2020CSP-S模拟赛1答案解析

一、单选

1、D 做法：四位二进制对应一位十六进制

2、D 离CPU越近，速度越快，寄存器是在CPU内部，其它都在外部。

寄存器>RAM>ROM>外存

3、D

解：假设这四只鸭子所在的点为a、b、c、d，这里我们不妨认为四只鸭子中，任意两只都不会处在一条直径上，这样并不影响我们计算概率。过这四个点做四条直径，这样我们可以得到8条半径（或者说是四对半径），不妨以其中一条半径开始，顺时针编号为1,2,3……7,8。如果a、b、c、d分别位于相邻的四条半径上，那么四只鸭子同一半圆。

由于a、b、c、d四个点分别位于四条不同的直径上，所以总共的情况有2^4=16种。其中四只鸭子位于同一半圆，a、b、c、d分别位于相邻的四条半径上的情况有8种（1,2,3,4和2,3,4,5和3,4,5,6和……一直到8,1,2,3共8种）。

所以概率是8/16，即1/2。

4、C

解：期望是O(n)的。先随便选一个数k，把数列分为左右两端，一边都小于k，一边都大于等于k，然后要么在左边要么在右边，每次期望缩小一半。所以复杂度就是n+n/2+n/4+n/8+…<2n

5、C

方法1：知道O(n)的建堆方法可直接得到答案。

方法2：递推式求解，可借助主定理快速求出。

6、C

每个点的度都为偶数<->无向连通图存在欧拉回路

7、D

在哈夫曼树（也叫最优树）中，只有两种类型的结点：度为0或N，即最优二叉树中只有度为0或2的结点，所以有2N-1个节点。

设n0表示叶子节点，n2表示度为2节点，n为节点总数，则有：

n0+n2=n

由树的性质，总边数为n-1

2n2=n-1

联立可得

2n0-1=n

8、B

9、D

10、C

分子可以计算有空车的方案。3\*2\*(C^{4}\_{5}+C^{5}\_{3})+3=93。答案为150/243=50/81

11、D

解：鸽巢原理

将任意连续11个数排成一圈：1 5 9 2 6 10 3 7 11 4 8，由题目条件知每相邻两个数至多有一个属于s，将这11个数按连续2个数为一组，分成6组：1 5，9 2，6 10，3 7，11 4，8，其中一组只有一个数8，若s含有这11个数至少6个，则必有两个数在同一组，与已知矛盾（我们考虑极端情况，每组取一个，第6组是8，那么可以发现不能取8，取8无论如何不能满足条件，因为取8就不能取1、4只能取5、11，取5、11就不能取9、7只能取2、3，取3就不能取10只能取6，取6就与取2矛盾，这样必然有一组取两数），所以s至多含有其中5个数。又因为2004=182\*11+2，所以s一共至多含有182\*5+2=912个元素。

12、D

13、D

14、B

15、C

第一个面向对象语言是smallstalk

二、阅读程序（以下共有三段程序。每段程序后面有6个题目，前4题为判断题，后2题为选择题。对于判断题，选择T代表正确，F代表错误；对于选择题，请在ABCD选项中选择正确的一项。判断题每题1分，选择题每题3分，共30分）

1、

1 #include<bits/stdc++.h>

2 using namespace std;

3 int arr[101];

4 int main() {

5 int n,gap;

6 cin>>n;

7 for(int i=1; i<=n; i++)

8 cin>>arr[i];

9 for (gap = n / 2; gap > 0; gap /= 2){

10 int k=0;

11 for (int i = gap; i <= n; i++){

12 for (int j= i - gap; j > 0 && arr[j] > arr[j + gap]; j -= gap){

13 swap(arr[j], arr[j + gap]);

14 k++;

15 }

16 }

17 cout<<k<<endl;

18 }

19 for(int i=1; i<=n; i++)

20 cout<<arr[i]<<" ";

21 return 0;

22 }

本题改编自shell排序

16、程序运行过程中，j一定小于等于n-gap。（ ） 是

17、若gap为1，20行输出结果不变。（ ） 是

gap为1实际上就是插入排序

18、任何时候，17行输出的k都小于n。（ ） 否

19、17行k值一定递减出现。（ ） 否

20、6个数，arr数组中数据为37 90 41 43 39 51，最后一次输出的k为（ ）。B

A．0 B．1 C．2 D．3

21、该程序时间复杂度最接近（ ）。 B

A．O(n) B．O(nlogn) C．O(n^2) D．O(n^2logn)

Shell排序没有稳定的时间复杂度，大约为n1.3

2、

1 struct Node {

2 ll d,id;

3 bool operator < (const Node &A) const {

4 return d > A.d);

5 }

6 };priority\_queue<Node> pq;

7 inline void f() {

8 for(int i=1; i<=n; i++) dis[i]=inf/3;

9 dis[1]=0,pq.push((Node) {0,1});

10 while(!pq.empty()) {

11 Node now=pq.top();

12 pq.pop();

13 int val=now.d,u=now.id;

14 if(dis[u]<val) continue;

15 for(int p=head[u]; p; p=nxt[p]) {

16 int v=a[p];

17 if(dis[u]+b[p]<dis[v]) {

18 dis[v]=dis[u]+b[p];

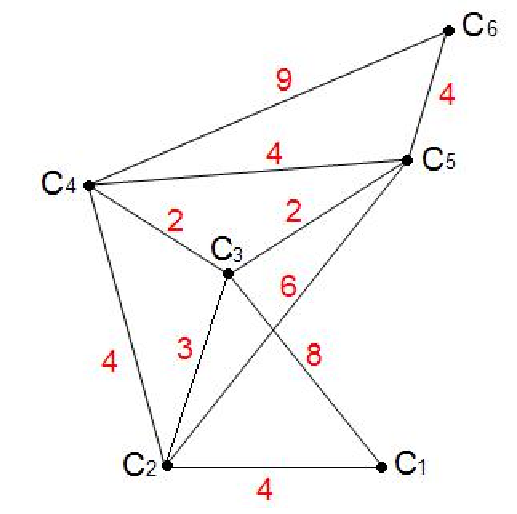
19 pq.push((Node) {dis[v],v});

20 }

21 }

22 }

23 }



注：本图仅与22、26题有关

本题是dijkstra算法堆优化非标准写法

22、对于上图，程序运行过程中，15行中的u一定不同。（ ） 是

正权情况如此

可证：一个点u只可能从堆中取出并松弛相联点一次

证明：反证，假设两次，那么dis[u]一定发生了变化（变小），因为如果没有变化，不会再次对点进行松弛操作。那么两次取出之间一定存在一个点v，使得dis[v]+g[u][v]<第一次取出时的dis[u]。但由于堆中每次取出的dis是单调不降的（可归纳证明），所以v不可能松弛u，矛盾。

23、对于任意输入，17行b[p]的值一定不能小于0。（ ） 否

此题出的不严谨，dij可以处理负权，不过复杂度会变成指数级，按道理程序应该有提前结束标记，本题程序中没有标记，似乎暗示不考虑负权，所以选“是”的同学也有道理。

24、11行每次pop出的now其val值一定单调不降。 （ ）是

25、堆中节点不会重复。 （ ）否

堆优化用stl实现，是没有删除节点操作的，具体的：堆顶弹出后，将这些关联点的dis放入堆中，新加入的dis显然比之前更小，这样会比原来的dis更早出队，原来的不会考虑相当于删除掉。

26、如上图，程序运行结束dis[6] C

A．8 B．9 C．13 D．17

27、该程序时间复杂度为（ ）。 C

A．O(nlogn) B．O((n+e)logn) C．O((n+e)loge) D．O(nloge)

可参见25题解析，堆包含很多相同的点，那么堆的大小是：

考虑相关联点的总数是度数之和，而度数之和是边\*2.

3、

如下代码用来计算字符串s的前缀函数。

定义前缀函数pi[i]=max(k=0-i){k:s[0…k-1]=s[i-(k-1)…i]}

简单来说pi[i]就是，子串s[0…i]最长的相等的真前缀与真后缀的长度。

例如：s[0…4]=ababc，pi[3]=2，pi[4]=0

规定：单个字符没有真前缀和真后缀，pi[0]=0

1 vector<int> prefix\_function(string s) {

2 int n = (int)s.length();

3 vector<int> pi(n);

4 for (int i = 1; i < n; i++) {

5 int j = pi[i - 1];

6 while (j > 0 && s[i] != s[j]) j = pi[j - 1];

7 if (s[i] == s[j]) j++;

8 pi[i] = j;

9 }

10 return pi;

11 }

本题代码来自OI-wiki 前缀函数优化版本

https://oi-wiki.org/string/kmp/

本程序相比n^3暴力优化点在两处：

1. 观察到相邻的前缀函数值至多增加1，因此字符串比较次数有上限
2. 考虑失配时，对于s[0…i]我们希望找到仅次于pi[i]的第二长度j，使得位置i的前缀性质仍能保留，可得j的状态转移方程

28、该算法是在线算法。（ ） 是

29、pi数组中元素值域为[0,n)。（ ） 否

30、6行中j一定单调递减。 （ ）是

由优化2

31、若第7行条件成立，第8行执行完时：一定有pi[j]=pi[j-1]+1。 （ ）是

32、若s为”abcabcd”，当i为5时，j的变化是：（ ） D

A．3 B．2->0->1 C．2->0->3 D．2->3

33、该程序时间复杂度为（ ）。 A

A．O(n) B．O(nlogn) C．O(nlog^2 n) D．O(n^2)

三、补充程序（本大题共含有2篇代码，共10小题，每小题4分，共40分。请在每道小题后所给的四条代码中选出最恰当的一项，使这段代码填入完整程序中对应的空缺处能符合题意。）

1 【Unique MST？】给定一棵无向连通图（点数n<=100），请你判断其最小生成树是否唯一。若不唯一，输出最小生成树大小，否则输出“Not Unique”。

核心算法：若MST不唯一，一定存在次小生成树的值与最小生成树相同，可统计在最小生成树添加新边后环上最大权值的边。

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF = 0x3f3f3f3f;

const int LEN = 101;

int Map[LEN][LEN];

int n, m, top, fa[LEN], dis[LEN][LEN], tag[LEN\*LEN], vis[LEN];

struct E {

int u, v, val;

} edge[LEN\*LEN];

bool cmp(E a, E b) {

return a.val < b.val;

}

void init() {

for(int i=0; i<LEN; i++) fa[i] = i;

}

int Find(int x) {

return fa[x] == x ? x : fa[x] = ① ;

}

int kruskal() {

int cnt = 0, ret = 0;

init();

memset(tag, 0, sizeof tag);

sort(edge, edge+m, cmp);

for(int i=0; i<m; i++) {

int pa = Find(edge[i].u), pb = Find(edge[i].v);

if(pa == pb) continue;

fa[pa] = pb;

cnt ++;

Map[edge[i].u][edge[i].v]=Map[edge[i].v][edge[i].u]=edge[i].val;

tag[i] = 1;

ret += edge[i].val;

if( ② ) break;

}

return ret;

}

void dfs(int bg, int v, int val) {

vis[v] = 1;

dis[bg][v] = val;

for(int i=1; i<=n; i++) {

if(Map[v][i] != INF && vis[i] == 0) dfs(bg, i, ③ );

}

}

int main() {

memset(Map, 0x3f, sizeof Map);

memset(dis, 0, sizeof dis);

scanf("%d%d", &n, &m);

for(int i=0; i<m; i++)

scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].val);

int ans = kruskal();

for(int i=1; i<=n; i++) {

memset(vis, 0, sizeof vis);

dfs(i, i, 0);

}

int f = 1;

for(int i=0; i<m; i++) {

if( ④ )continue;

if( ⑤ ) f = 0;

}

if(f)printf("%d\n", ans);

else printf("Not Unique!\n");

return 0;

}

34填入①处的代码是 B

A. fa[fa[x]]

B. Find(fa[x])

C. Find(Find(x))

D. 1

35填入②的代码是 C

A. cnt != n

B. cnt == 1

C. cnt == n-1

D. cnt == n

36.填入③处的代码是 A

A. max(val, Map[v][i])

B. max(val, Map[bg][i])

C. min(val, Map[v][i])

D. min(val, Map[bg][i])

37.填入④处的代码是 D

A. vis[i]

B. dis[i]

C. f==0

D. tag[i]

38.填入⑤处的代码分别是 C

A. edge[i].val == Map[edge[i].u][edge[i].v]

B. edge[i].val > Map[edge[i].u][edge[i].v]

C. edge[i].val == dis[edge[i].u][edge[i].v]

D. edge[i].val > dis[edge[i].u][edge[i].v]

2【好数】对于一个正整数X，如果把X化成二进制数后，如果X的二进制数至少有三个连续的1或者至少有3个连续的0（不能有前导0），那么X就是“好数”。例如8就是“好数”，因为8对应的二进制数是1000，有三个连续的0。整数15也是“好数”，因为15对应的二进制数是1111，也有三个连续的1。整数27就不是“好数”，因为27对应的二进制数是11011，既没有连续的三个1也没有连续三个0。

现在给出两个整数Low、UP，求Low和UP范围内有多少个“好数”。（0 <= Low <= UP <= 2147483647）

代码如下：

提示：F[i][j][k]表示从右往左数第i位的数字为j，第i-1位为k的“坏数”总数

核心算法：

好数难统计，正难则反，我们考虑指定范围内“坏数”的个数。

容易求0…0到11…1的坏数个数（设有i个1）：

F[i][1][0]+F[i][1][1]

因此若能求得0到up，求得0到low-1，相减即为low到up的坏数个数。

但是这里up和low并不等于11…11.

因此我们求出up的位数，假设为k

1、先求00…00到11…11（共k-1位）的坏数个数：

Sigma(F[i][1][0]+F[i][1][1])

2、再求100..0（k-1个0）到up的坏数个数：

方法是从左往右考虑除第一位之外第一个非0位，将它变成“0”.

例如10101010

除去第一位的1，第二位的1变0，就变成了100???，这样就是F[7][0][0]

第三位的1变0，就变成了10100???，这样就是F[5][0][0]

接着第四位1010100?，F[3][0][0]

三个加起来就是1000000-10101010中的坏数

注“中间过程由于‘1’变‘0’，可能会凑出三个连续0变“好数”，这种情况下的坏数个数不计入

最后，排除up和low本身是“好数“的情况

#include<bits/stdc++.h>

#define ll long long

using namespace std;

ll F[125][2][2];

ll Ans,Sum,len,k,n,m;

string sm;

ll Change(ll t) {

if(t<=0)return 0;

Ans=0,k=t,len=0;

sm="";

while(k) { //求t的二进制

char s=k%2+48;

sm=s+sm,k/=2;

}

len=sm.length();

for(int i=2; i<len; ++i) //求t的二进制长度-1位的坏数

Ans+= ① ;

if(t>0) ② ;

k=0;//求t的二进制长度位的坏数

while(++k<=len)

if(sm[k]=='1') {

sm[k]='0';

int l=1;

char r=sm[0];

bool B=0;

for(int i=1; i<=k; ++i)

if( ③ ) {

l++;

if(l==3) {

B=1;

break;

};

} else l=1,r=sm[i];

if(B==0)

Ans+= ④ ;

sm[k]='1';

}

int l=1;

char r=sm[1];

bool B=0;

for(int i=2; i<=len; ++i)

if( ⑤ ) {

l++;

if(l==3) {

B=1;

break;

};

} else l=1,r=sm[i];

if(!B)Ans++;

return t-Ans;

}

int main() {

scanf("%d%d",&n,&m);

F[2][1][0]=F[2][1][1]=F[2][0][0]=F[2][0][1]=1;//初值

for(int i=3; i<=31; ++i) {

F[i][0][0]=F[i-1][0][1];

F[i][1][0]= ⑥;

F[i][1][1]=F[i-1][1][0];

F[i][0][1]= ⑦;

}

Sum=Change(m)-Change(n-1);//前缀和

printf("%d",Sum);

return 0;

}

39.填入①处的代码是 C

A. F[i-1][1][0]+F[i-1][1][1]

B. F[i-1][0][1]+F[i-1][1][1]

C. F[i][1][0]+F[i][1][1]

D. F[i][0][1]+F[i][1][1]

40.填入②的代码是 B

A.len++

B. Ans++

C. F[0][0][0]

D. F[1][0][0]

41.填入③⑤处的代码是 A

A. sm[i]==r

B. sm[i]==’0’

C. sm[i]==’1’

D. sm[i]==sm[k]

42.填入④处的代码是 D

A. F[len-k][sm[k]-48][0]

B. F[len-k][sm[k-1]-48][0]

C. F[len-k+1][sm[k]-48][0]

D. F[len-k+1][sm[k-1]-48][0]

43.填入⑥和⑦处的代码分别是 D

A. F[i-1][1][1]；F[i-1][0][1]

B. F[i-1][0][1]；F[i-1][1][1]

C. F[i-1][1][1]+F[i-1][0][1]；F[i-1][0][1]+F[i-1][1][0]

D. F[i-1][0][0]+F[i-1][0][1]；F[i-1][1][1]+F[i-1][1][0]