10.1.3 链式前向星

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include<bits/stdc++.h>  using namespace std;  const int N = 1e6+5, M = 2e6+5; //1百万个点,2百万条边  int head[N],cnt; //cnt记录当前存储位置  struct {  int from, to, next; //from=边的起点u；to=边的终点v；next = u的下一个邻居  int w; //边权，根据题目设定有int，double等类型  }edge[M]; //存边  void init(){ //链式前向星初始化  for(int i=0; i<N; ++i) head[i] = -1; //点初始化  for(int i=0; i<M; ++i) edge[i].next = -1; //边初始化  cnt = 0;  }  void addedge(int u, int v, int w){ //前向星存边(u,v)，边的权值为w  edge[cnt].from = u; //一般情况下，这一句是多余的。  edge[cnt].to = v;  edge[cnt].w = w;  edge[cnt].next = head[u];  head[u] = cnt++;  }  int main(){  init(); //前向星初始化  int n, m; cin>>n>>m; // 输入n个点，m条边  for(int i=0;i<m;i++){int u,v,w; cin>>u>>v>>w; addedge(u,v,w);} //存m条有向边  for(int i=0;i<=n;i++) printf("h[%d]=%d,",i,head[i]); printf("\n"); //打印head[]  for(int i=0;i<m;i++) printf("e[%d].to=%d,",i,edge[i].to); printf("\n"); //打印edge[].to  for(int i=0;i<m;i++) printf("e[%d].nex=%d,",i,edge[i].next); printf("\n"); //打印edge[].next  for(int i=head[2]; ~i; i=edge[i].next) //遍历结点2的所有邻居。~i可写为i!=-1  printf("%d ",edge[i].to); //printf("%d-%d ",edge[i].from,edge[i].to);  return 0;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include<bits/stdc++.h>  using namespace std;  const int N = 1e6+5, M = 2e6+5; //1百万个点,2百万个边  int cnt=0,head[N]; //cnt等于其他值也行，根据题目要求赋值  struct {int to, next, w;} edge[M];  void addedge(int u,int v,int w) {  cnt++;  edge[cnt].to = v;  edge[cnt].w = w;  edge[cnt].next = head[u];  head[u] = cnt;  }  int main() {  int n, m; cin>>n>>m;  for(int i=0;i<m;i++){int u,v,w; cin>>u>>v>>w; addedge(u,v,w);}  for(int i=head[2]; i>0; i=edge[i].next) //遍历结点2的所有邻居  printf("%d ",edge[i].to); //输出：5 4 3 1  return 0;  } |

10.2.4 输出拓扑排序

|  |  |  |  |
| --- | --- | --- | --- |
| 例10-1. Following Orders Poj 1270 | | | |
|  | |  | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | //改写自 https://blog.csdn.net/iteye\_4476/article/details/82335629  #include <cstdio>  #include <algorithm>  #include <cstring>  using namespace std;  int n,a[25],dir[30][30]; //dir[i][j]=1表示i、j是先后关系  int topo[25],vis[30],indegree[30]; //topo[]记录拓扑排序，indegree记录度，vis标记是否访问  void dfs(int z,int cnt){  int i;  topo[cnt]=z; //记录拓扑排序  if(cnt==n-1) { //所有点取完了，输出一个拓扑排序  for(i=0;i<n;i++) printf("%c",topo[i]+'a');  printf("\n");  return ;  }  vis[z]=1; //标记为已访问  for(i=0;i<n;i++){  if(!vis[a[i]] && dir[z][a[i]] )  indegree[a[i]] --; //把所有下属的度数减一  }  for(i=0;i<n;i++)  if(!indegree[a[i]] && !vis[a[i]] ) //度数为0的继续取。  dfs(a[i],cnt+1);  for(i=0;i<n;i++){  if(!vis[a[i]] && dir[z][a[i]] )  indegree[a[i]] ++;  }  vis[z]=0; //恢复  }  int main(){  char s[100];  int len;  while(gets(s)!=NULL){  memset(dir,0,sizeof(dir));  memset(vis,0,sizeof(vis));  memset(indegree,0,sizeof(indegree));  len=strlen(s);  n=0;  for(int i=0;i<len;i++) //存字母到a[]  if(s[i]<='z' && s[i]>= 'a')  a[n++]= s[i]-'a';  sort(a,a+n); //对字母排序，这样就能按字典序输出了  gets(s);  len=strlen(s);  int first=1; //first = 1表示当前字母是起点  for(int i=0;i<len;i++) { //处理先后关系  int st,ed;  if(first && s[i]<='z' && s[i] >='a'){ //起点  first=0;  st=s[i]-'a'; //把字母转化为数字  continue;  }  if(!first && s[i]<='z' && s[i] >='a'){ //终点  first=1;  ed=s[i]-'a';  dir[st][ed]=1; //记录先后关系  indegree[ed]++; //记录度数，终点的入度加1  continue;  }  }  for(int i=0;i<n;i++)  if(!indegree[a[i]]) //从所有入度为0的点开始  dfs(a[i],0);  printf("\n");  }  return 0;  } | |

10.3.1 欧拉路和欧拉回路的存在性判断

|  |  |
| --- | --- |
| 例10-2. The Necklace uva 10054 https://vjudge.net/problem/UVA-10054 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | #include<bits/stdc++.h>  using namespace std;  const int N = 55;  int degree[N]; //记录度  int G[N][N]; //存图  void euler(int u){ //从u开始DFS  for(int v = 1; v <= 50; v++) { //v是u的邻居  if(G[u][v]) {  G[u][v]--;  G[v][u]--;  euler(v);  cout << v << " " << u << endl; //在euler()后打印，即回溯时打印  }  }  }  int main(){  int t; cin >> t;  int cnt = 0;  while (t--) {  cnt++;  if(cnt != 1) cout << endl;  cout << "Case #" << cnt << endl;  memset(degree, 0, sizeof(degree));  memset(G, 0, sizeof(G));  int n; cin >> n;  int color;  for(int i = 0; i < n; i++) { //输入n条边  int u, v; cin>>u>>v;  color = u; //记录一种颜色。测试的时候可能只出现某些颜色  degree[u]++;  degree[v]++; //记录点的度  G[u][v]++;  G[v][u]++; //存图：=0不连接，=1连接，>1有重边  }  int ok = 1;  for(int i = 1; i <= 50; i++)  if(degree[i] % 2) { //存在奇点，无欧拉路  cout<<"some beads may be lost"<<endl;  ok = 0;  break;  }  if(ok) euler(color); //有欧拉路。随便从某个存在的颜色开始  }  return 0;  } |

10.3.2 输出一个欧拉回路

|  |  |
| --- | --- |
| 例10-2. Code poj 1780 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include <stdio.h>  const int N = 1e5;  int num[N]; //num[v]：点v后加的数字，num[v]=0~9  int st\_edge[10\*N], top\_s; //栈，用于存边。top\_s指示栈顶  char st\_ans [10\*N]; int top\_a; //栈，存序列结果。top\_a指示栈顶  int m;  void no\_dfs(int v){ //模拟递归，递归搜点v的10条边，放进st\_edge中  int edge; //边的值  while(num[v]<10){ //在点v(是一个n-1位序列)后加0~9构成10条边  edge=10\*v + num[v]; //数字edge代表一个边  num[v]++; //点v添的下一个数字。按字典序递增  st\_edge[top\_s++] = edge; //把边存入到栈st\_edge中,它是字典序的  //printf("%02d -> ",v); //打印边的起点  v = edge%m; //更新起点为原来的终点，往下走。点值等于edge的后几位  //printf("%02d: edge=%03d\n",v,edge); //打印边的终点、边的权值  }  }  int main(){  int n, edge;  while(scanf("%d",&n)&&n!=0){  top\_s = top\_a = edge = 0;  m = 1;  for(int i=0;i<n-1;++i) m\*=10; //m是点的数量，共10^(n-1)个点  for(int i=0;i<m; i++) num[i]=0;  no\_dfs(0); //从起点0开始，递归点0的10条边  while(top\_s){ //继续走  edge = st\_edge[--top\_s];  st\_ans[top\_a++] = edge%10+'0'; //只需要存边值的最后一位  no\_dfs(edge/10); //边值的前n-1位，即上一个点，作用类似DFS的回溯  }  for(int i=1;i<n;++i) printf("0"); //打印第一组数，就是n个0  while(top\_a) printf("%c",st\_ans[--top\_a]); //打印其他组数，每组打印1位  printf("\n");  }  return 0;  } |

10.4 无向图的连通性

|  |  |
| --- | --- |
| 例10-3. network poj 1144 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include<algorithm>  #include<cstring>  #include<vector>  using namespace std;  const int N=109;  int low[N],num[N],dfn;  bool iscut[N];  vector <int> G[N];  void dfs(int u, int fa){ //u的父结点是fa  low[u] = num[u] = ++ dfn; //初始值  int child = 0; //孩子数目  for(int i = 0;i < G[u].size(); i++) { //处理u的所有子结点  int v = G[u][i];  if (!num[v]) { //v没访问过  child++;  dfs(v, u);  low[u] = min(low[v], low[u]); //用后代的返回值更新low值  if (low[v] >= num[u] && u !=1)  iscut[u] = true; //标记割点  }  else if(num[v]<num[u] && v!=fa) //处理回退边  low[u] = min(low[u], num[v]);  }  if (u == 1 && child >= 2) //根结点，有两个以上不相连的子树  iscut[1] = true;  }  int main(){  int ans,n;  while(scanf("%d",&n)!=-1){  if (n==0) break;  memset(low,0,sizeof(low));  memset(num,0,sizeof(num));  dfn = 0;  for (int i=0;i<=n;i++) G[i].clear();  int a,b;  while (scanf("%d",&a)&&a)  while (getchar()!='\n'){  scanf("%d",&b);  G[a].push\_back(b); G[b].push\_back(a); //双向边  }  memset(iscut,false,sizeof(iscut));  ans = 0;  dfs(1,1);  for (int i=1;i<=n;i++) ans+=iscut[i];  printf("%d\n",ans);  }  } |

10.4.2 双连通分量

|  |
| --- |
| 例10-4. Road Construction poj 3352 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | #include<cstring>  #include<vector>  #include<stdio.h>  using namespace std;  const int N = 1005;  int n, m, low[N], dfn;  vector<int> G[N]; //存图  void dfs(int u,int fa){ //计算每个点的low值  low[u]= ++dfn;  for(int i=0;i<G[u].size();i++){  int v = G[u][i];  if(v == fa) continue;  if(!low[v]) dfs(v,u);  low[u] = min(low[u], low[v]);  }  }  int tarjan(){  int degree[N]; //计算每个缩点的度数  memset(degree,0,sizeof(degree));  for(int i=1; i<=n; i++) //把有相同low值的点看成一个缩点  for(int j=0; j<G[i].size(); j++)  if(low[i] != low[G[i][j]])  degree[low[i]]++;  int res=0;  for(int i=1;i<=n;i++) //统计度数为1的缩点个数  if(degree[i]==1) res++;  return res;  }  int main(){  while(~scanf("%d%d", &n, &m)){  memset(low, 0, sizeof(low));  for(int i=0; i<=n; i++) G[i].clear();  for(int i=1; i<=m; i++){  int a, b; scanf("%d%d", &a, &b);  G[a].push\_back(b); G[b].push\_back(a);  }  dfn = 0;  dfs(1,-1);  int ans = tarjan();  printf("%d\n",(ans+1)/2);  }  return 0;  } |

10.5 有向图的连通性

|  |
| --- |
| 例10-5. 迷宫城堡 hdu 1269 |

下面给出hdu 1269 的Kosaraju算法代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | //部分代码参考《算法竞赛入门经典（第2版）》，刘汝佳，清华大学出版社，320页。  #include<bits/stdc++.h>  using namespace std;  const int N = 10005;  vector<int> G[N], rG[N];  vector<int> S; //存第一次dfs1的结果：标记点的先后顺序  int vis[N], sccno[N], cnt; // cnt：强连通分量的个数  void dfs1(int u) {  if(vis[u]) return;  vis[u] = 1;  for(int i=0; i<G[u].size(); i++) dfs1(G[u][i]);  S.push\_back(u); //记录点的先后顺序，标记大的放在S的后面  }  void dfs2(int u) {  if(sccno[u]) return;  sccno[u] = cnt;  for(int i=0; i < rG[u].size(); i++) dfs2(rG[u][i]);  }  void Kosaraju(int n) {  cnt = 0;  S.clear();  memset(sccno, 0, sizeof(sccno));  memset(vis, 0, sizeof(vis));  for(int i = 1; i <= n; i++) dfs1(i); //点的编号：1~n。递归所有点  for(int i = n-1; i >= 0; i--)  if(!sccno[S[i]]) { cnt++; dfs2(S[i]);}  }  int main(){  int n, m, u, v;  while(scanf("%d%d", &n, &m), n != 0 || m != 0) {  for(int i = 0; i < n; i++) { G[i].clear(); rG[i].clear();}  for(int i = 0; i < m; i++){  scanf("%d%d", &u, &v);  G[u].push\_back(v); //原图  rG[v].push\_back(u); //反图  }  Kosaraju(n);  printf("%s\n", cnt == 1 ? "Yes" : "No");  }  return 0;  } |

10.5.2 Tarjan算法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | #include<bits/stdc++.h>  using namespace std;  const int N = 10005;  int cnt; // 强连通分量的个数  int low[N], num[N], dfn;  int sccno[N], stack[N], top; // 用stack[]处理栈，top是栈顶  vector<int> G[N];  void dfs(int u){  stack[top++] = u; //u进栈  low[u]= num[u]= ++dfn;  for(int i=0; i<G[u].size(); ++i){  int v = G[u][i];  if(!num[v]){ //未访问过的点，继续dfs  dfs(v); //dfs的最底层，是最后一个SCC  low[u]= min( low[v], low[u] );  }  else if(!sccno[v]) //处理回退边  low[u]= min( low[u], num[v] );  }  if(low[u] == num[u]){ //栈底的点是SCC的祖先，它的low = num  cnt++;  while(1){  int v = stack[--top]; //v弹出栈  sccno[v]= cnt;  if(u==v) break; //栈底的点是SCC的祖先  }  }  }  void Tarjan(int n){  cnt = top = dfn = 0;  memset(sccno,0,sizeof(sccno));  memset(num,0,sizeof(num));  memset(low,0,sizeof(low));  for(int i=1; i<=n; i++)  if(!num[i])  dfs(i);  }  int main(){  int n,m,u,v;  while(scanf("%d%d", &n, &m), n != 0 || m != 0) {  for(int i=1; i<=n; i++){ G[i].clear();}  for(int i=0; i<m; i++){  scanf("%d%d", &u, &v);  G[u].push\_back(v);  }  Tarjan(n);  printf("%s\n", cnt == 1 ? "Yes" : "No" );  }  return 0;  } |

10.6 基环树

|  |  |
| --- | --- |
| 例10-6. 骑士 洛谷P2607 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N = 1e6 + 100;  vector<int> G[N];  int father[N],val[N],mark;  bool vis[N];  ll dp[N][2];  void addedge(int from,int to){  G[from].push\_back(to); //用邻接表建树  father[to] = from; //父子关系  }  void dfs(int u){ //和洛谷P1352 几乎一样  dp[u][0] = 0; //赋初值：不参加  dp[u][1] = val[u]; //赋初值：参加  vis[u] = true;  for(int v : G[u]){ //遍历u的邻居v  if(v == mark) continue;  dfs(v);  dp[u][1] += dp[v][0]; //父结点选择，子结点不选  dp[u][0] += max(dp[v][0],dp[v][1]); //父结点不选，子结点可选可不选  }  }  int check\_c(ll u){ //在基环树中找环上一个点  vis[u] = true;  int f = father[u];  if(vis[f]) return f; //第2次访问到，是环上一个点  else check\_c(f); //继续向父结点方向找  }  ll solve(int u){ //检查一棵基环树  ll res = 0;  mark = check\_c(u); //mark是基环树的环上一个点  dfs(mark); //做一次dfs  res = max(res,dp[mark][0]); //mark不参加  mark = father[mark];  dfs(mark); //mark的父结点不参加，再做一次dfs  res = max(res,dp[mark][0]);  return res;  }  int main(){  int n; scanf("%d",&n);  for(int i = 1;i <= n;i++){  int d; scanf("%d%d",&val[i],&d); addedge(d,i);  }  ll ans = 0;  for(int i = 1;i <= n;i++)  if(!vis[i]) ans += solve(i); //逐个检查每棵基环树  printf("%lld\n",ans);  return 0;  } |

10.7 2-SAT

|  |  |
| --- | --- |
| 例10-7. 2-SAT 问题 洛谷P4782 | |
|  |  |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #include<bits/stdc++.h>  using namespace std;  const int N = 1e6+10;  int cur, head[N<<1];  struct {int to,next;}edge[N<<2];  void addedge(int u,int v){  edge[++cur].to = v;  edge[cur].next = head[u];  head[u] = cur;  }  int low[N<<1],num[N<<1],st[N<<1],sccno[N<<1],dfn,top,cnt;  int n,m;  void tarjan(int u){  st[top++] = u;  low[u] = num[u] = ++dfn;  for(int i=head[u]; i; i=edge[i].next) {  int v=edge[i].to;  if(!num[v]){  tarjan(v);  low[u]=min(low[u],low[v]);  }else if(!sccno[v])  low[u]=min(low[u],num[v]);  }  if(low[u]==num[u]) {  cnt++;  while(1){  int v=st[--top];  sccno[v]=cnt;  if(u==v) break;  }  }  }  bool two\_SAT(){  for(int i=1; i<=2\*n; i++)  if(!num[i])  tarjan(i); //tarjan找强连通分量  for(int i=1; i<=n; i++)  if(sccno[i]==sccno[i+n]) //a和非a在同一个强连通分量，无解  return false;  return true;  }  int main(){  scanf("%d%d",&n,&m);  while(m--){  int a,b,va,vb; scanf("%d%d%d%d",&a,&va,&b,&vb);  int nota = va^1, notb = vb^1; //非a，非b  addedge(a+nota\*n, b+vb\*n); //连边(非a，b)  addedge(b+notb\*n, a+va\*n); //连边(非b，a)  }  if(two\_SAT()){  printf("POSSIBLE\n");  for(int i=1; i<=n; i++) printf("%d ",sccno[i]>sccno[i+n]);  }  else printf("IMPOSSIBLE");  return 0;  } |

10.8.1 Floyd-Warshall

用下面的例题给出Floyd的模板代码。

|  |  |
| --- | --- |
| 例10-8. 计算最短路 https://www.lanqiao.cn/problems/1121/learning/ | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include <bits/stdc++.h>  using namespace std;  const long long INF = 0x3f3f3f3f3f3f3f3fLL; //这样定义的好处是: INF <= INF+x  const int N = 405;  long long dp[N][N];  int n,m,q;  void input(){  // for(int i = 1; i <= n; i++) //第1种初始化方法  // for(int j = 1; j <= n; j++)  // dp[i][j] = INF;  memset(dp,0x3f,sizeof(dp)); //第2种初始化方法  for(int i = 1; i <= m; i++){  int u,v; long long w; cin >> u >> v >> w;  dp[u][v] = dp[v][u] = min(dp[u][v] , w); //防止有重边  }  }  void floyd(){ //floyd算法  for(int k = 1; k <= n; k++)  for(int i = 1; i <= n; i++)  for(int j = 1; j <= n; j++)  dp[i][j] = min(dp[i][j] , dp[i][k] + dp[k][j]);  }  void output(){  int s, t;  while(q--){  cin >> s >>t;  if(dp[s][t]==INF) cout << "-1" <<endl;  else if(s==t) cout << "0" <<endl; //floyd()后，**dp[i][i]并不等于0**  else cout << dp[s][t] << endl;  }  }  int main(){  cin >> n>> m >> q;  input(); floyd(); output();  return 0;  } |

**5**

|  |  |
| --- | --- |
| 例10-9. Minimum Transport Cost hdu 1385 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include<bits/stdc++.h>  const int INF = 0x3fffffff;  const int N = 505;  int n, map[N][N], tax[N], path[N][N];  void input(){  for(int i = 1; i <= n; i++)  for(int j = 1; j <= n; j++) {  scanf("%d", &map[i][j]);  if(map[i][j] == -1) map[i][j] = INF;  path[i][j] = j; //path[i][j]: 此时i、j相邻，或者断开  }  for(int i = 1; i <= n; i++) scanf("%d", &tax[i]); //交税  }  void floyd(){  for(int k = 1; k <= n; k++)  for(int i = 1; i <= n; i++)  for(int j = 1; j <= n; j++) {  int len = map[i][k] + map[k][j] + tax[k]; //计算最短路  if(map[i][j] > len) {  map[i][j] = len;  path[i][j] = path[i][k]; //标记到该点的前一个点  }  else if(len == map[i][j] && path[i][j] > path[i][k])  path[i][j] = path[i][k]; //若距离相同，按字典序  }  }  void output(){  int s, t;  while(scanf("%d %d", &s, &t)) {  if(s == -1 && t == -1) break;  printf("From %d to %d :\n", s, t);  printf("Path: %d", s);  int k = s;  while(k != t) { //输出路径从起点直至终点  printf("-->%d", path[k][t]);  k = path[k][t]; //一步一步往终点走  }  printf("\n");  printf("Total cost : %d\n\n", map[s][t]);  }  }  int main(){  while(scanf("%d", &n), n){  input(); floyd(); output();  }  return 0;  } |

|  |  |
| --- | --- |
| 例10-11. 跑路 洛谷1613 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include<bits/stdc++.h>  using namespace std;  const int N = 55;  bool p[N][N][34];  int mp[N][N];  int main(){  memset(mp,0x3f,sizeof(mp));  int n,m; cin >> n >> m;  for( int i = 1;i <= m; i++){  int u,v; cin >> u >> v;  mp[u][v] = 1;  p[u][v][0] = true;  }  for(int t = 1;t <= 32; t++) //长度为2^t的路径  for(int k = 1;k <= n; k++) //Floyd  for(int i = 1;i <= n; i++)  for(int j = 1;j <= n; j++)  if(p[i][k][t - 1] == true && p[k][j][t - 1] == true){  p[i][j][t] = true;  mp[i][j] = 1; //计算得到新图  }  for(int k = 1;k <= n; k++) //求最短路，就用Floyd  for(int i = 1;i <= n; i++)  for(int j = 1;j <= n; j++)  mp[i][j] = min(mp[i][j],mp[i][k] + mp[k][j]);  cout << mp[1][n] << endl;  return 0;  } |

10.8.2 传递闭包

**2. 用Floyd求解传递闭包**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | //(1)普通写法  for(int k=1; k<=n; k++) //floyd的3重循环  for(int i=1; i<=n; i++)  for(int j=1; j<=n; j++)  if(dp[i][k] && dp[k][j];)  dp[i][j] = 1; //5-6行可以合并为：dp[i][j] |= dp[i][k] & dp[k][j];  //(2)简单优化  for(int k=1; k<=n; k++)  for(int i=1; i<=n; i++)  if(dp[i][k]) //先判断dp[i][k]再进入j循环，计算量略少  for(int j=1; j<=n; j++)  if(dp[k][j])  dp[i][j] = 1;  //(3)用bitset优化，见下面的例题hdu 1704 |

|  |  |  |
| --- | --- | --- |
| 例10-12. Rank hdu 1704 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | #include <bits/stdc++.h>  using namespace std;  const int N=510;  int n, m;  /\* 第二种优化  int d[N][N];  void Floyd(){  for(int k = 1; k <= n; k++)  for(int i = 1; i <= n; i++)  if(d[i][k]) // 简单优化;  for(int j = 1; j <= n; j++)  if(d[k][j]) // 等价于 if(d[k][j] == 1)  d[i][j] = 1;  }  \*/  bitset<N> d[N]; //第三种优化：用bitset加速，能解决N = 1000的问题  void Floyd(){  for(int k = 1; k <= n; k++)  for(int i = 1; i <= n; i++)  if(d[i][k])  d[i] |= d[k]; //与11-13行等价  }  int main(){  int T; scanf("%d", &T);  while(T--){  scanf("%d%d", &n, &m);  for(int i = 1; i <= n; i++) //初始化  for(int j = 1; j <= n; j++) d[i][j] = (i==j);  int u, v;  for(int i = 0; i < m; i++){scanf("%d%d", &u, &v); d[u][v] = 1;}  Floyd();  int tot = 0;  for(int i = 1; i <= n; i++)  for(int j = i+1; j <= n; j++)  if(d[i][j] == 0 && d[j][i] == 0) ++tot;  printf("%d\n", tot);  }  return 0;  } |

10.8.3 Dijkstra

（poj 2449）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | // poj 2449代码  #include <cstdio>  #include <cstring>  #include <queue>  using namespace std;  const int INF = 0x3f3f3f3f;  const int N = 1005, M = 100005;  struct edge{ //记录边  int to, w;  //vector edge[i]:起点是i;它有很多边,其中一个边的to是边的终点,w是边长  edge(int a,int b){ to = a, w = b;} //赋值  };  vector <edge>G[M], G2[M]; //G:原图; G2:反图  struct node { //用于dijkstra。记录点，以及点到起点的路径  int id, dis; //id:点；dis：点id到起点的路径长度  node(int a, int b){ id = a, dis = b;} //赋值  bool operator < (const node &u) const { return dis > u.dis; }  };  int dist[N]; //dist[i]: 从s到点i的最短路长度  bool done[N]; //done[i]=ture: 表示到i的最短路已经找到  void dijkstra(int s) { //标准的dijkstra: 求s到其他所有点的最短路  for(int i =0;i<N;i++) {dist[i]=INF; done[i]=false;} //初始化  dist[s] = 0; //起点s到自己的距离是0  priority\_queue<node> q;  q.push(node(s, dist[s])); //从起点开始处理队列  while (!q.empty()) {  node u = q.top(); //pop出距起点s最近的点u  q.pop();  if (done[u.id]) continue; //丢弃已经找到最短路的点  done[u.id] = true; //标记：点u到s的最短路已经找到  for (int i = 0; i< G2[u.id].size(); i++) { //检查点u的所有邻居  edge y = G2[u.id][i];  if (done[y.to]) continue; //丢弃已经找到最短路的邻居  if (dist[y.to] > u.dis + y.w) {  dist[y.to] = u.dis + y.w;  q.push(node(y.to, dist[y.to])); //扩展新的邻居，放进优先队列  }  }  }  }  struct point { //用于 astar  int v, g, h; //评估函数 f = g + h, g是从s到i的长度，h是从i到t的长度  point(int a, int b, int c) { v=a, g=b, h=c; }  bool operator < (const point & b) const { return g + h > b.g + b.h;}  };  int times[N]; //times[i]: 点i被访问的次数  int astar(int s, int t, int k){  memset(times, 0, sizeof(times));  priority\_queue<point> q;  q.push(point(s, 0, 0));  while (!q.empty()) {  point p = q.top(); //从优先队列中弹出f = g + h最小的  q.pop();  times[p.v]++;  if (times[p.v] == k && p.v == t) //从队列中第k次弹出t，就是答案  return p.g + p.h;  for (int i = 0; i< G[p.v].size(); i++) {  edge y = G[p.v][i];  q.push(point(y.to, p.g + y.w, dist[y.to]));  }  }  return -1;  }  int main() {  int n, m;  scanf("%d%d", &n, &m);  while (m--) {  int a, b, w; //读边：起点、终点、边长  scanf("%d%d%d", &a, &b, &w); //本题是有向图  G[a].push\_back(edge(b,w)); //原图  G2[b].push\_back(edge(a,w)); //反图  }  int s, t, k;  scanf("%d%d%d", &s, &t, &k);  if (s == t) k++; //一个小陷阱  dijkstra(t); //在反图G2上，求终点t到其他点的最短路  printf("%d\n", astar(s, t, k)); //在原图G上，求第k短路  return 0;  } |

10.8.4 Bellman-ford

|  |  |  |
| --- | --- | --- |
| 例10-14. 最短路 hdu 2544 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | #include<bits/stdc++.h>  using namespace std;  const int INF = 1e6;  const int N = 105;  struct Edge { int u, v, w; } e[10005]; //边：起点u，终点v，权值w  int n, m, cnt;  int pre[N]; //记录前驱结点，用于打印路径。pre[x]=y：在最短路径上，结点x的前一个结点是y  void print\_path(int s, int t) { //打印从s到t的最短路  if(s==t){ printf("%d ", s); return; } //打印起点  print\_path(s, pre[t]); //先打印前一个点  printf("%d ", t); //后打印当前点。最后打印的是终点t  }  void bellman(){  int s=1; //定义起点  int d[N]; //d[i]记录第i个结点到起点s的最短距离  for (int i=1; i<=n; i++) d[i]=INF; //初始化为无穷大  d[s]=0;  for (int k=1; k<=n; k++) //一共有n轮操作  for (int i=0; i < cnt; i++){ //检查每条边  int x = e[i].u, y = e[i].v;  if (d[x] > d[y] + e[i].w){ // x通过y到达起点s：如果距离更短，更新  d[x] = d[y] + e[i].w;  pre[x] = y; //如果有需要，记录路径  }  }  printf("%d\n", d[n]);  // print\_path(s,n); //如果有需要，打印路径  }  int main() {  while(~scanf("%d%d", &n, &m)) {  if(n==0 && m==0) return 0;  cnt = 0; //记录边的数量。本题的边是双向的，共有cnt=2\*m条  while (m--) {  int a,b,c; scanf("%d%d%d",&a,&b,&c);  e[cnt].u=a; e[cnt].v=b; e[cnt].w=c; cnt++;  e[cnt].u=b; e[cnt].v=a; e[cnt].w=c; cnt++;  }  bellman();  }  return 0;  } |

10.8.5 SPFA

仍然用“例10-14 hdu 2544”给出SPFA的模板代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | #include<bits/stdc++.h>  using namespace std;  const int INF = 0x3f3f3f3f;  const int N = 1e6+5, M = 2e6+5; //1百万个点,2百万个边  int n, m;  int pre[N]; //记录前驱结点，用于打印路径  void print\_path(int s, int t) { //打印从s到t的最短路  if(s==t){ printf("%d ", s); return; } //打印起点  print\_path(s, pre[t]); //先打印前一个点  printf("%d ", t); //后打印当前点。最后打印的是终点t  }  int head[N],cnt; //链式前向星  struct {int to, next, w;}edge[M]; //存边  void init(){ //链式前向星初始化  for(int i = 0; i < N; ++i) head[i] = -1; //点初始化  for(int i = 0; i < M; ++i) edge[i].next = -1; //边初始化  cnt = 0;  }  void addedge(int u, int v, int w){ //前向星存边  edge[cnt].to=v; edge[cnt].w=w; edge[cnt].next=head[u]; head[u]=cnt++;  }  int dis[N]; //dis[i]，从起点到点i的距离  bool inq[N]; //inq[i] = true 表示点i在队列中  int Neg[N]; //判断负环(Negative loop)  int spfa(int s) { //返回1表示出现负环  memset(Neg, 0, sizeof(Neg));  Neg[s] = 1;  for(int i=1; i<=n; i++) {dis[i]=INF; inq[i]=false; } //初始化  dis[s] = 0; //起点到自己的距离是0  queue<int> Q; Q.push(s); //从s开始，s进队列  inq[s] = true; //起点在队列中  while(!Q.empty()) {  int u = Q.front(); Q.pop(); //队头出队  inq[u] = false; //u已经不在队列中  for(int i=head[u]; ~i; i = edge[i].next) { //~i也可以写成 i!=-1  int v = edge[i].to, w = edge[i].w; //v是u的第i个邻居  if (dis[u]+w < dis[v]) { //u的第i个邻居v，它借道u，到s更近  dis[v] = dis[u]+w; //更新邻居v到s的距离  pre[v] = u; //如果有需要，记录路径  if(!inq[v]) { //邻居v更新状态了，但v不在队列中，放进队列  inq[v] = true;  Q.push(v);  Neg[v]++; //点v进入队列的次数  if(Neg[v] > n) return 1; //出现负环  }  }  }  }  return 0;  }  int main() {  while(~scanf("%d%d",&n,&m)) {  init(); //前向星初始化  if(n==0 && m==0) return 0;  while(m--){  int u,v,w; scanf("%d%d%d",&u,&v,&w);  addedge(u,v,w); addedge(v,u,w); //双向边  }  spfa(1); //计算起点1到其他所有点的最短路  printf("%d\n",dis[n]); //打印从1到n的最短距离  //printf("path:"); print\_path(1,n); printf("\n"); //如有需要，打印从s到t的路径  }  return 0;  } |
| 例10-15. Sightseeing Cows 洛谷P2868，Poj 3621 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //洛谷P2868，Poj 3621的部分代码  #include<bits/stdc++.h>  using namespace std;  const int N = 1e4+5, M = 5e4+5;  double dis[N]; //距离  //下面的定义与上题hdu 2544的代码几乎一样，唯一的区别是把int w改为double w  //bool inq[N]; int Neg[N]; head[N],cnt; edge[M];init(); addedge(); spfa();  int f[N]; //乐趣值  int u[M],v[M],w[M]; //记录边  bool check(double x){  init(); //前向星初始化  for(int i=1;i<=m;++i) addedge(u[i],v[i],x \* w[i] - f[u[i]]); //修改边的权值  return spfa(1); //若有负环，从任意一个点出发都会遇到负环，这里选从1出发  }  int main(){  cin>>n>>m;  for(int i = 1;i <= n;++i) cin >> f[i]; //注意i从1开始，因为结点编号是1~n  for(int i = 1;i <= m;++i) cin >> u[i] >> v[i]>>w[i];  double L = 0,R = 0;  for (int i = 1; i <= n; i++) R += f[i]; //R的初值  for (int i = 0; i < 30; i++) { //实数二分。如果大于30，在poj提交会超时  double mid = L+(R-L)/2;  if(check(mid)) L = mid; //F < 0,放大  else R = mid; //F >= 0,缩小  }  printf("%.2f", L);  return 0;  } |

10.9.1 Kruskal算法

|  |
| --- |
| 例10-17. 最小生成树 洛谷 P3366 |

代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include <bits/stdc++.h>  using namespace std;  const int N = 5005,M = 2e5+1;  struct Edge{int u,v,w;}edge[M]; //用最简单且最省空间的结构体数组存边  bool cmp(Edge a, Edge b){ return a.w < b.w;} //从小到大排序  int s[N]; //并查集  int find\_set(int x){ //查询并查集，返回x的根  if(x != s[x])  s[x] = find\_set(s[x]); //路径压缩  return s[x];  }  int n,m; // n个点，m条边  void kruskal(){  sort(edge+1, edge+m+1, cmp); //对边做排序  for(int i=1; i<=n; i++) s[i]=i; //并查集初始化  int ans = 0, cnt=0; //cnt: 已经加入MST的边数  for(int i=1; i<=m; i++){ //贪心：逐一加入每条边  if(cnt == n-1) break; //小优化：不要也行  int e1 = find\_set(edge[i].u); //边的前端点u属于哪个集？  int e2 = find\_set(edge[i].v); //边的后端点v属于哪个集？  if(e1 == e2) continue; //属于同一个集：产生了圈，丢弃  else{ //不属于同一个集  ans += edge[i].w; //计算MST  s[e1]= e2; //合并  cnt++; //统计MST中的边数  }  }  if(cnt == n-1) cout << ans; //n-1条边  else cout << "orz"; //图不是连通的  }  int main(){  cin >> n >> m;  for(int i=1; i<=m; i++) cin >> edge[i].u >> edge[i].v >> edge[i].w;  kruskal();  return 0;  } |

10.9.2 Prim算法

下面是“例10-17. 洛谷 P3366”的代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | #include <bits/stdc++.h>  using namespace std;  const int N=5005,M = 2e5+1;  struct edge{ //记录边  int to, w;  edge(int a,int b){ to = a, w = b;} //赋值  };  vector <edge>G[M];  struct node {  int id, dis; //id:点；dis：边  node(int a, int b){ id = a, dis = b;} //赋值  bool operator < (const node &u) const { return dis > u.dis; }  };  int n, m;  bool done[N]; //done[i]=ture: 表示点i已经在MST中  void prim() { //对比dijkstra: 求s到其他所有点的最短路  int s = 1; //从任意点开始，例如从1开始  for(int i =1;i<=N;i++) done[i]=false; //初始化  priority\_queue<node> q;  q.push(node(s, 0)); //从s点开始处理队列  int ans=0,cnt=0;  while (!q.empty()) {  node u = q.top(); q.pop(); //pop出距集合U最近的点u  if (done[u.id]) continue; //丢弃已经在MST中的点，有判圈的作用  done[u.id] = true; //标记  ans += u.dis;  cnt++; //统计点数  for (int i = 0; i< G[u.id].size(); i++) { //检查点u的所有邻居  edge y = G[u.id][i]; //一个邻居y  if (done[y.to]) continue; //丢弃已经在MST中的点  q.push(node(y.to, y.w)); //扩展新的邻居，放进优先队列  }  }  if(cnt == n) cout << ans; //cnt=n个点。注意在kruskal代码中cnt是边数  else cout << "orz";  }  int main() {  cin>>n>>m;  for(int i=1; i<=m; i++) {  int a,b,w; cin>>a>>b>>w;  G[a].push\_back(edge(b,w)); G[b].push\_back(edge(a,w)); //双向边  }  prim();  return 0;  } |

10.10 最大流

|  |
| --- |
| 例10-19. 网络最大流 洛谷 P3376 |

10.10.2 Edmonds-Karp算法

下面是洛谷3376的代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | #include<bits/stdc++.h>  using namespace std;  const int INF = 1e9;  const int N = 250;  #define ll long long  int n, m;  ll graph[N][N], pre[N]; //graph[][]不仅记录图，还是残留网络  ll flow[N];  ll bfs(int s,int t){ //一次bfs找一条增广路  memset(pre,-1,sizeof pre);  flow[s] = INF; pre[s] = 0; //初始化起点  queue<int> Q; Q.push(s); //起点入栈，开始bfs  while(!Q.empty()){  int u = Q.front(); Q.pop();  if(u==t) break; //搜到一个路径，这次bfs结束  for(int i=1; i<=n; i++){ //bfs所有的点  if(i!=s && graph[u][i]>0 && pre[i]==-1){  pre[i] = u; //记录路径  Q.push(i);  flow[i] = min(flow[u], graph[u][i]); //更新结点流量  }  }  }  if(pre[t]==-1) return -1; //没有找到新的增广路  return flow[t]; //返回这个增广路的流量  }  ll maxflow(int s, int t){  ll Maxflow = 0;  while(1){  ll flow = bfs(s,t); //执行一次bfs，找到一条路径，返回路径的流量  if(flow == -1) break; //没有找到新的增广路，结束  int cur = t; //更新路径上的残留网络  while(cur!=s){ //一直沿路径回溯到起点  int father = pre[cur]; //pre[]记录路径上的前一个点  graph[father][cur] -= flow; //更新残留网络：正向边减  graph[cur][father] += flow; //更新残留网络：反向边加  cur = father;  }  Maxflow += flow;  }  return Maxflow;  }  int main(){  int s,t; scanf("%d%d%d%d",&n,&m,&s,&t);  memset(graph,0,sizeof graph);  for(int i=0; i<m; i++){  int u,v,w; scanf("%d%d%d",&u,&v,&w);  graph[u][v] += w; //可能有重边  }  printf("%ld\n",maxflow(s,t));  return 0;  } |

10.10.3 Dinic算法

下面重新用Dinic算法实现洛谷P3376。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | //代码改写自：www.luogu.com.cn/blog/Eleven-Qian-ssty/solution-p3376  #include <bits/stdc++.h>  using namespace std;  #define ll long long  const ll INF=1e9;  int n,m,s,t;  const int N=250, M=11000; //M定义为边的两倍：双向边  int cnt=1,head[N]; //8-16行：链式前向星。cnt初值不能是0，可以是1、3、5  struct {int to, nex, w;} e[M];  void add(int u,int v,int w) {  cnt++; //cnt初值是1，cnt++后第一个存储位置是cnt=2，是偶数位置  e[cnt].to = v;  e[cnt].w = w;  e[cnt].nex = head[u];  head[u] = cnt;  }  int now[N],dep[N]; //dep[]记录点所在的层次(深度)  int bfs() { //在残留网络中构造分层图  for(int i=1;i<=n;i++) dep[i]=INF;  dep[s] = 0; //从起点s开始分层  now[s] = head[s]; //当前弧优化。now是head的拷贝  queue<int>Q; Q.push(s);  while(!Q.empty()) {  int u = Q.front(); Q.pop();  for(int i=head[u]; i>0;i=e[i].nex) { //搜点u的所有邻居，邻居是下一层  int v = e[i].to;  if(e[i].w>0 && dep[v]==INF) { // e[i].w>0表示还有容量  Q.push(v);  now[v] = head[v];  dep[v] = dep[u]+1; //分层：u的邻居v是u的下一层  if(v==t) return 1; //搜到了终点，返回1  }  }  }  return 0; //如果通过有剩余容量的边无法到达终点t，即t不在残留网络中，返回0  }  int dfs(int u,ll sum) { //sum是这条增广路对最大流的贡献  if(u==t) return sum;  ll k,flow=0; //k是当前最小的剩余容量  for(int i=now[u]; i>0 && sum>0; i=e[i].nex) {  now[u]=i; //当前弧优化  int v=e[i].to;  if(e[i].w>0 && (dep[v]==dep[u]+1)){ //分层：用dep限制只能访问下一层  k = dfs(v,min(sum,(ll)e[i].w));  if(k==0) dep[v] = INF; //剪枝，去掉增广完毕的点。其实把INF写成0也行  e[i].w -= k; //更新残留网络：正向减  e[i^1].w += k; //更新残留网络：反向加。小技巧：奇偶边  flow += k; //flow表示经过该点的所有流量和  sum -= k; //sum表示经过该点的剩余流量  }  }  return flow;  }  int main() {  scanf("%d%d%d%d",&n,&m,&s,&t);  for(int i=1;i<=m;i++) {  int u,v,w; scanf("%d%d%lld",&u,&v,&w);  add(u,v,w); add(v,u,0); //双向边，反向边的容量是0  }  ll ans=0;  while(bfs()) ans += dfs(s,INF); //先后做BFS和DFS。当t不在残留网络中时退出  printf("%lld",ans);  return 0;  } |

10.10.4 ISAP算法

下面用ISAP重新实现洛谷P3376。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97 | #include <bits/stdc++.h>  using namespace std;  #define ll long long  const ll INF=1e9;  int n,m,s,t ;  const int N=250, M=11000; //M定义为边的两倍：双向边  int cnt=1,head[N]; //链式前向星  struct{int from, to, nex, w;} e[M];  void add(int u, int v, int w){  cnt++;  e[cnt].from = u;  e[cnt].to = v;  e[cnt].w = w;  e[cnt].nex = head[u];  head[u] = cnt;  }  int now[M], pre[M]; //pre[]用于记录路径，pre[i]是路径上点i(的存储位置)的前一个点(的存储位置)  int dep[M], gap[M]; //dep[i]: 点i的高度；gap[i]:高度为i的点的数量  void bfs(){ //用BFS确定各顶点到汇点的距离  memset(gap,0,sizeof(gap)); //初始化间隙数组  memset(dep,0,sizeof (dep)); //所有点的高度初始为0  dep[t] = 1; //汇点t的高度是1，其他点的高度都大于1  queue<int> Q; Q.push(t);  while(!Q.empty()){  int u = Q.front(); Q.pop();  gap[dep[u]]++; //间隙数组：计算高度为dep[u]的结点个数  for(int i=head[u]; i>0; i=e[i].nex){  int v = e[i].to; //v是u的邻居点  if(e[i^1].w && dep[v]==0){ //反向边不用处理；高度不等于0的已经处理过了  dep[v] = dep[u]+1;  Q.push(v);  }  }  }  }  ll Augment(){ //沿着增广路径更新残留网络  ll v = t, flow = INF;  while(v != s){ //找这个路径的流量  int u = pre[v]; //u是v向源点s方向回退的上一个点，相当于DFS的回溯  if(e[u].w < flow) flow = e[u].w; //路径上的最小容量就是这个路径的流量  v = e[u].from; //更新v，继续回退  }  v = t;  while(v != s){ //更新残留网络  int u=pre[v]; //向源点s方向回退  e[u].w -= flow; //正向边  e[u^1].w += flow; //反向边。用到了奇偶边的技巧  v = e[u].from; //更新v，继续回退  }  return flow; //返回这个路径的流量  }  void ISAP(){  bfs(); //用bfs()求每个点到汇点的距离（高度）  ll flow = 0; //计算最大流  int u = s; //从源点s开始找增广路径  memcpy(now, head, sizeof (head)); //当前弧优化。now是head的拷贝  while(dep[s] <= n){ //最大距离（高度）是n  if(u == t){ //找到了一条增广路径  flow += Augment(); //更新残留网络，更新流量  u = s; //回到s，重新开始找一条增广路径  }  bool ok = 0; //用于判断是否能顺利往下走  for(int i=now[u]; i; i=e[i].nex){ //在u的邻居中确定路径的下一个点  int v = e[i].to; //v是u的邻居点  if(e[i].w && dep[v]+1==dep[u]){ //沿着高度递减的方向找下一个点  ok = 1; //顺利找到了路径的下一个点  pre[v] = i; //记录路径  now[u] = i; //记录当前弧，下一次跳过它  u = v; //u更新为路径的下一个点v  break; //退出for，回到while循环，继续找路径的下一个点  }  }  if(!ok){ //路径走不下去了，需要更新u的高度重新走  if(!--gap[dep[u]]) break; //u的下一个深度的点没有了，断层了，退出while  int mindep = n+10; //mindep用于计算u的邻居点的最小高度。初值比n大就行  for(int i=head[u]; i; i=e[i].nex){ //在u的邻居中找最小的高度  int v = e[i].to;  if(dep[v]<mindep && e[i].w)  mindep = dep[v];  }  dep[u] = mindep+1; //更新u的高度，改为比v大1，从而能够生成一条路径，继续往后走  gap[dep[u]]++; //更新间隙数组：高度为dep[u]的结点个数加1  now[u] = head[u]; //记录当前弧  if(u != s) u = e[pre[u]].from; //回退一步，相当于DFS的回溯  }  }  printf("%ld", flow); //打印最大流  }  int main(){  scanf("%d%d%d%d", &n,&m,&s,&t);  for(int i=1; i<=m; ++i){  int u,v,w; scanf("%d%d%d",&u,&v,&w);  add(u,v,w); add(v,u,0); //反向边的初始容量是0  }  ISAP();  return 0;  } |

10.11 二分图

|  |
| --- |
| 例10-20. 二分图最大匹配 洛谷P3386（hdu 2063 过山车） |

**3. 用匈牙利算法求解二分图最大匹配**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include<bits/stdc++.h>  using namespace std;  int G[510][510];  int match[510], reserve\_boy[510]; //匹配结果在match[]中  int n, m;  bool dfs(int x){ //找一个增广路径，即给女孩x找一个配对男孩  for(int i=1; i<=m; i++)  if(!reserve\_boy[i] && G[x][i]){  reserve\_boy[i] = 1; // 预定男孩i，准备分给女孩x  if(!match[i] || dfs(match[i])){  //有两种情况：(1)如果男孩i还没配对，就分给女孩x；  // (2)如果男孩i已经配对，尝试用dfs()更换原配女孩，以腾出位置给女孩x  match[i] = x; //配对成功；如果原来有配对，更换成功。现在男孩i属于女孩x  return true;  }  }  return false; //女孩x没有喜欢的男孩，或者更换不成功  }  int main(){  int e; scanf("%d%d%d",&n,&m,&e);  while(e--){int a,b; scanf("%d%d",&a,&b); G[a][b]=1;} //矩阵存图  int sum=0;  for(int i=1; i<=n; i++){ //为每个女孩找配对  memset(reserve\_boy,0,sizeof(reserve\_boy));  if(dfs(i)) sum++; //第i个女孩配对成功，这个配对后面可能更换，但是保证她能配对  }  printf("%d\n",sum);  return 0;  } |

10.13 费用流

|  |
| --- |
| 例10-21. Farm Tour poj 2135 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | //poj 2135 , 邻接表存图 + SPFA + 最大流  #include <stdio.h>  #include <algorithm>  #include <cstring>  #include <queue>  using namespace std;  const int INF = 0x3f3f3f3f;  const int N = 1010;  int dis[N], pre[N], preve[N]; //dis[i]记录起点到i的最短距离。pre和 preve见下面注释  int n, m;  struct edge{  int to, cost, capacity, rev; //rev用于记录前驱点  edge(int to\_,int cost\_,int c,int rev\_){  to=to\_; cost=cost\_; capacity=c; rev=rev\_;}  };  vector<edge> e[N]; //e[i]：存第i个结点连接的所有的边  void addedge(int from,int to,int cost,int capacity){//把1个有向边再分为2个  e[from].push\_back(edge(to, cost, capacity, e[to].size()));  e[to].push\_back(edge(from, -cost, 0, e[from].size()-1));  }  bool spfa(int s, int t, int cnt){ //套SPFA模板  bool inq[N];  memset(pre, -1, sizeof(pre));  for(int i = 1; i <= cnt; ++i) {dis[i]=INF; inq[i]=false; }  dis[s] = 0;  queue <int> Q;  Q.push(s);  inq[s] = true;  while(!Q.empty()){  int u = Q.front();  Q.pop();  inq[u] = false;  for(int i=0; i < e[u].size(); i++)  if(e[u][i].capacity > 0){  int v = e[u][i].to, cost = e[u][i].cost;  if(dis[u]+cost < dis[v]){  dis[v] = dis[u]+cost;  pre[v] = u; //v的前驱点是u  preve[v] = i; // u的第i个边连接v点  if(!inq[v]){  inq[v] = true;  Q.push(v);  }  }  }  }  return dis[t] != INF; //s到t的最短距离（或者最小费用）是dis[t]  }  int mincost(int s,int t,int cnt){ //基本上是套最大流模板  int cost = 0;  while(spfa(s,t,cnt)){  int v = t, flow = INF; //每次增加的流量  while(pre[v] != -1){ //回溯整个路径，计算路径的流  int u = pre[v], i = preve[v]; //u是v的前驱点，u的第i个边连接v  flow = min(flow, e[u][i].capacity); //所有边的最小容量就是这条路的流  v = u; //回溯，直到源点  }  v = t;  while(pre[v] != -1){ //更新残留网络  int u = pre[v], i = preve[v];  e[u][i].capacity -= flow; //正向减  e[v][e[u][i].rev].capacity += flow; //反向加，注意rev的作用  v = u; //回溯，直到源点  }  cost += dis[t]\*flow; //费用累加。如果程序需要输出最大流，在这里累加flow  }  return cost; //返回总费用  }  int main(){  while(~scanf("%d%d", &n, &m)){  for(int i=0;i<N; i++) e[i].clear(); //清空  for(int i=1;i<=m;i++){  int u,v,w; scanf("%d%d%d",&u,&v,&w);  addedge(u,v,w,1); addedge(v,u,w,1); //把1个无向边分为2个有向边  }  int s = n+1, t = n+2;  addedge(s,1,0,2); //添加源点  addedge(n,t,0,2); //添加汇点  printf("%d\n", mincost(s,t,n+2));  }  return 0;  } |