icpc模板

[1.快读 快输 快速幂 快速乘](#Xa657005376bd03ff9f03916d6da1740bc155e9d)  
 [文件输入输出，测试运行时间](#文件输入输出测试运行时间)  
 [2.lcm gcd](#X39a17c968ef910a0ff5b9562d6eadb201e07968)  
 [3.dfs and bfs](#Xf993c4d98bfdfaf6d52c4583f33b7d1f13261a3)   
 [4二分](#X33798b6bbe68dd4c5f854a985484f15916dfac8)  
 [二分答案](#二分答案)  
 [lower\_bound upper\_bound](#lowerbound-upperbound)  
 [5.欧拉筛](#Xf93bdcec16ca3f3e850beffb478a2d9287e7cce)  
 [6.1-n全排列](#X7d58363298da42ff35c12004116cdffa9980eea)  
 [7.组合数](#X67ffa7a3da3a2462ecb573307e0054017243744)  
 [逆元法](#逆元法)  
 [递推法](#递推法)  
 [8.单调队列](#Xef2a7779d18e739e202e2c922641d796cd5e011)  
 [9.最小生成树](#X1bfca8a0703e06d2d1d8b0203a135d51ee01a47)  
 [1.kruskal](#Xcd7a2aded39ce2da1825214386c45521caaeeb2)  
 [2.prim](#X55f547bce6c0a216d7143f5e5e5cb08acb92eb5)  
 [10.最短路](#Xc0039f76dbed06fd9840c651ef1c923d2540199)  
 [1.Dijkstra](#Xac79c28fbc26e5d0ef73d05f953440e957b5952)  
 [2.SPFA](#X409c106ce8029557e7ecdf1939d9b002c6e20f6)  
 [3.Flody](#X2356549d185cdaf9e9d29f3ac759e09cb3a0062)  
 [11.dp](#Xd12542a5781671700d28c04221d6e44015bd44e)  
 [背包](#背包)  
 [LIS LCS](#lis-lcs)  
 [区间dp](#区间dp)  
 [12.欧拉函数](#Xecd0080e5043bbe1462f64bab213267ad545193)  
 [13.线段树](#Xa0550e8be462f6fdd7099deb4bcd5d07775cc98)  
 [点更新](#点更新)  
 [区间更新](#区间更新)  
 [14.st表](#X1beec0116f27014ff43d0b1210bb59d909677aa)  
 [15.矩阵快速幂](#X3ee6ab82fa5687d947b64faccb834d0880a8b67)  
 [16.拓扑排序](#X3a28d3d55ede61f6e40f664f3a31f70858c2125)  
 [17.lca](#X0241ba07509c5b0d28d864d6a5fa4bd5ddcf7fb)  
 [18.二维前缀和](#X8645f4f96b86a0793720ee45709932123768731)  
 [19.乘法逆元](#X6cc45c52c5eb7cfda278ded4c18e813bcc55fc0)  
 [20.set自定义排序](#X3c06515763ab3dab7e8f196ffd24f16c41e61d1)  
 [21.任意2-36进制转10进制](#X67ec1e97faab98e6d8bf10b27601abb88475ba7)  
 [22.10进制转换成任意进制](#X95efed8f7a0c09db22f16cb29a98b03f5fd3d6c)  
 [23.约瑟夫问题](#X0c922185534fa9f711ec78bbb5c117f19d17bf3)  
 [24.网络流](#X398c64eb63593cbc6446ef5af5978f690e0126e)  
 [Dinic 算法](#dinic-算法)

## 1.快读 快输 快速幂 快速乘

#pragma GCC optimize(1)  
#pragma GCC optimize(2)  
#pragma GCC optimize(3,"Ofast","inline")  
#include <bits/stdc++.h>  
#define endl '\n'  
#define inf 0x3f3f3f3f  
using namespace std;  
typedef long long ll;  
const ll mod=998244353;  
  
priority\_queue<int,vector<int>,greater<int> > small\_heap;   
  
ll ksm(ll a,ll b)  
{  
 ll ans=1;  
 while(b)  
 {  
 if(b&1)  
 {  
 ans=(ans\*a)%mod;  
 }  
 b/=2;  
 a=(a\*a)%mod;  
 }  
 return ans%mod;  
}  
ll ksc(ll a,ll b)  
{  
 ll ans=0;  
 while(b)  
 {  
 if(b&1)  
 {  
 b--;  
 ans=(ans+a)%mod;  
 }  
 b/=2;  
 a=(a+a)%mod;  
   
 }  
 return ans;  
}  
inline 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;  
}  
inline void write(int x)  
{  
 if(x<0) putchar('-'),x=-x;  
 if(x>9) write(x/10);  
 putchar(x%10+'0');  
}  
int main()  
{  
 ios::sync\_with\_stdio(false); cin.tie(0); cout.tie(0);  
}

### 文件输入输出，测试运行时间

freopen("in.cpp","r",stdin);  
 freopen("out.cpp","w",stdout);  
 clock\_t start,finish;  
 start=clock();  
 运行程序  
 finish=clock();  
 cerr<<((double)finish-start)/CLOCKS\_PER\_SEC<<endl;

## 2.lcm gcd

int gcd(int a,int b){  
 return b>0 ? gcd(b,a%b) : a;  
}  
int lcm(int a,int b){  
 return a/gcd(a,b)\*b;  
}  
// 拓展欧几里得算法  
void exgcd(int a, int b, int& x, int& y) {  
 if (b == 0) {  
 x = 1, y = 0;  
 return;  
 }  
 exgcd(b, a % b, y, x);  
 y -= a / b \* x;  
}

## 3.dfs and bfs

#include<bits/stdc++.h>  
using namespace std;  
struct edge{  
 int u,v;  
};  
vector<int> e[100001];  
vector<edge> s;  
int vis1[100001],vis2[100001];  
bool cmp(edge x,edge y){  
 if(x.v==y.v){  
 return x.u<y.u;  
 }  
 return x.v<y.v;  
}  
void dfs(int x){  
 vis1[x]=1;  
 cout<<x<<" ";  
 for(int i=0;i<e[x].size();i++){  
 int point=s[e[x][i]].v;  
 if(!vis1[point]) dfs(point);  
 }  
}  
void bfs(int x){  
 queue<int> q;  
 q.push(x);  
 cout<<x<<" ";  
 vis2[x]=1;  
 while(!q.empty()){  
 int f=q.front();  
 for(int i=0;i<e[f].size();i++){  
 int point=s[e[f][i]].v;  
 if(!vis2[point]){  
 q.push(point);  
 cout<<point<<" ";  
 vis2[point]=1;  
 }  
 }  
 q.pop();  
 }  
}  
int main()  
{  
 ios::sync\_with\_stdio(false); cin.tie(0); cout.tie(0);  
 int n,m;  
 cin>>n>>m;  
 for(int i=0;i<m;i++){  
 int u,v;  
 cin>>u>>v;  
 s.push\_back((edge){u,v});  
 }  
 sort(s.begin(),s.end(),cmp);  
 for(int i=0;i<m;i++){  
 e[s[i].u].push\_back(i);  
 }  
 dfs(1);  
 cout<<endl;  
 bfs(1);  
 return 0;  
}

## 4二分

### 二分答案

while (l <= r) {  
 int mid = (l + r) /2, pre = a[1], c = 0;  
 for (int i = 2; i <= n; i++) {  
 res = tmp = a[i] - pre, t = 2;  
 while (tmp > mid) {  
 tmp = (res - 1) / t + 1; //计算分成t段的最大值，每段长度可能不同  
 t++; //例如10分成3段是3，3，4  
 c++;  
 }  
 pre = a[i];  
 }  
 if (c <= k) { //如果这里是 c >= k 就是错的，没深究为啥  
 ans = mid;  
 r = mid - 1;  
 }  
 else l = mid + 1;  
 }

### lower\_bound upper\_bound

int p1=lower\_bound(a,a+n,7)-a; //返回数组中第一个大于或等于被查数的值  
 int p2=upper\_bound(a,a+n,7)-a; //返回数组中第一个大于被查数的值  
 int p3=lower\_bound(a,a+n,7,greater<int>())-num; //返回数组中第一个小于或等于被查数的值  
 int p4=upper\_bound(a,a+n,7,greater<int>())-num; //返回数组中第一个小于被查数的值

## 5.欧拉筛

const int N = 1e7 + 5;  
int st[N],p[N],cnt;  
void ola(int n)  
{  
 for(int i=2;i<=n;i++)  
 {  
 if(st[i]==0) p[cnt++]=i;//将质数存到pri中  
 for(int j=0;p[j]<=n/i;j++)  
 {  
 st[p[j]\*i]=1;  
 if (i%p[j]==0) break;  
 }  
 }  
}

## 6.1-n全排列

//求1-n 的全排列   
void per(int list[],int k,int n){ //前k个数不动   
 if(k==n-1){  
 for(int i=0;i<n;i++){  
 cout<<list[i];   
 }  
 cout<<endl;  
 }else{  
 for(int i=k;i<n;i++){  
 swap(list[k],list[i]);  
 per(list,k+1,n);  
 swap(list[k],list[i]);   
 }  
 }  
}

## 7.组合数

### 逆元法

ll fac[maxn],inv[maxn];  
//逆元法求组合数  
void init()  
{  
 fac[0]=inv[0]=1;  
 fac[1]=inv[1]=1;  
 for(int i=2;i<=maxn-9;++i){  
 fac[i]=fac[i-1]\*i%mod;  
 inv[i]=(mod-mod/i)\*inv[mod%i]%mod;  
 }  
 for(int i=1;i<=maxn-9;++i){  
 inv[i]=inv[i]\*inv[i-1]%mod;  
 }  
}  
ll C(ll n,ll m){  
 if(m>n) return 0ll;  
 return fac[n]\*inv[m]%mod\*inv[n-m]%mod;  
}

### 递推法

const int N = 2500, mod = 1e9 + 7;  
int n,   
int a[N];  
int c[N][N];  
//C(n, m) = C(n − 1, m) + C(n − 1, m − 1);  
void init()  
{  
 c[1][0] = c[1][1] = 1;  
 for(int i = 2; i < N; ++ i) {  
 c[i][0] = 1;  
 for(int j = 1; j < N; ++ j)  
 c[i][j] = (1ll \* c[i - 1][j] + c[i - 1][j - 1]) % mod;  
 }  
}

## 8.单调队列

//单调队列   
 head=1; tail=0; //min  
 for(int i=1;i<=n;i++){  
 while(head<=tail&&q[tail]>=a[i]) tail--;  
 q[++tail]=a[i];  
 p[tail]=i;  
 while(p[head]<=i-k) head++;  
 if(i>=k) cout<<q[head]<<" ";  
 }  
 cout<<endl;  
 head=1; tail=0; //max  
 for(int i=1;i<=n;i++){  
 while(head<=tail&&q[tail]<=a[i]) tail--;  
 q[++tail]=a[i];  
 p[tail]=i;  
 while(p[head]<=i-k) head++;  
 if(i>=k) cout<<q[head]<<" ";  
 }  
 cout<<endl;

## 9.最小生成树

### 1.kruskal

#include<bits/stdc++.h>  
#define rep(i,a,n) for (int i=a;i<=n;i++)//i为循环变量，a为初始值，n为界限值，递增  
#define per(i,a,n) for (int i=a;i>=n;i--)//i为循环变量， a为初始值，n为界限值，递减。  
#define pb push\_back  
#define IOS ios::sync\_with\_stdio(false);cin.tie(0); cout.tie(0)  
#define fi first  
#define se second  
#define mp make\_pair  
using namespace std;  
const int inf = 0x3f3f3f3f;//无穷大  
const int maxn = 1e5;//最大值。  
typedef long long ll;  
typedef long double ld;  
typedef pair<ll, ll> pll;  
typedef pair<int, int> pii;  
struct edge{  
 int s;//边的起始顶点。  
 int e;//边的终端顶点。  
 int w;//边权值。  
 bool operator < (const edge &a){  
 return w<a.w;  
 }  
};  
edge temp[maxn];//临时数组存储边。  
int verx[maxn];//辅助数组，判断是否连通。  
edge tree[maxn];//最小生成树。  
int n,m;//n\*n的图，m条边。  
int cnt;//统计生成结点个数，若不满足n个，则生成失败。  
int sum;//最小生成树权值总和。  
void print(){  
 //打印最小生成树函数。  
 cout<<"最小生成树的权值总和为："<<sum<<endl;  
 rep(i,0,cnt-1){  
 cout<<tree[i].s<<" "<<tree[i].e<<"边权值为"<<tree[i].w<<endl;  
 }  
}  
void Kruskal(){  
 rep(i,1,n)  
 verx[i]=i;//这里表示各顶点自成一个连通分量。  
 cnt=0;sum=0;  
 sort(temp,temp+m);//将边按权值排列。  
 int v1,v2;  
 rep(i,0,m-1){  
 v1=verx[temp[i].s];  
 v2=verx[temp[i].e];  
 if(v1!=v2){  
tree[cnt].s=temp[i].s;tree[cnt].e=temp[i].e;tree[cnt].w=temp[i].w;//并入最小生成树。  
 rep(j,1,n){  
 //合并v1和v2的两个分量，即两个集合统一编号。  
 if(verx[j]==v2)verx[j]=v1; //默认集合编号为v2的改为v1.  
 }  
 sum+=tree[cnt].w;  
 cnt++;  
 }  
 }  
 //结束双层for循环之后得到tree即是最小生成树。  
 print();  
}  
int main(){  
 //freopen("in.txt", "r", stdin);//提交的时候要注释掉  
 IOS;  
 while(cin>>n>>m){  
 int u,v,w;  
 rep(i,0,m-1){  
 cin>>u>>v>>w;  
 temp[i].s=u;temp[i].e=v;temp[i].w=w;  
 }  
 Kruskal();  
 }  
 return 0;  
}

### 2.prim

//|适用于 稠密图 求最小生成树|  
 //|堆优化版，时间复杂度：O(elgn)|  
struct node {   
 int v, len;   
 node(int v = 0, int len = 0) :v(v), len(len) {}   
 bool operator < (const node &a)const { // 加入队列的元素自动按距离从小到大排序   
 return len> a.len;   
 }   
};  
vector<node> G[maxn];  
int vis[maxn];  
int dis[maxn];  
void init() {   
 for (int i = 0; i<maxn; i++) {   
 G[i].clear();   
 dis[i] = INF;   
 vis[i] = false;   
 }   
}   
int Prim(int s) {   
 priority\_queue<node>Q; // 定义优先队列   
 int ans = 0;   
 Q.push(node(s,0)); // 起点加入队列   
 while (!Q.empty()) {   
 node now = Q.top(); Q.pop(); // 取出距离最小的点   
 int v = now.v;   
 if (vis[v]) continue; // 同一个节点，可能会推入2次或2次以上队列，这样第一个被标记后，剩下的需要直接跳过。   
 vis[v] = true; // 标记一下   
 ans += now.len;   
 for (int i = 0; i<G[v].size(); i++) { // 开始更新   
 int v2 = G[v][i].v;   
 int len = G[v][i].len;   
 if (!vis[v2] && dis[v2] > len) {   
 dis[v2] = len;   
 Q.push(node(v2, dis[v2])); // 更新的点加入队列并排序   
 }   
 }   
 }   
 return ans;   
}

## 10.最短路

单源最短路问题

### 1.Dijkstra

//Dijkstra 完整模板  
#include<bits/stdc++.h>  
#define ll long long   
using namespace std;  
const int maxn=1e5+100;  
const int INF=2147483647;   
struct Edge {  
 int from,to,dist;  
 Edge(int u,int v,int d):from(u),to(v),dist(d) {}  
};  
struct Heap{ //heap  
 int d,p; //distant ans position  
 bool operator < (const Heap& rhs) const {  
 return d>rhs.d;  
 }  
};  
vector<Edge> edges;  
vector<int> G[maxn];  
bool vis[maxn];  
int d[maxn];  
int p[maxn];  
int n,m,s; //n个点，m条边 ,s 起始点   
int u,v,w,num;//u v 相连 w为权重，num为每次加边G计数，确定边  
void init(int n) {  
 for(int i=0; i<=n; i++) G[i].clear();  
 edges.clear();  
}  
void AddEdge(int from,int to,int dist) {  
 edges.push\_back(Edge(from,to,dist));  
 num=edges.size();  
 G[from].push\_back(num-1);  
}  
void dijkstra(int s) {  
 priority\_queue<Heap> Q;//优先队列  
 for(int i=0;i<=n; i++){  
 d[i]=INF;  
 }  
 d[s]=0;  
 memset(vis,0,sizeof(vis));   
 Q.push((Heap){0,s});  
 while(!Q.empty()) {  
 Heap x=Q.top();  
 Q.pop();  
 int pos=x.p;  
 if(vis[pos]) continue;  
 vis[pos]=true;  
 for(int i=0;i<G[pos].size();i++) {  
 Edge& e=edges[G[pos][i]];  
 if(d[e.to]>d[pos]+e.dist) {  
 d[e.to]=d[pos]+e.dist;  
 p[e.to]=G[pos][i];  
 Q.push( (Heap){d[e.to],e.to} );  
 }  
 }  
 }  
}  
int main() {  
 cin>>n>>m>>s;  
 init(n);  
 for(int i=0;i<m;i++) {  
 cin>>u>>v>>w;  
 AddEdge(u,v,w);  
 }  
 dijkstra(s);  
 for(int i=1;i<=n;i++) {  
 cout<<d[i]<<" "; //起点到每个点的最短长度   
 }  
 return 0;  
}

### 2.SPFA

#include<bits/stdc++.h>  
using namespace std;  
const int maxn=1e6+100;  
const int INF=2147483647;  
struct edge{  
 int v,w;  
 edge(int to,int weight):v(to),w(weight){}  
};  
vector<edge> G[maxn];  
bool vis[maxn];  
int dis[maxn],cnt[maxn];  
void Init(int n)   
{   
 for(int i=0;i<=n+2;++i){   
 G[i].clear();   
 dis[i] = INF;   
 }   
}   
void SPFA(int s)   
{   
 int v1,v2,w;   
 queue<int> Q;   
 //memset(vis,false,sizeof(vis)); //标记数组  
 //memset(cnt,0,sizeof(cnt)); //计数数组,加入队列的次数  
 dis[s] = 0;   
 Q.push(s); //起点加入队列  
 vis[s] = true; // 标记  
 while(!Q.empty()){   
 v1 = Q.front();   
 Q.pop();   
 vis[v1] = false; // 取消标记  
 for(int i = 0 ; i < G[v1].size() ; ++i){ //搜索v1的链表  
 v2 = G[v1][i].v;  
 w = G[v1][i].w;   
 if(dis[v2] > dis[v1] + w){ //松弛操作  
 dis[v2] = dis[v1] + w;   
 if(vis[v2] == false){ //再次胶乳队列  
 vis[v2] = true;   
 //cnt[v2]++; //判断负环  
 //if(cnt[v2] > n) return -1;   
 Q.push(v2);   
 }   
 }   
 }   
 }   
}  
void addedge(int u,int v,int w){  
 G[u].push\_back(edge(v,w));   
 //G[v].push\_back(edge(v,u,w)); //区别有向还是无向  
}  
int main()  
{  
 int n,m,u,v,w,s; //n个点，m条边，u,v,w起点终点权值,是目标终点  
 cin>>n>>m>>s;  
 Init(n);//初始化  
 for(int i=0;i<m;i++){  
 cin>>u>>v>>w; //输入   
 addedge(u,v,w); //加边   
 }  
 SPFA(s);   
 for(int i=1;i<=n;i++){  
 cout<<dis[i]<<" "; //输出到每个点的最短距离   
 }  
 return 0;  
}

### 3.Flody

注意题目要求是否要更新输入点的距离

for(ll i=1;i<=n;i++){ //初始化  
 for(ll j=1;j<=n;j++){  
 if(i==j){  
 a[i][j]=0;  
 continue;  
 }   
 a[i][j]=INF;  
 }  
 }   
 for(ll k=1;k<=n;k++){ //flody主体  
 for(ll i=1;i<=n;i++){  
 for(ll j=1;j<=n;j++){  
 if(a[i][j]>a[i][k]+a[k][j]){  
 a[i][j]=a[i][k]+a[k][j];  
 }  
 }  
 }  
 }

## 11.dp

### 背包

for (int i = 1; i <= n; i++) // 01背包  
 for (int l = W; l >= w[i]; l--) f[l] = max(f[l], f[l - w[i]] + v[i]);  
void complete(int cost, int weight) { // 完全背包  
 for(i = cost ; i <= v; ++i)   
 dp[i] = max(dp[i], dp[i - cost] + weight);   
}   
void multiply(int cost, int weight, int amount) {// 多重背包   
 if(cost \* amount >= v)   
 complete(cost, weight);   
 else{   
 k = 1;   
 while (k < amount){   
 bag01(k \* cost, k \* weight);   
 amount -= k;   
 k += k;   
 }   
 bag01(cost \* amount, weight \* amount);   
 }   
}   
int dp[1000000]; // other  
int c[55], m[110];  
int sum;  
void CompletePack(int c) {  
 for (int v = c; v <= sum / 2; ++v){  
 dp[v] = max(dp[v], dp[v - c] + c);  
 }  
}  
void ZeroOnePack(int c) {  
 for (int v = sum / 2; v >= c; --v) {  
 dp[v] = max(dp[v], dp[v - c] + c);  
 }  
}  
void multiplePack(int c, int m） {  
 if (m \* c > sum / 2)  
 CompletePack(c);  
 else{  
 int k = 1;  
 while (k < m){  
 ZeroOnePack(k \* c);  
 m -= k;  
 k <<= 1;  
 }  
 if (m != 0){  
 ZeroOnePack(m \* c);  
 }  
 }  
}  
//有依赖的背包（选课问题）  
#include<bits/stdc++.h>  
using namespace std;  
int dp[110][110]; // 表达选择以x为子树的物品，在容量不超过v时所获得的最大价值  
vector<int>g[110];  
int v[110];  
int w[110];  
int V;  
void dfs(int x){  
 for(int i=v[x];i<=V;i++){  
 dp[x][i]=w[x];  
 }  
 for(int i=0;i<g[x].size();i++){  
 int u=g[x][i];  
 dfs(u);   
 for(int j=V;j>=v[x];j--){  
 for(int k=0;k<=j-v[x];k++){  
 dp[x][j]=max(dp[x][j],dp[x][j-k]+dp[u][k]);  
 }  
 }  
 }  
 return ;  
}  
int main(){  
 int n;  
 cin>>n>>V;  
 int s=0;  
 int x,y,z;  
 for(int i=1;i<=n;i++){  
 cin>>v[i]>>w[i]>>z;  
 if(z==-1){  
 s=i;  
 }else{  
 g[z].push\_back(i);  
 }  
 }  
 dfs(s);  
 cout<<dp[s][V];  
 return 0;  
}  
// C++ Version 二进制  
index = 0;  
for (int i = 1; i <= m; i++) {  
 int c = 1, p, h, k;  
 cin >> p >> h >> k;  
 while (k - c > 0) {  
 k -= c;  
 list[++index].w = c \* p;  
 list[index].v = c \* h;  
 c \*= 2;  
 }  
 list[++index].w = p \* k;  
 list[index].v = h \* k;  
}

### LIS LCS

/\* LIS   
 优化方法：  
 dp[i]表示长度为i+1的上升子序列的最末尾元素   
 找到第一个比dp末尾大的来代替   
\*/  
 void solve() {   
 for (int i = 0; i < n; ++i){  
 dp[i] = INF;  
 }  
 for (int i = 0; i < n; ++i) {   
 \*lower\_bound(dp, dp + n, a[i]) = a[i]; // 返回一个指针   
 }   
 printf("%d\n", \*lower\_bound(dp, dp + n, INF) - dp);   
 }  
/\*   
 函数lower\_bound()返回一个 iterator 它指向在[first,last)标记的有序序列中可以插入value，而不会破坏容器顺序的第一个位置，而这个位置标记了一个不小于value的值。  
\*/  
//LCS   
void solve() {   
 for (int i = 0; i < n; ++i) {   
 for (int j = 0; j < m; ++j) {   
 if (s1[i] == s2[j]) {   
 dp[i + 1][j + 1] = dp[i][j] + 1;   
 }else {   
 dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j]);   
 } } }  
}

### 区间dp

//区间dp核心：  
// C++ Version  
for (len = 1; len <= n; len++)  
 for (i = 1; i <= 2 \* n - 1; i++) {  
 int j = len + i - 1;  
 for (k = i; k < j && k <= 2 \* n - 1; k++)  
 f[i][j] = max(f[i][j], f[i][k] + f[k + 1][j] + sum[j] - sum[i - 1]);  
 }

## 12.欧拉函数

//求小于等于n和n互质的数的个数  
int euler\_phi(int n) {  
 int ans = n;  
 for (int i = 2; i \* i <= n; i++)  
 if (n % i == 0) {  
 ans = ans / i \* (i - 1);  
 while (n % i == 0) n /= i;  
 }  
 if (n > 1) ans = ans / n \* (n - 1);  
 return ans;  
}

## 13.线段树

### 点更新

struct node  
{  
 int left, right;  
 int max, sum;  
};  
node tree[maxn << 2];  
int a[maxn];  
int n;  
int k = 1;  
int p, q;  
string str;  
void build(int m, int l, int r)//m 是 树的标号  
{  
 tree[m].left = l;  
 tree[m].right = r;  
 if (l == r){  
 tree[m].max = a[l];  
 tree[m].sum = a[l];  
 return;  
 }  
 int mid = (l + r) >> 1;  
 build(m << 1, l, mid);  
 build(m << 1 | 1, mid + 1, r);  
 tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);  
 tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;  
}  
void update(int m, int a, int val)//a是节点位置,val是更新的值（加减的值）  
{  
 if (tree[m].left == a && tree[m].right == a){  
 tree[m].max += val;  
 tree[m].sum += val;  
 return;  
 }  
 int mid = (tree[m].left + tree[m].right) >> 1;  
 if (a <= mid){  
 update(m << 1, a, val);  
 }  
 else{  
 update(m << 1 | 1, a, val);  
 }  
 tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);  
 tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;  
}  
int querySum(int m, int l, int r)  
{  
 if (l == tree[m].left && r == tree[m].right){  
 return tree[m].sum;  
 }  
 int mid = (tree[m].left + tree[m].right) >> 1;  
 if (r <= mid){  
 return querySum(m << 1, l, r);  
 }  
 else if (l > mid){  
 return querySum(m << 1 | 1, l, r);  
 }  
 return querySum(m << 1, l, mid) + querySum(m << 1 | 1, mid + 1, r);  
}  
int queryMax(int m, int l, int r)  
{  
 if (l == tree[m].left && r == tree[m].right){  
 return tree[m].max;  
 }  
 int mid = (tree[m].left + tree[m].right) >> 1;  
 if (r <= mid){  
 return queryMax(m << 1, l, r);  
 }  
 else if (l > mid){  
 return queryMax(m << 1 | 1, l, r);  
 }  
 return max(queryMax(m << 1, l, mid), queryMax(m << 1 | 1, mid + 1, r));  
}   
build(1,1,n);   
update(1,a,b);   
query(1,a,b);

### 区间更新

typedef long long ll;   
const int maxn = 100010;   
int t,n,q;   
ll anssum;   
struct node{   
 ll l,r;   
 ll addv,sum;   
}tree[maxn<<2];   
void maintain(int id) {   
 if(tree[id].l >= tree[id].r)   
 return ;   
 tree[id].sum = tree[id<<1].sum + tree[id<<1|1].sum;   
}   
void pushdown(int id) {   
 if(tree[id].l >= tree[id].r)   
 return ;   
 if(tree[id].addv){   
 int tmp = tree[id].addv;   
 tree[id<<1].addv += tmp;   
 tree[id<<1|1].addv += tmp;   
 tree[id<<1].sum += (tree[id<<1].r - tree[id<<1].l + 1)\*tmp;   
 tree[id<<1|1].sum += (tree[id<<1|1].r - tree[id<<1|1].l + 1)\*tmp;   
 tree[id].addv = 0;   
 }   
}   
void build(int id,ll l,ll r) {   
 tree[id].l = l;   
 tree[id].r = r;   
 tree[id].addv = 0;   
 tree[id].sum = 0;   
 if(l==r) {   
 tree[id].sum = 0;   
 return ;   
 }   
 ll mid = (l+r)>>1;   
 build(id<<1,l,mid);   
 build(id<<1|1,mid+1,r);   
 maintain(id);   
}   
void updateAdd(int id,ll l,ll r,ll val) {   
 if(tree[id].l >= l && tree[id].r <= r)   
 {   
 tree[id].addv += val;   
 tree[id].sum += (tree[id].r - tree[id].l+1)\*val;   
 return ;   
 }   
 pushdown(id);   
 ll mid = (tree[id].l+tree[id].r)>>1;   
 if(l <= mid)   
 updateAdd(id<<1,l,r,val);   
 if(mid < r)   
 updateAdd(id<<1|1,l,r,val);   
 maintain(id);   
}   
void query(int id,ll l,ll r) {   
 if(tree[id].l >= l && tree[id].r <= r){   
 anssum += tree[id].sum;   
 return ;   
 }   
 pushdown(id);   
 ll mid = (tree[id].l + tree[id].r)>>1;   
 if(l <= mid)   
 query(id<<1,l,r);   
 if(mid < r)   
 query(id<<1|1,l,r);   
 maintain(id);   
}   
int main() {   
 scanf("%d",&t);   
 int kase = 0 ;   
 while(t--){   
 scanf("%d %d",&n,&q);   
 build(1,1,n);   
 int id;   
 ll x,y;   
 ll val;   
 printf("Case %d:\n",++kase);   
 while(q--){   
 scanf("%d",&id);   
 if(id==0){   
 scanf("%lld %lld %lld",&x,&y,&val);   
 updateAdd(1,x+1,y+1,val);   
 }   
 else{   
 scanf("%lld %lld",&x,&y);   
 anssum = 0;   
 query(1,x+1,y+1);   
 printf("%lld\n",anssum);   
 } } }   
 return 0;   
}

## 14.st表

const int logn = 21;  
const int maxn = 2000001;  
int f[maxn][logn + 1], Logn[maxn + 1];  
inline int read() { // 快读  
 char c = getchar();  
 int x = 0, f = 1;  
 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;  
}  
void pre() { // 准备工作，初始化  
 Logn[1] = 0;  
 Logn[2] = 1;  
 for (int i = 3; i < maxn; i++) {  
 Logn[i] = Logn[i / 2] + 1;  
 }  
}  
int main() {  
 int n = read(), m = read();  
 for (int i = 1; i <= n; i++) f[i][0] = read();  
 pre();  
 for (int j = 1; j <= logn; j++)  
 for (int i = 1; i + (1 << j) - 1 <= n; i++)  
 f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]); // ST表具体实现  
 for (int i = 1; i <= m; i++) {  
 int x = read(), y = read();  
 int s = Logn[y - x + 1];  
 printf("%d\n", max(f[x][s], f[y - (1 << s) + 1][s]));  
 }  
 return 0;  
}

## 15.矩阵快速幂

const int N=10;  
int tmp[N][N];  
void multi(int a[][N],int b[][N],int n){  
 memset(tmp,0,sizeof tmp);  
 for(int i=0;i<n;i++)  
 for(int j=0;j<n;j++)  
 for(int k=0;k<n;k++)  
 tmp[i][j]+=a[i][k]\*b[k][j];  
 for(int i=0;i<n;i++)  
 for(int j=0;j<n;j++)  
 a[i][j]=tmp[i][j];  
}  
int res[N][N];  
void Pow(int a[][N],int n)  
{  
 memset(res,0,sizeof res);//n是幂，N是矩阵大小  
 for(int i=0;i<N;i++) res[i][i]=1;  
 while(n){  
 if(n&1) multi(res,a,N);//res=res\*a;复制直接在multi里面实现了；  
 multi(a,a,N);//a=a\*a  
 n>>=1;  
 }  
}

## 16.拓扑排序

int n, m;  
vector<int> G[MAXN];  
int in[MAXN]; // 存储每个结点的入度  
bool toposort() {  
 vector<int> L;  
 queue<int> S;  
 for (int i = 1; i <= n; i++)  
 if (in[i] == 0) S.push(i);  
 while (!S.empty()) {  
 int u = S.front();  
 S.pop();  
 L.push\_back(u);  
 for (auto v : G[u]) {  
 if (--in[v] == 0) { S.push(v); }  
 }  
 }  
 if (L.size() == n) {  
 for (auto i : L) cout << i << ' ';  
 return true;  
 } else {  
 return false;  
 }  
}

## 17.lca

#define MXN 50007  
using namespace std;  
std::vector<int> v[MXN];  
std::vector<int> w[MXN];  
int fa[MXN][31], cost[MXN][31], dep[MXN];  
int n, m;  
int a, b, c;  
// dfs，用来为 lca 算法做准备。接受两个参数：dfs 起始节点和它的父亲节点。  
void dfs(int root, int fno) {  
 // 初始化：第 2^0 = 1 个祖先就是它的父亲节点，dep 也比父亲节点多 1。  
 fa[root][0] = fno;  
 dep[root] = dep[fa[root][0]] + 1;  
 // 初始化：其他的祖先节点：第 2^i 的祖先节点是第 2^(i-1) 的祖先节点的第  
 // 2^(i-1) 的祖先节点。  
 for (int i = 1; i < 31; ++i) {  
 fa[root][i] = fa[fa[root][i - 1]][i - 1];  
 cost[root][i] = cost[fa[root][i - 1]][i - 1] + cost[root][i - 1];  
 }  
 // 遍历子节点来进行 dfs。  
 int sz = v[root].size();  
 for (int i = 0; i < sz; ++i) {  
 if (v[root][i] == fno) continue;  
 cost[v[root][i]][0] = w[root][i];  
 dfs(v[root][i], root);  
 }  
}  
// lca。用倍增算法算取 x 和 y 的 lca 节点。  
int lca(int x, int y) {  
 // 令 y 比 x 深。  
 if (dep[x] > dep[y]) swap(x, y);  
 // 令 y 和 x 在一个深度。  
 int tmp = dep[y] - dep[x], ans = 0;  
 for (int j = 0; tmp; ++j, tmp >>= 1)  
 if (tmp & 1) ans += cost[y][j], y = fa[y][j];  
 // 如果这个时候 y = x，那么 x，y 就都是它们自己的祖先。  
 if (y == x) return ans;  
 // 不然的话，找到第一个不是它们祖先的两个点。  
 for (int j = 30; j >= 0 && y != x; --j) {  
 if (fa[x][j] != fa[y][j]) {  
 ans += cost[x][j] + cost[y][j];  
 x = fa[x][j];  
 y = fa[y][j];  
 }  
 }  
 // 返回结果。  
 ans += cost[x][0] + cost[y][0];  
 return ans;  
}  
int main() {  
 // 初始化表示祖先的数组 fa，代价 cost 和深度 dep。  
 memset(fa, 0, sizeof(fa));  
 memset(cost, 0, sizeof(cost));  
 memset(dep, 0, sizeof(dep));  
 // 读入树：节点数一共有 n 个。  
 scanf("%d", &n);  
 for (int i = 1; i < n; ++i) {  
 scanf("%d %d %d", &a, &b, &c);  
 ++a, ++b;  
 v[a].push\_back(b);  
 v[b].push\_back(a);  
 w[a].push\_back(c);  
 w[b].push\_back(c);  
 }  
 // 为了计算 lca 而使用 dfs。  
 dfs(1, 0);  
 // 查询 m 次，每一次查找两个节点的 lca 点。  
 scanf("%d", &m);  
 for (int i = 0; i < m; ++i) {  
 scanf("%d %d", &a, &b);  
 ++a, ++b;  
 printf("%d\n", lca(a, b));  
 }  
 return 0;  
}

## 18.二维前缀和

ans = sum[x2][y2] - sum[x2][y1-1] - sum[x1-1][y2] + sum[x1-1][y1-1];

## 19.乘法逆元

//线性求逆元  
inv[1] = 1;  
for (int i = 2; i <= n; ++i) {  
 inv[i] = (long long)(p - p / i) \* inv[p % i] % p;  
}  
//线性求n个数逆元  
s[0] = 1;  
for (int i = 1; i <= n; ++i) s[i] = s[i - 1] \* a[i] % p;  
sv[n] = qpow(s[n], p - 2);  
// 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.  
for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] \* a[i] % p;  
for (int i = 1; i <= n; ++i) inv[i] = sv[i] \* s[i - 1] % p;

## 20.set自定义排序

struct intComp  
{  
 bool operator()(const int&a, const int&b)  
 {  
 return a > b;  
 }  
};  
set<int,intComp>s;

## 21.任意2-36进制转10进制

int atoi(string s,int radix) //s是给定的radix进制字符串  
{  
 int ans=0;  
 for(int i=0;i<s.size();i++)  
 {  
 char t=s[i];  
 if(t>='0'&&t<='9') ans=ans\*radix+t-'0';  
 else ans=ans\*radix+t-'a'+10;  
 }  
 return ans;  
}

## 22.10进制转换成任意进制

string itoa(int n,int radix) //n是待转数字，radix是指定的进制  
{  
 string ans="";  
 do  
 {  
 int t=n%radix;  
 if(t>=0&&t<=9) ans+=t+'0';  
 else ans+=t-10+'a';  
 n/=radix;  
 }  
 while(n!=0); //使用do{}while（）以防止输入为0的情况  
 reverse(ans.begin(),ans.end());  
 return ans;  
}

## 23.约瑟夫问题

//约瑟夫问题 O(M\*logN):  
ll Josephus(ll n,ll m){  
    int f=1;  
    for(int i=2;i<=min(n,m);i++){  
        f=(f+(m-1)%i)%i+1;  
    }  
    int i;  
    for(int j=m;j<n;j=i){  
        i=min(n,(j\*m-f)/(m-1)+1);  
        f=(f+(i-j)\*m-1)%i+1;  
    }  
    return f;  
}

## 24.网络流

### Dinic 算法

#define maxn 250
  
#define INF 0x3f3f3f3f
  
struct Edge {
  
 int from, to, cap, flow;
  
  
 Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
  
};
  
struct Dinic {
  
 int n, m, s, t;
  
 vector<Edge> edges;
  
 vector<int> G[maxn];
  
 int d[maxn], cur[maxn];
  
 bool vis[maxn];
  
 void init(int n) {
  
 for (int i = 0; i < n; 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));
  
 m = edges.size();
  
 G[from].push\_back(m - 2);
  
 G[to].push\_back(m - 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) {
  
 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++) {
  
 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 s, int t) {
  
 this->s = s;
  
 this->t = t;
  
 int flow = 0;
  
 while (BFS()) {
  
 memset(cur, 0, sizeof(cur));
  
 flow += DFS(s, INF);
  
 }
  
 return flow;
  
 }
  
};