链式前向星

```
const int MAXN=1e5+10;
int tot=0;
struct Edge{
    int u;int v;int nxt;
};
int head[MAXN];
Edge e[MAXN];
void addEdge(int u,int v){
    e[tot].u=u;
    e[tot].v=v;
    e[tot].nxt=head[u];
    head[u]=tot;
    tot++;
}
```

bfs模板

```
void init(){
    for(int i=0;i<=n;i++){
        vis[i]=0;
    }
}
void bfs(int s){
    init();
    queue<int> q;
    q.push(s);
    vis[s]=1;
    while(!q.empty()){
        int u=q.front();q.pop();
        for(auto it=a[u].begin();it!=a[u].end();it++){
            int v=*it;
            if(!vis[v]){
                q.push(v);
                vis[v]=1;
            }
        }
    }
}
```

dfs模板：

```
void init(){
    for(int i=0;i<=n;i++){
        vis[i]=0;
    }
}
void dfs(int s){
    vis[s]=1;
    for(auto it=a[s].begin();it!=a[s].end();it++){
        int v=*it;
        if(!vis[v]){
            dfs(v);
```

```
        }
    }
}
```

dij模板：

```
struct node{
    int dis,pos;
    bool operator< (const node&x) const{
        return x.dis<dis;//注意，重载运算符
    }
};
priority_queue<node> q;//q是最大堆；结合重载的运算符
//如下是最大堆priority_queue<int,vector<int>,greater<int>> q
void Dij(int s){
    dis[s]=0;
    q.push((node){dis[s],s});
    while(!q.empty()){
        node n=q.top();q.pop();
        int u=n.pos;
        if(vis[u]) continue;
        vis[u]=1;
        for(int i=head[u];i!=-1;i=e[i].nxt){
            int v=e[i].v;
            if(dis[v]>dis[u]+e[i].w){
                dis[v]=dis[u]+e[i].w;
                q.push((node){dis[v],v});
            }
        }
    }
}
```

SPFA模板：

```
bool SPFA(int s){
    dis[s]=0;vis[s]=1;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[u]=0;
        for(int i=head[u];i!=-1;i=e[i].nxt){
            int v=e[i].v;
            if(dis[v]>dis[u]+e[i].w){
                dis[v]=dis[u]+e[i].w;
                if(!vis[v]){
                    q.push(v);
                    vis[v]=1;
                    cnt[v]++;
                    if(cnt[v]>=n) return true;
                }
            }
        }
    }
```

```
    return false;
}
```

Bellman-ford模板：

```
bool Bellman_ford(int s){
    dis[s]=0;
    for(int k=1;k<=n;k++){
        for(int i=0;i<tot;i++){
            int u=e[i].u,v=e[i].v,w=e[i].w;
            if((dis[v]>dis[u]+w) ){
                if(k==n) return true;
                dis[v]=dis[u]+w;
            }
        }
    }
    return false;
}
```

Floyed模板：

```
void Floyed(){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
            }
        }
    }
}
```

Kruskal模板：

```
int d[5010];
int find(int i){
    if(d[i]==i) return i;
    return d[i]=find(d[i]);
}
bool unite(int a,int b){
    int fa=find(a),fb=find(b);
    if(fa!=fb){
        d[fa]=fb;
        return true;
    }
    return false;
}
bool Kruskal(){
    int ans=0;
    for(int i=0;i<m;i++){
        int u=e[i].u,v=e[i].v,w=e[i].w;
        if(unite(u,v)){
            ans++;
            sum+=w;//sum代表最小生成树的权重
            if(ans==(n-1)){
```

```
                return true;
            }
        }
    }
    return false;
}
```

Prim模板

```
struct Node{
    int pos,dis;
    bool operator < (const Node& n) const{
        return dis>n.dis;
    }
};
bool Prim(int s){//返回true表示连通
    dis[s]=0;
    priority_queue<Node> q;
    q.push((Node){s,0});

    while(!q.empty()){
        int u=q.top().pos,tw=q.top().dis;q.pop();
        if(vis[u]) continue;
        vis[u]=1;
        cnt++;
        ans+=tw;
        if(cnt==n) return true;
        for(int i=head[u];i!=-1;i=e[i].nxt){
            int v=e[i].v,w=e[i].w;
            if(dis[v]>w){
                dis[v]=w;
                q.push((Node){v,dis[v]});
            }
        }
    }
    if(cnt!=n) return false;
}
```

STL

```
atoi(c);//字符转化成数字
stoi(s);//字符串转化成数字
```

字符串分割函数

```
void Stringsplit(string str, char split)
{
    istringstream iss(str); // 输入流
    string token;           // 接收缓冲区
    while (getline(iss, token,split))   // 以split为分隔符
    {
        cout << token << endl; // 输出
    }
}
```

定点数和浮点数

```
double a=30;
double b=0.000012;
cout<<a<<endl;//30
cout<<fixed<<a<<endl;//30.000000
cout<<setprecision(2)<<a<<endl;//30.00
cout<<fixed<<setprecision(2)<<a<<endl;//30.00

cout<<b<<endl;//0.00
cout<<fixed<<b<<endl;//0.00
cout<<setprecision(8)<<b<<endl;//0.00001200
cout<<fixed<<setprecision(8)<<b<<endl;//0.00001200
```

判断素数的方法

```
bool prime(int n)
{
    if (n==1) return false;
    if (n==2) return true;
    for(int i=2;i*i<=n;i++)
       if (n%i==0) return false;
    return true;
}
```

求强连通分量的算法

```
const int MAXN=1e6+10;
int tot_positive=0;
int tot_opposite=0;
int dcnt=0,cdnt=0;
int inDegree[MAXN];
struct Edge{
    int u;int v;int nxt;
};
int head1[MAXN],head2[MAXN];
Edge e1[MAXN],e2[MAXN];
int vis1[MAXN],vis2[MAXN];
int dfn[MAXN],cfn[MAXN];
void add_positive(int u,int v){
    e1[tot_positive].u=u;
    e1[tot_positive].v=v;
    e1[tot_positive].nxt=head1[u];
    head1[u]=tot_positive;
    tot_positive++;
}
void add_opposite(int u,int v){
    e2[tot_opposite].u=u;
    e2[tot_opposite].v=v;
    e2[tot_opposite].nxt=head2[u];
    head2[u]=tot_opposite;
    tot_opposite++;
};
void dfs1(int s){
```

```
    vis1[s]=1;
    for(int i=head1[s];i!=-1;i=e1[i].nxt){
        int v=e1[i].v;
        if(!vis1[v]) {
            dfs1(v);
        }
    }
    dfn[++dcnt]=s;//后序序列
}
void dfs2(int s){
    cfn[s]=cdnt;
    for(int i=head2[s];i!=-1;i=e2[i].nxt){
        int v=e2[i].v;
        if(!cfn[v]){
            dfs2(v);
        }
    }
}
void kosaruju(){
    for(int i=1;i<=n;i++){
        if(!vis1[i])  dfs1(i);
    }
    for(int i=n;i>=1;i--){//按照逆后序顺序访问
        if(!cfn[dfn[i]]){//其中cfn[i]中存储的是节点i属于哪一个scc。
            cdnt++;
            dfs2(dfn[i]);
        }
    }
}
```

树状数组模板

```
const int MAXN=1e6+10;
#define lb(x) x&(-x)
int d[MAXN]={0};
void upd(int p,int x){//单点修改
    for(int i=p;i<MAXN;i+=lb(i)){
        d[i]+=x;
    }
}
int ask(int p){//区间查询
    int sum=0;
    for(int i=p;i>=1;i-=lb(i)){
        sum+=d[i];
    }
    return sum;
}
```

线段树

```
const int MAXN=2e5+10;
int d[MAXN<<2];
int a[MAXN]={0};
```

```
//构造一棵线段树
void build(int x,int l,int r){
    if(l==r){
        d[x]=a[x];
        return;
    }
    int mid=(l+r)/2;
    build(2*x,l,mid);
    build(2*x+1,mid+1,r);
    d[x]=max(d[2*x],d[2*x+1]);
}
//将p的位置增加z
int p=0,z=0;
void upd(int x,int l,int r){
    if(l==r){
        d[x]+=z;
        return;
    }
    int mid=(l+r)/2;
    if(p<=mid)
        upd(2*x,l,mid);
    else if(p>mid)
        upd(2*x+1,mid+1,r);
    d[x]=max(d[2*x],d[2*x+1]);
}
//查找[p1,p2]之间的结果，示例为查找[p1,p2]之间的最大值
int ask(int x,int l,int r,int p1,int p2){
    if(l==p1&&r==p2){
        return d[x];
    }
    int mid=(l+r)/2;
    if(p1>mid){
        return ask(2*x+1,mid+1,r,p1,p2);
    }
    else if(p2<=mid){
        return ask(2*x,l,mid,p1,p2);
    }else{
        int lch=ask(2*x,l,mid,p1,mid);
        int rch=ask(2*x+1,mid+1,r,mid+1,p2);
        return max(lch,rch);
    }
}
//调用实例
build(1,1,2e5);//构建一颗线段树，树的范围是1-2e5
upd(1,1,2e5);//将p的位置增加z，p、z在上述中有说明
ask(1,1,2e5,l,r);//查找区间[l,r]上的结果
```

快速幂

```
inline long long quickPower(int a, int b, int p)  {
    long long ans = 1;
    while (b)
    {
        if (b & 1)
        {
            ans = ans * a % p;
        }
        a = a * a % p;
        b /= 2;
    }
    return ans;
}
```

矩阵快速幂

```
Matrix MatrixQuickPower(const Matrix &A, int k)  {
    Matrix res;
    for (int i = 1; i <= n; i++)
        res.m[i][i] = 1; // 单位矩阵
    while (k)
    {
        if (k & 1)
            res = res * A;
        A = A * A;
        k >>= 1;
    }
    return res;
}
```