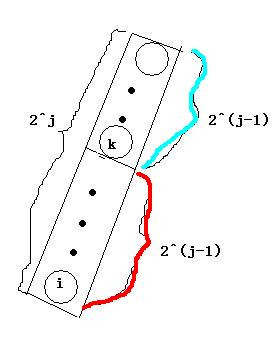
**v**

数据结构题。

对于每个点，分别求出其向上递增，递减，先增后减，先减后增的最长序列。

然后运用倍增思想，t[i][j]表示从结点i向上的长为2^i的序列中的答案。

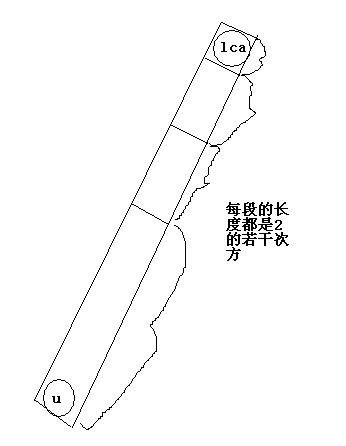
设k为i向上跳2^(j-1)步到达的点。

容易知道答案要么全部在红色区域内（答案为t[i][j-1])，要么全部在蓝色区域内（答案为t[k][j-1])，或者跨过红色区域和蓝色区域，这时必然经过k。

有4种情况：

k下方最长的递增序列长度+k上方最长的先增后减序列长度-1

k下方最长的递减序列长度+k上方最长的先减后增序列长度-1

k下方最长的先增后减序列长度+k上方最长的递减序列长度-1

k下方最长的先减后增序列长度+k上方最长的递增序列长度-1

对于询问u,v：

先求出lca。

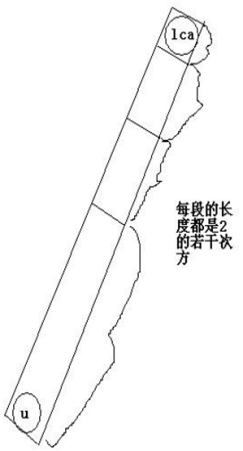
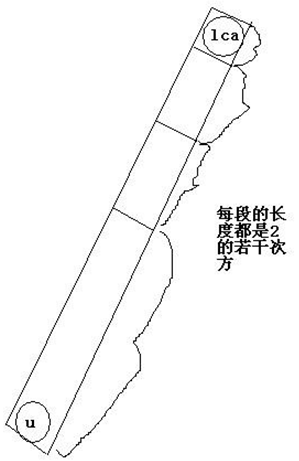
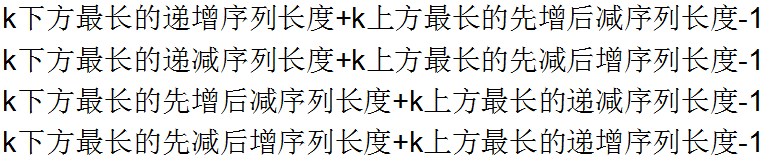
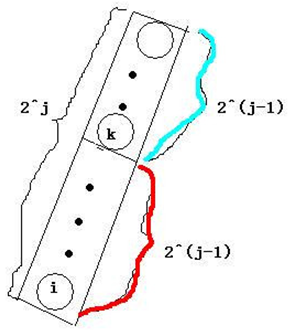
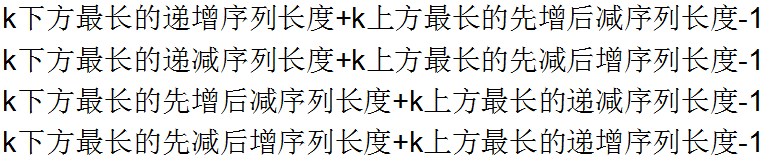
那么有2种情况，经过lca和不经过lca。

经过lca可以类似于上面的4种情况讨论。

不经过lca，就是从u到lca或者从v到lca，不失一般性，讨论从u到lca ：

我们把从u到lca的路径分成若干段，其中每段的长度都是2的若干次方，可以证明不超过logn段。

在每段内部，我们已经预处理好了，剩下的就是段与段之间的合并，还是利用上面的4种情况进行讨论。

#include<cstdio>  
#include<cstdlib>  
#include<iostream>  
#include<fstream>  
#include<algorithm>  
#include<cstring>  
#include<string>  
#include<cmath>  
#include<queue>  
#include<stack>  
#include<map>  
#include<utility>  
#include<set>  
#include<bitset>  
#include<vector>  
#include<functional>  
using namespace std;  
typedef long long LL;  
typedef double DB;  
typedef pair<int,int> PII;  
  
#define mmst(a,v) memset(a,v,sizeof(a))  
#define mmcy(a,b) memcpy(a,b,sizeof(a))  
#define re(i,a,b) for(i=a;i<=b;i++)  
#define red(i,a,b) for(i=a;i>=b;i--)  
#define fi first  
#define se second  
  
template<class T>inline T sqr(T x){return x\*x;}  
template<class T>inline void upmin(T &t,T tmp){if(t>tmp)t=tmp;}  
template<class T>inline void upmax(T &t,T tmp){if(t<tmp)t=tmp;}  
  
const DB EPS=1e-9;  
inline int dblcmp(DB x){if(abs(x)<EPS)return 0;return(x>0)?1:-1;}  
  
inline void SetOpen(string s)  
 {  
 freopen((s+".in").c\_str(),"r",stdin);  
 freopen((s+".out").c\_str(),"w",stdout);  
 }  
  
inline int Getin\_Int()  
 {  
 int res=0,flag=1;char z;  
 for(z=getchar();z!=EOF && z!='-' && !isdigit(z);z=getchar());  
 if(z==EOF)return 0;  
 if(z=='-'){flag=-flag;z=getchar();}  
 for(;z!=EOF && isdigit(z);res=res\*10+z-'0',z=getchar());  
 return res\*flag;  
 }  
inline LL Getin\_LL()  
 {  
 LL res=0,flag=1;char z;  
 for(z=getchar();z!=EOF && z!='-' && !isdigit(z);z=getchar());  
 if(z==EOF)return 0;  
 if(z=='-'){flag=-flag;z=getchar();}  
 for(;z!=EOF && isdigit(z);res=res\*10+z-'0',z=getchar());  
 return res\*flag;  
 }  
  
const int maxN=100000;  
  
int N,Q;  
int first[maxN+100],now;  
struct Tedge{int v,next;}edge[2\*maxN+100];  
int fa[maxN+100],dep[maxN+100];  
int jump[maxN+100][50];  
int p[maxN+100][4];//0上升 1下降 2先上后下 3先下后上   
int t[maxN+100][50];  
int ans;  
  
inline void addedge(int u,int v) //邻接表  
 {  
 now++;  
 edge[now].v=v;  
 edge[now].next=first[u];  
 first[u]=now;  
 }  
  
int que[maxN+100],head,tail;  
inline void BFS(int S) //计算各点的深度  
 {  
 dep[que[head=tail=0]=S]=1;  
 while(head<=tail)  
 {  
 int u=que[head++],i,v;  
 for(i=first[u],v=edge[i].v;i!=-1;i=edge[i].next,v=edge[i].v)dep[que[++tail]=v]=dep[u]+1;  
 }  
 }  
  
inline int swim(int x,int H) //爬到指定高度  
 {  
 for(int i=0;H!=0;H/=2,i++)if(H&1)x=jump[x][i];   
 return x;  
 }  
inline int Ask\_LCA(int x,int y)  
 {  
 if(dep[x]<dep[y]) swap(x,y);  
 x=swim(x,dep[x]-dep[y]);  
 if(x==y)return x;  
 int i;  
 red(i,30,0)if(jump[x][i]!=jump[y][i]){x=jump[x][i];y=jump[y][i];}  
 return jump[x][0];  
 }  
  
inline int up(int x,int y,int f)  
 {  
 return min(p[x][f],dep[x]-dep[y]+1);  
 }  
inline int down(int x,int y,int f)  
 {   
 int l=0,r=dep[x]-dep[y],mid;  
 while(l<=r)  
 {  
 mid=(l+r)/2;  
 int temp=swim(x,mid);  
 if(p[temp][f]>=dep[temp]-dep[y]+1) r=mid-1; else l=mid+1;  
 }  
 return dep[swim(x,l)]-dep[y]+1;  
 }  
  
inline int solve(int a,int b)  
 {  
 int i,res=0,H=dep[a]-dep[b]+1,x=a,y;  
 for(i=0;H!=0;H/=2,i++)if(H&1)  
 {  
 y=swim(x,(1<<i)-1);  
 upmax(res,t[x][i]);   
 if(a!=x)  
 {  
 upmax(res,down(a,x,0)+up(x,y,2)-1);  
 upmax(res,down(a,x,1)+up(x,y,3)-1);  
 upmax(res,down(a,x,2)+up(x,y,1)-1);  
 upmax(res,down(a,x,3)+up(x,y,0)-1);  
 }  
 x=fa[y];  
 }  
 return res;  
 }  
  
int main()  
 {  
 SetOpen("v");  
 int i,j;  
 N=Getin\_Int();  
 mmst(first,-1);now=-1;  
 re(i,2,N){fa[i]=Getin\_Int();addedge(fa[i],i);}  
 BFS(1);  
 re(i,0,30)jump[1][i]=1;  
 re(j,1,tail)  
 {  
 int u=que[j];  
 jump[u][0]=fa[u];  
 re(i,1,30)jump[u][i]=jump[jump[u][i-1]][i-1];  
 }  
 re(j,0,tail) //求出每个点向上递增，递减，先增后减，先减后增的最长序列。  
 {   
 int u=que[j];  
 if(u<fa[u])p[u][0]=p[fa[u]][0]+1; else p[u][0]=1;  
 if(u>fa[u])p[u][1]=p[fa[u]][1]+1; else p[u][1]=1;  
 p[u][2]=p[u][1];if(u<fa[u])p[u][2]=max(p[fa[u]][2],p[fa[u]][1])+1;  
 p[u][3]=p[u][0];if(u>fa[u])p[u][3]=max(p[fa[u]][3],p[fa[u]][0])+1;  
 }  
 re(j,0,tail)  
 {  
 int a=que[j],c,d;  
 t[a][0]=1;  
 re(i,1,30)  
 {  
 c=jump[a][i-1];  
 d=swim(c,dep[a]-dep[c]-1);  
 t[a][i]=max(t[a][i-1],t[c][i-1]);  
 upmax(t[a][i],down(a,c,0)+up(c,d,2)-1);  
 upmax(t[a][i],down(a,c,1)+up(c,d,3)-1);   
 upmax(t[a][i],down(a,c,2)+up(c,d,1)-1);   
 upmax(t[a][i],down(a,c,3)+up(c,d,0)-1);  
 }  
 }  
 ans=0;  
 Q=Getin\_Int();  
 while(Q--)  
 {  
 int u=Getin\_Int()^ans,v=Getin\_Int()^ans,lca;  
 ans=0;  
 lca=Ask\_LCA(u,v);  
 upmax(ans,solve(u,lca));  
 upmax(ans,solve(v,lca));  
 upmax(ans,down(u,lca,0)+down(v,lca,2)-1);  
 upmax(ans,down(u,lca,1)+down(v,lca,3)-1);  
 upmax(ans,down(u,lca,2)+down(v,lca,0)-1);  
 upmax(ans,down(u,lca,3)+down(v,lca,1)-1);   
 printf("%d\n",ans);  
 }   
 return 0;  
 }