南京工業大學
NANJING TECH UNIVERSITY

ICPC Template Manual

作者: 贺梦杰

August 20, 2019

Contents

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

```
#define ull unsigned long long
#define P 131
ull f[N], p[N];
void Init()
{
    p[0] = 1, f[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        p[i] = p[i - 1] * P;
        f[i] = f[i - 1] * P + str[i];
    }
}
ull Hash(int left, int right)
{
    return f[right] - f[left - 1] * p[right - left + 1];
}
```

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。

```
#define ull unsigned long long
#define P 131
const int N = 1e6 + 50;
ull pre[N], suf[N], p[N];
string str;
int n;
void Init()
{
    pre[0] = suf[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        pre[i] = pre[i - 1] * P + str[i - 1];
        suf[i] = suf[i - 1] * P + str[n - i];
    }
}
ull Hash(int left, int right)
{
    return pre[right] - pre[left - 1] * p[right - left + 1];
}
ull HashReverse(int left, int right)
{
    //WA 这里注意推导
    return suf[n - left + 1] - suf[n - right] * p[right - left + 1];
}
#define ODD 1
#define EVEN 2
bool Judge(int pos, int len, int flag)
{
    if (flag == ODD)
    {
        if (pos - len < 1 || pos > n || pos < 1 || pos + len > n)
            return false;
        return Hash(pos - len, pos) == HashReverse(pos, pos + len);
    }
    else
    {
        if (pos - len < 1 || pos - 1 > n || pos < 1 || pos + len - 1 > n)
            return false;
        return Hash(pos - len, pos - 1) == HashReverse(pos, pos + len - 1);
    }
}
```

```
int Solve(int pos)
{
    int ans = 1;
    //奇数
    int left = 0;
    int right = N;
    int mid;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (Judge(pos, mid, ODD))
            left = mid;
        else
            right = mid - 1;
    }
    ans = max(ans, 2 * left + 1);
    //偶数
    left = 0;
    right = N;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (Judge(pos, mid, EVEN))
            left = mid;
        else
            right = mid - 1;
    }
    ans = max(ans, 2 * left);
    return ans;
}
```

4.1.2 应用：后缀数组

```
#define ull unsigned long long
#define P 131
const int N = 3e5 + 50;
ull f[N], p[N];
char str[N];
int SA[N], n, Height[N];
void Init()
{
    p[0] = 1, f[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        p[i] = p[i - 1] * P;
        f[i] = f[i - 1] * P + str[i];
    }
}
ull Hash(int left, int right)
{
    return f[right] - f[left - 1] * p[right - left + 1];
}
// k:[0,n) 表示后缀S(k,n-1)
// 最长公共前缀
int LCP(int a, int b)
{
    int left = 0;
    int right = N;
    int mid;
    while (left < right)
    {
        mid = (left + right + 1) >> 1;
        if (a + mid - 1 <= n && b + mid - 1 <= n && Hash(a, a + mid - 1) == Hash(b, b + mid - 1))
            left = mid;
        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

```
const int MAXLEN = 1e6 + 50;
namespace SAM
{
    struct state
    {
        int len, link, next[26];
    } st[MAXLEN * 2];
    int sz, last;
    void SAM_Init()
    {
        st[0].len = 0, st[0].link = -1;
        sz++, last = 0;
    }
    void SAM_Extend(int c)
    {
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next[c])
        {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1)
            st[cur].link = 0;
        else
        {
            int q = st[p].next[c];
            if (st[q].len == st[p].len + 1)
                st[cur].link = q;
            else
            {
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                memcpy(st[clone].next, st[q].next, sizeof(st[clone].next));
                st[clone].link = st[q].link;
                while (p != -1 && st[p].next[c] == q)
                {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }
    int id[MAXLEN * 2], c[MAXLEN * 2];
    void Topo()
    {
        //计数排序
        for (int i = 1; i < sz; i++)
            c[st[i].len]++;
        for (int i = 1; i < MAXLEN; i++)
            c[i] += c[i - 1];
        for (int i = 1; i < sz; i++)
            id[c[st[i].len]--] = i;
    }
} // namespace SAM
```

4.3 KMP

```
// KMP
next[1] = 0;
for (int i = 2, j = 0; i <= n; i++) {
    while (j > 0 && a[i] != a[j+1]) j = next[j];
    if (a[i] == a[j+1]) j++;
    next[i] = j;
}

for (int i = 1, j = 0; i <= m; i++) {
    while (j > 0 && (j == n || b[i] != a[j+1])) j = next[j];
    if (b[i] == a[j+1]) j++;
    f[i] = j;
    // if (f[i] == n)，此时就是A在B中的某一次出现
}
```

字符串循环元可利用 next 数组求解。

4.4 最小表示法

```
// 最小表示法
int n = strlen(s + 1);
for (int i = 1; i <= n; i++) s[n+i] = s[i];
int i = 1, j = 2, k;
while (i <= n && j <= n) {
    for (k = 0; k < n && s[i+k] == s[j+k]; k++);
    if (k == n) break; // s likes "aaaaa"
    if (s[i+k] > s[j+k]) {
        i = i + k + 1;
        if (i == j) i++;
    } else {
        j = j + k + 1;
        if (i == j) j++;
    }
}
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

```
void Dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    priority_queue<pii, vector<pii>, greater<pii>> q;
    dist[1] = 0;
    q.push({dist[1], 1});
    while (!q.empty())
    {
        int x = q.top().second;
        q.pop();
        if (!vis[x])
        {
            vis[x] = 1;
            for (auto it : v[x])
            {
                int y = it.first;
                if (dist[y] > dist[x] + it.second)
                {
                    dist[y] = dist[x] + it.second;
                    q.push({dist[y], y});
                }
            }
        }
    }
}
```

6.1.1.2 Bellman-Ford 和 SPFA

```
void SPFA()
{
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    dis[1] = 0;
    vis[1] = 1;
    q.push(1);
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = 0; i < v[x].size(); i++)
        {
            int y = v[x][i].first;
            int z = v[x][i].second;
            if (dis[y] > dis[x] + z)
            {
                dis[y] = dis[x] + z;
                if (!vis[y])
                    q.push(y), vis[y] = 1;
            }
        }
    }
}
```

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

```
void get_path(int i, int j)
{
    if (!path[i][j])
        return;
    get_path(i, path[i][j]);
    p.push_back(path[i][j]);
    get_path(path[i][j], j);
}
void Floyd()
{
    memcpy(d, a, sizeof(d));
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i < k; i++)
        {
            for (int j = i + 1; j < k; j++)
            {
                //注意溢出
                ll temp = d[i][j] + a[i][k] + a[k][j];
                if (ans > temp)
                {
                    ans = temp;
                    p.clear();
                    p.push_back(i);
                    get_path(i, j);
                    p.push_back(j);
                    p.push_back(k);
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                ll temp = d[i][k] + d[k][j];
                if (d[i][j] > temp)
                {
                    d[i][j] = temp;
                    path[i][j] = k;
                }
            }
        }
    }
}
```

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

```
void Init()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    int ans=0;
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        int fu = Find(u), fv = Find(v);
        if (fu != fv)
        {
            fa[fu] = fv;
            ans += e[i].w;
        }
    }
}
```

6.2.2 Prim

```
void Prim()
{
    memset(vis, 0, sizeof(vis));
    memset(d, 0x3f, sizeof(d));
    d[1] = 0;
    int temp = n;
    int ret = 0;
    while (temp--)
    {
        int min_pos = 0;
        for (int i = 1; i <= n; i++)
            if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
                min_pos = i;
        if (min_pos)
        {
            vis[min_pos] = 1;
            ret += d[min_pos];
            for (int i = 1; i <= n; i++)
                if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
        }
    }
}
```

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）

```
#include <algorithm>
#include <cstring>
#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];
```

```
            }
            DFS(i, cur);
        }
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    string s1, s2;
    int len;
    name["Park"] = ++ptot;
    memset(a, 0x3f, sizeof(a));
    memset(d, 0x3f, sizeof(d));
    //Park: 1
    for (int i = 0; i < n; i++)
    {
        cin >> s1 >> s2 >> len;
        if (!name[s1])
            name[s1] = ++ptot;
        if (!name[s2])
            name[s2] = ++ptot;
        int u = name[s1], v = name[s2];
        a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
        e.push_back({u, v, len});
    }
    cin >> s; //度数限制
    ans = 0;
    Kruskal();
    for (int i = 2; i <= ptot; i++)
    {
        if (a[1][i] != inf)
        {
            int rt = Find(i);
            if (d[rt] > a[1][i])
                d[rt] = a[1][i], ver[rt] = i;
        }
    }
    for (int i = 2; i <= ptot; i++)
    {
        if (d[i] != inf)
        {
            s--;
            ans += d[i];
            vis[1][ver[i]] = vis[ver[i]][1] = 1;
        }
    }
    while (s-- > 0)
    {
        memset(dp, -1, sizeof(dp));
        dp[1].w = -inf;
        for (int i = 2; i <= ptot; i++)
        {
            if (vis[1][i])
                dp[i].w = -inf;
        }
        DFS(1, -1);
        int w = -inf;
        int v;
        for (int i = 2; i <= ptot; i++)
        {
            if (w < dp[i].w - a[1][i])
            {
                w = dp[i].w - a[1][i];
                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。

```
int dp[N];
//dp[rt] 以rt为根的子树 从rt出发最远可达距离
/*
    对于每个结点x f[x]:经过节点x的最长链长度
*/
void DP(int rt)
{
    dp[rt]=0;//单点
    vis[rt]=1;
    for(int i=head[rt];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            DP(s);
            diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
            dp[rt]=max(dp[rt],dp[s]+edge[i]);
        }
    }
}
```

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

```
void DFS(int start,bool record_path)
{
    vis[start]=1;
    for(int i=head[start];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            dis[s]=dis[start]+edge[i];
            if(record_path) path[s]=i;
            DFS(s,record_path);
        }
    }
    vis[start]=0;//清理
}
```

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

```
void BFS()
{
    queue<int> q;
    q.push(1);
    d[1] = 1;
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
```

```
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                fa[y][0] = x;
                for (int j = 1; j <= k; j++)
                {
                    fa[y][j] = fa[fa[y][j - 1]][j - 1];
                }
                q.push(y);
            }
        }
    }
}
int LCA(int x, int y)
{
    if (d[x] < d[y])
        swap(x, y);
    for (int i = k; i >= 0; i--)
        if (d[fa[x][i]] >= d[y])
            x = fa[x][i];
    if (x == y)
        return y;
    for (int i = k; i >= 0; i--)
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}
```

6.4.2 Tarjan

```
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Tarjan(int x)
{
    vis[x] = 1;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            Tarjan(y);
            fa[y] = x;
        }
    }
    for (int i = 0; i < q[x].size(); i++)
    {
        int y = q[x][i].first, id = q[x][i].second;
        if (vis[y] == 2)
            lca[id] = Find(y);
    }
    vis[x] = 2;
}
```

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）

```
#include <algorithm>
#include <iostream>
#include <math.h>
#include <queue>
#include <stdio.h>
using namespace std;
const int N = 1e5 + 50;
const int M = 6e5 + 50;
#define ll long long
#define pii pair<int, int>
#define inf 0x3f3f3f3f
int n, m, k, F[N][20], d[N], fa[N];
int head[N], ver[M], nxt[M], edge[M], tot;
ll G[N][20][2];
void add(int x, int y, int z)
{
    ver[++tot] = y, nxt[tot] = head[x], head[x] = tot, edge[tot] = z;
}
struct edge
{
    int x, y, z;
    bool used;
    bool operator<(const edge &e) const
    {
        return z < e.z;
    }
} e[M];
int Find(int x)
{
    if (fa[x] == x)
        return x;
    return fa[x] = Find(fa[x]);
}
ll Kruskal()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
    sort(e + 1, e + 1 + m);
    ll ans = 0;
    int cnt = 0;
    for (int i = 1; i <= m; i++)
    {
        int fx = Find(e[i].x), fy = Find(e[i].y);
        if (fx != fy)
        {
            fa[fx] = fy;
            ans += e[i].z;
            e[i].used = true;
            cnt++;
            if (cnt >= n - 1)
                break;
        }
    }
    return ans;
}
void BFS()
```

```
{
    k = log2(n) + 1;
    queue<int> q;
    q.push(1), d[1] = 1;
    for (int i = 0; i <= k; i++)
        G[1][i][0] = G[1][i][1] = -inf;
    while (q.size())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                F[y][0] = x;
                G[y][0][0] = edge[i];
                G[y][0][1] = -inf;
                for (int j = 1; j <= k; j++)
                {
                    F[y][j] = F[F[y][j - 1]][j - 1];
                    G[y][j][0] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][0]);
                    if (G[y][j - 1][0] == G[F[y][j - 1]][j - 1][0])
                        G[y][j][1] = max(G[y][j - 1][1], G[F[y][j - 1]][j - 1][1]);
                    else if (G[y][j - 1][0] > G[F[y][j - 1]][j - 1][0])
                        G[y][j][1] = max(G[F[y][j - 1]][j - 1][0], G[y][j - 1][1]);
                    else
                        G[y][j][1] = max(G[y][j - 1][0], G[F[y][j - 1]][j - 1][1]);
                }
                q.push(y);
            }
        }
    }
}
pii LCA(int x, int y)
{
    ll val1 = -inf, val2 = -inf;
    if (d[y] > d[x])
        swap(x, y);
    for (int i = k; i >= 0; i--)
    {
        if (d[F[x][i]] >= d[y])
        {
            if (G[x][i][0] > val1)
                val1 = G[x][i][0], val2 = max(val2, G[x][i][1]);
            else if (G[x][i][0] < val1)
                val2 = max(val2, G[x][i][0]);
            x = F[x][i];
        }
    }
    if (x == y)
        return make_pair(val1, val2);
    for (int i = k; i >= 0; i--)
    {
        if (F[x][i] != F[y][i])
        {
            val1 = max(val1, max(G[x][i][0], G[y][i][0]));
            val2 = max(val2, (val1 == G[x][i][0]) ? G[x][i][1] : G[x][i][0]);
            val2 = max(val2, (val1 == G[y][i][0]) ? G[y][i][1] : G[y][i][0]);
            x = F[x][i], y = F[y][i];
        }
    }
    val1 = max(val1, max(G[x][0][0], G[y][0][0]));
    val2 = max(val2, (val1 == G[x][0][0]) ? G[x][0][1] : G[x][0][0]);
    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

```
void Tarjan(int x, int in_edge)
{
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y, i);
            low[x] = min(low[x], low[y]);
            if (low[y] > dfn[x])
            {
                bridge[i] = bridge[i ^ 1] = true;
            }
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}
```

6.9.1.2 割点判定法则

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
```

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

```
void DFS(int x)
{
    color[x] = dcc;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!color[y] && !bridge[i])
```

```
            DFS(y);
    }
}
void e_DCC()
{
    dcc = 0;
    for (int i = 1; i <= n; i++)
        if (!color[i])
            ++dcc, DFS(i);
    totc = 1;
    for (int i = 2; i <= tot; i++)
    {
        int u = ver[i ^ 1], v = ver[i];
        if (color[u] != color[v])
            add_c(color[u], color[v]);
    }
}
```

6.9.2.2 点双连通分量 v-DCC 与其缩点

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    stack[++top] = x;
    if (x == root && !head[x])
    {
        dcc[++cnt].push_back(x);
        return;
    }
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
                cnt++;
                int z;
                do
                {
                    z = stack[top--];
                    dcc[cnt].push_back(z);
                } while (z != y);
                dcc[cnt].push_back(x);
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
void v_DCC()
{
    cnt = 0;
    top = 0;
    for (int i = 1; i <= n; i++)
    {
        if (!dfn[i])
            root = i, Tarjan(i);
    }
```

```
    // 给每个割点一个新的编号(编号从cnt+1开始)
    num = cnt;
    for (int i = 1; i <= n; i++)
        if (cut[i]) new_id[i] = ++num;
    // 建新图，从每个v-DCC到它包含的所有割点连边
    tc = 1;
    for (int i = 1; i <= cnt; i++)
        for (int j = 0; j < dcc[i].size(); j++)
        {
            int x = dcc[i][j];
            if (cut[x]) {
                add_c(i, new_id[x]);
                add_c(new_id[x], i);
            }
            else c[x] = i; // 除割点外，其它点仅属于1个v-DCC
        }
}
```

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

```
// 模拟系统栈，答案栈
void Euler() {
    stack[++top] = 1;
    while (top > 0) {
        int x = stack[top], i = head[x];
        // 找到一条尚未访问的边
        while (i && vis[i]) i = Next[i];
        // 沿着这条边模拟递归过程，标记该边，并更新表头
        if (i) {
            stack[++top] = ver[i];
            head[x] = Next[i];
            vis[i] = vis[i ^ 1] = true;
        }
        // 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
        else {
            top--;
            ans[++t] = x;
        }
    }
}
```

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

```
void Tarjan(int x)
{
    dfn[x]=low[x]=++num;
    stack[++top]=x,in_stack[x]=true;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            Tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(in_stack[y])
            low[x]=min(low[x],dfn[y]);
    }
    if(dfn[x]==low[x])
    {
        cnt++;
        int y;
        do
        {
            y=stack[top--],in_stack[y]=false;
            color[y]=cnt, scc[cnt].push_back(y);
        } while (x!=y);
    }
}
```

6.10.2 SCC -> DAG

```
void SCC()
{
    for (int i = 0; i <= n; i++)
        if (!dfn[i])
            Tarjan(i);
    //缩点
    for (int x = 1; x <= n; x++)
    {
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver1[i];
            if (color[x] != color[y])
                add_c(color[x], color[y]);
        }
    }
}
```

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）

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

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）。

```
//染色法判定奇环
bool DFS(int x,int color)
{
    vis[x]=color;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!vis[y])
        {
            if(!DFS(y,3-color)) return false;
        }
        else if(vis[y]==color) return false;
    }
    return true;
}
```

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”

```
\\匈牙利算法
\\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。
bool DFS(int x)
{
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            vis[y] = 1;
            if (!match[y] || DFS(match[y]))
            {
                match[y] = x;
                return true;
            }
        }
    }
    return false;
}
```

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。

```
/*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。
w[][]:边权
la[], lb[]: 左，右部点顶标
visa[], visb[]: 访问标记，是否在交错树中
ans: Σw[match[i]][i]
*/
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 G2 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）

6.13 网络流初步

6.13.1 最大流

6.13.1.1 Edmonds Karp 增广路

```
bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    q.push(S); vis[S] = 1;
    incf[S] = inf; // 增广路上各边的最小剩余容量
    while (q.size()) {
        int x = q.front(); q.pop();
        for (int i = head[x]; i; i = Next[i])
            if (edge[i]) {
                int y = ver[i];
                if (vis[y]) continue;
                incf[y] = min(incf[x], edge[i]);
                pre[y] = i; // 记录前驱，便于找到最长路的实际方案
                q.push(y), vis[y] = 1;
                if (y == t) return 1;
            }
    }
    return 0;
}
void Update() { // 更新增广路及其反向边的剩余容量
    int x = t;
    while (x != s) {
        int i = pre[x];
        edge[i] -= incf[t];
        edge[i ^ 1] += incf[t]; // 利用“成对存储”的xor 1技巧
        x = ver[i ^ 1];
    }
    maxflow += incf[t];
}
```

6.13.1.2 Dinic

```
//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。
bool BFS()
{
    memset(d, 0, sizeof(d));
    queue<int> q;
    q.push(S);
    d[S] = 1; //不为1  陷入死循环
    while (q.size())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (edge[i] && !d[y])
            {
                d[y] = d[x] + 1;
                q.push(y);
                if (y == T)
                    return true;
            }
        }
    }
    return false;
}
int Dinic(int x, int flow)
{
```

```
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;
```

```
                }
            }
        }
    }
    if (dis[T] == 0xcfcfcfcf)
        return false;
    return true;
}
int max_flow, ans;
void Update()
{
    int x = T;
    while (x != S)
    {
        int i = pre[x];
        edge[i] -= incf[T];
        edge[i ^ 1] += incf[T];
        x = ver[i ^ 1];
    }
    max_flow += incf[T];
    ans += incf[T] * dis[T];
}
```

例题分析

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