

Here are all the questions from the uploaded image:

1. **A)Prove that uniform-cost search and breadth-first search with constant step costs are optimal when used with the GRAPH-SEARCH algorithm. B)Compare the four evaluation criteria set of several uninformed search strategies.** C)**Show a state space with constant step costs in which GRAPH-SEARCH using iterative deepening finds a suboptimal solution.**

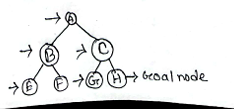
**A):**

Got it 👍  
I’ll rewrite the handwritten answer from your uploaded pages in a clean, structured form:

## **Breadth-First Search (BFS)**

In BFS, all edges have the same cost, and it explores nodes **level by level**.  
It explores all nodes at the current level before moving on to the next level. Since all step costs are constant, BFS ensures that all nodes at a given level have the same cost.

BFS will always find the optimal path to a node if it exists, because it explores nodes in the order of their distance from the start node. The goal node is guaranteed to be found.



**Example:**

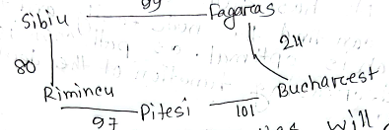
* Start from node **A**.
* First explore **B** and **C** (level 1).
* Next explore **E, F, G, H** (level 2).
* Since all costs are the same, BFS can easily find the goal node (H).
* Thus, BFS is optimal when all edge costs are equal.

## **Uniform Cost Search (UCS)**

UCS explores nodes in **increasing order of path cost**.  
It uses the lowest cumulative cost to find a path from the source to the destination.

* Nodes are expanded starting from the root, according to the minimum cumulative cost.
* Implemented using a **priority queue**.
* At every step, UCS chooses the path with the least cost so far.
* As nodes are added, the paths get shorter, ensuring that the first time the goal is expanded, it is the optimal solution.

UCS guarantees finding the optimal solution because it explores paths in increasing order of cost.



## **Example with Romania Map**

Consider the graph with the following edges:

* **Sibiu → Rimnicu** = 80
* **Rimnicu → Pitesti** = 97
* **Pitesti → Bucharest** = 101
* **Sibiu → Fagaras** = 99
* **Fagaras → Bucharest** = 211

**Step-by-step UCS expansion:**

1. Start at **Sibiu**.
2. Expand **Rimnicu**: cost = 80.
3. Expand **Pitesti**: cost = 80 + 97 = **177**.
4. Expand **Fagaras**: cost = 99.
5. From Fagaras → Bucharest = 99 + 211 = **310**.
6. From Pitesti → Bucharest = 177 + 101 = **278**.

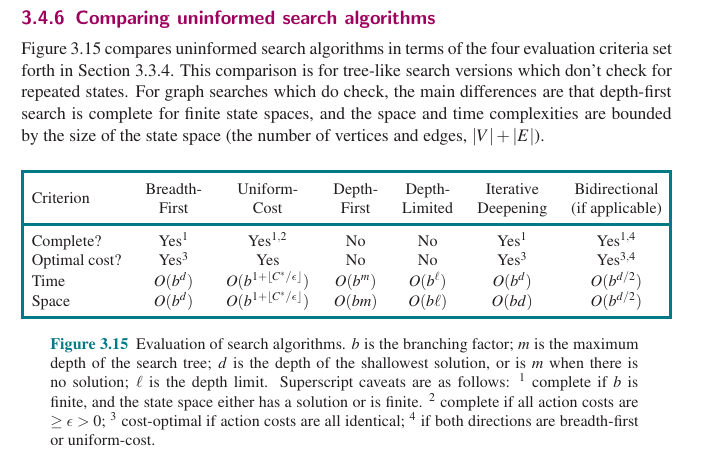
Since UCS always picks the lowest cost path, it will choose 278 over 310.

✅ **Optimal Path Found:**  
**Sibiu → Rimnicu → Pitesti → Bucharest**  
with total cost = **278**.

✨ Final Notes:

* **BFS** is best when all edges have equal cost.
* **UCS** is best when edge costs vary, as it guarantees the least-cost path.

B):



**C): Show a state space with constant step costs in which GRAPH-SEARCH using iterative deepening finds a suboptimal solution.**

**What is the question asking?**

👉 We need to show that if **Iterative Deepening Search (IDS)** is implemented as a **GRAPH-SEARCH** (i.e., using an explored set), then sometimes it can return a **wrong (suboptimal) solution**.

**Main Reason**

* IDS runs DFS with a depth limit, gradually increasing the limit.
* But in GRAPH-SEARCH, there is an **explored set**.
* If a node is reached first through a “bad/longer path,” it gets stored as explored.
* Later, if we try to reach the same node again through a “better/shorter path,” GRAPH-SEARCH will say: *“No, this node was already visited, we won’t expand it again.”*
* As a result, the optimal path is discarded.
* IDS then proceeds to the next iteration and finds the goal through a **longer/worse path** → giving a **suboptimal solution**.

**Small Example**

S

├─ A ─ C ─ D ─ G (path length = 4)

└─ B ─ C ─ G (path length = 3) ✅ optimal

* All step costs = 1
* Optimal path = S-B-C-G (3 steps)

**But what happens in GRAPH-SEARCH IDS:**

1. At depth limit = 2, IDS first explores A-C and marks C as explored.
2. Still at limit = 2, when going through B-C, GRAPH-SEARCH says “C is already explored,” so it does not expand C->G.
3. As the depth limit increases, eventually IDS discovers S-A-C-D-G (length = 4).
4. But the shorter path of 3 steps was skipped because of the explored set.

**Summary**

* **Tree-search IDS → always optimal (with unit costs).**
* **Graph-search IDS (with an explored set) → can return suboptimal solutions.**

ঠিক আছে 🙂 এখন আমি একদম **সহজ ভাষায়** ছোট করে বুঝিয়ে দিচ্ছি —

**প্রশ্নে কী চাওয়া হয়েছে?**

👉 দেখাতে হবে যে **Iterative Deepening Search (IDS)** যদি **GRAPH-SEARCH** আকারে করা হয় (মানে explored set ব্যবহার করি), তখন কখনো কখনো **ভুল (suboptimal) solution** দিতে পারে।

**মূল কারণ**

* IDS তো depth-limit করে করে DFS চালায়।
* কিন্তু GRAPH-SEARCH-এ **explored set থাকে**।
* যদি কোনো নোডকে আগে “খারাপ পথ” দিয়ে দেখি, তাহলে সেটা explored হয়ে যাবে।
* পরে যখন “ভালো/ছোট পথ” দিয়ে আবার সেই নোডে আসতে চাই, GRAPH-SEARCH বলবে — *“না, এই নোড তো আগেই দেখা, আবার ঢুকবো না”*।
* ফলে optimal পথটা বাদ পড়ে যায়।
* IDS পরের iteration-এ গিয়ে একটা **লম্বা/খারাপ পথ** দিয়েই goal পায় → **suboptimal solution**।

**ছোট উদাহরণ**

S

├─ A ─ C ─ D ─ G (পথের দৈর্ঘ্য = 4)

└─ B ─ C ─ G (পথের দৈর্ঘ্য = 3) ✅ optimal

* সব step cost = 1
* Optimal path = S-B-C-G (3 ধাপ)

**কিন্তু যা হলো:**

1. IDS limit=2 এ গিয়ে প্রথমে A-C দেখে ফেলে, C কে explored সেটে ঢুকিয়ে রাখে।
2. limit=2 তে B-C এ গেলে GRAPH-SEARCH বলবে: “C তো আগেই দেখা হয়েছে” → তাই C->G আর দেখা যাবে না।
3. Limit বাড়াতে বাড়াতে শেষে S-A-C-D-G (4 ধাপ) পাই।
4. অথচ ৩ ধাপের ছোট পথ ছিল, যেটা বাদ গেলো explored সেটের কারণে।

**সারসংক্ষেপ**

* **Tree-search IDS → সবসময় optimal (unit cost হলে)।**
* **Graph-search IDS (explored set রাখা হলে) → suboptimal হতে পারে।**

1. **A)What is the heuristic function of an informed search strategy? B)How to minimize the total estimated solution cost using the best-first search, A\* search, algorithm? C)Show the heuristic must be admissible and consistent for the optimal solution in the A\* search algorithm.\*\***

**A:** The **heuristic function** in an **informed search strategy** is a function that estimates how close a given **state** is to the **goal state**. It is denoted as:

h(n)

where n is a node in the search space.

### Key points:

1. **Purpose:**  
   The heuristic function guides the search algorithm to explore more promising paths first, rather than blindly searching all possible paths (like in uninformed search).
2. **Nature:**
   * It is usually **problem-specific**.
   * Provides an **estimate**, not the exact cost to reach the goal.
   * Should be **fast to compute** for efficiency.
3. **Example:**
   * In a **pathfinding problem**, h(n) could be the straight-line distance from the current location to the destination.
   * In the **8-puzzle problem**, h(n) could be the number of misplaced tiles or the sum of Manhattan distances of tiles from their goal positions.
4. **Role in Algorithms:**  
   Heuristic functions are used in algorithms like **A\***, **Greedy Best-First Search**, etc., to prioritize nodes.
   * **A\***: f(n)=g(n)+h(n) where g(n) is the cost to reach n from the start.
   * **Greedy Best-First**: chooses node with minimum h(n) only

**B:**

To **minimize the total estimated solution cost** using **Best-First Search** and **A\***, we need to understand how these algorithms use **heuristics** and **path costs**. Let’s break it down clearly:

### ****1. Best-First Search****

* **Idea:** Select the node that appears “closest” to the goal according to a heuristic function h(n).
* **Node selection:** Node with **minimum h(n)** is expanded first.
* **Limitation:**
  + Best-First Search only considers h(n) (estimate to goal) and **ignores the path cost g(n)**.
  + It **does not guarantee the minimum total cost solution**, only finds a solution quickly.

✅ **So, to minimize total estimated cost:**

* Best-First Search alone is **not enough**; you must combine it with path cost consideration → that’s where **A\*** comes in.

### 2. A Search Algorithm\*

* **Idea:** Minimize the total estimated cost f(n) for a node:

f(n)=g(n)+h(n)

where:

* g(n) = actual cost from start to node n
* h(n) = estimated cost from n to goal
* **Node selection:** Expand the node with **lowest f(n)** first.
* **How it guarantees minimal cost:**
  + If the heuristic h(n) is **admissible** (never overestimates the true cost to goal), A\* is **optimal**.
  + If h(n) is **consistent (monotone)**, A\* is efficient and guarantees that once a node is expanded, its best path cost is final.

### ****Steps to Minimize Total Estimated Cost in A****\*

1. Compute f(n)=g(n)+h(n) for all frontier nodes.
2. Choose the node with **lowest f(n)** to expand next.
3. Update g(n) for neighbors if a cheaper path is found.
4. Repeat until the goal node is expanded.

### ****Summary Table****

| **Algorithm** | **Uses Heuristic** | **Considers Path Cost** | **Guarantees Minimal Solution** |
| --- | --- | --- | --- |
| Best-First Search | Yes (h(n)) | No | No |
| A\* Search | Yes (h(n)) | Yes (g(n)) | Yes, if h(n) is admissible |

ঠিক আছে, আমি বাংলায় সহজভাবে ব্যাখ্যা করি।

### ****১. Best-First Search (সেরা-প্রথম অনুসন্ধান)****

* **কাজের ধারণা:** প্রতিটি নোডের জন্য হিসাব করা হয় **heuristic** h(n), যা নোডটি goal-এর কত কাছে তা অনুমান করে।
* **নোড নির্বাচন:** সর্বনিম্ন h(n) মানের নোডকে প্রথমে এক্সপ্যান্ড করে।
* **সমস্যা:**
  + শুধু h(n) ব্যবহার করে, **start থেকে নোড পর্যন্ত আসা , path cost g(n)** বিবেচনা করে না।
  + তাই এটি সর্বনিম্ন মোট খরচ সমাধান দেয় না, শুধু দ্রুত কোনো সমাধান খুঁজে দেয়।

✅ **অর্থাৎ:** Best-First Search শুধুমাত্র goal-এর দিকে দ্রুত যেতে সাহায্য করে, কিন্তু সর্বনিম্ন খরচ নিশ্চিত করে না।

### ২. A Search (এ-স্টার অনুসন্ধান)\*

* **কাজের ধারণা:** প্রতিটি নোডের **মোট অনুমানিত খরচ** f(n) হিসাব করে:

f(n)=g(n)+h(n)

যেখানে:

* g(n) = start থেকে নোড n পর্যন্ত আসা প্রকৃত খরচ
* h(n) = নোড n থেকে goal পর্যন্ত অনুমানিত খরচ
* **নোড নির্বাচন:** সর্বনিম্ন f(n) মানের নোডকে এক্সপ্যান্ড করে।
* **কেন এটি সর্বনিম্ন খরচ নিশ্চিত করে:**
  + যদি h(n) **admissible** হয় (goal পর্যন্ত খরচের বেশি অনুমান না করে), A\* সর্বদা **optimal solution** দেয়।
  + যদি h(n) **consistent** হয়, তখন প্রতিটি নোড একবার এক্সপ্যান্ড করলে তার সেরা path cost নিশ্চিত হয়।

### A এ খরচ কমানোর ধাপগুলো:\*

1. প্রতিটি frontier নোডের জন্য f(n)=g(n)+h(n) হিসাব করা।
2. সবচেয়ে ছোট f(n) মানের নোডকে এক্সপ্যান্ড করা।
3. যদি neighbor নোডে কম খরচের path পাওয়া যায়, g(n) আপডেট করা।
4. goal নোড এক্সপ্যান্ড হওয়া পর্যন্ত ধাপ ১–৩ পুনরাবৃত্তি করা।

### ****সারাংশ****

| **Algorithm** | **Heuristic ব্যবহার করে** | **Path cost বিবেচনা করে** | **সর্বনিম্ন সমাধান দেয়?** |
| --- | --- | --- | --- |
| Best-First Search | হ্যাঁ (h(n)h(n)) | না | না |
| A\* Search | হ্যাঁ (h(n)h(n)) | হ্যাঁ (g(n)g(n)) | হ্যাঁ, যদি h(n)h(n) admissible হয় |

C:

## **1. Admissible heuristic**

* **Definition:**  
  A heuristic h(n) is **admissible** if it never overestimates the true cost h\*(n) from node n to the goal.

**h(n)≤h\*(n) ∀n**

* **Reasoning:**
  + A\* selects nodes based on f(n) = g(n) + h(n).
  + If h(n) never overestimates, then f(n) will never be smaller than the actual total cost.
  + As a result, A\* can safely prioritize and always choose the **lowest-cost path** to the goal.
* **Example:**
  + Suppose you’re finding the shortest path between two cities using **straight-line distance** as the heuristic.
  + Straight-line distance will never exceed the actual road distance → therefore it is **admissible**.

## **2. Consistent (Monotone) heuristic**

* **Definition:**  
  A heuristic h(n) is **consistent** if, for every node n and its neighbor n',

**h(n)≤c(n,n′)+h(n′)**

where c(n, n') is the actual step cost from n to n'.

* **Reasoning:**
  + Consistency means the heuristic is **monotone**: as you move closer to the goal, f(n) = g(n) + h(n) never decreases.
  + This guarantees that once a node is expanded, its path cost is **final and optimal**.
  + Therefore, A\* does not need to re-expand nodes → it remains both efficient and optimal.
* **Example:**
  + In a grid puzzle using **Manhattan distance** as the heuristic:  
    each step reduces the heuristic by at most 1, which matches the actual step cost c(n, n') = 1.
  + Hence, Manhattan distance is **consistent**.

## **3. In summary: Why A\* is optimal**

* **Admissible:** ensures A\* never overestimates, so it always finds the **cheapest path** to the goal.
* **Consistent:** ensures A\* expands nodes safely and efficiently, with no need to revisit old nodes.

***A\* guarantees optimality if h(n) is admissible (and ideally consistent).***

ঠিক আছে, আমি বাংলায় ধাপে ধাপে বুঝিয়ে দিচ্ছি কেন **heuristic অবশ্যই admissible এবং consistent হতে হবে**, যাতে **A\* search সর্বোত্তম (optimal) সমাধান** দিতে পারে।

## **১. Admissible heuristic**

* **সংজ্ঞা:**  
  একটি heuristic h(n)h(n) **admissible** হয় যদি এটি কখনো সত্যিকারের goal পর্যন্ত খরচ h∗(n)h^\*(n) এর বেশি অনুমান না করে।

h(n)≤h∗(n)∀nh(n) \le h^\*(n) \quad \forall n

* **কারণ:**
  + A\* নোডগুলোর f(n)=g(n)+h(n)f(n) = g(n) + h(n) মান দেখে নির্বাচন করে।
  + যদি h(n)h(n) কখনো বেশি অনুমান না করে, তাহলে f(n)f(n) কখনো কম দেখাবে না যে প্রকৃত খরচের চেয়ে কম।
  + ফলে, A\* সর্বোচ্চ priority দিয়ে **সর্বনিম্ন প্রকৃত খরচের path** নির্বাচন করতে পারে।
* **উদাহরণ:**
  + যদি তুমি একটি শহর থেকে অন্য শহরে shortest path খুঁজছো, এবং straight-line distance (as-the-crow-flies) ব্যবহার করছো।
  + Straight-line distance কখনো বাস্তব খরচের চেয়ে বেশি হবে না → এটি **admissible**।

## **২. Consistent (Monotone) heuristic**

* **সংজ্ঞা:**  
  একটি heuristic h(n)h(n) **consistent** হয় যদি প্রতিটি নোড nn এবং neighbor n′n' এর জন্য:

h(n)≤c(n,n′)+h(n′)h(n) \le c(n, n') + h(n')

যেখানে c(n,n′)c(n, n') হলো nn থেকে n′n' পর্যন্ত আসল খরচ।

* **কারণ:**
  + Consistency মানে heuristic **monotone**: তুমি goal-এর দিকে এগোলে f(n)=g(n)+h(n)f(n) = g(n) + h(n) কখনো কমবে না।
  + এটি নিশ্চিত করে যে **একবার নোড expand হলে, তার path cost final**।
  + এতে A\* আর পুরানো নোড পুনরায় এক্সপ্যান্ড করতে হবে না → efficient এবং optimal।
* **উদাহরণ:**
  + Manhattan distance puzzle-এর জন্য heuristic consistent। কারণ তুমি এক ধাপ এগালে h(n)h(n) সর্বাধিক ১ কমতে পারে, যা step cost c(n,n′)=1c(n, n')=1 এর সমান।

## ৩. সংক্ষেপে A এ optimality\*

* **Admissible:** নিশ্চিত করে A\* কখনো underestimate না করে, তাই **goal node পর্যন্ত cheapest path** পাওয়া যায়।
* **Consistent:** নিশ্চিত করে A\* efficient ও safe → একবার expand করা node-এর cost final।

Optimality guaranteed if h(n) is admissible (and ideally consistent).\text{Optimality guaranteed if } h(n) \text{ is admissible (and ideally consistent).}

1. **What is an uninformed search? Compare breadth-first search (BFS), and depth-first search (DFS) algorithms. Show that the 8-puzzle states are divided into two disjoint sets, such that no state in one set can be transformed into a state in the other set by any number of moves.**

An **uninformed search strategy** means a search method where we have no information about which path will lead to the goal more quickly or at a lower cost.  
In other words, the agent has no idea which step will bring it closer to the goal.

**Example:**  
If your agent wants to go from **Arad to Bucharest**, but it does not know where the cities in Romania are located or which city is closer to the goal. It doesn’t know whether going to **Zerind** or **Sibiu** first would be better.

## **Breadth-First Search (BFS)**

* BFS is a search method where, starting from the root node (initial state), it first expands all nodes at the same level, then moves to the next level, then the next… and so on.
* In other words—first at **Depth = 0** (the root), then all nodes at **Depth = 1**, then all nodes at **Depth = 2**, and so on in sequence.

**How does BFS work?**

* First, the root node is placed in a **FIFO queue** (i.e., first-in-first-out).
* Then, all of its child (successor) nodes are added to the back of the queue.
* Next, the node at the front of the queue is removed and its children are added.
* This process continues, level by level, until the goal is found.

**Properties:**

* **Complete:** Yes (if the branching factor is finite).
* **Optimal:** Yes (if all step costs are equal).
* **Time complexity:** O(b^d), where b= branching factor, d = goal depth.
* **Space complexity:** O(b^d) (since all frontier nodes must be stored in memory).

## **Depth-First Search (DFS)**

* **Working principle:** Depth-first search is a method where we go as deep as possible down one branch (to the deepest node), then backtrack.
* It first reaches the lowest-level nodes.
* When there is no way to go deeper, it backs up and explores another branch.
* This continues until a solution is found.
* The solution may **not** be the best or cheapest (cost-optimal), since it returns the first solution it finds.
* **Data structure:** Stack (LIFO) or recursion.

**Properties:**

* **Complete:** No (if there is infinite depth, the goal may never be reached).
* **Optimal:** No (no guarantee of finding the shallowest or least-cost goal).
* **Time complexity:** O(b^m), where m= maximum depth.
* **Space complexity:** O(bm) (only needs to track the path, so memory is much less than BFS).

Would you like me to also **compare BFS vs DFS in a table format** (side-by-side) so it’s easier to memorize for exams?

## **Breadth-First Search (BFS)**

* BFS হচ্ছে এমন একটা অনুসন্ধান পদ্ধতি যেখানে প্রথমে রুট নোড (মূল অবস্থান) থেকে শুরু করে, প্রথমে সে সব নোড একসাথে এক লেভেলে এক্সপ্যান্ড করে, তারপর পরের লেভেলে, তারপর পরের লেভেলে... এমনভাবে সার্চ করে।
* অর্থাৎ—প্রথমে Depth=0 (মূল নোড), তারপর Depth=1 এর সব নোড, তারপর Depth=2 এর সব নোড... এরকম ক্রমান্বয়ে।

**BFS কিভাবে কাজ করে?**

* প্রথমে মূল নোডকে একটি **FIFO queue** তে রাখে (অর্থাৎ, প্রথম আসা নোড প্রথম বের হবে)।
* এরপর সেই নোডের সব চাইল্ড (successors) নোড গুলো queue-র পিছনে যোগ করে।
* পরের ধাপে queue-র প্রথমে থাকা নোডটি বের করে তার চাইল্ড গুলো যোগ করে।
* এরকম করে এক এক করে লেভেল অনুসারে সব নোড এক্সপ্লোর করে।
* **গুণাবলী:**
  + **Complete:** হ্যাঁ (যদি branching factor সীমিত হয়)।
  + **Optimal:** হ্যাঁ (যদি প্রতিটি step cost সমান হয়)।
  + **Time complexity:** O(b^d), যেখানে b = branching factor, d = goal depth।
  + **Space complexity:** O(b^d) (কারণ সব frontier মেমরিতে রাখতে হয়)।

## **Depth-First Search (DFS)**

* কাজের ধরণ: Depth-first search হলো একটা খোঁজ করার পদ্ধতি যেখানে আমরা সর্বপ্রথম যত গভীরভাবে পারি তত নিচে (deepest node) যাই, পরে backtrack করে।
* প্রথমে এটি সবচেয়ে নিচের স্তরে নোডগুলোতে পৌঁছে যায়।
* যখন আর নিচে যাওয়ার জায়গা থাকে না, তখন পিছনে ফিরে গিয়ে (back up) সেই নোড থেকে অন্য একটা শাখায় যায়।
* এরকম করে সব শাখায় অনুসন্ধান করে যতক্ষণ না প্রথম সমাধান পাওয়া যায়।
* এই সমাধান **সর্বোত্তম বা সস্তা (cost-optimal)** নাও হতে পারে, কারণ এটি প্রথম যে সমাধান পায় সেটাই দেয়।
* **ডেটা স্ট্রাকচার:** Stack (LIFO) বা recursion।
* **গুণাবলী:**
  + **Complete:** না (infinite depth থাকলে goal নাও পেতে পারে)।
  + **Optimal:** না (সবচেয়ে ছোট cost বা কম depth-এর goal খুঁজে পাওয়ার নিশ্চয়তা নেই)।
  + **Time complexity:** O(b^m), যেখানে m হলো সর্বোচ্চ depth।
  + **Space complexity:** O(bm) (শুধু path track রাখতে হয়, তাই BFS এর চেয়ে অনেক কম মেমরি লাগে)।

1. Answer any 5 of the following questions:

**(a)** What does artificial intelligence mean? List the learning outcomes of AI.

**(b)** Define in your own words the following terminologies: agent, agent program, rationality, autonomy, deterministic, and stochastic.

**(c)** What is PEAS in specifying the task environment? Illustrate and describe the structure of the model-based reflex agent.

Let me know if you'd like me to help with answering these questions!

**Compare the four evaluation criteria set of several uninformed search strategies:**