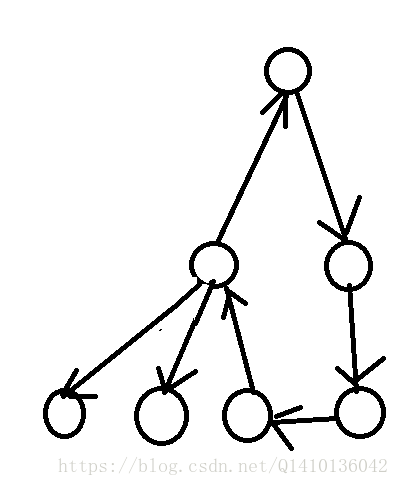
# ****基环+外向树****

与基环内向树相反，它有且只有一个入度（基环内向树是出度），并且并且由环指向环外（额，不知道这个怎么解释，直接看图吧，和上面内向树的图比较一下就OK了）



基环树DP

概念

在图论中，树被视作为一种特殊的图G=(V, E)，其中|V| = |E|+1。其存在如下特性：

树G上任意两点必定能够通过途经若干边后到达

任意两点间的路径必然唯一，即不存在环

将树G上任意一条边删去，该图即成为非连通图

在G中任意不相连两点间插入一条边，该新图G’ =(V, E’)正好含有一个环

基环树的概念即是从上述特性4所引申出的特殊的树。虽然其不符合树|V| = |E| + 1的特征，但由于其特殊性——删除环上任意一条边即可成为树，故仍将其视作”树”来解决问题。

问题提出

​ 给定N个点，N条边，保证任意两点间至少存在一条路径。其中每个点均有其权值vali，问如何选择点，使得在保证任意直接相连的两点不会同时被选中的情况下，被选中的点的权值和最大？

问题简化自[BZOJ 1040] 骑士 ，将其所描述的基环树林简化做基环树以探究该算法。

分析

​ 如何解决上述问题？很显然，若题中所给出的图为N点N-1条边的树，那么可以直接用树形dp解决。其状态转移方程为：

 dp[0][parent]=max(dp[0][son],dp[1][son])dp[1][parent]=dp[0][son]

dp[0][parent]=max(dp[0][son],dp[1][son]) dp[1][parent]=dp[0][son]

其中dp[0][u]表示以点u为根的子树不选u点时的最大权值，dp[1][u]表示以点u为根的子树必选u点时的最大权值。

​ 但是此题很明显不是树，那么如何解决？

​ 基环树dp考虑提出了一种奇妙的想法：考虑找到图中的环，去掉环上的任意一条边，那么图就能变成树，套用树规很容易解决。

​ 假设将要删去的环上的边为Eii，边上两端点为u，v。考虑将点u作为新树的根，作树规将得到dp[0][u]和dp[1][u]。由于作树规假设边Ei不存在，则dp[1][u]作为必须u时的最大权值，可能是同时选择点v得到的最大值。在实际图中，u与v并不能同时取到，故只能将dp[0][u]暂时作为答案保存。很显然，这个答案并不能保证是最优的，因为可能最优的答案要包括u（例如其余N-1个点权值val均为1，点u权值为∞∞）

​ 如何解决不能取到u的问题？再以点v作为新树的根，再做一遍树规，取上次暂存答案与dp[0][v]求最大值即可。

PROBLEM：是否能够保证所求答案必然为题中最大权值？

无向树可以以任意一点u为根，做树形dp求最大值，其答案将保存在dp[0][u]和dp[1][u]中。基环树不考虑dp[1][u]的值，则答案将保存在dp[0][u]中，此时，已遍历所有情况的最优值——除了必须选择点u的情况。将需删除的边的另一端点作为根求值，此时考虑了选与不选u的情况。

同时由于不能同时选择u与v，则答案必然为可行方案的最大值。

模板

此模板中ringPt1、ringPt2作为将图改为树需要删除的边的两个端点。

const int N = 1e6+10; //基环树中点的个数

const int E = 1e6+10; //基环树中无向边的个数

struct Edge{ int nxt, to; }e[E\*2];

int head[N], cnt;

bool vis[N];

int val[N], ringPt1, ringPt2, not\_pass; //ringPt1,ringPt2 -> not\_pass边的端点

long long dp[2][N];

void add(int u,int v){

e[++cnt].nxt = head[u];

e[cnt].to = v;

head[u] = cnt;

}

void getDP(int rt, int fa) {

dp[0][rt] = 0, dp[1][rt] = val[rt];

for(int i=head[rt];i;i=e[i].nxt) {

if(e[i].to == fa) continue;

if(i == not\_pass || i == (not\_pass^1)) continue;

getDP(e[i].to, rt);

//具体树形dp策略(根据实际修改)

dp[0][rt] += max(dp[0][e[i].to], dp[1][e[i].to]);

dp[1][rt] += dp[0][e[i].to];

}

}

void dfs(int rt, int fa) {

vis[rt] = 1;

for(int i=head[rt];i;i=e[i].nxt) {

if(e[i].to == fa) continue;

if(!vis[e[i].to]) dfs(e[i].to, rt);

else {

//记录基环上一条特定边的标号

//e[not\_pass]为该边

//e[not\_pass]^1为该边的反向边

//hint: 邻接表中 cnt应初始化为1

not\_pass = i;

ringPt1 = e[i].to; ringPt2 = rt;

}

}

}

void init() {

memset(head,0,sizeof(head));

cnt = 1; //为配合基环边及其反向边的记录，特将其初始化为1

memset(vis,0,sizeof(vis));

}

int main() {

init();

dfs(1, -1);

getDP(ringPt1, -1);

long long ans = dp[0][ringPt1];

getDP(ringPt2, -1);

ans = max(ans, dp[0][ringPt2]);

}

[BZOJ 1040] 骑士

题目大意：有n个骑士，每个骑士有一个权值。每个骑士都有一个特别讨厌的另一个骑士，他们不应该被同时选中。问选择若干骑士，使得被选中的骑士的权值和最大，最大为多少？

分析：每个骑士都有一个特别讨厌的另一个骑士，看似是一个有向图，实际上还是无向图（例如u讨厌v，则选u就不能v，选v就不能选u）。由于n个点n条边，很容易想到基环树求解，但是，实际上此图并不保证两点间一定存在至少一条路径。综合上述情况，可以将其视作由若干基环树构成的基环树林。对每个基环树单独求解后求ΣΣ。

代码

#include<iostream>

#include<cmath>

#include<cstring>

#include<algorithm>

#include<cstdio>

using namespace std;

const int N = 1e6+10;

const int E = 1e6 + 10;

struct Edge{

int nxt, to;

}e[E\*2];

int head[N], cnt = 1;

bool vis[N];

int val[N], ringPt1, ringPt2, not\_pass;

long long dp[2][N];

void add(int u,int v){

e[++cnt].nxt = head[u];

e[cnt].to = v;

head[u] = cnt;

}

void getDP(int rt, int fa)

{

dp[0][rt] = 0, dp[1][rt] = val[rt];

for(int i=head[rt];i;i=e[i].nxt)

{

if(e[i].to == fa) continue;

if(i == not\_pass || i == (not\_pass^1)) continue;

getDP(e[i].to, rt);

dp[0][rt] += max(dp[0][e[i].to], dp[1][e[i].to]);

dp[1][rt] += dp[0][e[i].to];

}

}

void dfs(int rt, int fa)

{

vis[rt] = 1;

for(int i=head[rt];i;i=e[i].nxt)

{

if(e[i].to == fa) continue;

if(!vis[e[i].to]) dfs(e[i].to, rt);

else{

not\_pass = i;

ringPt1 = e[i].to; ringPt2 = rt;

}

}

}

int main()

{

int n;

scanf("%d",&n);

for(int i=1,y;i<=n;i++)

scanf("%d %d",&val[i], &y), add(i,y), add(y,i);

long long ans = 0;

for(int i=1;i<=n;i++)

{

if(vis[i]) continue;

dfs(i, -1);

getDP(ringPt1, -1);

long long tmp = dp[0][ringPt1];

getDP(ringPt2, -1);

ans += max(tmp, dp[0][ringPt2]);

}

cout<<ans<<endl;

}