RMQ 问题及解决算法

> By Billchenchina
>
> The newest version can be found here

RMQ 问题

RMQ 问题，即区间最值查询，是在长度为 N 的序列中求出其连续的子序列中最大/最小值的问题。

ST 算法

算法介绍

ST 算法适用于解决 RMQ 问题，是一个较长时间预处理，（时间复杂度为 O(NlogN)），在 O(1) 的时间内回答每个查询的算法。

算法思想及类型

算法思想为`人人为我 我为人人`，本质为动态规划。

实现顺序为：

1. 区间大小为单位大小，计算出每个单位中最大/最小值（即为本身）
2. 逐渐增大区间大小（每次乘二），利用其两个子连续区间动态规划出这次计算的区间的最大/最小值

这里`M[i][j]`的最值即为`[i,i+1-(2^j)]`的最值，也就是`[i,i+1-(2^(j-1))]`的最值和`[i+(2^(j-1)),i+1-(2^j)]`最值的max/min，所以可以这么DP

代码实现

（from TopCoder）

```
void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N) {
    int i, j;
    // initialize M for the intervals with length 1
    // 将每个单位对应的最大/最小值都设为自身
    // 为上图的 1,2,3,4 的处理过程
    for (i = 0; i < N; i++)
        M[i][0] = i;
    // compute values from smaller to bigger intervals
```

```
    // 开始DP
    for (j = 1; 1 << j <= N; j++)
        for (i = 0; i + (1 << j) - 1 < N; i++)
            if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])
                M[i][j] = M[i][j - 1];
            else
                M[i][j] = M[i + (1 << (j - 1))][j - 1];
}
```

参数

int **M: 二维数组，用于接收返回结果。其中第一维为从 i 个数字开始，第二维为表示 [i,i+2^j]的区间，数组值为最大/最小值在A数组中的位置。

int *A: 一位数组，为原数组。

int N: A的长度

求区间最值（O(1)）

当求 [a,b] 区间最值时，应找到两个长度为2^k的重叠区间，使得第一个区间的初始位置为 a，第二个区间的结尾位置为 b。那么第一个区间的最值即为 `M[a][k]` ，第二个区间最值为 `M[b+1-2^k][k]` ，再求出 `ans=max(max1,max2)` （或 `ans=min(min1,min2)` ）

需要注意的是，上面求的 M[a][k] 和 M[d][k] 虽然有可能有重叠部分，但是由于查询时间复杂度为 常数，这里可以忽略

例题

POJ 3264

> 这道题是个奇葩题？
>
> 在 OpenJudge上的 这道题 线段树可过，POJ 却过不了
>
> 几乎就是模板题？（大雾

Luogu 3379

这个题是树的题，需要从根节点把dfs序记录下来并转成欧拉序，对欧拉序做区间最小值查询即为最近公共祖先。

> err...不太友好的一点是卡 `vector` ，所以我换成链式前向星了
>
> 啥？链式前向星也卡？链式前向星+快速妥妥的A

```cpp
#include <bits/stdc++.h>
using namespace std;

#define maxn 1000001

struct edge
{
    int to;
    int next;
};
edge edges[maxn];
int edges_size=0;
int first[maxn];

int N,M,root;

int dfs_list[maxn];
int dfs_list_size=0;
int dfn[maxn];
bool vis[maxn];
int first_pos[maxn];
int defaultdfn=1;
int _M[maxn][21];
int another_dfn[maxn];
void preprocess();
void process();
int getmin(int a,int b);
void connect();
void dfs(int i);
// 快速读入
int read()
{
    int x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return x*f;
}

int main()
{
    memset(first,-1,sizeof first);
    N=read();M=read();root=read();
    //cin>>N>>M>>root;
    for(int i=0; i<N-1; ++i)
    {
        connect();
    }
    dfs(root);

    preprocess();
    for(int i=0; i<M; ++i)
    {
        process();
    }
}



void connect()
{
```

```
    int x,y;
    x=read();y=read();
    //cin>>x>>y;
    edges[edges_size].to=y;
    edges[edges_size].next=first[x];
    first[x]=edges_size++;
    edges[edges_size].to=x;
    edges[edges_size].next=first[y];
    first[y]=edges_size++;
}



void dfs(int i)
{
    vis[i]=1;
    dfn[i]=defaultdfn++;
    another_dfn[dfn[i]]=i;
    dfs_list[dfs_list_size++]=dfn[i];
    first_pos[dfn[i]]=dfs_list_size-1;
    for(int j=first[i]; j!=-1; j=edges[j].next)
    {
        int nextp=edges[j].to;
        if(!vis[nextp])
        {
            dfs(nextp);
            dfs_list[dfs_list_size++]=dfn[i];
        }
    }
}

void process()
{
    int x,y;
    //cin>>x>>y;
    x=read();y=read();
    x=dfn[x];
    y=dfn[y];
    if(x>y)
    {
        swap(x,y);
    }
    x=first_pos[x];y=first_pos[y];
    printf("%d\n",another_dfn[getmin(x,y)]);
    //cout<<another_dfn[getmin(x,y)]<<endl;
}

int getmin(int a,int b)
{
    int k=log2(b-a+1);
    if((b-a+1)&(b-a))
    {
        int min1=dfs_list[_M[a][k]];
        int min2=dfs_list[_M[b+1-(1<<(k))][k]];
        return min1<min2?min1:min2;
    }
    else
    {
        return dfs_list[_M[a][k]];
    }
}
```

```
void preprocess() {
    int *A=dfs_list;
    int N=dfs_list_size;
    int i, j;
    for (i = 0; i < N; i++)
        _M[i][0] = i;
    for (j = 1; 1 << j <= N; j++)
        for (i = 0; i + (1 << j) - 1 < N; i++)
            if (A[_M[i][j - 1]] < A[_M[i + (1 << (j - 1))][j - 1]])
                _M[i][j] = _M[i][j - 1];
            else
                _M[i][j] = _M[i + (1 << (j - 1))][j - 1];
}
```