Bellman-Ford 算法 & SPFA 算法

一、Bellman-Ford 算法

图：暂略；

【C++版】

```
// 单源最短路径的 Bellman-ford 算法
// 执行 v-1 次，每次对每条边进行松弛操作
// 如有负权回路则输出"Error"
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;

const int MAXN = 100;
const int MAXE = MAXN * (MAXN - 1)/ 2;

struct edge_type{
   int  a, b, w;
};

edge_type  edges[MAXE+1];
int dist[MAXN+1];
int e, n, s;

void init()
{
   int i;

   freopen("g.in", "r", stdin);
   freopen("g.out", "w", stdout);

   cin >> n >> s;   // 读入顶点个数和源点编号
   e = 1;

   //读入每条边的信息，并存入边集数组
   while (cin>>edges[e].a >> edges[e].b >> edges[e].w )
      e++;
```

```
   //while(scanf("%d%d%d", &edges[e].a, &edges[e].b,
&edges[e].w)!=EOF){…}

   memset(dist, 0x7f, sizeof(dist));
   //所有点的最短距离的初始值为无穷大
   dist[s] = 0;
}

void relax(int u, int v, int w)   // 能否解决负权边问题?
{
   if ( dist[u] + w < dist[v] )
      dist[v] = dist[u] + w;
}

bool bellman_ford()
{
   int i, j;

   for(i=1; i<=n-1; i++)
      for(j=1; j<=e; j++)
         relax(edges[j].a, edges[j].b, edges[j].w);

   for (i=1; i<=e; i++)
      if (dist[edges[i].a] + edges[i].w < dist[edges[i].b])
         return false;

   return true;
}

void print()
{
   int i;

   for (i=1; i<=n; i++)
      cout <<i<<':'<<dist[i]<<endl;
}

int main() {
   init();

   if ( bellman_ford() )
      print();
   else
      cout << "Error!!" << endl;
```

```
   return 0;
}
```

【Pascal 版】

```
// 单源最短路径的 Bellman-ford 算法
// 执行 v-1 次，每次对每条边进行松弛操作
// 如有负权回路则输出"Error"
const
   maxn = 100;
   maxe = maxn * (maxn-1)/ 2;
type
   edge = record
            a,b,w: integer;
          end;
var
   edges: array[1..maxe]of edge;
   dist: array[1..maxn]of integer;
   pre: array[1..maxn]of integer;
   e, n, s: integer;

procedure init;
var i: integer;
begin
   assign(input, 'g.in');
   reset(input);
   readln(n, s);   // 读入顶点个数和源点编号

   e := 0;

   while not eof do
   begin
      inc(e);
      with edges[e] do readln(a,b,w);
        //读入每条边的信息，并存入边集数组
   end;

   fillchar(dis, sizeof(dist), $7f);
      //所有点的最短距离的初始值为无穷大
   dist[s] := 0;     pre[s] := s;
end;

procedure relax(u, v, w: integer);
// 能否解决负权边问题？
```

```
begin
    if (dist[u] + w < dist[v])
    begin
       dist[v] := dist[u] + w;
       pre[v] := u;
    end
end;

function bellman_ford: boolean;
var i, j: integer;
begin
   for i:=1 to n-1 do
      for j:=1 to e do
         with edges[j] do
            relax(a,b,w);

   for i:=1 to e do
      with edges[i] do
         if dist[a] + w < dist[b] then exit(false);

   exit(true)
end;

procedure print_path(i:integer);
begin
   if pre[i]<>s then   // 利用递归来倒序打印
      print_path(pre[i]);
   write('-->',i)
end;

procedure show;
var i: integer;
begin
   for i:=1 to n do
   begin
      write(i:3,':',dist[i]:3,':',s);
      print_path(i);
      writeln
   end;
end;

int main() {
   init();
```

```
  if ( bellman_ford() )
    show();
  else
    cout << "Error!!" << endl;
end.
```

二、SPFA 算法

2.1 算法简介

SPFA(Shortest Path Faster Algorithm)是 Bellman-Ford 算法的一种队列实现，减少了不必要的冗余计算。也有人说 SPFA 本来就是 Bellman-Ford 算法，现在广为流传的 Bellman-Ford 算法实际上是山寨版。

2.2 算法流程

SPFA 算法采用了一个队列来进行维护和实现：

（1）初始时，将源点加入队列。

（2）每次从队列中（队首）取出一个元素，并对所有与该队首顶点相邻的点进行松弛（“松驰”的含义与上面 Bellman-Ford 算法中描述的一样）。若某个相邻的点松弛成功，则将其入队（加入队尾）。

（3）一直对队列进行操作，直到队列为空时算法结束。

简单地说，这个算法就是利用队列优化过的 Bellman-Ford 算法，是利用了每个点的更新次数不会太多这一特点而发明的算法。

SPFA 可以在 O(kE)的时间复杂度内，求出源点到其他所有点的最短路径，并且可以处理负边。

SPFA 的实现甚至比 Dijkstra 或者 Bellman-Ford 还要简单：

设 Dist[i]代表源点 S 到任一顶点 i 的当前最短距离，Fa 代表 S 到 i 的当前最短路径中，i 点之前的一个点的编号。开始时，Dist 全部为+∞，只有 Dist[S]=0，Fa 全部为 0。

维护一个队列，里面存放所有需要进行迭代的点。初始时，队列中只有一个点 S。此外，再用一个布尔数组，记录每个点当前是否在队列中。

每次迭代时，取出队首的点 v，依次枚举从 v 出发的边 v->u：设该边的长度为 len，判断 Dist[v] + len 是否小于 Dist[u]，若小于则改进 Dist[u]（即松弛成功），将 Fa[u]记为 v（即 v 是 u 的父亲或前趋）。

进一步地，由于 S 到 u 的最短距离变小了，进而有可能通过 u 来改进其他点的最短距离，所以若 u 不在队列中，就将它放入队尾。

这样一直迭代下去，直到队列变空，也就是 S 到所有点的最短距离都确定下来时，算法结束。

若一个点的入队次数超过了 n(??)，则说明图中有负权的环（回路）{ Why? }。

SPFA 在形式上和宽度优先搜索非常类似，不同的是：

- 在宽搜过程中，一个点出了队列后就不可能再重新进入队列了；
- 相反地，在 SPFA 中，一个点可能在出队之后再次被放入队列。
- 也就是说，一个点改进过其他的点之后，过了一段时间，可能该点本身又被改进了，于是，它又被再次用来改进其他的点，就这样反复迭代下去。

设一个点用来作为迭代点对其他点进行改进的平均次数为 k，有办法证明对于通常的情况，k 在 2 左右。

2.3 SPFA 的实现代码

【C++版】

```
void spfa()
{
    memset(que, 0, sizeof(que));
    head = 0;  tail = 0;
    memset(inque ,false, sizeof(inque));
    // inque[i] 用于判断点 i 当前是否在队列中

    for (i=1; i<=n; i++)
        dist[i] = MAXINT;  // 初始化

    tail++;
    que[tail] = 1;
    inque[1] = true;
    dist[1] = 0;   // 这里把顶点 1 作为源点

    while (head != tail) {
        head++;
        x = que[head];
        inque[x] = false;    // 队首元素出队了
        for (i=1; i<=n; i++)    // 找相邻的、可改进的点
           if ( weight[x][i]>0 && dist[x]+weight[x][i]<dist[i]) {
              // 此条件合理否?
              dist[i] = dist[x] + weight[x][i];
              if (! inque[i]) {
                  tail++;
                  que[tail] = i;
                  inque[i] = true;
              }
          }
    }
}
```

【算法伪代码】{加入“某一个点入队次数达 n-1 即停止”}

```
procedure SPFA;
begin
   initialize-single-source(G, s);
   initialize-queue(Q);
   enqueue(Q, s);
```

```
  while not empty(Q) do
  begin
     u := dequeue(Q);
     for each v ∈ adj[u] do
     begin
        old := dist[v];
        relax(u, v);
        if ( old <> dist[v] ) and (not v in Q) then
           enqueue(Q,v);
     end;
  end;
end;
```

【或者】

```
procedure spfa;
begin
   fillchar(que, sizeof(que), 0);
   head := 0;  tail := 0;
   fillchar(inque ,sizeof(inque), false);
      // inque[i]用于判断点 i 当前是否在队列中
   for i:=1 to n do
      dist[i] := maxint;  // 初始化

   inc(tail);
   que[tail] := 1;
   inque[1] := true;
   dist[1] := 0;   // 这里把顶点 1 作为源点

   while head <> tail do
   begin
      head := (head mod n) + 1;
      x := que[head];
      inque[x] := false;    // 队首元素出队了
      for i := 1 to n do    // 找相邻的、可改进的点
        if (weight[x,i]>0) and    // 此条件合理否?
           (dist[x] + weight[x,i] < dist[i]) then
        begin
           dist[i] := dist[x] + weight[x,i];
           if not(inque[i]) then
           begin
              tail := (tail mod n)+1;
              que[tail] := i;
              inque[i] := true;
           end;
```

```
            end;
        end;
end;
```

三、SPFA 算法的应用实例

1、遇见（meet.cpp/c/pas）

【故事背景】

聪聪是个非常单纯的小朋友。有一天他在食堂吃饭的时候遇见了一位 MM。正当他想上前搭讪的时候，天空突然黑了，一个巨大的蚕宝宝从天而降，竟然把该 MM 吃了下去！蚕宝宝狂笑着对聪聪说：“这个 MM，你给她东西吃，她就吃，你不给她吃，她自己就死掉了是吧？我已经把她藏到了我体内的最深处，再过不久她就饿死了！”聪聪当然不会放过这个英雄救美的机会，于是他立刻施展法术，进入蚕宝宝体内去营救 MM！

【问题描述】

进入蚕宝宝体内后，聪聪发现蚕宝宝的身体是一个含有 N 个结点的**有向图**。聪聪所在的位置为 1 号结点，MM 所在的位置为 N 号结点。两个结点之间最多只有一条道路{什么意思？}。通过每条路都需要一定的时间，当聪聪第一次通过某条路时能获得这条道路上的金币。聪聪现在从 1 号结点出发，要到 MM 所在的 N 号结点。聪聪虽然救 MM 心切，但他也希望能多拿走一些金币，所以现在你的目标是：在保证所需时间最小的前提下，拿走最多的金币。

【输入格式】

输入文件 meet.in 包含 M+1 行。

第 1 行包含两个正整数 N、M，分别表示结点总数和道路的数量。

第 2 行到 M+1 行，每行包含四个正整数 A、B、C、D，表示有一条从 A 到 B 的道路，通过这条路所需花费的时间为 C，这条路上的金币数为 D。

【输出格式】

输出文件 meet.out 只包含 1 行，依次输出最优方案所需的时间和金币数，中间用一个空格分开。

【输入输出样例】

meet.in	meet.out
3 3 1 2 3 8 2 3 2 5 1 3 20 10	5 13

【输入输出样例解释】

最优路线为 1→2→3。

【数据规模和约定】

对于 20%的数据，N≤10，M≤8。

对于 40%的数据，N≤200，M≤2000。

对于 80%的数据，N≤2000，M≤30000。

对于全部的数据，N≤6000，M≤200000。

通过每条道路所需的时间和通过每条道路所获得的金币数均不超过 100。

保证不会出现无解的情况，即至少存在一条从结点 1 通向结点 N 的路径。

{HJS}

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;

struct node{
	int y;
	int t;
	int m;
};

bool in[6001];
queue<int> q;
vector<node> g[6001];
int d[6001],m[6001];

int main(){
	int n,e;

	cin>>n>>e;

	for (int i=1;i<=e;i++){
		int x,y,t,m;
		cin>>x>>y>>t>>m;
		node a;
		a.y=y;	a.t=t;	a.m=m;
		g[x].push_back(a);
	}

	for (int i=1;i<=n;i++){
		d[i]=2147483646;
	}

	q.push(1);
	in[1]=true;
	d[1]=0;

	while (!q.empty()){
		int x=q.front();
		q.pop();
		in[x]=false;
		for (int i=0;i<g[x].size();i++){
			int dist=d[x]+g[x][i].t;
			if (dist<d[g[x][i].y]){
```

```
                    d[g[x][i].y]=dist;
                    if (m[x]+g[x][i].m>m[g[x][i].y]) m[g[x][i].y]=m[x]+g[x][i].m;
                    if (!in[g[x][i].y]) {q.push(g[x][i].y);    in[g[x][i].y]=true;}
                }
            }
        }
        cout<<d[n]<<" "<<m[n]<<endl;
        return 0;
}

{TKN}
#include <iostream>
#include <cstdio>
#include <queue>
#include <cstring>
using namespace std;

int n,m,w[6001][6001],c[6001][6001],dist[6001],co[6001];
bool in[6001];
queue<int>q;

int main(){
        cin>>n>>m;
        int x,y,w1,c1;
        for(int i=1;i<=m;i++){
                cin>>x>>y>>w1>>c1;
                w[x][y]=w1;
                c[x][y]=c1;
        }
        memset(dist,0x7f,sizeof(dist));
        dist[1]=0;
        q.push(1);
        in[1]=true;
        int p=0;
        while(!q.empty()){
                p=q.front();
                in[p]=false;
                q.pop();
                for(int i=1;i<=n;i++){
                        if(dist[p]+w[p][i]<dist[i]){
                                dist[i]=dist[p]+w[p][i];
                                co[i]=co[p]+c[p][i];
                                if(!in[i]){
                                        q.push(i);
```

```
                        in[i]=true;
                    }
                }
                if(dist[p]+w[p][i]==dist[i]){
                    co[i]=max(co[p]+c[p][i],co[i]);
                }
            }
        }
        cout<<dist[n]<<' '<<co[n]<<endl;;
        return 0;
}
{YYP}
#include<iostream>
#include<cstdio>
#include<queue>
using namespace std;
const int maxint=2147483647;
int dist[100],ga[100][100],i,n,e,x,y,w;
bool inque[100];
queue<int> q;
int main(){
        cin>>n>>e;
        for(i=1;i<=e;i++){
            cin>>x>>y>>w;
            ga[x][y]=w;
        }
        for(i=2;i<=n;i++)
            dist[i]=maxint;
        q.push(1);
        inque[1]=true;
        while(!q.empty()){
            x=q.front();
            inque[x]=false;
            q.pop();
            for(i=1;i<=n;i++){
                if(ga[x][i]>0&&dist[x]+ga[x][i]<dist[i]){
                    dist[i]=dist[x]+ga[x][i];
                    if(!inque[i]){
                        inque[i]=true;
                        q.push(i);
                    }
                }
            }
        }
```

```
    for(i=2;i<=n;i++)
        cout<<dist[i]<<endl;
    return 0;
    return 0;
}
```

2、 最优贸易 (trade.cpp/c/pas) //洛谷 P1073

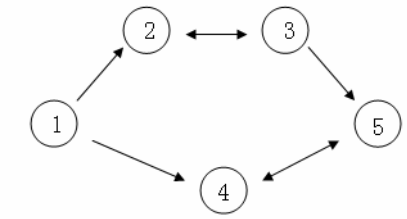
【问题描述】

C 国有 n 个大城市和 m 条道路，每条道路连接这 n 个城市中的某两个城市。任意两个 城市之间最多只有一条道路直接相连。这 m 条道路中有一部分为单向通行的道路，一部分 为双向通行的道路，双向通行的道路在统计条数时也计为 1 条。

C 国幅员辽阔,各地的资源分布情况各不相同,这就导致了同一种商品在不同城市的价格不一定相同。但是，同一种商品在同一个城市的买入价和卖出价始终是相同的。

商人阿龙来到 C 国旅游。当他得知同一种商品在不同城市的价格可能会不同这一信息之后，便决定在旅游的同时，利用商品在不同城市中的差价赚回一点旅费。设C国n个城市的标号从1~ n,阿龙决定从1号城市出发,并最终在n号城市结束自己的旅行。在旅游的过程中,任何城市可以重复经过多次,但不要求经过所有n个城市。阿龙通过这样的贸易方式赚取旅费:他会选择一个经过的城市买入他最喜欢的商品——水晶球,并在之后经过的另一个城市卖出这个水晶球，用赚取的差价当做旅费。由于阿龙主要是来 C 国旅游，他决定这个贸易只进行最多一次，当然，在赚不到差价的情况下他就无需进行贸易。

假设C国有5个大城市，城市的编号和道路连接情况如下图，单向箭头表示这条道路为单向通行，双向箭头表示这条道路为双向通行。



假设 1~n 号城市的水晶球价格分别为 4，3，5，6，1。

阿龙可以选择如下一条线路：1->2->3->5，并在 2 号城市以 3 的价格买入水晶球，在 3号城市以 5 的价格卖出水晶球，赚取的旅费数为 2。

阿龙也可以选择如下一条线路 1->4->5->4->5，并在第 1 次到达 5 号城市时以 1 的价格买入水晶球，在第 2 次到达 4 号城市时，以 6 的价格卖出水晶球，赚取的旅费数为 5。

现在给出 n 个城市的水晶球价格、m 条道路的信息(每条道路所连接的两个城市的编号以及该条道路的通行情况），请你告诉阿龙，他最多能赚取多少旅费。

【输入】(trade.in)

第1行：包含 2 个正整数 n 和 m，中间用一个空格隔开，分别表示城市的数目和道路的数目。

第2行:n 个正整数,每两个整数之间用一个空格隔开,按标号顺序分别表示这 n 个城市的商品价格。

接下来 m 行:每行有 3 个正整数x、y、z,每两个整数之间用一个空格隔开。如果z=1，

表示这条道路是城市x到城市y之间的单向道路；如果 z=2，表示这条道路为城市 x 和城市 y 之间的双向道路。

【输出】(trade.out)

1 行：包含 1 个整数,表示最多能赚取的旅费。如果没有进行贸易，则输出 0。

【输入输出样例】

trade.in	trade.out
5 5 4 3 5 6 1 1 2 1 1 4 1 2 3 2 3 5 1 4 5 2	5

【数据范围】

输入数据保证 1 号城市可以到达 n 号城市。

10%的数据： 1≤n≤6。

30%的数据： 1≤n≤100。

50%的数据： 不存在一条旅游路线，可以从一个城市出发，再回到这个城市。

100%的数据： 1≤n≤100000，1≤m≤500000，1≤x，y≤n，1≤z≤2，1≤各城市水晶球价格≤100。

【参考程序】

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
using namespace std;

const int maxn = 100010;
const int maxm = 1000010;

int first[maxn];
int next[maxm];
int w[maxm];
int rfirst[maxn];
int rnext[maxm];
int rw[maxm];
int l = 0;

int adde(int a, int b)
{
    ++L;
    w[L] = b;
    next[L] = first[a];
    first[a] = L;
```

```
    rw[L] = a;
    rnext[L] = rfirst[b];
    rfirst[b] = L;
}

int n, m;
int v[maxn], mv[maxn];
bool c1[maxn], c2[maxn], mark[maxn];
int q[maxn];
int lq;

void findv()
{
    memset(c1, 0, sizeof(c1));
    memset(c2, 0, sizeof(c2));

    lq = 1;           // 队尾指针
    q[0] = 1;         // 1#顶点入队
    int en = 0;       // 队首指针
    c1[1] = true;     // 打标记——已访问过，用于宽搜判重

    while (en < lq) {      // 队首指针赶上队尾指针，队列即为空
        int p = q[en++];   // 队首元素出队
        for (int e = first[p]; e; e = next[e])  // 宽搜扩展 p 的相邻点
        {
            if (!c1[w[e]])      // 只要该相邻点未被访问过
            {
                c1[w[e]] = true;
                q[lq++] = w[e];
            }
        }
    }

    lq = 1;      // 再做一次反向 BFS
    q[0] = n;
    en = 0;
    c2[n] = true;

    while (en < lq)
    {
        int p = q[en++];
        for (int e = rfirst[p]; e; e = rnext[e])
        {
            if (!c2[rw[e]])
```

```
            {
                c2[rw[e]] = true;
                q[lq++] = rw[e];
            }
        }
    }

    for (int i = 1; i <= n; ++i)
        c1[i] &= c2[i];      // 应该是“&&”，表示从起点、终点都可达？
}

int qv[maxn][2];

int cmp(const void *a, const void *b)
{
    int *x = (int*)a, *y = (int*)b;
    if (*x==*y)
        return x[1] - y[1];
    return *x - *y;
}

void expand(int x)
{
    int lq = 1;
    q[0] = x;
    int en = 0;
    while (en < lq)   // 找 x 出发可求得的最小价格
    {
        int p = q[en++];
        for (int e = first[p]; e; e = next[e])
            if (mv[w[e]] > mv[p])
            {
                mv[w[e]] = mv[p];
                q[lq++] = w[e];
            }
    }
}

int ans;

void getbest()
{
    for (int i = 1; i <= n; ++i)
        mv[i] = v[i];
```

```
    int lq = 0;
    for (int i = 1; i <= n; ++i)
        if (c1[i])
        {
            qv[lq][0] = v[i];
            qv[lq][1] = i;
            ++lq;
        }

    qsort(qv, lq, sizeof(qv[0]), cmp);

    for (int i = 0; i < lq; ++i)
    {
        expand(qv[i][1]);
    }

    ans = 0;
    for (int i = 1; i <= n; ++i)
        if (c1[i])    // 若顶点 i 可最终到达终点
            if (v[i] - mv[i] > ans)
                ans = v[i] - mv[i];
}

int main()
{
    freopen("trade.in", "rt", stdin);
    freopen("trade.out", "wt", stdout);

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

    for (int i = 1; i <= n; ++i)   // 读入各城市的价格
        scanf("%d", &v[i]);

    memset(first, 0, sizeof(first));
    memset(rfirst, 0, sizeof(rfirst));

    for (int i = 0; i < m; ++i)     // 读入所有的边
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        adde(a, b);
        if (c==2)           // 如果是双向边
            adde(b, a);
    }
```

```
    findv();

    getbest();

    cout << ans << endl;

    return 0;
}
```

3、 **Word Morph** (wmorph.cpp/c/pas, Jacob Steinhardt, USACO 2010 Jan. Bronze)

Description

Farmer John is playing a word game against his cows. It starts like this:

* Farmer John chooses a word, such as 'cat'
* The cows then choose their own word, perhaps 'dog'

Farmer John then must morph his word to the cows' word by performing a number of steps, each one of which changes one single letter at a time to make a new, valid word.

Your program should read a dictionary of valid words from file dict.txt. The file has no more than 25,000 words, each with length in the range 1..20. These 'words' contain only letters in the range 'a'..'z'. The same dictionary is used for all tests.

For this example, Farmer John could make the following sequence of four words:

'cat' -> 'cot' -> 'cog' -> 'dog'

to morph words from the first word 'cat' to the final word 'dog' in just three moves. The cows will never give Farmer John an impossible task. Farmer John must get from his word to the cows' word in as few moves as possible.

You will be given a starting word and an ending word. Determine and output a number which is the least number of legal letter changes required to morph the starting word to the ending word.

PROBLEM NAME: wmorph

INPUT FORMAT:

* Line 1: A single string that is the starting word

* Line 2: A single string that is the end word

SAMPLE INPUT (file wmorph.in):

```
cat
dog
```

OUTPUT FORMAT:

* Line 1: A single integer that is the number of times a letter must be changed to transform the starting word into the ending word.

SAMPLE OUTPUT (file wmorph.out):

```
3
```

Analysis

简单的 BFS 显然会出问题。// (26*k)^k * logN
图论建模：建图+经典算法。
问题：图如何建起来（或者，各顶点如何产生出来？）

This task asks us to use a dictionary to 'morph' words, one letter at a time, from one valid dictionary word to another valid dictionary word (e.g., 'cat' -> 'cot' -> 'cog' -> 'dog').

Clearly, we need to choose some searching algorithm. Its goal will be to search from 'cat' to get all the possible one-letter changes (in this dictionary: bat, cat, cit, cot, cut, eat, fat, hat, mat, oat, pat, rat, sat, tat, vat). Those words must somehow be used to seatch for the next single letter change until the chain cat, cot, cog, dog (or shorter!) is found.

Noting that 'cat' and 'dog' share no common letters, it should be fairly clear that a depth-first search will be a catastrophe. Wandering around in the space of (as it turns out) almost 600 three letter words could take a very very long time, especially when searching for a 'shortest' path. In fact, that 'shortest' part is the cue that we should be using a *breadth-first* search.

Breadth-first searches are characterized by the use of a queue and a simple algorithm:

```
     Enqueue first element (or first set of elements) to be
searched
     while (no solution found) {
         de-queue a term from the front of the queue
         foreach possible next term
              solution? --> DONE
              enqueue that term at the end of the queue
             [optimization: do not enqueue terms already
explored]
     }
```

The solution below stores queue elements in a struct q_f (the _f reminds us that this is a 'form' -- a datatype) with a word (and plenty of room for its newline and '\0' terminating character) and the length of **the chain that led to that word**, not the number of characters in the word. This data structure should make the code that implements enqueue() and dequeue() clear enough.

The main logic of the program follows these general steps:

- Get the start and end words.
- Determine their length.
- Read the dictionary, keeping only words of that length (without this optimization, longer cases will time out). Note that each word has its terminating newline, if present, removed. **Programming Note:** One should submit the program for debugging at this point to make sure that the dictionary is being read correctly and to peek at some of the entries to make sure one's assumptions about the dictionary are correct. For instance, the dictionary is NOT sorted into any discernible order.
- Mark the start word in the dictionary as having been 'seen'.
- Enqueue the single start word.
- Implement the algorithm fragment shown above with dequeueing, generating of new words, and enqueueing them. **Optimization:** Marking the queued words as 'seen' eliminates the possibility of looking at a word more than once, which is almost a requirement here.

【参考程序】

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
using namespace std;

char dict[25000][20+2];
char wordseen[25000];
int ndict = 0;
char startword[20+2];
char endword[20+2];

const int NQUEUE = 10000;

struct q_f {
    char q_word[20+2];
    int q_len;
} q[NQUEUE];

int qin, qout;

void enqueue(word, len)
char *word;
{
    strcpy (q[qin].q_word, (const char *)word);
    q[qin].q_len = len;
    if (++qin >= NQUEUE) qin = 0;
    if (qin == qout) {
        fprintf (stderr, "queue full\n");
        exit (1);
    }
}

struct q_f  *dequeue()
{
    int oldqout = qout;
    if (qout == qin)          /* empty queue? */
        return (struct q_f *)0;
    if (++qout >= NQUEUE) qout = 0;
    return &q[oldqout];
}

worddiff (w1, w2)
char *w1, *w2;
{
```

```
    int ndiffs = 0;
    for ( ; *w1 && *w2; w1++, w2++) {
        if (*w1 == *w2) continue;
        ndiffs++;
        if (ndiffs > 1) return ndiffs;
    }
    if (*w1 || *w2) return -1;         /* differing lengths */
    return ndiffs;
}

chomp (w)
char *w;
{
    for (; *w; w++)
        if (*w == '\n') {
            *w = '\0';
            break;
        }
}

int main() {
    FILE *fdict = fopen ("dict.txt", "r");
    FILE *fin = fopen ("wmorph.in", "r");
    FILE *fout = fopen ("wmorph.out", "w");
    int i, wordlen;

    fscanf (fin, "%s", startword);
    fscanf (fin, "%s", endword);
    for (wordlen = 0; startword[wordlen]; wordlen++)
        ;

    // get dictionary of words of length wordlen:
    char *p;
    while ( fgets (dict[ndict], 20, fdict))  {
        chomp (dict[ndict]);
        if (strlen(dict[ndict]) != wordlen) continue;
        ndict++;
    }

    for (i = 0; i < ndict; i++) {
        if (strcmp (startword, dict[i]) == 0) {
            wordseen[i] = 1;
            break;
        }
```

```
    }

    enqueue (startword, 0);
    struct q_f *curword;

    while (curword = dequeue()) {
        if (strcmp (curword->q_word, endword) == 0) {
            fprintf (fout, "%d\n", curword->q_len);
            exit (0);
        }
        // find potential new words to morph to
        for (i = 0; i < ndict; i++) {
            if (wordseen[i]) continue;
            if (worddiff(curword->q_word, dict[i])==1) {
                wordseen[i] = 1;
                enqueue (dict[i], curword->q_len + 1);
            }
        }
    }

    fprintf (stderr, "failed to morph\n");
    return 0;
}
```

```
{
ID:maziyux1
LANG:PASCAL
PROG:wmorph
}

program wmorph(input,output);
type
  pointer = ^node;
  node = record
          num: integer;
          next: pointer;
        end;
var
  fin, fout, fdict: text;
  st: array[0..25001] of string;
  dis: array[0..25001] of integer;
  num, i, j: integer;
  ge: array[0..25001] of pointer;
```

```
function dif (x, y: integer): boolean;
var sum, i: integer;
begin
   sum := 0;

   for i:=1 to length( st[x] ) do
      if st[x,i] <> st[y,i] then
      begin
          inc(sum);
          if sum > 1 then exit(false);
      end;

   if sum = 1 then exit(True);
   exit(false);
end;

procedure insert (i, j: integer);
var p: pointer;
begin
   new(p);
   p^.num := j;
   p^.next := nil;
   if ge[i] = nil then
      ge[i] := p
   else
      begin
         p^.next := ge[i]^.next;
         ge[i]^.next := p;
      end;
end;

procedure init;
var str: string;
begin
   assign(fin, 'wmorph.in');
   reset( fin );
   assign(fdict, 'dict.txt');
   reset( fdict );
   assign(fout, 'wmorph.out');
   rewrite( fout );

   readln(fin, st[0]);   { 起始单词 }
   readln(fin, st[1]);   { 目标单词 }
```

```
  num:=1;
  while not( eof(fdict) ) do
  begin
    readln(fdict, str);
    if length(str) = length(st[1]) then
    begin
      inc(num);
      st[num] := str;    { 与起始单词长度一样的单词 }
    end;
  end;

  for i:=0 to num do
    ge[i]:=nil;

  for i:=0 to num-1 do        { 在仅有一位字母不同的单词间建立一条边 }
    for j:=i+1 to num do
      if dif(i,j) then
      begin
        insert(i,j);
        insert(j,i);
      end;

  dis[0]:=0;
  for i := 1 to num do   { 准备好求以起始单词为起点的单源最短路径 }
    dis[i]:=maxint;
end;

procedure clearall;
begin
  writeln(fout, dis[1]);   { 打印起始单词到终止单词之间的最少变化次数 }
  close(fin);
  close(fout);
end;

procedure work;   { SPFA 求起始单词至其他单词之间的最短路径,即最少变化次数 }
var
  que: array[1..50000] of integer;
  head, tail: integer;
  p: pointer;
begin
  que[1] := 0;  { 第 0 号单词(即起始单词)入队 }
  head := 1;
  tail := 1;
```

```
  while head <= tail do
  begin
    p := ge[ que[head] ];
    while p <> nil do
    begin
      if dis[que[head]] + 1 < dis[p^.num]  then
      begin
        dis[p^.num] := dis[que[head]] + 1;
        inc(tail);
    { 将已成功松弛的结点入队——还应加一个判断：看待入队的结点是否已在队中 }
        que[tail] := p^.num;
      end;
      p := p^.next;
    end;
    inc(head);   { 至此，队首结点出队 }
  end;
end;

begin  { main }
  init;
  work;
  clearall;
end.
```

4、**逃离遗迹**(escape.cpp/c/pas)

【问题描述】

根据外星人的回信得知，在遗迹中分布着三样道具。当三样道具都拿走后，遗迹很快就会自动毁灭，所以必须在最短的时间内离开。遗迹可以看作是由 N 个房间（编号 1..N）和 N-1 条长度不等的通道组成。任意两个房间之间有且只有一条路可以相互到达。

现在，我们的队员已经在编号为 A、B、C 的房间内拿到了道具，并且准备撤退。由于只有一架直升机，所以只能在一个房间上停留。现在，请你决定将直升机停在哪一个房间之上，才能够使三人到达该房间的距离之和最短。

【输入格式】(escape.in)

第 1 行：4 个整数 N、A、B、C。

第 2..N 行：每行 3 个整数 u、v、w，表示存在连接房间 u、v 的通道，长度为 w。

【输出格式】(escape.out)

第 1 行：一个整数，表示汇合房间的编号。若存在多个解，输出字典序最小的。

第 2 行：一个整数，表示三人到达该房间的距离之和。

【输入样例】

```
5 3 1 4
3 5 5
4 3 9
```

4 1 7

1 2 1

【输出样例】

4

16

【数据规模】

50%的数据： 1<=N<=1000；

100%的数据： 1<=N<=20,000。

1<=A, B, C, u, v <=N 且 A、B、C 不相等；u、v 不相等。

1<=w<=1000。

5、 **文化之旅** (culture.cpp/c/pas, NOIP2012 普及组复赛第四题)

【问题描述】

有一位使者要游历各国，他每到一个国家，都能学到一种文化，但他不愿意学习任何一种文化超过一次（即如果他学习了某种文化，则他就不能到达其他有这种文化的国家）。不同的国家可能有相同的文化。不同文化的国家对其他文化的看法不同，有些文化会排斥外来文化（即如果他学习了某种文化，则他不能到达排斥这种文化的其他国家）。

现给定各个国家间的地理关系、各个国家的文化、每种文化对其他文化的看法，以及这位使者游历的起点和终点（在起点和终点也会学习当地的文化），国家间的道路距离，试求从起点到终点最少需要走多少路。

【输入】

输入文件 culture.in。

第一行为五个整数 N，K，M，S，T，每两个整数之间用一个空格隔开，依次代表国家个数（国家编号为 1 到 N），文化种数（文化编号为 1 到 K），道路的条数，以及起点和终点的编号（保证 S 不等于 T）；

第二行为 N 个整数，每两个整数之间用一个空格隔开，其中第 i 个数 C_i，表示国家 i 的文化为 C_i。

接下来的 K 行，每行 K 个整数，每两个整数之间用一个空格隔开，记第 i 行的第 j 个数为 a_{ij}，a_{ij} = 1 表示文化 i 排斥外来文化 j（i 等于 j 时表示排斥相同文化的外来人），a_{ij}=0 表示不排斥（注意 i 排斥 j 并不保证 j 一定也排斥 i）。

接下来的 M 行，每行三个整数 u，v，d，每两个整数之间用一个空格隔开，表示国家 u 与国家 v 之间有一条距离为 d 的可双向通行的道路（保证 u 不等于 v，两个国家之间可能有多条道路）。

【输出】

输出文件名为 culture.out。

输出只有一行，一个整数，表示使者从起点国家到达终点国家最少需要走的距离数（如果无解则输出-1）。

【输入输出样例 1】

culture.in	culture.out
2 2 1 1 2 1 2 0 1	-1

1 0 1 2 10	

【输入输出样例说明】

由于到国家 2 必须要经过国家 1，而国家 2 的文明却排斥国家 1 的文明，所以不可能到达国家 2。

【输入输出样例 2】

culture.in	culture.out
2 2 1 1 2 1 2 0 1 0 0 1 2 10	10

【输入输出样例说明】

路线为 1 -> 2。

【数据范围】

对于 20%的数据： 有 2<=N<=8，K<=5；

对于 30%的数据： 有 2<=N<=10，K<=5；

对于 50%的数据： 有 2<=N<=20，K<=8；

对于 70%的数据： 有 2<=N<=100，K<=10；

对于 100%的数据： 有 2<=N<=100，1<=K<=100，1<=M<=N^2，1<=k_i<=K，1<=u,v<=N，1<=d<=1000，S<>T，1<=S,T<=N。

【参考程序】{ZXY}{周星宇}

```
var
   n,k,m,s,t,i,j,u,v,d,x:longint;
   ch,dis,pa,c:array[1..100] of longint;
   ch1,ch2,pai:array[1..100,1..100] of longint;
   bo:array[1..100,1..100] of boolean;
   ins,boo:array[1..100] of boolean;

function cmp(a,b:longint):boolean;
  var i:longint;
  begin
    for i:=1 to pa[c[b]] do
      if bo[a,pai[c[b],i]]=true then
        exit(false);
    exit(true);
end;

procedure spfa;
  var head,tail,j:longint;
      de:array[1..100] of longint;
  begin
    head:=0;
```

```
    tail:=1;
    de[tail]:=s;
    while head<>tail do
      begin
        head:=(head mod n)+1;
        ins[de[head]]:=false;
        for i:=1 to ch[de[head]] do
          if cmp(de[head],ch1[de[head],i]) then
            if boo[c[ch1[de[head],i]]] then
              if  dis[de[head]]+ch2[de[head],i]<dis[ch1[de[head],i]]
then
                begin

dis[ch1[de[head],i]]:=dis[de[head]]+ch2[de[head],i];
                for j:=1 to k do
                  bo[ch1[de[head],i],j]:=bo[de[head],j];
                bo[ch1[de[head],i],c[ch1[de[head],i]]]:=true;
                if ins[ch1[de[head],i]]=false then
                  begin
                    ins[ch1[de[head],i]]:=true;
                    tail:=(tail mod n)+1;
                    de[tail]:=ch1[de[head],i];
                  end;
        end;
    end;
end;

begin
  assign(input,'culture.in');
  assign(output,'culture.out');
  reset(input);
  rewrite(output);
  readln(n,k,m,s,t);
  for i:=1 to n do read(c[i]);
  fillchar(pa,sizeof(pa),0);

  for i:=1 to k do
    for j:=1 to k do
      begin
        read(x);
        if x=1 then
          begin
            pa[i]:=pa[i]+1;
            pai[i,pa[i]]:=j;
```

```
      end;
  end;

  fillchar(ch,sizeof(ch),0);
  for i:=1 to m do
    begin
      readln(u,v,d);
      ch[u]:=ch[u]+1;
      ch1[u,ch[u]]:=v;
      ch2[u,ch[u]]:=d;
      ch[v]:=ch[v]+1;
      ch1[v,ch[v]]:=u;
      ch2[v,ch[v]]:=d;
  end;
  fillchar(boo,sizeof(boo),true);
  for i:=1 to pa[c[t]] do
    boo[pai[c[t],i]]:=false;
  for i:=1 to n do dis[i]:=2147483640;
  fillchar(bo,sizeof(bo),false);
  bo[s,c[s]]:=true;

  dis[s]:=0;

  fillchar(ins,sizeof(ins),false);
  ins[s]:=true;

  spfa;

  if dis[t]=2147483640 then
    writeln(-1)
  else
    writeln(dis[t]);

  close(input);
  close(output);
end.
```

【参考程序】{YD,C++}

```
#include <cstdio>
#include <iostream>
using namespace std;

int  n, i, j, k, l, m, s, be, en, j1, j2, u, v, w;
int  ga[200+1][200+1], v1[200+1][200+1];
bool  v[200+1];
```

```
int dist[200+1], b[200+1];

int main()
{
   freopen("culture.in", "r", stdin);
   freopen("culture.out", "w", stdout);
   cin >> n >> k >> m >> be >> en;

   for (i=1; i<=n; i++)    cin >> b[i];

   for (i=1; i<=k; i++)
     for (j=1; j<=k; j++)
        cin >> v1[i][j];

   for (i=1; i<=n; i++)
     for (j=1; j<=n; j++)
        ga[i][j] = 10000000;

   for (i=1; i<=m; i++) {
      cin >> u >> v >> w;    //u, v, w
      a[u][v] = w;
      a[v][u] = w;
      if (v1[b[u]][b[v]]==1)   ga[v][u] = 10000000;
      if (v1[b[v]][b[u]]==1)   ga[u][v] = 10000000;
   }

   for (i=1; i<=n; i++)
     for (j=1; j<=n; j++)
       if (b[i]==b[j]) {
          ga[i][j] = 10000000;
          ga[j][i] = 10000000;
       }

   for (k=1; k<=n; k++)
     for (i=1; i<=n; i++)
       for (j=1; j<=n; j++)
          if (i!=j && i!=k && j!=k)
             if (ga[i][k]+ga[k][j]<ga[i][j])
                ga[i][j] = ga[i][k] + ga[k][j];

   v[1] = true;
   dist[be] = 0;

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

```
      dist[i] = 10000000;
      v[i] = false;
    }

    if (ga[be][en]>=10000000 || b[be]==b[en])
      cout <<"-1" << endl;
    else
      cout << ga[be][en] << endl;

    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

【参考程序】{YD, Pascal}

```
var
  n,i,j,k,l,m,s,be,en,j1,j2:longint;
  a,v1:array[1..200,1..200] of longint;
  v:array[1..200] of boolean;
  dist,b:array[1..200] of longint;
begin
  assign(input,'culture.in');reset(input);
  assign(output,'culture.out');rewrite(output);
  readln(n,m,k,be,en);
  for i:=1 to n do
    read(b[i]);
  for i:=1 to m do
    for j:=1 to m do
      read(v1[i,j]);
  for i:=1 to n do
    for j:=1 to n do
      a[i,j]:=10000000;
  for i:=1 to k do
    begin
      readln(j,j1,j2);
      a[j,j1]:=j2;
      a[j1,j]:=j2;
      if v1[b[j],b[j1]]=1 then a[j1,j]:=10000000;
      if v1[b[j1],b[j]]=1 then a[j,j1]:=10000000;
    end;
  for i:=1 to n do
    for j:=1 to n do
      if b[i]=b[j] then begin a[i,j]:=10000000;a[j,i]:=10000000;end;
  for i:=1 to n do
    for j:=1 to n do
```

```
      for k:=1 to n do
        if (i<>j) and (i<>k) and (j<>k) then
          if (a[i,k]+a[k,j]<a[i,j]) then a[i,j]:=a[i,k]+a[k,j];
  v[1]:=true;dist[be]:=0;
  for i:=1 to n do
    begin
      dist[i]:=10000000;
      v[i]:=false;
    end;
  if (a[be,en]>=10000000) or (b[be]=b[en]) then writeln('-1') else
writeln(a[be,en]);
  close(input);
  close(output);
end.
```