快读

int in()

{

int k=0,f=1;

char c=getchar();

while(c<'0'||c>'9')

{

if(c=='-')f=-1;

c=getchar();

}

while(c>='0'&&c<='9')k=k\*10+c-'0',c=getchar();

return k\*f; // 别忘记标记的负数要乘进去

}

// 调用时用 n=in();

快输

void out(int x)

{

if(x<0)putchar('-'),x=-x;

if(x<10)putchar(x+'0');

else out(x/10),putchar(x%10+'0');

}

// 直接调用 out(n) 就行了

栈 模板

#include <stack>

#include <string>

#include <iostream>

int main() {

std::cin.tie(0);

std::ios::sync\_with\_stdio(false);

int T, n;

for (std::cin >> T; T; --T) {

std::stack<unsigned long long int> s;

for (std::cin >> n; n; --n) {

std::string t;

std::cin >> t;

if (t == "push") {

unsigned long long x;

std::cin >> x; s.push(x);

} else if (t == "pop") {

if (s.empty()) std::cout << "Empty\n";

else s.pop();

} else if (t == "query") {

if (s.empty()) std::cout << "Anguei!\n";

else std::cout << s.top() << '\n';

} else {

std::cout << s.size() << '\n';

}

}

}

return 0;

}

字符串哈希（匹配）

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdio>

using namespace std;

typedef unsigned long long ull;

ull base=131;

ull a[10010];

char s[10010];

int n,ans=1;

int prime=233317;

ull mod=212370440130137957ll;

ull hashe(char s[])

{

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=(ans\*base+(ull)s[i])%mod+prime;

return ans;

}

int main()

{

scanf("%d",&n);

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

{

scanf("%s",s);

a[i]=hashe(s);

}

sort(a+1,a+n+1);

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

{

if(a[i]!=a[i+1])

ans++;

}

printf("%d",ans);

}

给你三个整数 *a*,*b*,*p*，求 *a^b* mod*p*。

int quickPower(int a, int b)//是求a的b次方

{

int ans = 1, base = a;//ans为答案，base为a^(2^n)

while(b > 0)//b是一个变化的二进制数，如果还没有用完

{

if(b & 1)//&是位运算，b&1表示b在二进制下最后一位是不是1，如果是：

ans \*= base;//把ans乘上对应的a^(2^n)

base \*= base;//base自乘，由a^(2^n)变成a^(2^(n+1))

b >>= 1;//位运算，b右移一位，如101变成10（把最右边的1移掉了），10010变成1001。现在b在二进制下最后一位是刚刚的倒数第二位。结合上面b & 1食用更佳

}

return ans;

}

Nim游戏标准模板

nim 游戏的规则是这样的：地上有 *n* 堆石子（每堆石子数量小于 104），每人每次可从任意一堆石子里取出任意多枚石子扔掉，可以取完，不能不取。每次只能从一堆里取。最后没石子可取的人就输了。假如甲是先手，且告诉你这 *n* 堆石子的数量，他想知道是否存在先手必胜的策略。

**输入格式**

**本题有多组测试数据。**

第一行一个整数 *T* （*T*≤10），表示有 *T* 组数据

接下来每两行是一组数据，第一行一个整数 *n*，表示有 *n* 堆石子，*n*≤104。

第二行有 *n* 个数，表示每一堆石子的数量.

**输出格式**

共 *T* 行，每行表示如果对于这组数据存在先手必胜策略则输出 Yes，否则输出 No。

#include <bits/stdc++.h> //万能头

using namespace std;

int t, n, x, ans;

int main()

{

cin >> t;

while(t--)

{

ans = 0;

cin >> n;

for(int i = 0; i < n; i++)

{

cin >> x;

ans ^= x; //异或

}

if(ans)

{

printf("Yes\n");

}

else

{

printf("No\n");

}

//也可以用三目运算符 printf(ans ? "Yes\n" : "No\n");

}

return 0;

}

黑白棋染色

**题目描述**

一共 *n*×*m* 个硬币，摆成 *n*×*m* 的长方形。dongdong 和 xixi 玩一个游戏，每次可以选择一个连通块，并把其中的硬币全部翻转，但是需要满足存在一个硬币属于这个连通块并且所有其他硬币都在它的左上方(可以正左方也可以正上方)，并且这个硬币是从反面向上翻成正面向上。dongdong 和 xixi 轮流操作。如果某一方无法操作，那么他(她)就输了。dongdong 先进行第一步操作，假设双方都采用最优策略。问 dongdong 是否有必胜策略。

**输入格式**

第一行一个数 *T*，表示他们一共玩 *T* 局游戏。

接下来是 *T* 组游戏描述。每组游戏第一行两个数 *n*,*m*。

接下来 *n* 行每行 *m* 个字符，第 *i* 行第 *j* 个字符如果是 H 表示第 *i* 行第 *j* 列的硬币是正面向上，否则是反面向上。第 *i* 行 *j* 列的左上方是指行不超过 *i* 并且列不超过 *j* 的区域。

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int T, sg[105][105], n, m, a[10005], cnt, mp[1200];

bool f[300];

int lowbit(int x) {

return x & (-x);

}

int getsg(int a, int b){

if(a == 1 || b == 1){

return mp[lowbit(a + b - 1)];

}else return a + b - 2;

}

int main() {

for(int i = 0 ; i <=9 ; i++) {

mp[(1<<i)] = i;

}

cin>>T;

while(T--) {

memset(f, 0, sizeof(f));

cin>>n>>m;

int ans = 0;

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

for(int j = 1 ; j <= m ; j++) {

char c ;

scanf(" %c ", &c);

if(c !='H'){

f[getsg(i, j)] ^= 1;

}

}

}

for(int i = m + n - 1 ; i >= 0 ; i--) if(f[i]) {ans = 1;break;}

if(ans) printf("-\_-\n"); //存在

else printf("=\_=\n"); //不存在

}

return 0;

}

**题目描述**

如题，现在有一个并查集，你需要完成合并和查询操作。

**输入格式**

第一行包含两个整数 *N*,*M* ,表示共有 *N* 个元素和 *M* 个操作。

接下来 *M* 行，每行包含三个整数 *Zi*​,*Xi*​,*Yi*​ 。

当 *Zi*​=1 时，将 *Xi*​ 与 *Yi*​ 所在的集合合并。

当 *Zi*​=2 时，输出 *Xi*​ 与 *Yi*​ 是否在同一集合内，是的输出 Y ；否则输出 N 。

**输出格式**

对于每一个 *Zi*​=2 的操作，都有一行输出，每行包含一个大写字母，为 Y 或者 N 。

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5+5;

int fa[N];

int n, m;

int find(int x){

if(fa[x] == x) return x; // 已经是根节点

else return find(fa[x]); // 继续找

}

int main(){

cin >> n >> m;

for(int i = 1; i <= n; ++i) fa[i] = i; // 初始化

for(int i = 1; i <= m; ++i){

int op, x, y;

cin >> op >> x >> y;

if(op == 1){ // 合并操作

x = find(x), y = find(y); // 查询各自的代表元素

fa[x] = y; // 这里的合并顺序可以任意

}

else{ // 查询操作的一种变形

x = find(x), y = find(y); // 查询各自的代表元素

if(x == y) cout << "Y" << endl; // 是否在同一个集合

else cout << "N" << endl;

}

}

return 0;

}

Dij模板

#include <iostream>

#include <vector>

#include <queue>

#include <climits> // For INT\_MAX

using namespace std;

// 定义边的结构体：连接到哪个节点，以及边的权重

struct Edge {

int to;

int weight;

Edge(int t, int w) : to(t), weight(w) {}

};

// 定义用于优先队列的元素类型： (距离, 节点编号)

// 注意：pair的第一个元素是距离，因为pair默认按第一个元素排序

using PII = pair<int, int>; // <dist, node>

/\*\*

\* @brief Dijkstra 算法求单源最短路径

\* @param graph 邻接表表示的图。graph[i] 存储从节点 i 出发的所有边

\* @param start 源点

\* @return 一个 vector<int>，表示从 start 到每个点的最短距离。如果不可达，则为 INT\_MAX。

\*/

vector<int> dijkstra(const vector<vector<Edge>>& graph, int start) {

int n = graph.size();

// 初始化距离数组，所有距离初始为“无穷大”

vector<int> dist(n, INT\_MAX);

// 初始化一个最小堆（优先队列）

priority\_queue<PII, vector<PII>, greater<PII>> pq;

// 从源点开始

dist[start] = 0;

pq.push({0, start}); // 将源点及其距离加入队列

while (!pq.empty()) {

// 1. 取出当前距离最小的节点

auto [d, u] = pq.top();

pq.pop();

// 2. 重要：如果队列中记录的距离大于我们当前计算出的最短距离，说明这个记录是旧的、无效的，直接跳过。

// 这是一种“懒惰删除”的技巧，避免直接操作优先队列中的元素。

if (d > dist[u]) {

continue;

}

// 3. 松弛操作：遍历当前节点 u 的所有邻居

for (const Edge& edge : graph[u]) {

int v = edge.to;

int w = edge.weight;

// 尝试从 u 走到 v，看距离是否能缩短

if (dist[v] > dist[u] + w) {

// 找到了更短的路径，更新 dist[v]

dist[v] = dist[u] + w;

// 将新的距离和节点加入优先队列（即使旧记录还在队列中，新的更小的记录会先被处理，并通过上面的continue跳过旧记录）

pq.push({dist[v], v});

}

}

}

return dist;

}

int main() {

// 示例：构建一个图

int n = 5; // 节点数量为 5 (0 to 4)

vector<vector<Edge>> graph(n);

// 添加边

graph[0].push\_back(Edge(1, 4));

graph[0].push\_back(Edge(2, 1));

graph[1].push\_back(Edge(3, 1));

graph[2].push\_back(Edge(1, 2));

graph[2].push\_back(Edge(3, 5));

graph[3].push\_back(Edge(4, 3));

int start = 0;

vector<int> dist = dijkstra(graph, start);

cout << "从节点 " << start << " 到各节点的最短距离：" << endl;

for (int i = 0; i < n; ++i) {

if (dist[i] == INT\_MAX) {

cout << "节点 " << i << ": 不可达" << endl;

} else {

cout << "节点 " << i << ": " << dist[i] << endl;

}

}

return 0;

}

#include <iostream>

#include <vector>

#include <queue>

#include <climits>

#include <cstring>

using namespace std;

const int MAXN = 100010;

const int MAXM = 200010;

struct Edge {

int to;

int weight;

int next;

};

Edge edges[MAXM];

int head[MAXN];

int cnt = 0;

void init() {

cnt = 0;

memset(head, -1, sizeof(head));

}

void addEdge(int u, int v, int w) {

edges[cnt].to = v;

edges[cnt].weight = w;

edges[cnt].next = head[u];

head[u] = cnt++;

}

// 用于优先队列的元素类型

using PII = pair<int, int>; // <distance, node>

/\*\*

\* @brief 使用链式前向星的Dijkstra算法

\* @param n 节点数量

\* @param start 起始节点

\* @return 最短距离数组

\*/

vector<int> dijkstra(int n, int start) {

vector<int> dist(n + 1, INT\_MAX); // 节点从1开始编号，多分配一个

vector<bool> visited(n + 1, false); // 可选的访问标记，有时比懒惰删除更清晰

priority\_queue<PII, vector<PII>, greater<PII>> pq;

dist[start] = 0;

pq.push({0, start});

while (!pq.empty()) {

auto [d, u] = pq.top();

pq.pop();

// 如果已经用更短距离处理过这个节点，跳过

if (d > dist[u]) {

continue;

}

// 遍历节点u的所有出边

// 注意：这里使用链式前向星的遍历方式

for (int i = head[u]; i != -1; i = edges[i].next) {

int v = edges[i].to;

int w = edges[i].weight;

if (dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

pq.push({dist[v], v});

}

}

}

return dist;

}

int main() {

init(); // 初始化链式前向星

int n = 5, m = 6; // 5个节点，6条边（示例）

// 添加边（示例图）

addEdge(0, 1, 4);

addEdge(0, 2, 1);

addEdge(1, 3, 1);

addEdge(2, 1, 2);

addEdge(2, 3, 5);

addEdge(3, 4, 3);

int start = 0;

vector<int> dist = dijkstra(n, start);

cout << "从节点 " << start << " 到各节点的最短距离：" << endl;

for (int i = 0; i < n; ++i) {

if (dist[i] == INT\_MAX) {

cout << "节点 " << i << ": 不可达" << endl;

} else {

cout << "节点 " << i << ": " << dist[i] << endl;

}

}

return 0;

}

给定一个 *n* 个点，*m* 条有向边的带非负权图，请你计算从 *s* 出发，到每个点的距离。

数据保证你能从 *s* 出发到任意点。

**输入格式**

第一行为三个正整数 *n*,*m*,*s*。 第二行起 *m* 行，每行三个非负整数 *ui*​,*vi*​,*wi*​，表示从 *ui*​ 到 *vi*​ 有一条权值为 *wi*​ 的有向边。

**输出格式**

输出一行 *n* 个空格分隔的非负整数，表示 *s* 到每个点的距离。

#include<bits/stdc++.h>

const int MaxN = 100010, MaxM = 500010;

struct edge

{

int to, dis, next;

};

edge e[MaxM];

int head[MaxN], dis[MaxN], cnt;

bool vis[MaxN];

int n, m, s;

inline void add\_edge( int u, int v, int d )

{

cnt++;

e[cnt].dis = d;

e[cnt].to = v;

e[cnt].next = head[u];

head[u] = cnt;

}

struct node

{

int dis;

int pos;

bool operator <( const node &x )const

{

return x.dis < dis;

}

};

std::priority\_queue<node> q;

inline void dijkstra()

{

dis[s] = 0;

q.push( ( node ){0, s} );

while( !q.empty() )

{

node tmp = q.top();

q.pop();

int x = tmp.pos, d = tmp.dis;

if( vis[x] )

continue;

vis[x] = 1;

for( int i = head[x]; i; i = e[i].next )

{

int y = e[i].to;

if( dis[y] > dis[x] + e[i].dis )

{

dis[y] = dis[x] + e[i].dis;

if( !vis[y] )

{

q.push( ( node ){dis[y], y} );

}

}

}

}

}

int main()

{

scanf( "%d%d%d", &n, &m, &s );

for(int i = 1; i <= n; ++i)dis[i] = 0x7fffffff;

for( register int i = 0; i < m; ++i )

{

register int u, v, d;

scanf( "%d%d%d", &u, &v, &d );

add\_edge( u, v, d );

}

dijkstra();

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

printf( "%d ", dis[i] );

return 0;

}

SPFA

如题，给出一个有向图，请输出从某一点出发到所有点的最短路径长度。

**输入格式**

第一行包含三个整数 *n*,*m*,*s*，分别表示点的个数、有向边的个数、出发点的编号。

接下来 *m* 行每行包含三个整数 *u*,*v*,*w*，表示一条 *u*→*v* 的，长度为 *w* 的边。

**输出格式**

输出一行 *n* 个整数，第 *i* 个表示 *s* 到第 *i* 个点的最短路径，若不能到达则输出 231−1。

#include <iostream>

#include <fstream>

#include <queue>

#include <cstdlib>

using namespace std;

const int MAXN = 5e5 + 5;

int head[MAXN], next[MAXN], to[MAXN], weight[MAXN];

int tot = 0;

int n, m, s;

int dis[MAXN], vis[MAXN];

bool inQueue[MAXN];

queue<int> q;

void AddEdge(int u, int v, int w) {

next[++tot] = head[u];

to[tot] = v;

weight[tot] = w;

head[u] = tot;

}

bool Relax(int u, int v, int w) {

if(dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

return true;

}

return false;

}

void Error() {

cout << "negetive loop!" << endl;

exit(0);

}

void Spfa(int start) {

for(int i = 0; i < MAXN; i++) {

dis[i] = 2147483647;

}

q.push(start);

inQueue[start] = true;

dis[start] = 0;

vis[start] = 1;

while(!q.empty()) {

int u = q.front();

q.pop();

vis[u]++;

inQueue[u] = false;

if(vis[u] > n) { //判负环

Error();

}

for(int i = head[u]; i; i = next[i]) {

int v = to[i];

int w = weight[i];

if(Relax(u, v, w) && !inQueue[v]) {

q.push(v);

inQueue[v] = false;

}

}

}

}

int main() {

ifstream inFile("spfa.in");

cin >> n >> m >> s;

for(int i = 0; i < m; i++) {

int u, v, w;

cin >> u >> v >> w;

AddEdge(u, v, w);

}

Spfa(s);

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

cout << dis[i] << ' ';

}

cout << endl;

return 0;

}