

# 笛卡尔树

## 算法简介

一种二叉树，主要用于解决有关序列  $\text{RMQ}$  问题和直方图问题，建树时间复杂度  $O(n)$

## 算法思想

笛卡尔树具有如下三条性质：

1. 对笛卡尔树中序遍历可以得到原序列
2. 笛卡尔树的点权符合堆性质
3. 任意区间的  $\text{RMQ}$  操作可以转换为笛卡尔树上两端点的  $\text{LCA}$  操作

事实上，笛卡尔树可以对任意二元组进行建树，对第一关键字满足二叉搜索树性质，对第二关键字满足堆性质。

从上面性质也可以看出，笛卡尔树实际上就是一棵特殊的  $\text{Treap}$

接下来考虑建树，事实上可以按顺序插入序列，每次插入时由于插入结点下标最大，所以只能不停跳右子树。

找到合适结点，取代该结点位置，并将该结点所在子树变成插入结点的左子树。

发现只需要用一个单调栈维护一下从根结点起沿途所有右儿子构成的树链即可，时间复杂度  $O(n)$

## 代码模板

[洛谷p5854](#)

```
const int MAXN=1e7+5,Inf=2e9;
int Stack[MAXN],v[MAXN],ch[MAXN][2],root;
void build(int n){//小根堆笛卡尔树
    int top=0;
    v[0]=-Inf;Stack[++top]=0;
    _rep(i,1,n){
        while(v[i]<v[Stack[top]])top--;
        ch[i][0]=ch[Stack[top]][1];
        ch[Stack[top]][1]=i;
        Stack[++top]=i;
    }
    root=Stack[2];
}
```

# 算法练习

## 习题一

### SP3734

#### 题意

给定一个  $N$  列的表格，每列的高度为  $a_i$  且底部对齐。

要求向表格中填入  $k$  个相同的数，填写时要求不能有两个数在同一列，或在同一行且中间表格不间断。

问在取模意义下有多少种填法。

#### 题解 1

建立根据表格高度建立笛卡尔树，将表格划分成  $N$  个矩形。

设  $f(i,j)$  表示在以结点  $i$  为根结点的子树所代表的区域中填写  $j$  个数的填法种数， $H$  为结点代表矩阵高度， $sz$  代表结点子树大小。

$g(i,j)$  表示在以结点  $i$  为根结点的子树所代表的区域中填写  $j$  个数且不在根结点代表的矩形填写数字的填法种数。

先在考虑从某个结点所代表的矩阵中选取  $k$  个结点的选法。

易知从  $n \times m$  的矩阵中选取  $k$  个结点的总数为  $k! C_n^k C_m^k$  预处理  $k!$  和  $k!$  的逆即可快速求出。

容易得出状态转移方程

$$H = H_u - H_{fa}, sz_u = 1 + sz_{lson} + sz_{rson}$$

$$g(u,i) = \sum_{j=0}^i f(lson,j) \times f(rson,i-j)$$

$$f(u,i) = \sum_{j=0}^i g(u,j) \times (i-j)! C_H^{i-j} C_{sz_u-j}^{i-j}$$

边界条件为  $f(0,0) = g(0,0) = 1$  时间复杂度  $O(nk^2)$

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
```

```

#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=505,MAXH=1e6+5,mod=1e9+7;
LL frac[MAXN],inv[MAXN];
LL quick_pow(LL a,LL b,LL mod){
    LL t=1;
    while(b){
        if(b&1)
            t=t*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return t%mod;
}

```

```
void get_inv(){
    frac[0]=1;
    _for(i,1,MAXH)
        frac[i]=frac[i-1]*i%mod;
    inv[MAXH-1]=quick_pow(frac[MAXH-1],mod-2,mod);
    for(int i=MAXH-2;i>=0;i--)
        inv[i]=inv[i+1]*(i+1)%mod;
}
int C(int n,int m){return frac[n]*inv[m]%mod*inv[n-m]%mod;}
int cal(int r,int c,int k){
    if(r<k||c<k)
        return 0;
    return 1LL*C(r,k)*C(c,k)%mod*frac[k]%mod;
}
int Stack[MAXN],v[MAXN],ch[MAXN][2],root;
void build(int n){
    int top=0,last=0;
    _rep(i,1,n){
        while(top&&v[i]<v[Stack[top]])top--;
        if(top)ch[Stack[top]][1]=i;
        if(top<last)ch[i][0]=Stack[top+1];
        Stack[++top]=i;
        last=top;
    }
    root=Stack[1];
}
int sz[MAXN],n,k;
LL f[MAXN][MAXN],g[MAXN][MAXN];
void dfs(int pos,int fa){
    if(!pos)
        return;
    dfs(ch[pos][0],pos);
    dfs(ch[pos][1],pos);
    sz[pos]=sz[ch[pos][0]]+sz[ch[pos][1]]+1;
    _rep(i,0,k)
        _rep(j,0,i)
            g[pos][i]=(g[pos][i]+f[ch[pos][0]][j]*f[ch[pos][1]][i-j])%mod;
    int H=v[pos]-v[fa];
    _rep(i,0,k)
        _rep(j,0,i){
            if(!g[pos][j])
                break;
            f[pos][i]=(f[pos][i]+g[pos][j]*cal(H,sz[pos]-j,i-j))%mod;
        }
}
int main()
{
    get_inv();
    f[0][0]=g[0][0]=1;
    n=read_int(),k=read_int();
```

```

    _rep(i,1,n)
    v[i]=read_int();
    build(n);
    dfs(root,0);
    enter(f[root][k]);
    return 0;
}

```

## 题解2

另外，还可以用笛卡尔树+树上背包解决上述问题，常数较小，其中  $\text{dp}$  数组第一次转移表示  $g$  第二次转移表示  $f$

```

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n' ||c=='\r')c=getchar();
}

```

```
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=505,MAXH=1e6+5,mod=1e9+7;
LL frac[MAXN],inv[MAXN];
LL quick_pow(LL a,LL b,LL mod){
    LL t=1;
    while(b){
        if(b&1)
            t=t*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return t%mod;
}
void get_inv(){
    frac[0]=1;
    _for(i,1,MAXH)
        frac[i]=frac[i-1]*i%mod;
    inv[MAXH-1]=quick_pow(frac[MAXH-1],mod-2,mod);
    for(int i=MAXH-2;i>=0;i--)
        inv[i]=inv[i+1]*(i+1)%mod;
}
int C(int n,int m){return frac[n]*inv[m]%mod*inv[n-m]%mod;}
int Stack[MAXN],v[MAXN],ch[MAXN][2],root;
void build(int n){//小根堆的笛卡尔树
    int top=0,last=0;
    _rep(i,1,n){
        while(top&&v[i]<v[Stack[top]])top--;
        if(top)ch[Stack[top]][1]=i;
        if(top<last)ch[i][0]=Stack[top+1];
        Stack[++top]=i;
        last=top;
    }
    root=Stack[1];
}
int sz[MAXN],n,k;
LL dp[MAXN][MAXN];
void dfs(int pos,int fa){
    if(!pos)
        return;
    dfs(ch[pos][0],pos);
```

```

dfs(ch[pos][1],pos);
sz[pos]=1,dp[pos][0]=1;
_for(dir,0,2){//计算从子树中选取i个结点的背包总数
    if(!ch[pos][dir])continue;
    sz[pos]+=sz[ch[pos][dir]];
    for(int i=min(k,sz[pos]);i;i--)
        _rep(j,1,min(sz[ch[pos][dir]],i))
            dp[pos][i]=(dp[pos][i]+dp[pos][i-j]*dp[ch[pos][dir]][j])%mod;
}
int H=v[pos]-v[fa],Inf=min(k,sz[pos]);
for(int i=Inf-1;i>=0;i--)//计算从子树中选取i个结点,从当前矩阵中选取j个结点的贡献
    _rep(j,1,min(Inf-i,H))
        dp[pos][i+j]=(dp[pos][i+j]+dp[pos][i]*C(H,j)%mod*C(sz[pos]-
i,j)%mod*frac[j]%mod)%mod;
}
int main()
{
    get_inv();
    n=read_int(),k=read_int();
    _rep(i,1,n)
        v[i]=read_int();
    build(n);
    dfs(root,0);
    enter(dp[root][k]);
    return 0;
}

```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: [https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E7%AC%9B%E5%8D%A1%E5%B0%94%E6%A0%91&rev=1595602574](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E7%AC%9B%E5%8D%A1%E5%B0%94%E6%A0%91&rev=1595602574)

Last update: 2020/07/24 22:56