CSP 2020 入门组第一轮

一、单项选择题

第 1 题

题目

在内存储器中每个存储单元都被赋予一个唯一的序号，称为 ( ) ；

A. 地址 B. 序号 C. 下标 D. 编号

分析

内储存器直接与CPU相连接，储存容量较小，但速度快，用来存放当前运行程序的指令和数据，并直接与CPU交换信息；每个存储单元的序号称为地址；

答案

A

第 2 题

题目

编译器的主要功能是 ( ) ；

A. 将源程序翻译成机器指令代码 B. 将源程序重新组合

C. 将低级语言翻译成高级语言 D. 将一种高级语言翻译成另一种高级语言

分析

编译器将一种语言 (通常为高级语言) 翻译为另一种语言 (通常为低级语言) 的程序；

编译器完成的功能为把源码 (SourceCode) 编译成通用中间语言 (MSIL/CIL) 的字节码 (ByteCode) ，最后运行的时候通过通用语言运行库的转换，变程最终可以被 CPU 直接计算的机器码 (NativeCode) ；

答案

A

第 3 题

题目

设 x=true, y=true, z=false，以下逻辑运算表达式值为真的是 ( ) ；

A. (y∨z)∧x∧z B. x∧(z∨y) ∧z C. (x∧y) ∧z D. (x∧y)∨(z∨x)

分析

∨ ∨∨ 为 ∣ |∣ ，或的数学运算符， ∧ ∧∧ 为 & \&& ，且的数学运算符， ¬ ¬¬ 为 ~，取反的数学运算符；

A，(y∨z)∧x∧z=(true∨false)∧true∧false

=true∧true∧false

=true∧false

=false

B，x∧(z∨y)∧z=true∧(false∨true)∧false

=true∧true∧false

=true∧false

=false

C，(x∧y)∧z =(true∧true)∧false

=true∧false

=false

D，(x∧y)∨(z∨x)=(true∧true)∨(false∨true)

=true∨(false∨true)

=true∨true

=true

答案

D

第 4 题

题目

现有一张分辨率为 2048×1024 像素的 32 位真彩色图像；请问要存储这张图像，需要多大的存储空间？ ( ) ；

A. 16MB B. 4MB C. 8MB D. 2MB

分析

每 8 位真彩色图像中每一个像素的图片存储大小为 1 B，则，

32 位真彩色图像中每一个像素的图片存储大小为 4 B；

则计算为

4 B ∗ 2048 ∗ 1024 = 8388608 B = 8192 K B = 8 M B 4B \* 2048 \* 1024 = 8388608B = 8192KB = 8MB4B∗2048∗1024=8388608B=8192KB=8MB

答案

C

第 5 题

题目

冒泡排序算法的伪代码如下

输入数组L, n ≥ k；输出按非递减顺序排序的 L；

算法 BubbleSort

1. FLAG ← n //标记被交换的最后元素位置

2. while FLAG > 1 do

3. k ← FLAG -1

4. FLAG ← 1

5. for j=1 to k do

6. if L(j) > L(j+1) then do

7. L(j) ↔ L(j+1)

8. FLAG ← j

对 n 个数用以上冒泡排序算法进行排序，最少需要比较多少次? ( ) ；

A. n^2 B. n-2 C. n-1 D. n

分析

最少情况下，则为数组均为有序排列，比较次数为k次，k为n-1；

答案

C

第 6 题

题目

设 A 是介个实数的数组，考虑下面的递归算法

XYZ (A[1..n])

1. if n= 1 then return A[1]

2. else temp ← XYZ (A[l..n-1])

3. if temp < A[n]

4. then return temp

5. else return A[n]

请问算法 XYZ 的输出是什么？ ( ) ；

A. A数组的平均 B. A数组的最小值

C. A数组的中值 D. A数组的最大值

分析

分析递归函数，传入一个数组；

出口为 n = = 1 ，返回数组第一个数，

其他情况返回最后一个数与在对 1~n-1 数组进行计算的最小值；

即为返回最小值；

答案

B

第 7 题

题目

链表不具有的特点是 ( ) ；

A. 可随机访问任一元素 B. 不必事先估计存储空间

C. 插入删除不需要移动元素 D. 所需空间与线性表长度成正比

分析

链表为了表示每个数据元素 与其直接后继数据元素 之间的逻辑关系，对数据元素 来说，除了存储其本身的信息之外，还需存储一个指示其直接后继的信息 (即直接后继的存储位置 ) ，由这两部分信息组成一个"结点" (如概述旁的图所示 ) ，表示线性表中一个数据元素；线性表的链式存储表示，有缺点就是要找一个数，必须要从头开始找起，也无法随机访问任意元素；

答案

A

第 8 题

题目

有 10 个顶点的无向图至少应该有 ( ) 条边才能确保是一个连通图；

A. 9 B. 10 C. 11 D. 12

分析

连通图即为任意两点都联通的图，则 10 个顶点确保连通图将编号相邻两点链接即可；

答案

A

第 9 题 二进制数 1011 转换成十进制数是 ( ) ；

A. 11 B. 10 C. 13 D. 12

分析

使用权值相加法，

( 1011 ) 2 = 1 ∗20 + 1∗21 + 1∗23 = ( 11 ) 10

答案 A

第 10 题

5 个小朋友并排站成一列，其中有两个小朋友是双胞胎，如果要求这两个双胞胎必须相邻，则有 ( ) 种不同排列方法?

A. 48 B. 36 C. 24 D. 72

分析

两个双胞胎由于必须站一起，所以将他们看作一个人，则将排队看作4个人无顺序排，再乘上2个双胞胎的站列情况，即为

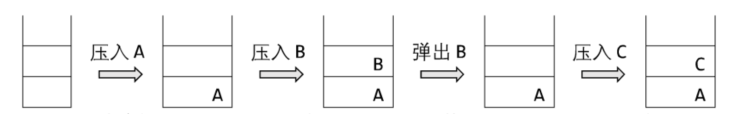
A44∗A22 = 48

答案A

第 11 题

题目

下图中所使用的数据结构是 ( )；



A. 栈 B. 队列 C. 二叉树 D. 哈希表

分析

图中数据为先进后出，即为栈；

答案

A

第 12 题

题目

独根树的高度为 1；

具有 61 个结点的完全二叉树的高度为 ( ) ；

A. 7 B. 8 C. 5 D. 6

分析

n结点数的二叉树深度为floor(log\_2^n) + 1，则为 floor(log\_2^61) + 16 = 6

答案

D

第 13 题 干支纪年法是中国传统的纪年方法，由10个天干和12个地支组合成60个天干地支；由公历年份可以根据以下公式和表格换算出对应的天干地支；

天干 = (公历年份 ) 除以10所得余数

地支 = (公历年份 ) 除以12所得余数



例如，今年是 2020 年，2020 除以 10 余数为 0，查表为"庚”；2020 除以 12，余数为 4，查表为“子” 所以今年是庚子年；

请问 1949 年的天干地支是 ( )

A. 己酉 B. 己亥 C. 己丑 D. 己卯

分析

天干 =1949 % 10 = 9 = 己

地支 =1949 % 12 = 5 = 丑

答案C

第 14 题

10 个三好学生名额分配到 7 个班级，每个班级至少有一个名额，一共有 ( ) 种不同的分配方案；

A. 84 B. 72 C. 56 D. 504

分析

+ ∗+= 84

答案 A

第 15 题

有五副不同颜色的手套 (共 10 只手套，每副手套左右手各 1 只 ) ，一次性从中取 6 只手套，请问恰好能配成两副手套的不同取法有 ( ) 种；

A. 120 B. 180 C. 150 D. 30

分析

6只中取2副手套的情况为所有的情况减去超过两副的，即为-3，由于可以选取任一两种颜色，再乘上即可；

\*(-3 )=120

答案A

二、阅读程序

第 1 题

#include <cstdlib>

#include <iostream>

using namespace std；

char encoder[26] = {'C','S','P',0}；

char decoder[26]；

string st；

int main() {

int k = 0；

for (int i = 0； i < 26； ++i)

if (encoder[i] != 0) ++k；

for (char x ='A'； x <= 'Z'； ++x) {

bool flag = true；

for (int i = 0； i < 26； ++i)

if (encoder[i] ==x) {

flag = false；

break；

}

if (flag) {

encoder[k]= x；

++k；

}

}

for (int i = 0； i < 26； ++i)

decoder[encoder[i]- 'A'] = i + 'A'；

cin >> st；

for (int i = 0； i < st.length( )； ++i)

st[i] = decoder[st[i] -'A']；

cout << st；

return 0；

}

程序解释

#include <cstdlib>

#include <iostream>

using namespace std；

char encoder[26] = {'C','S','P',0}；

char decoder[26]；

string st；

int main() {

int k = 0；

for (int i = 0； i < 26； ++i)

if (encoder[i] != 0) ++k； // k 统计 encoder 的字符数量, 计算完后，k 为 3

for (char x ='A'； x <= 'Z'； ++x) { // 此循环将 encoder 中没有的字母添加在 encoder 的末尾

bool flag = true；

for (int i = 0； i < 26； ++i) // 寻找 encoder 中是否有 x

if (encoder[i] ==x) {

flag = false；

break；

}

if (flag) { // 未找到 x 将其添加到末尾

encoder[k]= x；

++k；

}

}

//此时，encoder 中为 "C, S, P, A, B, D, E, F, G, H, I, J, K, L, M, N, O, Q, R, T, U, V, W, X, Y, Z"

for (int i = 0； i < 26； ++i)

decoder[encoder[i]- 'A'] = i + 'A'；

//此时，decoder 中为 "D, E, A, F, G, H, I, J, K, L, M, N, O, P, Q, C, R, S, B, T, U, V, W, X, Y, Z"

cin >> st；

for (int i = 0； i < st.length( )； ++i)

st[i] = decoder[st[i] -'A']；

cout << st；

return 0；

}

**判断题**

1.输入的字符串应当只由大写字母组成，否则在访问数组时可能越界； ( )

**分析**

若输入为小写字母，则字符减去’A’时，将会小于0，则越界；

**答案**

正确

2.若输入的字符串不是空串，则输入的字符串与输出的字符串一定不一样； ( )

**分析**

当输入T~Z时，输出为与输入一样；

**答案**

错误

3.将第 12 行的“i < 26”改为“i < 16”，程序运行结果不会改变； ( )

**分析**

由于只有C,S,P三个字母，P为第16个，所以只需枚举到16即可；

**答案**

正确

4.将第 26 行的"i < 26”改为“i < 16”，程序运行结果不会改变； ( )

**分析**

此时，encoder 中已有26个字符，需要枚举到26；

**答案**

错误

**单选题**

5.若输出的字符串为“ABCABCABCA”，则下列说法正确的是 ( ) ；

A. 输入的字符串中既有S又有P

B. 输入的字符串中既有S又有B

C. 输入的字符串中既有A又有P

D. 输入的字符串中既有A又有B

**分析**

A在decoder中出现在C位， B在decoder中出现在S位， C在decoder出现在P位，所以输入应为"CSPCSPCSPC"；

**答案**

A

6.若输出的字符串为“CSPCSPCSPCSP”，则下列说法正确的是 ( ) ；

A. 输入的字符串中既有P又有K

B. 输入的字符串中既有J又有R

C. 输入的字符串中既有J又有K

D. 输入的字符串中既有P又有R

**分析**

C在decoder中出现在P位， S在decoder中出现在R位， P在decoder中出现在N，所以输入应为“PRNPRNPRNP”

**答案**

D

第 2 题

#include <iostream>

using namespace std；

long long n, ans；

int k, len；

long long d[1000000]；

int main() {

cin >> n >> k；

d[0] = 0；

len= 1；

ans = 0；

for (long long i = 0； i <n； ++i) {

++d[0]；

for (int j = 0； j + 1<len； ++j) {

if (d[j] == k) {

d[j] = 0；

d[j + 1] += 1；

++ans；

}

}

if (d[len- 1] == k) {

d[len - 1] = 0；

d[len] =1；

++len；

++ans；

}

}

cout << ans << endl；

return 0；

}

假设输入的 n 是不超过 262的正整数，k都是不超过 10000 的正整数，完成下面的判断题和单选题

程序解释

#include <iostream>

using namespace std；

long long n, ans； // 将n转换为k进制数，len记录转换后的位数，ans记录进位次数；

int k, len；

long long d[1000000]；

int main() {

cin >> n >> k；

d[0] = 0；

len= 1；

ans = 0；

for (long long i = 0； i <n； ++i) { // n次循环，每次循环将d[0]增加1；

++d[0]；

for (int j = 0； j + 1<len； ++j) {

if (d[j] == k) { // 检查第0位到第len-2是否为k；

d[j] = 0； // 进行进位处理；

d[j + 1] += 1；

++ans；

}

}

if (d[len- 1] == k) { // 判断最高位是否需要向前进位；

d[len - 1] = 0；

d[len] =1；

++len；

++ans；

}

}

cout << ans << endl；

return 0；

}

**判断题**

1.若 k=1，则输出ans时，len=n； ( )

**分析**

k=1 时，len 只会在判断最高位进位时加一次；

**答案**

错误

2.若 k>1，则输出 ans 时，len —定小于 n； ( )

**分析**

k=2,n=1 时，len=1, 即 len=n ；

**答案**

错误

3.若 k>1，则输出ans时，klen—定大于n； ( )

**分析**

若k!=1,则n转化为len位的k进制，klen>n；

**答案**

正确

**单选题**

4.若输入的 n 等于1015，输入的 k 为 1，则输出等于 ( ) ；

A. 1 B. (1030-1015)/2 C.(1030+1015)/2 D. 1015

**分析**

k=1时，len 只会加一次,而 ans 会加 n 次，所以 ans 即为 n ；

**答案**

D

5.若输入的 n 等于205,891,132,094,649 (即330 )，输入的 k 为 3，则输出等于 ( ) ；

A. 330 B. (330-1 ) /2 C.330-1 D. (330+1 ) /2

**分析**

题目要计算的是将 n 转换为 3 进制，进位次数 ans ，其中 n 为 330 ；

先计算 3 3时的进位次数，寻找规律；

33 = ( 1000 ) 3 ，求进位次数，可以将进位情况分为以下几类

向最高位进位，有 1 种情况(222)3 ;

向次高位进位，有 3 种情况 ( 022 )3 , ( 122 ) 3 , ( 222 )3 ;

向第二位进位，有 9 种情况 ( 002 )3, ( 012 ) 3 , ( 022 ) 3 , ( 102 ) 3 , ( 112 )3 , ( 122 )3 , ( 202 )3 , ( 212 ) 3 , ( 222 ) 3 ;

换过程中总的进位次数 ans = 1 + 3 + 9 = 13；

则对3m的进位情况进行分类，每类的进位次数满足公比为 3 的等比数列，因此总的进位次数ans = 30 + 31 + 32 + ... + 3m- 1 =1× ​= ；

当输入n=km时，ans = ；

则输出为ans =  **=**  ；

答案

B

6.若输入的 n 等于 100,010,002,000,090，输入的 k 为 10，则输出等于 ( ) ；

A. 11,112,222,444,543 B. 11,122,222,444,453

C. 11,122,222,444,543 D. 11,112,222,444,453

分析

则1.累加到100 , 000 , 000 , 000 , 000 ，需要进位 = 11,111,111,111,111 次；

2.累加到10 , 000 , 000 , 000 ，需要进位 =1,111,111,111次；

3.累加到2 , 000 , 000 ，需要进位 2× = 2×111 , 111 = 222 , 222次；

4.累加到90，需要进位9× =9×1=1 次；

ans = 11,111,111,111,111 + 1,111,111,111 + 222,222 + 9 = 11,112,222,444,453 ；

答案

D

第 3 题

#include <algorithm>

#include <iostream>

using namespace std；

int n；

int d[50][2]；

int ans；

void dfs(int n, int sum) {

if (n == 1) {

ans = max(sum, ans)；

return；

}

for (int i = 1； i < n； ++i) {

int a = d[i - 1][0], b = d[i - 1][1]；

int x = d[i][0], y = d[i][1]；

d[i - 1][0] = a + x；

d[i - 1][1] = b + y；

for (int j = i； j < n - 1； ++j)

d[j][0] = d[j + 1][0], d[j][1] = d[j + 1][1]；

int s = a + x + abs(b - y)；

dfs(n - 1, sum + s)；

for (int j = n - 1； j > i； --j)

d[j][0] = d[j - 1][0], d[j][1] = d[j - 1][1]；

d[i - 1][0] = a, d[i - 1][1] = b；

d[i][0] = x, d[i][1] = y；

}

}

int main() {

cin >> n；

for (int i = 0； i < n； ++i)

cin >> d[i][0]；

for (int i = 0； i < n；++i)

cin >> d[i][1]；

ans = 0；

dfs(n, 0)；

cout << ans << endl；

return 0；

}

假设输入的n是不超过50的正整数，d[i][0]、d[i][i]都是不超过10000的正整数，完成下面的判断题和单选题

**程序解释**

程序输入了两个长度为n的序列A和B，分别保存到d[i][0]和d[i][1]；dfs求解A序列相邻项的和与B序列相邻项差的绝对值，求它们之和的最大值；

**判断题**

1.若输入 n 为 0,此程序可能会死循环或发生运行错误； ( )

**分析**

n 为 0 时，dfs 中 for 循环不会被执行，因此直接输出 0，不会发生死循环；

**答案**

错误

2.若输入 n 为 20,接下来的输入全为 0，则输出为 0； ( )

**分析**

当两个序列同时为 0，dfs 中 sum 也始终为 0，则输出为 0；

**答案**

正确

3.输出的数一定不小于输入的 d[i][0] 和 d[i][l] 的任意一个； ( )

**分析**

n = 1时，ans = 0，小于d[1][0] 和 d[1][1] ；

**答案**

错误

**单选题**

4.若输入的 n 为 20,接下来的输入是 20 个 9 和 20 个 0,则输出为 ( ) ；

A. 1890 B. 1881 C. 1908 D. 1917

**分析**

则对A序列进行迭代求和；

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | … | 20 |
| s | 0 | 9 + 9 = 18 | 18 + 9 = 27 | 27 + 9 = 36 | … | 20 \* 9 = 180 |
| ans | 0 | 0 + 18 = 18 | 18 + 27 = 45 | 45 + 36 = 81 | … | 2 \* 9 + 3 \* 9 + 4 \* 9 + … + 20 \* 9 |

ans= 9=1881

答案

B

5.若输入的 n 为 30,接下来的输入是 30 个 0 和 30 个 5,则输出为 ( ) ；

A. 2000 B. 2010 C. 2030 D. 2020

分析

则对B序列相邻项差进行迭代求和,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | … | 30 |
| s | 0 | 5 - 5 = 0 | 10 - 5 = 5 | 15 - 5 = 10 | … | 29 \* 5 - 5 = 140 |
| ans | 0 | 0 | 0 + 5 = 5 | 5 + 10 = 15 | … | 1 \* 5 + 2 \* 5 + 3 \* 5 + … + 28 \* 5 |

ans= 5=2030

答案

C

6.若输入的 n 为 15，接下来的输入是 15 到 1，以及 15到1，则输出为 ( ) ；

A. 2440 B. 2220 C. 2240 D. 2420

分析

A序列相邻项的和与B序列相邻项差的绝对值和的最大值；

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | … | 30 |
| a | 15 + 14 = 29 | 15 + 14 + 13 = 42 | 15 + 14 + 13 + 12 = 54 | 15 + 14 + 13 + 12 + 11 = 65 | … | 15 + 14 + 13 + 12 + 11 + 2 + 1| s | 15 + 14 + 15 - 14 = 30 | (15 + 14) + 13 + (15 + 14) - 13 = 6 | (15 + 14 + 13) + 12 + (15 + 14 + 13) - 12 = 84 | (15 + 14 + 13 + 12) + 11 + (15 + 14 + 13 + 12) - 11 = 130| … | (15 + 14 + … + 2) - 1 + (15 + 14 + … + 2) - 1 = 238| ans| 0 + 30 = 30 | 30 + 58 = 88 | 88 + 84 = 172| 172 + 108 = 280 | … | 15 \* 2 + (15 + 14) \* 2 + … + (15 + 14 + … + 2) \* 2| |

ans=15×2+(15+14)×2+...+(15+14+...+2)×2

=(15+29+42+54+65+75+84+92+99+105+110+114+117+119)×2

=1120×2

=2240

答案

C

三、完善程序

第 1 题

(质因数分解 ) 给出正整数 n，请输出将 n 质因数分解的结果，结果从小到大输出； 例如输入 n=120，程序应该输出 2 2 2 3 5，表示120 = 2 × 2 × 2 × 3 × 5；输入保证 2 ≤ n ≤ 1 09 ；提示先从小到大枚举变量 i，然后用 i 不停试除 n来寻找所有的质因子；

试补全程序；

#include <cstdio>

using namespace std；

int n, i；

int main() {

scanf("%d", &n)；

for(i = ①； ② <=n； i ++){

③{

printf("%d ", i)；

n = n / i；

}

}

if(④)

printf("%d ", ⑤)；

return 0；

}

①处应填 ( )

A. 1 B. n-1 C. 2 D. 0

**分析**

①处枚举正整数n的因数，从2开始枚举；

**答案**

C

②处应填 ( )

A. n/i B. n/(i\*i) C. i\*i D. i\*i\*i

分析

②处为正整数n因数的最大值，只需要枚举到n的平方根即可，所以为 i\*i<=n

**答案**

C

③处应填 ( )

A. if(n%i==0) B. if(i\*i<=n) C. while(n%i==0) D. while(i\*i<=n)

**分析**

③处为分解出n中所有的i，即一直除以i，直到分解完所有的i为止；

**答案**

C

④处应填 ( )

A. n>1 B. n<=1 C. i<n/i D. i+i<=n

**分析**

由于唯一分解定理，任何数大于 1 的正整数都可以表示为素数的积，所以当 n 还有因数，即$n>1 $时，输出 n 剩下的因数即可；

**答案**

A

⑤处应填 ( )

A. 2 B. n/i C. n D. i

分析

输出 n 剩下的因数,即为 n ；

答案

C

第 2 题

(最小区间覆盖 ) 给出 n 个区间，第 i 个区间的左右端点是 [ ai , bi ]；现在要在这些区间中选出若干个，使得区间 [0, m] 被所选区间的并覆盖 (即每一个 0≤i≤m 都在某个所选的区间中 ) ；保证答案存在，求所选区间个数的最小值；

输入第一行包含两个整数 n 和 m ( 1 ≤ n ≤ 5000 , 1 ≤ m ≤ 1 0 9 )，接下来 n 行，每行两个整数ai，bi ( 0 ≤ ai , bi ≤ m )；

提示使用贪心法解决这个问题；先用 O ( n2 ) 的时间复杂度排序，然后贪心选择这些区间；

试补全程序；

#include <iostream>

using namespace std；

const int MAXN = 5000；

int n, m；

struct segment { int a, b； } A[MAXN]；

void sort() // 排序

{

for (int i = 0； i < n； i++)

for (int j = 1； j < n； j++)

if (①)

{

segment t = A[j]；

②

}

}

int main()

{

cin >> n >> m；

for (int i = 0； i < n； i++)

cin >> A[i].a >> A[i]・b；

sort()；

int p = 1；

for (int i = 1； i < n； i++)

if (③)

A[p++] = A[i]；

n = p；

int ans =0, r = 0；

int q = 0；

while (r < m)

{

while (④)

q++；

⑤；

ans++；

}

cout << ans << endl；

return 0；

}

1.①处应填 ( )

A. A[j].b>A[j-1].b B. A[j].a<A[j-1].a C. A[j].a>A[j-1].a D. A[j].b<A[j-1].b

**分析**

冒泡排序时，按照左端点从小到大排；

**答案**

B

2.②处应填 ( )

A. A[j+1]=A[j]；A[j]=t； B. A[j-1]=A[j]；A[j]=t； C. A[j]=A[j+1]；A[j+1]=t； D. A[j]=A[j-1]；A[j-1]=t；

分析

交换两个数的位置；

答案

D

3.③处应填 ( )

A. A[i].b>A[p-1].b B. A[i].b<A[i-1].b C. A[i].b>A[i-1].b D. A[i].b<A[p-1].b

**分析**

此循环为去除重复段，则若当前区间右端点大于上一个区间右端点，就将其保留；

**答案**

A

4.④处应填 ( )

A. q+1<n&&A[q+1].a<=r B. q+1<n&&A[q+1].b<=r

C. q<n&&A[q].a<=r D. q<n&&A[q].b<=r

**分析**

从之前覆盖到的面积起，一直覆盖到r；

**答案**

A

5.⑤处应填 ( )

A. r=max(r,A[q+1].b) B. r=max(r,A[q].b)

C. r=max(r,A[q+1].a) D. q++

**分析**

取覆盖面积的最大右端点；

**答案**

B