# acm-template

langman

January 31, 2018

## 1 头文件

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
#include<algorithm>
#include<queue>
#include<stack>
#include<vector>
#include<cmath>
#include<set>
#include<cstdlib>
#include<functional>
#include<climits>
#include<cctype>
#include<iomanip>
using namespace std;
typedef long long ll;
#define INF 0x3f3f3f3f
const int mod = 1e9+7 ;
#define clr(a,x) memset(a,x,sizeof(a))
#define cle(a,n) for(int i=1;i<=n;i++) a.clear();
const double eps = 1e-6;
int main()
{
  freopen("in.txt","r",stdin);
  freopen("out.txt","w",stdout);
  //舒服了
  return 0;
}
```

# 2 图论

## 2.1 并查集

```
int par[maxn];
int rank[maxn];

void init()
{
  for(int i = 1;i<=n;i++)
  {
    par[i] = i;
    rank[i] = 0;
  }
}

int find(int x)
{
  return x==par[x]?x:find(par[x]);
}
```

## 2.2 最短路

两种算法 但是要注意dijkstra无法处理负边的情况

### 2.2.1 dijkstra

需要注意的在于 可以更优化 我没写了 而且需要注意重边的情况

```
//dijkstra 算法
//无负权
int map[2005][2005];//记录路径 注意双向
int dp[2005];//单源最短路径记录
bool vis[2005];//记录是否用过
int N;//顶点数
void dijkstra(int s)
{
  clr(dp,INF);
  clr(vis,0);
  dp[s] = 0;
  while(true)
  {
    int v = -1;
    for(int u = 1;u<=N;u++)//从没用过的点中找一个距离最小的顶点
    {
      if(!vis[u] && (v==-1 || dp[u]<dp[v]) )
      v = u;
    }
    if(v == -1) break;
    vis[v] = true;
    for(int i = 1;i<=N;i++)
    {
      dp[i] = min(dp[i],dp[v]+map[v][i]);
    }
  }
}
```

### 2.2.2 spfa

需要注意的是怎么建边 双向边?

```
// Bellman Ford 存在最短路 这个比较难看感觉
//这个可以有负权
//但是  spfa 是在bellmen Ford 的基础上的加强
//这里用的是用前向星的方法去建图
//这里不用判重边的还是很舒服
int N,M;
```

```
int cnt;
struct edge{
int to,Next,w;
}E[maxn];
int pre[maxn],dp[maxn];//pre 路径结点 dp 最短路
bool vis[maxn];
int in[maxn];//这个的作用在于处理进去过多少次 就能看出是不是存在负环
void addedge(int x,int y,int z)
{
E[++cnt] .to = y;
E[cnt].Next = pre[x];
E[cnt].w = z;
pre[x] = cnt;
return;
}
bool spfa(int s)//这个算法还能判断是否存在负环
{
int i,t,temp;
queue<int>Q;
clr(vis,0);
clr(dp,INF);
clr(in,0);

Q.push(s);
vis[s] = true;
dp[s] = 0;

while(!Q.empty())
{
  t = Q.front();Q.pop();vis[t] = false;
  for(i = pre[t];i;i=E[i].Next)
  {
    temp = E[i].to;
    if(dp[temp] > dp[t]+E[i].w)
    {
      dp[temp] = dp[t]+E[i].w;
      if(!vis[temp])
      {
        Q.push(temp);
        vis[temp] = true;
        if(++in[temp]>N) return false; //负环判定关键
      }
    }
  }
```

```
}
return true;
}
```

### 2.2.3 Flody

这个就不写了,一个小dp

## 2.3 最小生成树

这是个什么玩意呢 图里面是吧,找到n-1条边使得生成一颗树,然后他的边权之和最小

```
//prime 算法
//还有一个不想去写了 没有这个必要
//注意重边
// 还有这个算法 在树生成不起来的情况下 需要特判一下
// dfs一遍就行 看是否全联通
int map[maxn][maxn];
int dp[maxn];
int vis[maxn];
int N;
int prime()
{
  clr(dp,INF);
  clr(vis,0);
  dp[1] = 0;
  int res = 0;
  while(true)
  {
    int v = -1;
    for(int u = 1;u<=N;u++)
    {
      if(!vis[u] && (v==-1 || dp[u]<dp[v])) v = u;
    }
    if(v == -1) break;
    vis[v] = 1;
    res += dp[v];
    for(int u = 1;u<=N;u++)
    {
      dp[u] = min(dp[u],map[v][u]);
    }
  }
  return res;
}
```

## 2.4 最大流

### 2.4.1 Dinic

板子先存着,坑定用的着

```
//最大流 dinic算法

const int MAXN = 1000;
struct edge{int to,cap,rev;};    //用边来存图
vector<edge>G[MAXN];             //图的链接表表示
int level[MAXN];                 //顶点到源点的距离标号
int iter[MAXN];                  //当前弧在其之前的边已经没有用了

void addedge(int from,int to,int cap)     //为图加一条从from到to的容量为cap的边
{
  G[from].push_back((edge){to,cap,(int)G[to].size()});
  G[to].push_back((edge){from,0,(int)G[from].size()-1});
}
void bfs(int s)               //bfs计算从源点出发的距离标号
{
  clr(level,-1);
  queue<int>que;
  level[s] = 0;
  que.push(s);
  while(!que.empty())
  {
    int v = que.front();
    que.pop();
    for(int i = 0;i<G[v].size();i++)
    {
      edge &e = G[v][i];
      if(e.cap>0 && level[e.to]<0)
      {
        level[e.to] = level[v]+1;
        que.push(e.to);
      }
    }
  }
}
int dfs(int v, int t,int f)              //通过dfs寻找增广路
{
  if(v == t) return f;
  for(int i = iter[v] ;i<G[v].size();i++)
  {
```

```
    edge &e = G[v][i];
    if(e.cap > 0 && level[v] < level[e.to])
    {
       int d = dfs(e.to,t,min(f,e.cap));
       if(d>0)
       {
          e.cap -=d;
          G[e.to][e.rev].cap += d;
          return d;
       }
    }
  }
  return 0;
}
int max_flow(int s,int t)    //从 s 到 t 的最大流
{
  int flow = 0;
  while(true)
  {
    bfs(s);
    if(level[t] < 0 ) return flow;
    clr(iter,0);
    int f;
    while((f = dfs(s,t,INF)) > 0 )
    flow += f;
  }
}
```