Algorithm Library

mental2008

December 6, 2018

Contents

1 图论

1.1 网络流

1.1.1 Dinic

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=2005;
const int INF=(1<<30)-1;
int n,m,s,t; //结点数，边数（包括反向弧），源点编号和汇点编号
struct Edge {
    int from,to,cap,flow;
    Edge(int _from,int _to,int _cap,int _flow) {
        from=_from,to=_to,cap=_cap,flow=_flow;
    }
};
vector<Edge> edges;  //边表，edges[e] 和 edges[e^1] 互为反向弧
vector<int> G[maxn]; //邻接表，G[i][j] 表示结点 i 的第 j 条边在
↪  e 数组中的序号
bool vis[maxn]; //BFS 使用
int d[maxn]; //从起点到 i 的距离
int cur[maxn]; //当前弧下标
void init() {
    for(int i=0;i<maxn;++i) G[i].clear();
    edges.clear();
}
void AddEdge(int from,int to,int cap) {
    edges.push_back(Edge(from,to,cap,0));
    edges.push_back(Edge(to,from,0,0));
    int k=edges.size();
    G[from].push_back(k-2);
    G[to].push_back(k-1);
}
bool BFS() {
    memset(vis,0,sizeof(vis));
    queue<int> Q;
    Q.push(s);
    d[s]=0;
    vis[s]=1;
    while(!Q.empty()) {
        int x=Q.front();
        Q.pop();
```

```
        for(int i=0;i<G[x].size();++i) {
            Edge e=edges[G[x][i]];
            if(!vis[e.to]&&e.cap>e.flow) { //只考虑残量网络中的
            ↪ 弧
                vis[e.to]=1;
                d[e.to]=d[x]+1;
                Q.push(e.to);
            }
        }
    }
    return vis[t];
}
int DFS(int x,int a) {
    if(x==t||a==0) return a;
    int flow=0,f;
    for(int& i=cur[x];i<G[x].size();++i) { //从上次考虑的弧
        Edge& e=edges[G[x][i]];

        ↪ if(d[x]+1==d[e.to]&&(f=DFS(e.to,min(a,e.cap-e.flow)))>0)
        ↪ {
            e.flow+=f;
            edges[G[x][i]^1].flow-=f;
            flow+=f;
            a-=f;
            if(a==0) break;
        }
    }
    return flow;
}
int Maxflow(int u,int v) {
    s=u,t=v;
    int flow=0;
    while(BFS()) {
        memset(cur,0,sizeof(cur));
        flow+=DFS(u,INF);
    }
    return flow;
}
```

1.1.2 ISAP

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=2005;
```

```
const int INF=(1<<30)-1;
// ISAP 中的 n 非常重要，必须准确的定义为结点数，如有必要，建议将
↪  ISAP 算法封装成一个结构体
int n,m,s,t; //结点数，边数（包括反向弧），源点编号和汇点编号
struct Edge {
    int from,to,cap,flow;
    Edge(int _from,int _to,int _cap,int _flow) {
        from=_from,to=_to,cap=_cap,flow=_flow;
    }
};
vector<Edge> edges;
vector<int> G[maxn];
int p[maxn];
int d[maxn];
int num[maxn];
bool vis[maxn];
int cur[maxn];
void init() {
    for(int i=0;i<maxn;++i) G[i].clear();
    edges.clear();
}
void AddEdge(int from,int to,int cap) {
    edges.push_back(Edge(from,to,cap,0));
    edges.push_back(Edge(to,from,0,0));
    int k=edges.size();
    G[from].push_back(k-2);
    G[to].push_back(k-1);
}
void BFS() {
    memset(vis,0,sizeof(vis));
    queue<int> Q;
    Q.push(t);
    vis[t]=1;
    d[t]=0;
    while(!Q.empty()) {
        int x=Q.front();
        Q.pop();
        for(int i=0;i<G[x].size();++i) {
            Edge e=edges[G[x][i]^1];
            if(!vis[e.from]&&e.cap>e.flow) {
                d[e.from]=d[x]+1;
                vis[e.from]=1;
                Q.push(e.from);
```

```
            }
        }
    }
}
int Augment() {
    int x=t,a=INF;
    while(x!=s) {
        Edge e=edges[p[x]];
        a=min(a,e.cap-e.flow);
        x=edges[p[x]].from;
    }
    x=t;
    while(x!=s) {
        edges[p[x]].flow+=a;
        edges[p[x]^1].flow-=a;
        x=edges[p[x]].from;
    }
    return a;
}
int Maxflow(int u,int v) {
    s=u,t=v;
    int flow=0;
    BFS();
    memset(num,0,sizeof(num));
    for(int i=0;i<=n;++i) num[d[i]]++;
    int x=s;
    memset(cur,0,sizeof(cur));
    while(d[s]<n) {
        if(x==t) {
            flow+=Augment();
            x=s;
        }
        int ok=0;
        for(int i=cur[x];i<G[x].size();++i) {
            Edge& e=edges[G[x][i]];
            if(e.cap>e.flow&&d[x]==d[e.to]+1) {
                ok=1;
                p[e.to]=G[x][i];
                cur[x]=i;
                x=e.to;
                break;
            }
        }
        if(!ok) {
```

```
            int k=n-1;
            for(int i=0;i<G[x].size();++i) {
                Edge& e=edges[G[x][i]];
                if(e.cap>e.flow) k=min(k,d[e.to]);
            }
            if(--num[d[x]]==0) break;
            num[d[x]=k+1]++;
            cur[x]=0;
            if(x!=s) x=edges[p[x]].from;
        }
    }
    return flow;
}
```

2 字符串

3 数学

3.1 阶乘逆元

```
/*
 * fact[i], i 的阶乘
 * fiv[i], i 的阶乘的逆元
 * inv[i], i 的逆元
 */
typedef long long ll;
const int maxn=1e5+5;
const int mod=1e9+7;
ll fact[maxn];
ll fiv[maxn];
ll inv[maxn];
void init() {
    fact[0]=fact[1]=1;
    fiv[0]=fiv[1]=1;
    inv[1]=1;
    for(int i=2;i<maxn;++i) {
        fact[i]=fact[i-1]*i%mod;
        inv[i]=inv[mod%i]*(mod-mod/i)%mod;
        fiv[i]=fiv[i-1]*inv[i]%mod;
    }
}
```

3.2 组合数

```
ll C(ll m,ll k) {
    if(m<k||k<0) return 0;
    return (fact[m]*fiv[k]%mod)*fiv[m-k]%mod;
}
```

3.3 卡特兰数

卡特兰数是一种经典的组合数，经常出现在各种计算中，其前几项为：1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

卡特兰数满足以下性质：

令 $h(0)=1, h(1)=1$,catalan 数满足递推式。$h(n)=h(0)*h(n-1)+h(1)*h(n-2)+...+h(n-1)*h(0)$。如果能将公式化为上面这种类型，就是卡特兰数。

通项公式：$h(n)=\binom{2n}{n}-\binom{2n}{n+1}$，或者 $h(n)=\frac{1}{n+1}\binom{2n}{n}$。

经典问题：出栈次序、二叉树构成问题、凸多边形的三角形划分。

4 黑科技

4.1 输入输出外挂

4.1.1 简单快读

```
/*
 * 适用于正负整数 (包括 int,long long 与 __int128)
 */
template<typename T>
inline bool Read(T &r) {
        char c;
        int sgn;
        if(c=getchar(),c==EOF) return 0; // EOF
        while(c!='-'&&(c<'0'||c>'9')) c=getchar();
        sgn=(c=='-')?-1:1;
        r=(c=='-')?0:(c-'0');
        while(c=getchar(),c>='0'&&c<='9') r=r*10+(c-'0');
        return r*sgn;
}
template<typename T>
inline void out(T x) {
        if(x>9) out(x/10);
        putchar(x%10+'0');
}
```

4.1.2 真・快速读入

```
/*
 * 读入时候这样写:
 * int x;
 * FastIO::Read(x);
 *
 * 若要处理到文件末尾可以这样写：
 * while(FastIO::Read(x), FastIO::IOError == 0);
 */
#include<cstdio>
namespace FastIO {
        #define BUF_SIZE 10000000 //缓冲区大小可修改
        bool IOError = 0; //IOError == false 时表示处理到文件结
        ↪ 尾
        inline char NextChar() {
                static char buf[BUF_SIZE], *p1 = buf +
                ↪ BUF_SIZE, *pend = buf + BUF_SIZE;
                if(p1 == pend) {
```

```
                        p1 = buf;
                        pend = buf + fread(buf, 1, BUF_SIZE,
                        ↪  stdin);
                        if(pend == p1) {
                                IOError = 1;
                                return -1;
                        }
                }
                return *p1++;
        }
        inline bool Blank(char c) {
                return c == ' ' || c == '\n' || c == '\r' || c
                ↪  == '\t';
        }

        template<class T> inline void Read(T &x) {
                char c;
                while(Blank(c = NextChar()));
                if(!IOError) {
                        for(x = 0; '0' <= c && c <= '9'; c =
                        ↪  NextChar())
                                x = (x << 3) + (x << 1) + c -
                                ↪  '0';
                }
        }
}
```

4.2 pbds 库

4.2.1 Hash

```
/*
 * 头文件：
 * #include<ext/pb_ds/assoc_container.hpp>
 * #include<ext/pb_ds/hash_policy.hpp>
 * 用法：
 * cc_hash_table 是拉链法
 * gp_hash_table 是查探法
 * 除了当数组用外，还支持 find 和 operator[]
 * 例如：__gnu_pbds::gp_hash_table<int,bool> h;
 */
#include<cstdio>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
```

```
__gnu_pbds::gp_hash_table<int,bool> h;

int main() {
    int n,m,k;
    scanf("%d%d",&n,&m);
    h.clear();
    while(n--) {
        scanf("%d",&k);
        h[k]=true;
    }
    while(m--) {
        scanf("%d",&k);
        puts(h[k]?"YES":"NO");
    }
    return 0;
}
```

4.2.2 可并堆

```
#include<bits/stdc++.h>
#include<ext/pb_ds/priority_queue.hpp>
using namespace std;
__gnu_pbds::priority_queue<int,greater<int> > a,b; //小的优先
__gnu_pbds::priority_queue<int> c; //大的优先
int main() {
    a.push(2);a.push(5);
    b.push(4);b.push(1);
    a.join(b);
    cout<<a.top()<<"\n";
    return 0;
}
// output: 1
```

4.2.3 Rope

```
/*
 * rope test;
 * test.push_back(x); //在末尾添加 x
 * test.insert(pos,x); //在 pos 位置插入 x
 * test.erase(pos,x); //从 pos 开始删除 x 个
 * test.copy(pos,len,x); //从 pos 开始到 pos+len 为止用 x 代替
 * test.replace(pos,x); //从 pos 开始换乘 x
 * test.substr(pos,x); //从 pos 开始提取 x 个
 * test.at(x); //访问第 x 个元素
```

```
 * test[x]; 访问第 x 个元素
 * 算法复杂度为 n^(3/2), 可以在很短的时间内实现快速的插入, 删除和
↪  查找字符串, 是一个很厉害的神器！
 */
#include<bits/stdc++.h>
#include<ext/rope>
using namespace std;
using namespace __gnu_cxx;
rope<int> a;
int main() {
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;++i) a.push_back(i);
    while(m--) {
        int pos,s;
        cin>>pos>>s;
        pos--;

        ↪  a=a.substr(pos,s)+a.substr(0,pos)+a.substr(pos+s,n-pos-s);
    }
    for(int i=0;i<n;++i) cout<<a[i]<<" ";
    cout<<"\n";
    return 0;
}
```

4.3 FWT

例子：$C_k = \sum_{i|j=k} A_i \times B_j, C_k = \sum_{i\&j=k} A_i \times B_j, C_k = \sum_{i\oplus j=k} A_i \times B_j$

```
/*
 * FWT(快速沃尔什变换)——解决多项式的位运算卷积
 * 能不取模尽量不取
 */
#include<bits/stdc++.h>

typedef long long ll;
const ll mod=1e9+7;
const ll inv2=(mod+1)>>1;
struct FWT {
    int N;
    void init(int n) {
        N=1;
        while(N<n) N<<=1;
    }
```

```
    void FWT_or(ll *a,int opt) {
        for(int i=1;i<N;i<<=1) {
            for(int p=i<<1,j=0;j<N;j+=p) {
                for(int k=0;k<i;++k) {
                    if(opt==1)
                        a[i+j+k]=(a[j+k]+a[i+j+k])%mod;
                    else
                        a[i+j+k]=(a[i+j+k]+mod-a[j+k])%mod;
                }
            }
        }
    }
    void FWT_and(ll *a,int opt) {
        for(int i=1;i<N;i<<=1) {
            for(int p=i<<1,j=0;j<N;j+=p) {
                for(int k=0;k<i;++k) {
                    if(opt==1)
                        a[j+k]=(a[j+k]+a[i+j+k])%mod;
                    else
                        a[j+k]=(a[j+k]+mod-a[i+j+k])%mod;
                }
            }
        }
    }
    void FWT_xor(ll *a,int opt) {
        for(int i=1;i<N;i<<=1) {
            for(int p=i<<1,j=0;j<N;j+=p) {
                for(int k=0;k<i;++k) {
                    ll x=a[j+k],y=a[i+j+k];
                    a[j+k]=(x+y)%mod;
                    a[i+j+k]=(x+mod-y)%mod;
                    if(opt==-1) {
                        a[j+k]=a[j+k]*inv2%mod;
                        a[i+j+k]=a[i+j+k]*inv2%mod;
                    }
                }
            }
        }
    }
} fwt;
```

4.4 曼哈顿距离与切比雪夫距离

曼哈顿距离: $d(i,j) = |x1-x2|+|y1-y2|$ 切比雪夫距离: $d(i,j) = max(|x1-x2|, |y1-y2|)$
将一个点 (x,y) 的坐标变为 $(x+y, x-y)$ 后原坐标系中的曼哈顿距离 = 新坐标系中的切比雪夫距离
将一个点 (x,y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后原坐标系中的切比雪夫距离 = 新坐标系中的曼哈顿距离

4.5 Bitset

```
/*
 * 优化了空间复杂度和时间复杂度
 */
#include<bitset>
using namespace std;
bitset<length> b;
b[2] = 1; // 下标从 0 开始

常用函数
b1 = b2 & b3;
b1 = b2 | b3;
b1 = b2 ^ b3;
b1 = ~b2;
b1 = b2 << 3;

常用成员函数
b.any()          // b 中是否存在置为 1 的二进制位?
b.none()         // b 中不存在置为 1 的二进制位吗?
b.count()        // b 中置为 1 的二进制位的个数
b.size()         // b 中二进制位数的个数
b[pos]           // 访问 b 中在 pos 处的二进制位
b.test(pos)      // b 中在 pos 处的二进制位置为 1 吗?
b.set()          // 把 b 中所有二进制位置为 1
b.set(pos)       // 把 b 中在 pos 处的二进制位置为 1
b.reset()        // 把 b 中所有二进制位置为 0
b.reset(pos)     // 把 b 中在 pos 处的二进制位置为 0
b.flip()         // 把 b 中所有二进制位逐位取反
b.flip(pos)      // 把 b 中在 pos 处的二进制位取反
b.to_ulong()     // 把 b 中同样的二进制位返回一个 unsigned long
b.to_ullong()    // 把 b 中同样的二进制位返回一个 unsigned long
↪  long
```