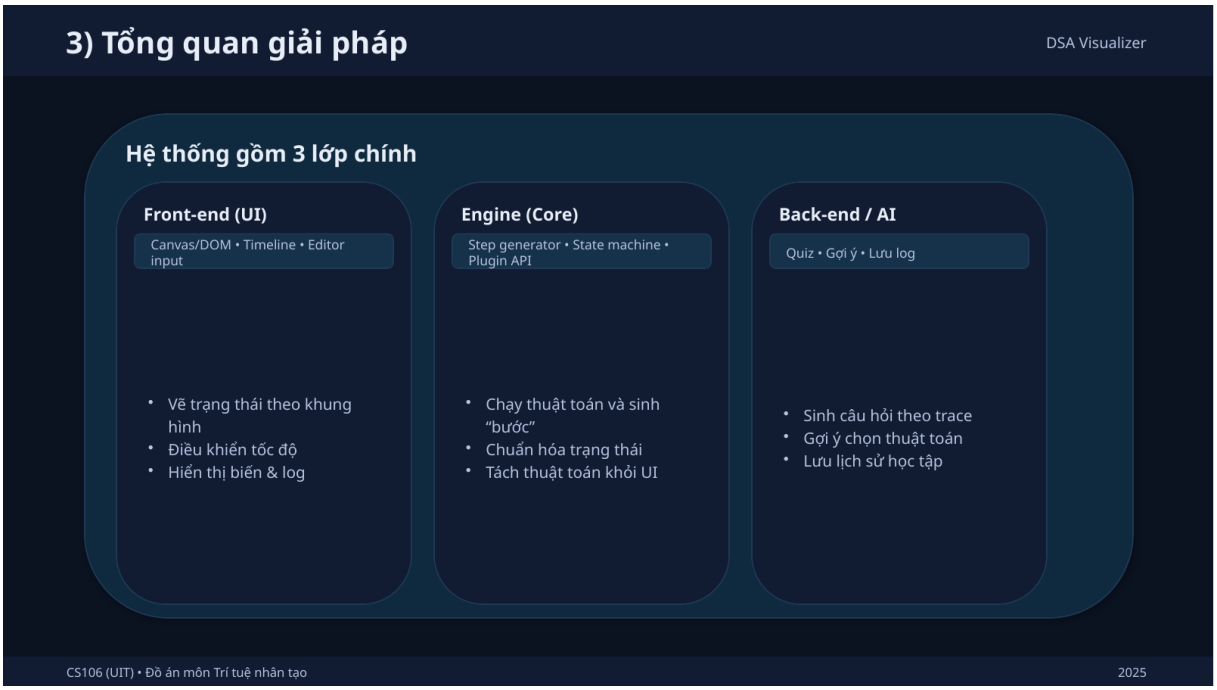
Phụ lục B. module gợi ý AI

Các luật dưới đây chỉ là ví dụ. Ý tưởng: dựa trên mục tiêu người học (tìm kiếm, đường đi ngắn, sắp xếp) và đặc điểm dữ liệu (n, có trọng số, có chu trình, yêu cầu ổn định...) để gợi ý thuật toán phù hợp.

```
RULES (mẫu):
IF task == "sort" AND n <= 30
  THEN suggest = "Insertion Sort"    // dễ quan sát, phù hợp n nhỏ
IF task == "sort" AND need_stable == true
  THEN suggest = "Merge Sort"
IF task == "shortest_path" AND has_negative_edge == true
  THEN suggest = "Bellman-Ford"
IF task == "shortest_path" AND weighted == true AND non_negative == true
  THEN suggest = "Dijkstra"
IF task == "traverse" AND want_level_order == true
  THEN suggest = "BFS"
IF task == "MST"
  THEN suggest = "Kruskal (Union-Find)"
```

Phụ lục C. Checklist nộp bài (tham khảo)

- Repository code + README + hướng dẫn chạy.
- Báo cáo Word (>= 30 trang) có hình minh họa và mô tả đầy đủ.
- Slide PowerPoint (>= 20 trang) có hình/GIF/video demo.
- Video demo (mp4) + link (nếu upload).
- Kết quả thử nghiệm: bảng/biểu đồ + mô tả môi trường test.



Hình PL.1. Tổng quan 3 lớp chính của hệ thống.

Hình PL.2. Thành phần AI trong đồ án (từ slide).

Hình PL.3. Thiết kế plugin-based (từ slide).

PHẦN BỔ SUNG – ĐỒNG BỘ AI TRỢ GIẢNG (OFFLINE)

Hỏi: AI trong đồ án này là AI gì?

Đáp: AI trong đồ án là AI trợ giảng offline, kết hợp giữa Hệ chuyên gia (rule-based) và truy xuất tri thức TF-IDF (RAG-lite). Hệ thống không sử dụng Internet và không dùng mô hình LLM online.

Hỏi: Vì sao gọi là AI mà không dùng Machine Learning?

Đáp: Theo định nghĩa môn học, AI bao gồm cả hệ luật (Expert System). Đồ án áp dụng AI biểu tượng: suy luận dựa trên luật IF–THEN, phân tích intent câu hỏi và truy xuất tri thức.

Hỏi: AI hoạt động như thế nào?

Đáp: AI gồm 4 bước: (1) Tiền xử lý câu hỏi, (2) Nhận diện ý định, (3) Truy xuất đoạn lý thuyết liên quan bằng TF-IDF, (4) Sinh câu trả lời + nút minh hoạ thuật toán.

Hỏi: Ưu điểm của AI offline?

Đáp: Không phụ thuộc Internet, dễ kiểm soát, phù hợp môi trường học tập và minh bạch logic xử lý.

Hỏi: Hạn chế của hệ thống AI?

Đáp: Chưa học được từ dữ liệu mới và chưa hiểu ngữ nghĩa sâu như LLM. Đây là hướng phát triển trong tương lai.

Hỏi: AI khác gì so với chatbot thông thường?

Đáp: AI trợ giảng tập trung vào nội dung DSA, có khả năng dẫn sang mô phỏng trực quan và chấm bài tự động.