



南京工業大學  
NANJING TECH  
UNIVERSITY

## ICPC Template Manual



作者: 贺梦杰

August 20, 2019

# Contents

<b>1</b>	<b>基础</b>	<b>3</b>
1.1	测试	4
<b>2</b>	<b>搜索</b>	<b>5</b>
<b>3</b>	<b>动态规划</b>	<b>6</b>
<b>4</b>	<b>字符串</b>	<b>7</b>
4.1	字符串 Hash	8
4.1.1	应用：最长回文子串	8
4.1.2	应用：后缀数组	9
4.1.3	应用：二维 Hash	10
4.1.4	应用：一类同构判定的问题	10
4.2	后缀自动机	11
4.3	KMP	12
4.4	最小表示法	12
<b>5</b>	<b>数据结构</b>	<b>13</b>
<b>6</b>	<b>图论</b>	<b>14</b>
6.1	最短路	15
6.1.1	单源最短路径	15
6.1.1.1	Dijkstra	15
6.1.1.2	Bellman-Ford 和 SPFA	15
6.1.2	任意两点间最短路径	16
6.1.2.1	Floyd	16
6.2	最小生成树	17
6.2.1	Kruskal	17
6.2.2	Prim	17
6.3	树的直径	21
6.3.1	树形 DP 求树的直径	21
6.3.2	两次 BFS/DFS 求树的直径	21
6.4	最近公共祖先 (LCA)	21
6.4.1	树上倍增	21
6.4.2	Tarjan	22
6.5	树上差分	22
6.6	LCA 的综合应用	23
6.7	基环树	26
6.8	负环与差分约束	27
6.8.1	负环	27
6.8.2	差分约束系统	27
6.9	Tarjan 算法与无向图连通性	28
6.9.1	无向图的割点与桥	28
6.9.1.1	割边判定法则	28
6.9.1.2	割点判定法则	28
6.9.2	无向图的双连通分量	28
6.9.2.1	边双连通分量 e-DCC 与其缩点	28
6.9.2.2	点双连通分量 v-DCC 与其缩点	29
6.9.3	欧拉路问题	30
6.10	Tarjan 算法与有向图连通性	31

6.10.1	强连通分量 (SCC) 判定法则	31
6.10.2	SCC $\rightarrow$ DAG	31
6.10.3	有向图的必经点与必经边	31
6.10.4	2-SAT 问题	32
6.11	二分图的匹配	33
6.11.1	二分图判定	33
6.11.2	二分图最大匹配	33
6.11.3	二分图带权匹配	33
6.12	二分图的覆盖与独立集	35
6.12.1	二分图最小点覆盖	35
6.12.1.1	König's theorem	35
6.12.2	二分图最大独立集	35
6.12.3	有向无环图的最小路径点覆盖	35
6.13	网络流初步	36
6.13.1	最大流	36
6.13.1.1	Edmonds Karp 增广路	36
6.13.1.2	Dinic	36
6.13.1.3	二分图最大匹配的必须边与可行边	37
6.13.2	最小割	37
6.13.2.1	最大流最小割定理	37
6.13.3	费用流	37
6.13.3.1	Edmonds Karp 增广路	37

# Chapter 1

## 基础

## 1.1 测试

## Chapter 2

# 搜索

## Chapter 3

# 动态规划

## Chapter 4

# 字符串



## 4.1 字符串 Hash

```

1  #define ull unsigned long long
2  #define P 131
3  ull f[N], p[N];
4  void Init()
5  {
6      p[0] = 1, f[0] = 0;
7      for (int i = 1; i <= n; i++)
8      {
9          p[i] = p[i - 1] * P;
10         f[i] = f[i - 1] * P + str[i];
11     }
12 }
13 ull Hash(int left, int right)
14 {
15     return f[right] - f[left - 1] * p[right - left + 1];
16 }

```

### 4.1.1 应用：最长回文子串

枚举回文子串的中心位置，求解从中心位置出发向左右两侧最长可扩展出多长的回文串，分奇、偶回文子串，二分长度。

```

1  #define ull unsigned long long
2  #define P 131
3  const int N = 1e6 + 50;
4  ull pre[N], suf[N], p[N];
5  string str;
6  int n;
7  void Init()
8  {
9      pre[0] = suf[0] = 0;
10     for (int i = 1; i <= n; i++)
11     {
12         pre[i] = pre[i - 1] * P + str[i - 1];
13         suf[i] = suf[i - 1] * P + str[n - i];
14     }
15 }
16 ull Hash(int left, int right)
17 {
18     return pre[right] - pre[left - 1] * p[right - left + 1];
19 }
20 ull HashReverse(int left, int right)
21 {
22     //WA 这里注意推导
23     return suf[n - left + 1] - suf[n - right] * p[right - left + 1];
24 }
25 #define ODD 1
26 #define EVEN 2
27 bool Judge(int pos, int len, int flag)
28 {
29     if (flag == ODD)
30     {
31         if (pos - len < 1 || pos > n || pos < 1 || pos + len > n)
32             return false;
33         return Hash(pos - len, pos) == HashReverse(pos, pos + len);
34     }
35     else
36     {
37         if (pos - len < 1 || pos - 1 > n || pos < 1 || pos + len - 1 > n)
38             return false;
39         return Hash(pos - len, pos - 1) == HashReverse(pos, pos + len - 1);
40     }
41 }

```

```

42 int Solve(int pos)
43 {
44     int ans = 1;
45     //奇数
46     int left = 0;
47     int right = N;
48     int mid;
49     while (left < right)
50     {
51         mid = (left + right + 1) >> 1;
52         if (Judge(pos, mid, ODD))
53             left = mid;
54         else
55             right = mid - 1;
56     }
57     ans = max(ans, 2 * left + 1);
58     //偶数
59     left = 0;
60     right = N;
61     while (left < right)
62     {
63         mid = (left + right + 1) >> 1;
64         if (Judge(pos, mid, EVEN))
65             left = mid;
66         else
67             right = mid - 1;
68     }
69     ans = max(ans, 2 * left);
70     return ans;
71 }

```

#### 4.1.2 应用：后缀数组

```

1  #define ull unsigned long long
2  #define P 131
3  const int N = 3e5 + 50;
4  ull f[N], p[N];
5  char str[N];
6  int SA[N], n, Height[N];
7  void Init()
8  {
9      p[0] = 1, f[0] = 0;
10     for (int i = 1; i <= n; i++)
11     {
12         p[i] = p[i - 1] * P;
13         f[i] = f[i - 1] * P + str[i];
14     }
15 }
16 ull Hash(int left, int right)
17 {
18     return f[right] - f[left - 1] * p[right - left + 1];
19 }
20 // k:[0,n) 表示后缀S(k,n-1)
21 // 最长公共前缀
22 int LCP(int a, int b)
23 {
24     int left = 0;
25     int right = N;
26     int mid;
27     while (left < right)
28     {
29         mid = (left + right + 1) >> 1;
30         if (a + mid - 1 <= n && b + mid - 1 <= n && Hash(a, a + mid - 1) == Hash(b, b + mid - 1))
31             left = mid;
32         else

```

```

33         right = mid - 1;
34     }
35     return left;
36 }
37 bool cmp(int a, int b)
38 {
39     int len = LCP(a, b);
40     return str[a + len] < str[b + len];
41 }
42 void calc_height()
43 {
44     Height[1] = 0;
45     for (int i = 2; i <= n; i++)
46         Height[i] = LCP(SA[i], SA[i - 1]);
47 }
48 int main()
49 {
50     scanf("%s", str + 1);
51     n = strlen(str + 1);
52     for (int i = 1; i <= n; i++)
53         SA[i] = i;
54     Init();
55     sort(SA + 1, SA + n + 1, cmp);
56     calc_height();
57 }

```

### 4.1.3 应用：二维 Hash

给定一个  $M$  行  $N$  列的 01 矩阵（只包含数字 0 或 1 的矩阵），再执行  $Q$  次询问，每次询问给出一个  $A$  行  $B$  列的 01 矩阵，求该矩阵是否在原矩阵中出现过。

做法：选取两个不同的  $P$  值分别对行列进行 Hash 处理，应用二维前缀和求取矩阵 Hash 值。

```

1  #define P 131
2  #define Q 13331
3  #define ull unsigned long long
4  void Init()
5  {
6      char ch;
7      for (int i = 1; i <= m; i++)
8          for (int j = 1; j <= n; j++)
9              cin >> ch, Hash[i][j] = Hash[i][j - 1] * P + ch;
10     for (int i = 1; i <= m; i++)
11         for (int j = 1; j <= n; j++)
12             Hash[i][j] += Hash[i - 1][j] * Q;
13 }
14 ull temp = Hash[i][j] - Hash[i - a][j] * q[a] - Hash[i][j - b] * p[b] + Hash[i - a][j - b] * q[a]
    * p[b];

```

### 4.1.4 应用：一类同构判定的问题

参考：杨弋《Hash 在信息学竞赛中的一类应用》

## 4.2 后缀自动机

```

1  const int MAXLEN = 1e6 + 50;
2  namespace SAM
3  {
4      struct state
5      {
6          int len, link, next[26];
7      } st[MAXLEN * 2];
8      int sz, last;
9      void SAM_Init()
10     {
11         st[0].len = 0, st[0].link = -1;
12         sz++, last = 0;
13     }
14     void SAM_Extend(int c)
15     {
16         int cur = sz++;
17         st[cur].len = st[last].len + 1;
18         int p = last;
19         while (p != -1 && !st[p].next[c])
20         {
21             st[p].next[c] = cur;
22             p = st[p].link;
23         }
24         if (p == -1)
25             st[cur].link = 0;
26         else
27         {
28             int q = st[p].next[c];
29             if (st[q].len == st[p].len + 1)
30                 st[cur].link = q;
31             else
32             {
33                 int clone = sz++;
34                 st[clone].len = st[p].len + 1;
35                 memcpy(st[clone].next, st[q].next, sizeof(st[clone].next));
36                 st[clone].link = st[q].link;
37                 while (p != -1 && st[p].next[c] == q)
38                 {
39                     st[p].next[c] = clone;
40                     p = st[p].link;
41                 }
42                 st[q].link = st[cur].link = clone;
43             }
44         }
45         last = cur;
46     }
47     int id[MAXLEN * 2], c[MAXLEN * 2];
48     void Topo()
49     {
50         //计数排序
51         for (int i = 1; i < sz; i++)
52             c[st[i].len]++;
53         for (int i = 1; i < MAXLEN; i++)
54             c[i] += c[i - 1];
55         for (int i = 1; i < sz; i++)
56             id[c[st[i].len]--] = i;
57     }
58 } // namespace SAM

```

## 4.3 KMP

```

1  // KMP
2  next[1] = 0;
3  for (int i = 2, j = 0; i <= n; i++) {
4      while (j > 0 && a[i] != a[j+1]) j = next[j];
5      if (a[i] == a[j+1]) j++;
6      next[i] = j;
7  }
8
9  for (int i = 1, j = 0; i <= m; i++) {
10     while (j > 0 && (j == n || b[i] != a[j+1])) j = next[j];
11     if (b[i] == a[j+1]) j++;
12     f[i] = j;
13     // if (f[i] == n), 此时就是A在B中的某一次出现
14 }

```

字符串循环元可利用 next 数组求解。

## 4.4 最小表示法

```

1  // 最小表示法
2  int n = strlen(s + 1);
3  for (int i = 1; i <= n; i++) s[n+i] = s[i];
4  int i = 1, j = 2, k;
5  while (i <= n && j <= n) {
6      for (k = 0; k < n && s[i+k] == s[j+k]; k++);
7      if (k == n) break; // s likes "aaaaa"
8      if (s[i+k] > s[j+k]) {
9          i = i + k + 1;
10         if (i == j) i++;
11     } else {
12         j = j + k + 1;
13         if (i == j) j++;
14     }
15 }
16 ans = min(i, j);

```

## Chapter 5

# 数据结构

## Chapter 6

# 图论

## 6.1 最短路

### 6.1.1 单源最短路径

#### 6.1.1.1 Dijkstra

```

1 void Dijkstra()
2 {
3     memset(dist, 0x3f, sizeof(dist));
4     memset(vis, 0, sizeof(vis));
5     priority_queue<pii, vector<pii>, greater<pii>> q;
6     dist[1] = 0;
7     q.push({dist[1], 1});
8     while (!q.empty())
9     {
10         int x = q.top().second;
11         q.pop();
12         if (!vis[x])
13         {
14             vis[x] = 1;
15             for (auto it : v[x])
16             {
17                 int y = it.first;
18                 if (dist[y] > dist[x] + it.second)
19                 {
20                     dist[y] = dist[x] + it.second;
21                     q.push({dist[y], y});
22                 }
23             }
24         }
25     }
26 }

```

#### 6.1.1.2 Bellman-Ford 和 SPFA

```

1 void SPFA()
2 {
3     memset(dis, 0x3f, sizeof(dis));
4     memset(vis, 0, sizeof(vis));
5     queue<int> q;
6     dis[1] = 0;
7     vis[1] = 1;
8     q.push(1);
9     while (!q.empty())
10    {
11        int x = q.front();
12        q.pop();
13        vis[x] = 0;
14        for (int i = 0; i < v[x].size(); i++)
15        {
16            int y = v[x][i].first;
17            int z = v[x][i].second;
18            if (dis[y] > dis[x] + z)
19            {
20                dis[y] = dis[x] + z;
21                if (!vis[y])
22                    q.push(y), vis[y] = 1;
23            }
24        }
25    }
26 }

```

#### 例题分析

POJ3662 Telephone Lines (分层图最短路/二分答案, 双端队列 BFS)

P1073 最优贸易 (原图与反图, 枚举节点)

P3008 [USACO11JAN] 道路和飞机 Roads and Planes (DAG, 拓扑序, 连通块)



## 6.1.2 任意两点间最短路径

### 6.1.2.1 Floyd

```

1 void get_path(int i, int j)
2 {
3     if (!path[i][j])
4         return;
5     get_path(i, path[i][j]);
6     p.push_back(path[i][j]);
7     get_path(path[i][j], j);
8 }
9 void Floyd()
10 {
11     memcpy(d, a, sizeof(d));
12     for (int k = 1; k <= n; k++)
13     {
14         for (int i = 1; i < k; i++)
15         {
16             for (int j = i + 1; j < k; j++)
17             {
18                 //注意溢出
19                 ll temp = d[i][j] + a[i][k] + a[k][j];
20                 if (ans > temp)
21                 {
22                     ans = temp;
23                     p.clear();
24                     p.push_back(i);
25                     get_path(i, j);
26                     p.push_back(j);
27                     p.push_back(k);
28                 }
29             }
30         }
31         for (int i = 1; i <= n; i++)
32         {
33             for (int j = 1; j <= n; j++)
34             {
35                 ll temp = d[i][k] + d[k][j];
36                 if (d[i][j] > temp)
37                 {
38                     d[i][j] = temp;
39                     path[i][j] = k;
40                 }
41             }
42         }
43     }
44 }

```

#### 例题分析

POJ1094 Sorting It All Out (传递闭包)

POJ1734 Sightseeing trip (无向图最小环)

POJ3613 Cow Relays (离散化, 广义矩阵乘法, 快速幂)

## 6.2 最小生成树

### 6.2.1 Kruskal

基于并查集

```

1 void Init()
2 {
3     for (int i = 1; i <= n; i++)
4         fa[i] = i;
5 }
6 int Find(int x)
7 {
8     if (x == fa[x])
9         return x;
10    return fa[x] = Find(fa[x]);
11 }
12 void Kruskal()
13 {
14     Init();
15     sort(e.begin(), e.end());
16     int ans=0;
17     for (int i = 0; i < e.size(); i++)
18     {
19         int u = e[i].u, v = e[i].v;
20         int fu = Find(u), fv = Find(v);
21         if (fu != fv)
22         {
23             fa[fu] = fv;
24             ans += e[i].w;
25         }
26     }
27 }
```

### 6.2.2 Prim

```

1 void Prim()
2 {
3     memset(vis, 0, sizeof(vis));
4     memset(d, 0x3f, sizeof(d));
5     d[1] = 0;
6     int temp = n;
7     int ret = 0;
8     while (temp--)
9     {
10         int min_pos = 0;
11         for (int i = 1; i <= n; i++)
12             if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
13                 min_pos = i;
14         if (min_pos)
15         {
16             vis[min_pos] = 1;
17             ret += d[min_pos];
18             for (int i = 1; i <= n; i++)
19                 if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
20         }
21     }
22 }
```

例题分析

走廊泼水节 (Kruskal, 最小生成树扩充为完全图)

POJ1639 Picnic Planning (度限制最小生成树, 连通块, 树形 DP)

```

1 #include <algorithm>
2 #include <cstring>
3 #include <iostream>
```

```

4  #include <map>
5  #include <string>
6  #include <vector>
7  using namespace std;
8  #define inf 0x3f3f3f3f
9  #define N 25
10 #define M 500
11 map<string, int> name;
12 struct edge
13 {
14     int u, v, w;
15     bool operator<(const edge &e) const
16     {
17         return w < e.w;
18     }
19 };
20 int n, s, ptot = 0, a[N][N], ans, fa[N], d[N], ver[N];
21 vector<edge> e;
22 bool vis[N][N];
23 edge dp[N]; //dp[i] 1...i路径上的最大边
24 void Init()
25 {
26     for (int i = 1; i <= ptot; i++)
27         fa[i] = i;
28 }
29 int Find(int x)
30 {
31     if (x == fa[x])
32         return x;
33     return fa[x] = Find(fa[x]);
34 }
35 void Kruskal()
36 {
37     Init();
38     sort(e.begin(), e.end());
39     for (int i = 0; i < e.size(); i++)
40     {
41         int u = e[i].u, v = e[i].v;
42         if (u != 1 && v != 1)
43         {
44             int fu = Find(u), fv = Find(v);
45             if (fu != fv)
46             {
47                 fa[fu] = fv;
48                 vis[u][v] = vis[v][u] = 1;
49                 ans += e[i].w;
50             }
51         }
52     }
53 }
54 void DFS(int cur, int pre)
55 {
56     for (int i = 2; i <= ptot; i++)
57     {
58         if (i != pre && vis[cur][i])
59         {
60             if (dp[i].w == -1)
61             {
62                 if (dp[cur].w < a[cur][i])
63                 {
64                     dp[i].u = cur;
65                     dp[i].v = i;
66                     dp[i].w = a[cur][i];
67                 }
68                 else
69                     dp[i] = dp[cur];

```

```

70         }
71         DFS(i, cur);
72     }
73 }
74 }
75 int main()
76 {
77     ios::sync_with_stdio(false);
78     cin.tie(0);
79     cin >> n;
80     string s1, s2;
81     int len;
82     name["Park"] = ++ptot;
83     memset(a, 0x3f, sizeof(a));
84     memset(d, 0x3f, sizeof(d));
85     //Park: 1
86     for (int i = 0; i < n; i++)
87     {
88         cin >> s1 >> s2 >> len;
89         if (!name[s1])
90             name[s1] = ++ptot;
91         if (!name[s2])
92             name[s2] = ++ptot;
93         int u = name[s1], v = name[s2];
94         a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
95         e.push_back({u, v, len});
96     }
97     cin >> s; //度数限制
98     ans = 0;
99     Kruskal();
100    for (int i = 2; i <= ptot; i++)
101    {
102        if (a[1][i] != inf)
103        {
104            int rt = Find(i);
105            if (d[rt] > a[1][i])
106                d[rt] = a[1][i], ver[rt] = i;
107        }
108    }
109    for (int i = 2; i <= ptot; i++)
110    {
111        if (d[i] != inf)
112        {
113            s--;
114            ans += d[i];
115            vis[1][ver[i]] = vis[ver[i]][1] = 1;
116        }
117    }
118    while (s-- > 0)
119    {
120        memset(dp, -1, sizeof(dp));
121        dp[1].w = -inf;
122        for (int i = 2; i <= ptot; i++)
123        {
124            if (vis[1][i])
125                dp[i].w = -inf;
126        }
127        DFS(1, -1);
128        int w = -inf;
129        int v;
130        for (int i = 2; i <= ptot; i++)
131        {
132            if (w < dp[i].w - a[1][i])
133            {
134                w = dp[i].w - a[1][i];
135                v = i;

```

```
136         }
137     }
138     if (w <= 0)
139         break;
140     ans -= w;
141     vis[1][v] = vis[v][1] = 1;
142     vis[dp[v].u][dp[v].v] = vis[dp[v].v][dp[v].u] = 0;
143 }
144 cout << "Total miles driven: " << ans << endl;
145 system("pause");
146 return 0;
147 }
```

POJ2728 Desert King (最优比率生成树, 0/1 分数规划, 二分)  
黑暗城堡 (最短路径生成树计数, 最短路, 排序)

## 6.3 树的直径

### 6.3.1 树形 DP 求树的直径

仅能求出直径长度，无法得知路径信息，可处理负权边。

```

1  int dp[N];
2  //dp[rt] 以rt为根的子树 从rt出发最远可达距离
3  /*
4   对于每个结点x f[x]:经过节点x的最长链长度
5  */
6  void DP(int rt)
7  {
8      dp[rt]=0;//单点
9      vis[rt]=1;
10     for(int i=head[rt];i;i=nxt[i])
11     {
12         int s=ver[i];
13         if(!vis[s])
14         {
15             DP(s);
16             diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
17             dp[rt]=max(dp[rt],dp[s]+edge[i]);
18         }
19     }
20 }
```

### 6.3.2 两次 BFS/DFS 求树的直径

无法处理负权边，容易记录路径

```

1  void DFS(int start,bool record_path)
2  {
3      vis[start]=1;
4      for(int i=head[start];i;i=nxt[i])
5      {
6          int s=ver[i];
7          if(!vis[s])
8          {
9              dis[s]=dis[start]+edge[i];
10             if(record_path) path[s]=i;
11             DFS(s,record_path);
12         }
13     }
14     vis[start]=0;//清理
15 }
```

例题分析

P3629 [APIO2010] 巡逻（两种求树直径方法的综合应用）  
P1099 树网的核（枚举）

## 6.4 最近公共祖先（LCA）

### 6.4.1 树上倍增

```

1  void BFS()
2  {
3      queue<int> q;
4      q.push(1);
5      d[1] = 1;
6      while (!q.empty())
7      {
8          int x = q.front();
9          q.pop();
10         for (int i = head[x]; i; i = nxt[i])
```

```

11     {
12         int y = ver[i];
13         if (!d[y])
14         {
15             d[y] = d[x] + 1;
16             fa[y][0] = x;
17             for (int j = 1; j <= k; j++)
18             {
19                 fa[y][j] = fa[fa[y][j - 1]][j - 1];
20             }
21             q.push(y);
22         }
23     }
24 }
25 }
26 int LCA(int x, int y)
27 {
28     if (d[x] < d[y])
29         swap(x, y);
30     for (int i = k; i >= 0; i--)
31         if (d[fa[x][i]] >= d[y])
32             x = fa[x][i];
33     if (x == y)
34         return y;
35     for (int i = k; i >= 0; i--)
36         if (fa[x][i] != fa[y][i])
37             x = fa[x][i], y = fa[y][i];
38     return fa[x][0];
39 }

```

### 6.4.2 Tarjan

```

1 int Find(int x)
2 {
3     if (x == fa[x])
4         return x;
5     return fa[x] = Find(fa[x]);
6 }
7 void Tarjan(int x)
8 {
9     vis[x] = 1;
10    for (int i = head[x]; i; i = nxt[i])
11    {
12        int y = ver[i];
13        if (!vis[y])
14        {
15            Tarjan(y);
16            fa[y] = x;
17        }
18    }
19    for (int i = 0; i < q[x].size(); i++)
20    {
21        int y = q[x][i].first, id = q[x][i].second;
22        if (vis[y] == 2)
23            lca[id] = Find(y);
24    }
25    vis[x] = 2;
26 }

```

## 6.5 树上差分

例题分析

POJ3417 Network (LCA, 树上差分, 边覆盖)

6302 雨天的尾巴 (LCA, 树上差分, 点覆盖, 权值线段树, 线段树合并)

P1600 天天爱跑步 (LCA, 树上差分)

## 6.6 LCA 的综合应用

例题分析

CH56C 异象石 (dfn 时间戳, LCA)

P4180 【模板】严格次小生成树 [BJWC2010] (树上倍增)

```

1  #include <algorithm>
2  #include <iostream>
3  #include <math.h>
4  #include <queue>
5  #include <stdio.h>
6  using namespace std;
7  const int N = 1e5 + 50;
8  const int M = 6e5 + 50;
9  #define ll long long
10 #define pii pair<int, int>
11 #define inf 0x3f3f3f3f
12 int n, m, k, F[N][20], d[N], fa[N];
13 int head[N], ver[M], nxt[M], edge[M], tot;
14 ll G[N][20][2];
15 void add(int x, int y, int z)
16 {
17     ver[++tot] = y, nxt[tot] = head[x], head[x] = tot, edge[tot] = z;
18 }
19 struct edge
20 {
21     int x, y, z;
22     bool used;
23     bool operator<(const edge &e) const
24     {
25         return z < e.z;
26     }
27 } e[M];
28 int Find(int x)
29 {
30     if (fa[x] == x)
31         return x;
32     return fa[x] = Find(fa[x]);
33 }
34 ll Kruskal()
35 {
36     for (int i = 1; i <= n; i++)
37         fa[i] = i;
38     sort(e + 1, e + 1 + m);
39     ll ans = 0;
40     int cnt = 0;
41     for (int i = 1; i <= m; i++)
42     {
43         int fx = Find(e[i].x), fy = Find(e[i].y);
44         if (fx != fy)
45         {
46             fa[fx] = fy;
47             ans += e[i].z;
48             e[i].used = true;
49             cnt++;
50             if (cnt >= n - 1)
51                 break;
52         }
53     }
54     return ans;
55 }
56 void BFS()

```



```

57 {
58     k = log2(n) + 1;
59     queue<int> q;
60     q.push(1), d[1] = 1;
61     for (int i = 0; i <= k; i++)
62         G[1][i][0] = G[1][i][1] = -inf;
63     while (q.size())
64     {
65         int x = q.front();
66         q.pop();
67         for (int i = head[x]; i; i = nxt[i])
68         {
69             int y = ver[i];
70             if (!d[y])
71             {
72                 d[y] = d[x] + 1;
73                 F[y][0] = x;
74                 G[y][0][0] = edge[i];
75                 G[y][0][1] = -inf;
76                 for (int j = 1; j <= k; j++)
77                 {
78                     F[y][j] = F[F[y][j - 1]][j - 1];
79                     G[y][j][0] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][0]);
80                     if (G[y][j - 1][0] == G[F[y][j - 1]][j - 1][0])
81                         G[y][j][1] = max(G[y][j - 1][1], G[F[y][j - 1]][j - 1][1]);
82                     else if (G[y][j - 1][0] > G[F[y][j - 1]][j - 1][0])
83                         G[y][j][1] = max(G[F[y][j - 1]][j - 1][0], G[y][j - 1][1]);
84                     else
85                         G[y][j][1] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][1]);
86                 }
87                 q.push(y);
88             }
89         }
90     }
91 }
92 pii LCA(int x, int y)
93 {
94     ll val1 = -inf, val2 = -inf;
95     if (d[y] > d[x])
96         swap(x, y);
97     for (int i = k; i >= 0; i--)
98     {
99         if (d[F[x][i]] >= d[y])
100         {
101             if (G[x][i][0] > val1)
102                 val1 = G[x][i][0], val2 = max(val2, G[x][i][1]);
103             else if (G[x][i][0] < val1)
104                 val2 = max(val2, G[x][i][0]);
105             x = F[x][i];
106         }
107     }
108     if (x == y)
109         return make_pair(val1, val2);
110     for (int i = k; i >= 0; i--)
111     {
112         if (F[x][i] != F[y][i])
113         {
114             val1 = max(val1, max(G[x][i][0], G[y][i][0]));
115             val2 = max(val2, (val1 == G[x][i][0]) ? G[x][i][1] : G[x][i][0]);
116             val2 = max(val2, (val1 == G[y][i][0]) ? G[y][i][1] : G[y][i][0]);
117             x = F[x][i], y = F[y][i];
118         }
119     }
120     val1 = max(val1, max(G[x][0][0], G[y][0][0]));
121     val2 = max(val2, (val1 == G[x][0][0]) ? G[x][0][1] : G[x][0][0]);
122     val2 = max(val2, (val1 == G[y][0][0]) ? G[y][0][1] : G[y][0][0]);

```

```
123     return make_pair(val1, val2);
124 }
125 int main()
126 {
127     scanf("%d%d", &n, &m);
128     for (int i = 1; i <= m; i++)
129         scanf("%d%d%d", &e[i].x, &e[i].y, &e[i].z), e[i].used = false;
130     ll sum = Kruskal(), ans = 0x3f3f3f3f3f3f3f3f;
131     for (int i = 1; i <= m; i++)
132         if (e[i].used)
133             add(e[i].x, e[i].y, e[i].z), add(e[i].y, e[i].x, e[i].z);
134     BFS();
135     for (int i = 1; i <= m; i++)
136     {
137         if (!e[i].used)
138         {
139             pii temp = LCA(e[i].x, e[i].y);
140             if (e[i].z > temp.first)
141                 ans = min(ans, sum - temp.first + e[i].z);
142             else if (e[i].z == temp.first)
143                 ans = min(ans, sum - temp.second + e[i].z);
144         }
145     }
146     cout << ans << endl;
147     //system("pause");
148     return 0;
149 }
```

P1084 疫情控制（二分，树上倍增，贪心）

## 6.7 基环树

## 6.8 负环与差分约束

### 6.8.1 负环

例题分析

POJ3621 Sightseeing Cows (0/1 分数规划, SPFA 判定负环)

### 6.8.2 差分约束系统

例题分析

POJ1201 Intervals (单源最长路)

## 6.9 Tarjan 算法与无向图连通性

### 6.9.1 无向图的割点与桥

#### 6.9.1.1 割边判定法则

```

1 void Tarjan(int x, int in_edge)
2 {
3     dfn[x] = low[x] = ++num;
4     for (int i = head[x]; i; i = nxt[i])
5     {
6         int y = ver[i];
7         if (!dfn[y])
8         {
9             Tarjan(y, i);
10            low[x] = min(low[x], low[y]);
11            if (low[y] > dfn[x])
12            {
13                bridge[i] = bridge[i ^ 1] = true;
14            }
15        }
16        else if (i != (in_edge ^ 1))
17            low[x] = min(low[x], dfn[y]);
18    }
19 }

```

#### 6.9.1.2 割点判定法则

```

1 void Tarjan(int x)
2 {
3     dfn[x] = low[x] = ++num;
4     int flag = 0;
5     for (int i = head[x]; i; i = nxt[i])
6     {
7         int y = ver[i];
8         if (!dfn[y])
9         {
10            Tarjan(y);
11            low[x] = min(low[x], low[y]);
12            if (low[y] >= dfn[x])
13            {
14                flag++;
15                if (x != root || flag >= 2)
16                    cut[x] = true;
17            }
18        }
19        else
20            low[x] = min(low[x], dfn[y]);
21    }
22 }

```

例题分析

P3469 [POI2008]BLO-Blockade (割点, 连通块计数)

### 6.9.2 无向图的双连通分量

#### 6.9.2.1 边双连通分量 e-DCC 与其缩点

```

1 void DFS(int x)
2 {
3     color[x] = dcc;
4     for (int i = head[x]; i; i = nxt[i])
5     {
6         int y = ver[i];
7         if (!color[y] && !bridge[i])

```

```

8         DFS(y);
9     }
10 }
11 void e_DCC()
12 {
13     dcc = 0;
14     for (int i = 1; i <= n; i++)
15         if (!color[i])
16             ++dcc, DFS(i);
17     totc = 1;
18     for (int i = 2; i <= tot; i++)
19     {
20         int u = ver[i ^ 1], v = ver[i];
21         if (color[u] != color[v])
22             add_c(color[u], color[v]);
23     }
24 }

```

### 6.9.2.2 点双连通分量 v-DCC 与其缩点

```

1 void Tarjan(int x)
2 {
3     dfn[x] = low[x] = ++num;
4     int flag = 0;
5     stack[++top] = x;
6     if (x == root && !head[x])
7     {
8         dcc[++cnt].push_back(x);
9         return;
10    }
11    for (int i = head[x]; i; i = nxt[i])
12    {
13        int y = ver[i];
14        if (!dfn[y])
15        {
16            Tarjan(y);
17            low[x] = min(low[x], low[y]);
18            if (low[y] >= dfn[x])
19            {
20                flag++;
21                if (x != root || flag >= 2)
22                    cut[x] = true;
23                cnt++;
24                int z;
25                do
26                {
27                    z = stack[top--];
28                    dcc[cnt].push_back(z);
29                } while (z != y);
30                dcc[cnt].push_back(x);
31            }
32        }
33        else
34            low[x] = min(low[x], dfn[y]);
35    }
36 }
37 void v_DCC()
38 {
39     cnt = 0;
40     top = 0;
41     for (int i = 1; i <= n; i++)
42     {
43         if (!dfn[i])
44             root = i, Tarjan(i);
45     }

```

```

46 // 给每个割点一个新的编号(编号从cnt+1开始)
47 num = cnt;
48 for (int i = 1; i <= n; i++)
49     if (cut[i]) new_id[i] = ++num;
50 // 建新图, 从每个v-DCC到它包含的所有割点连边
51 tc = 1;
52 for (int i = 1; i <= cnt; i++)
53     for (int j = 0; j < dcc[i].size(); j++)
54     {
55         int x = dcc[i][j];
56         if (cut[x]) {
57             add_c(i, new_id[x]);
58             add_c(new_id[x], i);
59         }
60         else c[x] = i; // 除割点外, 其它点仅属于1个v-DCC
61     }
62 }

```

例题分析

POJ3694 Network (e-DCC 缩点, LCA, 并查集)

POJ2942 Knights of the Round Table (补图, v-DCC, 染色法奇环判定)

### 6.9.3 欧拉路问题

欧拉图的判定

无向图连通, 所有点度数为偶数。

欧拉路的存在性判定

无向图连通, 恰有两个节点度数为奇数, 其他节点度数均为偶数

```

1 // 模拟系统栈, 答案栈
2 void Euler() {
3     stack[++top] = 1;
4     while (top > 0) {
5         int x = stack[top], i = head[x];
6         // 找到一条尚未访问的边
7         while (i && vis[i]) i = Next[i];
8         // 沿着这条边模拟递归过程, 标记该边, 并更新表头
9         if (i) {
10             stack[++top] = ver[i];
11             head[x] = Next[i];
12             vis[i] = vis[i ^ 1] = true;
13         }
14         // 与x相连的所有边均已访问, 模拟回溯过程, 并记录于答案栈中
15         else {
16             top--;
17             ans[++t] = x;
18         }
19     }
20 }

```

例题分析

POJ2230 Watchcow (欧拉回路)

## 6.10 Tarjan 算法与有向图连通性

### 6.10.1 强连通分量 (SCC) 判定法则

```

1 void Tarjan(int x)
2 {
3     dfn[x]=low[x]=++num;
4     stack[++top]=x,in_stack[x]=true;
5     for(int i=head[x];i;i=nxt[i])
6     {
7         int y=ver[i];
8         if(!dfn[y])
9         {
10             Tarjan(y);
11             low[x]=min(low[x],low[y]);
12         }
13         else if(in_stack[y])
14             low[x]=min(low[x],dfn[y]);
15     }
16     if(dfn[x]==low[x])
17     {
18         cnt++;
19         int y;
20         do
21         {
22             y=stack[top--],in_stack[y]=false;
23             color[y]=cnt, scc[cnt].push_back(y);
24         } while (x!=y);
25     }
26 }

```

### 6.10.2 SCC -> DAG

```

1 void SCC()
2 {
3     for (int i = 0; i <= n; i++)
4         if (!dfn[i])
5             Tarjan(i);
6     //缩点
7     for (int x = 1; x <= n; x++)
8     {
9         for (int i = head[x]; i; i = nxt[i])
10        {
11            int y = ver1[i];
12            if (color[x] != color[y])
13                add_c(color[x], color[y]);
14        }
15    }
16 }

```

例题分析

POJ1236 Network of Schools (SCC->DAG, 入度出度)

P3275 [SCOI2011] 糖果 (SPFA TLE, SCC->DAG, Topo, DP)

### 6.10.3 有向图的必经点与必经边

对于有向无环图 (DAG):

在原图中按照拓扑序进行动态规划, 求出起点  $S$  到图中每个点  $x$  的路径条数  $fs[x]$ 。

在反图上再次按照拓扑序进行动态规划, 求出每个点  $x$  到终点  $T$  的路径条数  $ft[x]$ 。

显然,  $fs[T]$  表示从  $S$  到  $T$  的路径总条数。根据乘法原理:

1: 对于一条有向边  $(x,y)$ , 若  $fs[x]*ft[y]=fs[T]$ , 则  $(x,y)$  是有向无环图从  $S$  到  $T$  的必经边。

2: 对于一个点  $x$ , 若  $fs[x]*ft[x]=fs[T]$ , 则  $x$  是有向无环图从  $S$  到  $T$  的必经点。

路径条数规模较大, 可对大质数取模后保存, 但有概率误判。

例题分析

6703 PKU ACM Team's Excursion (DAG 必经边, 枚举, DP)



#### 6.10.4 2-SAT 问题

原命题与逆否命题互为等价命题，在建立有向边关系时要注意对称性。

例题分析

POJ3678 Katu Puzzle (2-SAT, Tarjan, SCC)

POJ3683 Priest John's Busiest Day (2-SAT 方案构造)

```
1 for (int i = 1; i <= 2 * n; i++)
2 {
3     val[i] = color[i] > color[opp[i]];
4     //若c[i] 大于 c[opp[i]]   opp[i]赋0
5 }
```

## 6.11 二分图的匹配

### 6.11.1 二分图判定

一张无向图是二分图，当且仅当图中不存在奇环（长度为奇数的环）。

```

1 //染色法判定奇环
2 bool DFS(int x,int color)
3 {
4     vis[x]=color;
5     for(int i=head[x];i;i=nxt[i])
6     {
7         int y=ver[i];
8         if(!vis[y])
9         {
10             if(!DFS(y,3-color)) return false;
11         }
12         else if(vis[y]==color) return false;
13     }
14     return true;
15 }
```

例题分析

P1525 关押罪犯（判定二分图，二分）

### 6.11.2 二分图最大匹配

二分图匹配的模型要素

- 1: 节点能分成独立的两个集合，每个集合内部有 0 条边。“0 要素”
- 2: 每个节点只能与 1 条匹配边相连。“1 要素”

```

1 \\匈牙利算法
2 \\在主函数中对每个左部节点调用寻找增广路时，需要对 vis 重置。
3 bool DFS(int x)
4 {
5     for (int i = head[x]; i; i = nxt[i])
6     {
7         int y = ver[i];
8         if (!vis[y])
9         {
10             vis[y] = 1;
11             if (!match[y] || DFS(match[y]))
12             {
13                 match[y] = x;
14                 return true;
15             }
16         }
17     }
18     return false;
19 }
```

例题分析

6801 棋盘覆盖（奇偶染色）

6802 車的放置（行列）

6803 导弹防御塔（二分，拆点多重匹配）

### 6.11.3 二分图带权匹配

二分图带权最大匹配的前提是匹配数最大，然后再最大化匹配边的权值总和。

```

1 /*KM 稠密图上效率高于费用流，但是有较大局限性，只能在满足“带权最大匹配一定是完备匹配”的图中正确求解。
2 w[][]:边权
3 la[], lb[]: 左，右部点顶标
4 visa[], visb[]: 访问标记，是否在交错树中
5 ans: Σw[match[i]][i]
6 */
7 bool DFS(int x)
```

```

8 {
9     visa[x] = true;
10    for (int y = 1; y <= n; y++)
11    {
12        if (!visb[y])
13        {
14            double temp = fabs(la[x] + lb[y] - w[x][y]); //对于浮点数, 相等子图的判定
15            if (temp < eps)
16            {
17                visb[y] = true;
18                if (!match[y] || DFS(match[y]))
19                {
20                    match[y] = x;
21                    return true;
22                }
23            }
24            else
25                upd[y] = min(upd[y], la[x] + lb[y] - w[x][y]);
26        }
27    }
28    return false;
29 }
30 void KM()
31 {
32     for (int i = 1; i <= n; i++)
33     {
34         la[i] = -inf;
35         lb[i] = 0;
36         for (int j = 1; j <= n; j++)
37             la[i] = max(la[i], w[i][j]);
38     }
39     for (int i = 1; i <= n; i++)
40     {
41         while (true)
42         {
43             memset(visa, 0, sizeof(visa));
44             memset(visb, 0, sizeof(visb));
45             for (int j = 1; j <= n; j++)
46                 upd[j] = inf;
47             if (DFS(i))
48                 break;
49             else
50             {
51                 delta = inf;
52                 for (int j = 1; j <= n; j++)
53                     if (!visb[j])
54                         delta = min(delta, upd[j]);
55                 for (int j = 1; j <= n; j++)
56                 {
57                     if (visa[j])
58                         la[j] -= delta;
59                     if (visb[j])
60                         lb[j] += delta;
61                 }
62             }
63         }
64     }
65 }

```

例题分析

POJ3565 Ants (三角形不等式, 二分图带权最小匹配)

## 6.12 二分图的覆盖与独立集

### 6.12.1 二分图最小点覆盖

二分图最小覆盖模型特点：

每条边有 2 个端点，二者至少选择一个。“2 要素”

#### 6.12.1.1 König's theorem

二分图最小点覆盖包含的点数等于二分图最大匹配包含的边数。

例题分析

POJ1325 Machine Schedule (二分图最小覆盖)

POJ2226 Muddy Fields (行列连续块，二分图最小覆盖)

### 6.12.2 二分图最大独立集

无向图  $G$  的最大团等于其补图  $G'$  的最大独立集。(补图转化)

设  $G$  是有  $n$  个节点的二分图， $G$  的最大独立集的大小等于  $n$  减去最大匹配数。

例题分析

6901 骑士放置 (奇偶染色)

### 6.12.3 有向无环图的最小路径点覆盖

给定一张有向无环图，要求用尽量少的不相交的简单路径，覆盖有向无环图的所有顶点（也就是每个顶点恰好被覆盖一次）。这个问题被称为有向无环图的最小路径点覆盖，简称“最小路径覆盖”。

有向无环图  $G$  的最小路径点覆盖包含的路径条数，等于  $n$  (有向无环图的点数) 减去拆点二分图  $G_2$  的最大匹配数。

若简单路径可相交，即一个节点可被覆盖多次，这个问题称为有向无环图的最小路径可重复点覆盖。

对于这个问题，可先对  $G$  求传递闭包，得到有向无环图  $G'$ ，再在  $G'$  上求一般的（路径不可相交的）最小路径点覆盖。

例题分析

6902 Vani 和 Cl2 捉迷藏 (最小路径可重复点覆盖，构造方案)

```

1 // 构造方案，先把所有路径终点（左部非匹配点）作为藏身点
2 for (int i = 1; i <= n; i++) succ[match[i]] = true;
3 for (int i = 1, k = 0; i <= n; i++)
4     if (!succ[i]) hide[++k] = i;
5 memset(vis, 0, sizeof(vis));
6 bool modify = true;
7 while (modify) {
8     modify = false;
9     // 求出 next(hide)
10    for (int i = 1; i <= ans; i++)
11        for (int j = 1; j <= n; j++)
12            if (cl[hide[i]][j]) vis[j] = true;
13    for (int i = 1; i <= ans; i++)
14        if (vis[hide[i]]) {
15            modify = true;
16            // 不断向上移动
17            while (vis[hide[i]]) hide[i] = match[hide[i]];
18        }
19 }
20 for (int i = 1; i <= ans; i++) printf("%d ", hide[i]);
21 cout << endl;

```

## 6.13 网络流初步

### 6.13.1 最大流

#### 6.13.1.1 Edmonds Karp 增广路

```

1  bool BFS() {
2      memset(vis, 0, sizeof(vis));
3      queue<int> q;
4      q.push(S); vis[S] = 1;
5      incf[S] = inf; // 增广路上各边的最小剩余容量
6      while (q.size()) {
7          int x = q.front(); q.pop();
8          for (int i = head[x]; i; i = Next[i])
9              if (edge[i]) {
10                 int y = ver[i];
11                 if (vis[y]) continue;
12                 incf[y] = min(incf[x], edge[i]);
13                 pre[y] = i; // 记录前驱, 便于找到最长路的实际方案
14                 q.push(y), vis[y] = 1;
15                 if (y == t) return 1;
16             }
17      }
18      return 0;
19  }
20  void Update() { // 更新增广路及其反向边的剩余容量
21      int x = t;
22      while (x != s) {
23          int i = pre[x];
24          edge[i] -= incf[t];
25          edge[i ^ 1] += incf[t]; // 利用“成对存储”的xor 1技巧
26          x = ver[i ^ 1];
27      }
28      maxflow += incf[t];
29  }

```

#### 6.13.1.2 Dinic

```

1  //可加入当前弧优化(&)：在增广时复制head[]到cur[]，在增广时同步修改cur[]，目的是递归时跳过已增广的边。
2  bool BFS()
3  {
4      memset(d, 0, sizeof(d));
5      queue<int> q;
6      q.push(S);
7      d[S] = 1; //不为1 陷入死循环
8      while (q.size())
9      {
10         int x = q.front();
11         q.pop();
12         for (int i = head[x]; i; i = nxt[i])
13             {
14                 int y = ver[i];
15                 if (edge[i] && !d[y])
16                     {
17                         d[y] = d[x] + 1;
18                         q.push(y);
19                         if (y == T)
20                             return true;
21                     }
22             }
23     }
24     return false;
25 }
26 int Dinic(int x, int flow)
27 {

```

```

28     if (x == T)
29         return flow;
30     int rest = flow, k;
31     for (int i = head[x]; i && rest; i = nxt[i])
32     {
33         int y = ver[i];
34         if (edge[i] && d[y] == d[x] + 1)
35         {
36             k = Dinic(y, min(edge[i], rest));
37             if (!k)
38                 d[y] = 0;
39             edge[i] -= k;
40             edge[i ^ 1] += k;
41             rest -= k;
42         }
43     }
44     return flow - rest;
45 }

```

### 6.13.1.3 二分图最大匹配的必须边与可行边

在一般的二分图中，可以用最大流计算任一组最大匹配。

此时：必须边的判定条件为：(x,y) 流量为 1，并且在残量网络上属于不同的 SCC。

可行边的判定条件为：(x,y) 流量为 1，或者在残量网络上属于同一个 SCC。

例题分析

CH17C 舞动的夜晚 (Dinic, Tarjan, 二分图可行边)

## 6.13.2 最小割

### 6.13.2.1 最大流最小割定理

任何一个网络的最大流量等于最小割中边的容量之和。

例题分析

POJ1966 Cable TV Network (枚举, 点边转化)

## 6.13.3 费用流

### 6.13.3.1 Edmonds Karp 增广路

BFS 寻找增广路 -> SPFA 寻找单位费用之和最小的增广路 (将费用作为边权, 在残量网络上求最短路)。

注意：反向边的费用为相反数。

```

1 bool SPFA()
2 {
3     memset(dis, 0xcf, sizeof(dis)); // -inf
4     memset(vis, 0, sizeof(vis));
5     queue<int> q;
6     dis[S] = 0, vis[S] = 1, incf[S] = 1 << 30;
7     q.push(S);
8     while (q.size())
9     {
10         int x = q.front();
11         q.pop();
12         vis[x] = 0;
13         for (int i = head[x]; i; i = nxt[i])
14         {
15             if (edge[i])
16             {
17                 int y = ver[i];
18                 if (dis[y] < dis[x] + cost[i])
19                 {
20                     dis[y] = dis[x] + cost[i];
21                     incf[y] = min(incf[x], edge[i]);
22                     pre[y] = i;
23                     if (!vis[y])
24                         q.push(y), vis[y] = 1;

```

```
25         }
26     }
27 }
28 }
29 if (dis[T] == 0xcfcfcfcf)
30     return false;
31     return true;
32 }
33 int max_flow, ans;
34 void Update()
35 {
36     int x = T;
37     while (x != S)
38     {
39         int i = pre[x];
40         edge[i] -= incf[T];
41         edge[i ^ 1] += incf[T];
42         x = ver[i ^ 1];
43     }
44     max_flow += incf[T];
45     ans += incf[T] * dis[T];
46 }
```

例题分析

POJ3422 Kaka's Matrix Travels (点边转化, 费用流)