2016级数据结构第十二次上机解题报告

A

该题类似与进制转换，不过要注意的是数据可能比较大，要用long long存储。

参考代码：

#include <iostream>

#include <cstdio>

using namespace std;

long long titleToNumber(string s) {

long long res = 0, cur = 1;

int len = s.length();

for (int i = len - 1; i >= 0; i--) {

res += cur \* (s[i] - 'A' + 1);

cur \*= 26;

}

return res;

}

int main() {

string s;

while (cin >> s) {

cout << titleToNumber(s) << "\n";

}

}

B

此题为最小生成树题目的一个变种：有一个点的度只能为1。

从如下角度思考：设度有限制的点为P，所有点的最小生成树为T0，去掉点P的最小生成树为T1（先去掉点再求最小生成树），T0中去掉P形成的树为T2（先求最小生成树再去掉点），那么T1和T2是等价的（参考kruskal算法的流程），并且因为P点的度最大只有1，去掉P以后，原来的最小生成树不会分裂成两棵。

所以，此题的做法为：先求出T1，然后将P加入T1。由于T1是一个已经形成的最小生成树，只要找到与P相连的权值最小的边，将其加入T1即可。

参考代码：

#include <iostream>

#include <algorithm>

using namespace std;

#define MaxN 10000

#define MaxM 600000

struct edge {

int x, y, w;

edge(int x = 0, int y = 0, int w = 0) : x(x), y(y), w(w) {}

} e[MaxM];

bool cmp(const edge &edge1, const edge &edge2) {

return (edge1.w < edge2.w);

}

int father[MaxN];

int getfather(int x) {

if (x == father[x]) return x;

else return father[x] = getfather(father[x]);

}

void unite(int x, int y) {

father[x] = father[y];

}

int kruscal(int N, int M) {

sort(e, e + M, cmp);

int cnt = N;

int ans = 0;

for (int i = 1; i <= N; ++i) father[i] = i;

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

int t1 = getfather(e[i].x);

int t2 = getfather(e[i].y);

if (t1 != t2) {

unite(t1, t2);

ans += e[i].w;

if (cnt == 1) break;

}

}

return ans;

}

#define Min(x, y) ((x)<(y)?(x):(y))

#define Adjust(x, k) ((x)>(k)?(x-1):(x))

int main() {

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

cin.tie(0);

cout.tie(0);

int n, m, k, M = 0, minDist = INT\_MAX;

cin >> n >> m >> k;

while (m--) {

int x, y, w;

cin >> x >> y >> w;

if (x != k && y != k)

e[M++] = edge(Adjust(x, k), Adjust(y, k), w);

else

minDist = Min(minDist, w);

}

cout << kruscal(n - 1, M) + minDist << "\n";

return 0;

}

C

这道题就是判断一个图的连通性，我们只需要从某个节点开始DFS，如果遍历了所有的点，便表示这个图是连通的。

参考代码：

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int maxn = 100;

int m, k;

bool ways[maxn][maxn];

bool vis[maxn];

void dfs(int s) {

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

if (!vis[i] && ways[s][i]) {

vis[i] = true;

dfs(i);

}

}

}

int main() {

while (~scanf("%d%d", &k, &m)) {

int u, v, res = 1;

memset(ways, 0, sizeof(ways));

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

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

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

ways[u][v] = true;

}

dfs(0);

vis[0] = true;

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

if (!vis[i]) {

res = 0;

break;

}

}

printf(res ? "Yes\n" : "No\n");

}

}

D

此题即为，给出一张图，求出图中连通块的个数。

使用并查集可以优雅高效的完成这一题。直接DFS遍历也是可以的。

参考代码：

#include <iostream>

using namespace std;

#define MaxN 1001

int father[MaxN];

int getfather(int x) {

if (x == father[x]) return x;

else return father[x] = getfather(father[x]);

}

void unite(int x, int y) {

father[getfather(x)] = getfather(y);

}

int main() {

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

cin.tie(0);

cout.tie(0);

int n, m, x, y;

cin >> n >> m;

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

father[i] = i;

int cnt = n;

while (m--) {

cin >> x >> y;

if (getfather(x) != getfather(y)) {

unite(x, y);

cnt--;

}

}

cout << cnt << "\n";

return 0;

}

E

这题看起来和上一题比较类似，不过有一点不同的是，如果我们用上一题一样的方法，内存占用会特别大，而且还会超时，所以我们可以用并查集，每次输入一条边的时候，如果他们父节点不同，便将其中一个节点设为另一个的父节点，同时加上子节点的总数。

参考代码：

#include <iostream>

#include <cstdio>

#include <cstring>

int n, m;

int node[100000], father[100000];

int getFather(int u) {

if (u != father[u]) {

node[father[u]] += node[u];

node[u] = 0;

father[u] = getFather(father[u]);

}

return father[u];

}

int main() {

int u, v;

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

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

father[i] = i;

}

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

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

u = getFather(u);

v = getFather(v);

if (u != v) {

father[v] = u;

node[u] += node[v] + 1;

node[v] = 0;

}

}

int ans = 0;

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

if (i == father[i] && ans < node[i]) {

ans = node[i];

}

}

printf("%d\n", ans + 1);

}

F

先跑一遍最短路算出每两个主节点之间的最短距离，然后根据每一个t判断是否可以坐这辆车从一个节点到另一个，把c当做边权重新建图，再跑一遍最短路算出结果。

参考代码：

#include <cstdio>

#include <algorithm>

#include <vector>

#include <queue>

using namespace std;

#define inf 1000000000000000000LL

long long mp[1001][1001];

long long mp2[1001][1001];

vector<pair<long long,long long> > edge[1001];

long long n,m,x,y,u,v,w,t,c;

void dijkstra(long long v,long long \*d)

{

long long t;

priority\_queue<pair<long long,long long>,vector<pair<long long,long long> >,greater<pair<long long,long long> > > pq;

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

d[i]=inf;

d[v]=0;

pq.push(make\_pair(0,v));

while(!pq.empty())

{

t=pq.top().second;

pq.pop();

for(long long i=0;i<edge[t].size();i++)

if(d[edge[t][i].first]>d[t]+edge[t][i].second)

{

d[edge[t][i].first]=d[t]+edge[t][i].second;

pq.push(make\_pair(d[edge[t][i].first],edge[t][i].first));

}

}

return;

}

int main()

{

while(scanf("%lld%lld",&n,&m)!=EOF)

{

scanf("%lld%lld",&x,&y);

for(long long i=0;i<m;i++)

{

scanf("%lld%lld%lld",&u,&v,&w);

edge[u].push\_back(make\_pair(v,w));

edge[v].push\_back(make\_pair(u,w));

}

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

dijkstra(i,mp[i]);

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

edge[i].clear();

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

{

scanf("%lld%lld",&t,&c);

for(long long j=1;j<=n;j++)

if(j!=i&&mp[i][j]<=t)

edge[i].push\_back(make\_pair(j,c));

}

dijkstra(x,mp2[x]);

if(mp2[x][y]!=inf)

printf("%lld\n",mp2[x][y]);

else printf("-1\n");

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

edge[i].clear();

}

return 0;

}

G

二分图，把行和列看成点，把’#’的行列连上一条边，把所有边权值都看作1，跑最短路即可。

参考代码：

#include <cstdio>

#include <vector>

#include <queue>

using namespace std;

#define inf 0x7fffffff

char mp[1001][1001];

vector<int> edge[2001];

int d[2001];

int n,m;

void dijkstra(int v,int \*d)

{

int t;

priority\_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int> > > pq;

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

d[i]=inf;

d[v]=0;

pq.push(make\_pair(0,v));

while(!pq.empty())

{

t=pq.top().second;

pq.pop();

for(long long i=0;i<edge[t].size();i++)

if(d[edge[t][i]]>d[t]+1)

{

d[edge[t][i]]=d[t]+1;

pq.push(make\_pair(d[edge[t][i]],edge[t][i]));

}

}

return;

}

int main()

{

while(scanf("%d%d",&n,&m)!=EOF)

{

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

scanf("%s",mp[i]);

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

for(int j=0;j<m;j++)

if(mp[i][j]=='#')

{

edge[i].push\_back(j+n);

edge[j+n].push\_back(i);

}

dijkstra(0,d);

if(d[n-1]!=inf)

printf("%d\n",d[n-1]);

else printf("-1\n");

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

edge[i].clear();

}

return 0;

}