



AP[®] Computer Science AB 2005 Scoring Guidelines

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT[®], the PSAT/NMSQT[®], and the Advanced Placement Program[®] (AP[®]). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2005 by College Board. All rights reserved. College Board, AP Central, APCD, Advanced Placement Program, AP, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Entrance Examination Board. Admitted Class Evaluation Service, CollegeEd, Connect to college success, MyRoad, SAT Professional Development, SAT Readiness Program, and Setting the Cornerstones are trademarks owned by the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: <http://www.collegeboard.com/inquiry/cbpermit.html>.

Visit the College Board on the Web: www.collegeboard.com.

AP Central is the official online home for the AP Program and Pre-AP: apcentral.collegeboard.com.

**AP[®] COMPUTER SCIENCE AB
2005 SCORING GUIDELINES**

2005 AB Question 1: Salmon

Part A:	<code>nextLocation</code>	5 points
----------------	---------------------------	-----------------

- +1/2 test if `age < matureAge`
- +1/2 return `super.nextLocation()` if juvenile
(no credit for reimplementing)
- +1 check neighboring locations
 - +1/2 correctly call `emptyNeighbors()` or `neighborsOf`
ex: `environment().neighborsOf(<location>)` (must also call `isEmpty`)
 - +1/2 reference all (N) neighbors (must not remove the location behind)
- +2 compare neighbor distance with current distance
 - +1/2 get current location (lose this if reference inaccessible field)
 - +1/2 attempt to identify closer location
 - +1 correctly compare distances from present location using `distanceHome`
(may not (re)implement `distanceHome`, must potentially test all possibilities)
- +1 return value
 - +1/2 return any empty neighbor that is closer to home
 - +1/2 return current location if none closer

Part B:	<code>act</code>	4 points
----------------	------------------	-----------------

- +1 distinguish juvenile from mature
 - +1/2 test if `age < matureAge`
 - +1/2 juvenile actions or mature actions, based on decision
- +1/2 juvenile action: `move()`
- +2 mature actions
 - +1/2 test if `location().equals(homeLocation)` (lose this if reference inaccessible field)
 - +1/2 call `breed()` if and only if at `homeLocation`
 - +1/2 call `die()` if and only if `breed()` succeeds
 - +1/2 call `move()` in context of `homeLocation` test

Note: Methods `breed()`, `die()` and `move()` may not be reimplemented, but `equals()` can be replaced by `distanceHome()` or `compareTo()` if done correctly.

- +1/2 increment age (after processing)

**AP[®] COMPUTER SCIENCE AB
2005 SCORING GUIDELINES**

2005 AB Question 2: Postal Codes

Part A:	efficiency	1 point
----------------	------------	----------------

+1/2 O(1) for `getCitiesForCode`

+1/2 O(1) for `addCityCodePair`

Part B:	design	4 points
----------------	--------	-----------------

+1 data structure

+1/2 supports storing city names (list or set OK)

+1/2 supports storing city \rightarrow codes mapping

+1 initialization and declaration

+1/2 attempt (`new Map()` OK)

+1/2 correct (for data structures provided) (must be `private`)

+1 explanation

+1/2 describe how cities are stored

+1/2 describe how codes are stored and mapped

+1 `addCityCodePair` update

+1/2 describe how cities are updated

+1/2 describe how codes are updated (mapped)

Part C:	<code>getCodesForCity</code> efficiency	1 point
----------------	---	----------------

+1 O(log N) or better

+1/2 data structure supports O(log N)

+1/2 explanation (must identify big-Oh for structure used)

Part D:	<code>printAllCities</code>	3 points
----------------	-----------------------------	-----------------

+1 iterate and print all cities

+1/2 attempt to iterate/traverse and print the cities

+1/2 prints all, no dups

`(System.out.println(cityToCodeMap.keySet()));` is OK

+1 cities printed in alphabetical order

+1 O(N) efficiency (must iterate over data structure that contains cities)

**AP[®] COMPUTER SCIENCE AB
2005 SCORING GUIDELINES**

2005 AB Question 3: Successor Nodes

Part A:	<code>verifyParentLinks</code>	5 points
----------------	--------------------------------	-----------------

- +1/2 return `true` if empty
- +1/2 check parent link of root node is `null`
- +1 1/2 check parent link of at least one non-root node
 - +1/2 attempt (call to `getParent` is enough)
 - +1 correct relationship test
- +1 1/2 traversal
 - +1/2 attempt (recursion or iteration)
 - +1/2 traverse to bottom of tree somewhere
 - +1/2 traverse every node
- +1 return correct boolean in all cases for non-empty trees

Part B:	<code>successor</code>	4 points
----------------	------------------------	-----------------

- +1 1/2 identify and handle case where `successor` is below `t`:
 - +1/2 attempt (must test if `t.getRight() != null` and do something with the right subtree OR put in another data structure and look for successor of `t`)
 - +1 return minimum node from right subtree,
e.g., `return minNode(t.getRight())`
- +1 1/2 handle case where `successor` is above `t`:
 - +1/2 attempt traversal (loop) above `t`
e.g., up to parent OR down from root to `t` OR
put in another data structure and look for successor of `t`
 - +1 return `successor` node
- +1 in the case where there's no successor, return `null`

Note: Must accept any correct solution, regardless of efficiency (e.g., can do inorder traversal, store nodes in a list, search for `t`, then get next entry).

Note: `equals` method not redefined in `TreeNode`, so equivalent to `==` as long as the invoking object is not `null`

AP[®] COMPUTER SCIENCE AB
2005 SCORING GUIDELINES

2005 AB Question 4: Expand Aliases

Part A:	<code>appendSetToQueue</code>	3 points
+2	iterate over set	} no loop, no credit
+1	attempt (must access set members)	
+1	correct	
+1	each item added to end of <code>q</code>	

Part B:	<code>expandAlias</code>	6 points
+1	instantiate set	
+1/2	attempt (<code>new Set()</code> OK)	
+1/2	correct	
+1/2	create queue*	
+1/2	enqueue <code>alias</code> or its expansion	
+1	access all items in queue	
+1/2	attempt (must access queue in body)	
+1/2	correct	
+1/2	get next alias/address (in context of loop)	
+2	process alias/address (in context of loop)	
+1/2	test whether alias or address	
+1/2	expand alias	
+1/2	store expansion	
+1/2	store address in set	
+1/2	return a set of addresses	

*Note: Alternative data structures for queue are acceptable.

Usage: No penalty for use of “enque” for “enqueue” or “deque” for “dequeue”

Workshop Exam Materials
Canonical Solutions
2005 AP[®] Computer Science AB

Question 1

PART A:

```
protected Location nextLocation()
{
    if (age < matureAge)
    {
        return super.nextLocation();
    }
    else
    {
        Location currentLoc = location();
        int currentDistance = distanceHome(currentLoc);

        ArrayList possLocs = emptyNeighbors();
        for (int i = 0; i < possLocs.size(); i++)
        {
            if (distanceHome((Location)possLocs.get(i)) < currentDistance)
            {
                return (Location)possLocs.get(i);
            }
        }
        return currentLoc;
    }
}
```

PART B:

```
public void act()
{
    if ( ! isInEnv() )
    {
        return;
    }

    if (age >= matureAge && location().equals(homeLocation))
    {
        if ( breed() )
        {
            die();
        }
    }
    else
    {
        move();
    }

    age++;
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP[®] Computer Science AB

Question 2

PART A:

```
getCitiesForCode:    O(1)
addCityCodePair:    O(1)
```

PART B:

```
private Map cityToCodeMap;

public PostalCodeDB()
{
    . . .
    cityToCodeMap = new TreeMap();
}
```

`cityToCodeMap` will have cities for keys, and sets of codes for that city as corresponding value.

`addCityCodePair` will need to similarly update `cityToCodeMap` (adding to the set of codes for that city, adding a city entry to the map if first occurrence).

PART C:

Since `cityToCodeMap` is a `TreeMap`, the `get` method is $O(\log N)$.

PART D:

```
public void printAllCities()
{
    Set cities = cityToCodeMap.keySet();
    Iterator iter = cities.iterator();
    while (iter.hasNext())
    {
        System.out.println(iter.next());
    }
}
```

OR

```
public void printAllCities()
{
    Set cities = cityToCodeMap.keySet();
    System.out.println(cities);
}
```

```
private Map cityToCodeMap;
private Set cities;

public PostalCodeDB()
{
    . . .
    cityToCodeMap = new HashMap();
    cities = new TreeSet();
}
```

`cityToCodeMap` will have cities for keys, and sets of codes for that city as corresponding value. `cities` will store the city names.

`addCityCodePair` will need to similarly update `cityToCodeMap` (adding to the set of codes for that city, adding a city entry to the map if first occurrence). For each new city, its name must be entered into `cities`.

Since `cityToCodeMap` is a `HashMap`, the `get` method is $O(1)$.

```
public void printAllCities()
{
    Iterator iter = cities.iterator();
    while (iter.hasNext())
    {
        System.out.println(iter.next());
    }
}
```

OR

```
public void printAllCities()
{
    System.out.println(cities);
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP[®] Computer Science AB

Question 3

PART A:

```
private boolean verifyParentLinks()
{
    return verifyParent(root, null);
}

private boolean verifyParent(TreeNode t, TreeNode parent)
{
    return (t == null ||
           (t.getParent() == parent && verifyParent(t.getLeft(), t) &&
            verifyParent(t.getRight(), t)));
}
```

OR

```
private boolean verifyParentLinks()
{
    if (root == null)
        return true;
    if (root.getParent() != null)
        return false;
    return verifyChildren (root);
}

private boolean verifyChildren(TreeNode parent)
{
    if (parent == null)
        return true;
    if (parent.getLeft() != null && parent.getLeft().getParent() != parent)
        return false;
    if (parent.getRight() != null && parent.getRight().getParent() != parent)
        return false;
    return verifyChildren(parent.getLeft()) && verifyChildren(parent.getRight());
}
```

PART B:

```
private TreeNode successor(TreeNode t)
{
    if (maxNode(root) == t)
        return null;

    if (t.getRight() != null)
        return minNode(t.getRight());

    while (t.getParent() != null && t.getParent().getRight() == t)
        t = t.getParent();
    return t.getParent();
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP[®] Computer Science AB

Question 4

PART A:

```
private void appendSetToQueue(Set items, Queue q)
{
    Iterator iter = items.iterator();
    while (iter.hasNext())
    {
        q.enqueue(iter.next());
    }
}
```

PART B:

```
public Set expandAlias(String alias)
{
    Set expanded = new HashSet();

    Queue partial = new ListQueue();
    partial.enqueue(alias);
    while (!partial.empty())
    {
        String front = (String)partial.dequeue();
        if (addressBook.containsKey(front))
        {
            appendSetToQueue((Set)addressBook.get(front), partial);
        }
        else
        {
            expanded.add(front);
        }
    }
    return expanded;
}
```