知识点补充

递推和递归

高精度递推

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 10000007
typedef long long LL;
using namespace std;
inline int read() {
    int X=0,w=1;
    char ch=0;
    while(ch<'0' || ch>'9') {
        if(ch=='-') w=-1;
```

```
        ch=getchar();
    }
    while(ch>='0' && ch<='9') X=(X<<3)+(X<<1)+ch-'0',ch=getchar();
    return X*w;
}
inline void write(int x) {
    if(x < 0)putchar('-'),x=-x;
    if (x > 9)write(x / 10);
    putchar(x % 10 + 48);
}
int dp[5001][5001];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n,m;
    cin >> n >> m;
    n = m - n;
    dp[0][1] = 0;
    dp[1][1] = 1;
    dp[2][1] = 2;
    for(int i = 3; i <=n ; i++)
    {
        for(int j = 1; j < 5001 ; j++)      //加每一位
        {
            dp[i][j] = dp[i-1][j] + dp[i-2][j];
        }
        for(int j = 1; j < 5001 ; j++)      //处理每一位，保证每一位小于10
        {
            if(dp[i][j]>9)
            {
                dp[i][j+1]++;
                dp[i][j]-=10;
            }
        }
    }
    int flag = 0;
    for(int i = 5001 ; i > 1 ; i--)
    {
        if(flag == 0 && dp[n][i]==0)    continue;
        else
        {
            flag = 1;
            write(dp[n][i]);
        }
    }
    write(dp[n][1]);        //最后一位要单独输出，否则会WA
    return 0;
}
```

动态规划

单串

打家劫舍

限制n次的环形打家劫舍问题（3n披萨）

我们可以用 dp[i][j]表示在前 i 个数中选择了 j个不相邻的数的最大和

$$dp[i][j] = max(dp[i-2][j-1] + slices[i], dp[i-1][j])$$

```
int calculate(const vector<int>& slices) {
    int n = slices.size();
    int choose = (n + 1) / 3;
    vector<vector<int>> dp(n + 1, vector<int>(choose + 1));
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= choose; ++j) {
            dp[i][j] = max(dp[i - 1][j], (i - 2 >= 0 ? dp[i - 2][j - 1] : 0) +
slices[i - 1]);
        }
    }
    return dp[n][choose];
}

int maxSizeSlices(vector<int>& slices) {
    vector<int> v1(slices.begin() + 1, slices.end());
    vector<int> v2(slices.begin(), slices.end() - 1);
    int ans1 = calculate(v1);
    int ans2 = calculate(v2);
    return max(ans1, ans2);
}
```

双串

LCS

最长公共子序列

状态定义: dp[i][j]表示字符串text1 = 0..i和字符串text2=0..j的最大公共子序列长度
状态转移
`dp[i][j] = max(dp[i-1][j], dp[i][j-1]), if text1[i] != text2[j]`
`dp[i][j] = dp[i-1][j-1] + 1, if text1[i] == text2[j]`

```
int longestCommonSubsequence(char * text1, char * text2){
    int n = strlen(text1),m = strlen(text2);
    int dp[m+1];
    memset(dp,0,sizeof(dp));
    for(int i = 1;i<=n;i++)
    {
        int upleft = dp[0];
        for(int j=1;j<=m;j++)
        {
            int tmp = dp[j];
            if(text1[i-1] == text2[j-1])
                dp[j] = upleft + 1;
            else
```

```
                dp[j] = fmax(dp[j-1],dp[j]);
            upleft = tmp;
        }
    }
    return dp[m];
}
```

```
int longestCommonSubsequence(char* text1, char* text2) {
    int m = strlen(text1), n = strlen(text2);
    int dp[m + 1][n + 1];
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= m; i++) {
        char c1 = text1[i - 1];
        for (int j = 1; j <= n; j++) {
            char c2 = text2[j - 1];
            if (c1 == c2) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = fmax(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
    return dp[m][n];
}
```

两个字符串的最小 ASCII 删除和

给定两个字符串`s1, s2`，找到使两个字符串相等所需删除字符的ASCII值的最小和。

- 其实就是LCS的变形，先统计两个字符串的所有ASCII值，再减去公共最长子序列的ASCII值

```
int longestCommonSubsequence(char * text1, char * text2){
    int n = strlen(text1),m = strlen(text2);
    int dp[m+1];
    memset(dp,0,sizeof(dp));
    for(int i = 1;i<=n;i++)
    {
        int upleft = dp[0];
        for(int j=1;j<=m;j++)
        {
            int tmp = dp[j];
            if(text1[i-1] == text2[j-1])
                dp[j] = upleft + (int)text1[i-1];
            else
                dp[j] = fmax(dp[j-1],dp[j]);
            upleft = tmp;
        }
    }
    return dp[m];
}
int minimumDeleteSum(char * s1, char * s2){
    int sum = 0;
    for(int i = 0 ; s1[i] != '\0' ; i++)
    {
        sum += (int)s1[i];
```

```
    }
    for(int i = 0 ; s2[i] != '\0' ; i++)
    {
        sum += (int)s2[i];
    }
    return sum - 2 * longestCommonSubsequence(s1,s2);
}
```

最长重复子数组（子数组是连续的，子序列是可以不连续的）

• 状态定义: `dp[i][j]表示数组A[0..i]和数组B[0..j]的最大公共子数组长度`
• 状态转移
○ `dp[i][j] = 0 , if A[i] != B[j]`
○ `dp[i][j] = dp[i-1][j-1] + 1, if A[i] == B[j]`
○ `Res = max{dp[i][j]}`

```
int findLength(int* nums1, int nums1Size, int* nums2, int nums2Size){
    int n = nums1Size,m = nums2Size;
    int l = fmax(n,m);
    int dp[l+1];
    int ans = -1;
    memset(dp,0,sizeof(dp));
    for(int i = 1;i<=n;i++)
    {
        int upleft = dp[0];
        for(int j=1;j<=m;j++)
        {
            int tmp = dp[j];
            if(nums1[i-1] == nums2[j-1])
                dp[j] = upleft + 1;
            else
                dp[j] = 0;
            upleft = tmp;
            ans = fmax(dp[j],ans);
        }
    }
    return ans;
}
```

字符串匹配

边界条件及其重要（比如`dp[0][0] = 0 , dp[i][0] = ? , dp[0][j] = ?`）

编译距离

给你两个单词 word1 和 word2，请你计算出将 word1 转换成 word2 所使用的最少操作数

你可以对一个单词进行如下三种操作：

- 插入一个字符
- 删除一个字符
- 替换一个字符

`状态定义: dp[i][j]表示从word1[0..i]变为word2[0..j]的最少操作步骤`
状态转移:
`If word1[i] == word2[j]`
• `dp[i][j] = dp[i - 1][j - 1]`

`If word1[i] != word2[j]`

- `deleteCost = dp[i-1][j] + 1`
- `insertCost = dp[i][j - 1] + 1`
- `updateCost = dp[i -1][j - 1] + 1`
- `dp[i][j] = min(deleteCost, insertCost, updateCost)`

```
int minDistance(char * word1, char * word2){
    int l1 = strlen(word1), l2 = strlen(word2);
    int dp[l1 + 1][l2 + 1];
    dp[0][0] = 0;//重要的边界条件
    for(int i = 1 ; i <= l1 ; i++)
        dp[i][0] = i;//重要的边界条件
    for(int i = 1 ; i <= l2 ; i++)
        dp[0][i] = i;//重要的边界条件
    for(int i = 1 ; i <= l1 ; i++)
    {
        for(int j = 1 ; j <= l2 ; j++)
        {
            if(word1[i-1] == word2[j-1])
                dp[i][j] = dp[i-1][j-1];
            else
                dp[i][j] = fmin(dp[i-1][j],fmin(dp[i][j-1],dp[i-1][j-1])) + 1;
        }
    }
    return dp[l1][l2];
}
```

通配符匹配

给定一个字符串 (s) 和一个字符模式 (p) ，实现一个支持 '?' 和 '*' 的通配符匹配。

`'?' 可以匹配任何单个字符。`
`'*' 可以匹配任意字符串（包括空字符串）。`
两个字符串**完全匹配**才算匹配成功。

说明:

- s 可能为空，且只包含从 a-z 的小写字母。
- p 可能为空，且只包含从 a-z 的小写字母，以及字符 ? 和 *。

状态定义: `dp[i][k]表示字符串s[0..i], p[0..j]是否匹配`
状态转移:

- `dp[i][k] = (dp[i - 1][k - 1]) if (s[i] == p[j]) or p[j] == '?'`
  `dp[i][k] = or(dp[i-1][j], dp[i][j - 1]) , if p[k] == '*', 匹配时使用或不使用星号`
  `dp[i][k] = false, 其他情况`

边界条件

- `dp[0][0] = true`
  `dp[i][0] = false`
  `dp[0][j] = true, if p[0..j]都是星号`

```
bool isMatch(char * s, char * p){
    int l1 = strlen(s), l2 = strlen(p);
```

```
    bool dp[l1 + 1][l2 + 1];
    memset(dp,false,sizeof(dp));
    dp[0][0] = 1;
    for(int i = 1 ; i <= l2 ; i++)
    {
        if(p[i-1] == '*')
            dp[0][i] = true;
        else    break;
    }
    for(int i = 1 ; i <= l1 ; i++)
    {
        for(int j = 1; j <= l2 ; j++)
        {
            if(s[i-1] == p[j-1] || p[j-1] == '?')
                dp[i][j] = dp[i-1][j-1];
            else if (p[j-1] == '*')
                dp[i][j] = dp[i-1][j] | dp[i][j-1];
            //需要*则为dp[i - 1][j] 不需要则为dp[i][j - 1]
        }
    }
    return dp[l1][l2];
}
```

二分查找（答案）

(l和r分别为初始时区间的下界和上界)
①当二分区间为[l,mid] [mid+1,r]时：**最大值最小化问题**

```
while(l<r)
{
    int mid=(l+r)>>1;
    if(check(mid))
    {
        r=mid;
    }
    else
    {
        l=mid+1;
    }
}
write(l);
```

②当二分区间为[l,mid-1] [mid,r]时：**最小值最大化问题**

```
while(l<r)
{
    int mid=(l+r+1)>>1;
    if(check(mid))
    {
        l=mid;
    }
    else
    {
        r=mid-1;
    }
}
write(l);
```

- 一般情况

```
while(l<=r)
{
    mid = (l+r)/2;
    if(judge(mid))
    {
        l = mid+1;
        ans = mid;
    }
    else
    {
        r = mid-1;
    }
}
write(ans);
```

图论

Dij

s到其他所有节点+其他所有节点回s的最短路

思路： 建正向反向边，可以放在一张图里面，最后相加的时候注意控制变量的范围就行

```
    dijkstra(1);
    for (int i = 2; i <= n ; i++)
    {
        ans+=dis[i];
    }
    dijkstra(1+n);
    for (int i = n+2; i <= 2*n; i++)
    {
        ans+=dis[i];
    }
```

最短路计数

```
void dijkstra(int s)
{
    memset(dis, INF, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    memset(num, 0, sizeof(num));
    dis[s] = 0;
    num[s] = 1;          //初始点数量为1
    while (pq.size())
        pq.pop();
    pq.push(node(s, 0));
    while (!pq.empty())
    {
        node tmp = pq.top();
        pq.pop();
        int u = tmp.v;
        if (vis[u])
            continue;
        vis[u] = true;
        for (int i = head[u]; i; i = edges[i].next)
        {
            int v = edges[i].v;
            int w = edges[i].w;
            if (!vis[v] && dis[v] > w + dis[u])
            {
                dis[v] = w + dis[u];
                num[v] = num[u] ;                //传递
                pq.push(node(v, dis[v]));
            }
            else if(dis[v] == w + dis[u])
            {
                num[v] = (num[v] + num[u] ) % 100003;       //相加
            }
        }
    }
}
```

Floyd

更新单个点后求最短路

```
for (LL i = 0; i < n; i++)//ptr为更新的点的编号
    for (LL j = 0; j < n; j++) {
        dis[i][j] = dis[j][i] = min(dis[i][j], dis[i][ptr] + dis[ptr][j]); //跑一遍Floyd
    }
```

SPFA

二分答案的01检验

```
bool check(int mid) {
    queue<int>q;
    memset(vis,0,sizeof(vis));//标记是否在队列中
```

```
    memset(dis,0x3f3f3f3f,sizeof(dis));
    vis[1]=1;
    dis[1]=0;
    q.push(1);
    while(!q.empty()) {
        int x=q.front();
        q.pop();
        vis[x]=0;
        for(int i=head[x]; ~i; i=e[i].next) {
            int v=e[i].to;
            int w;
            if(e[i].w<=mid)w=0;      //01检验
            else w=1;
            if(dis[v]>dis[x]+w) {    //判断条件
                dis[v]=dis[x]+w;
                if(!vis[v]) {
                    vis[v]=1;
                    q.push(v);
                }
            }
        }
    }
    return dis[n]<=k;        //二分结果
}
```

二分图判断

判断二分图的某一部分最小点数

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;
using namespace std;
const int N=400005;
const int M=400500;
int cnt;
int head[N];
int n,m,white,black,ans;
int vis[N],sum[3];
struct Edge{
   int v,next;
}E[M<<1];//双边
void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}
void add(int u,int v){
    E[cnt].v=v;
    E[cnt].next=head[u];
    head[u]=cnt++;
}
queue<int> q;
bool BFS(int start)
{
    vis[start] = 1;
```

```
    sum[1] = 1, sum[2] = 0;
    q.push(start);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int i = head[u] ;~i;i = E[i].next)
        {
            int v = E[i].v;
            if(vis[v] == vis[u])
                return 0;
            if(vis[v]==0)
            {
                vis[v] = vis[u] % 2 + 1;
                sum[vis[v]]++;
                q.push(v);
            }
        }
    }
    return 1;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    init();
    for(int i = 0 ; i < m ; i++)
    {
        int u,v;
        cin >> u >> v;
        add(u,v);
        add(v,u);
    }

    while(!q.empty())   q.pop();
    memset(vis,0,sizeof(vis));

    for(int i = 1;i<=n;i++)
    {
        if(vis[i])  continue;
        if(!BFS(i))
        {
            puts("Impossible");
            return 0;
        }
        else    ans += min(sum[1],sum[2]);
    }
    cout << ans;
    return 0;
}
```

二分图判定二分答案

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;
using namespace std;
const int N=400005;
const int M=400500;
int cnt;
int head[N];
int n,m,ans,l,r,c;
int vis[N],sum[3];
struct Edge{
   int v,next,w;
}E[M<<1];//双边
void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}
void add(int u,int v, int c){
    E[cnt].v=v;
    E[cnt].w=c;
    E[cnt].next=head[u];
    head[u]=cnt++;
}
queue<int> q;
bool BFS(int start,int mid)
{
    vis[start] = 1;
    sum[1] = 1, sum[2] = 0;
    q.push(start);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int i = head[u] ;~i;i = E[i].next)
        {
            int v = E[i].v;
            int w = E[i].w;
            if(w>mid)
            {
                if(vis[v] == vis[u])
                    return 0;
                if(vis[v]==0)
                {
                    vis[v] = vis[u] % 2 + 1;
                    sum[vis[v]]++;
                    q.push(v);
                }
            }

        }
    }
    return 1;
}
```

```
bool check(int mid)
{
    while(!q.empty())   q.pop();
    memset(vis,0,sizeof(vis));
    for(int i = 1;i<=n;i++)
    {
        if(vis[i])  continue;
        if(!BFS(i,mid))
        {
            return 0;
        }
    }
    return 1;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    l = 0x7fffffff, r = -1;
    init();
    for(int i = 1 ; i <= m ; i++)
    {
        int u,v;
        cin >> u >> v >> c;
        add(u,v,c);
        add(v,u,c);
        r= max(r,c);
    }
    l = 0;
    while(l<r)
    {
        int mid=(l+r)>>1;
        if(check(mid))
        {
            r=mid;
        }
        else
        {
            l=mid+1;
        }
    }
    cout << l;
    return 0;
}
```

KMP

算法实现

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000000+5;
int slen,tlen;
int nxt[maxn];
char s[maxn],t[maxn];
void get_nxt(char *t){//求模式串T的nxt函数值
    int j=0,k=-1;
    nxt[0]=-1;
    while(j<tlen){//模式串t的长度
        if(k==-1||t[j]==t[k])
            nxt[++j]=++k;
        else
            k=nxt[k];
    }
}

void KMP(char *s,char *t){
    int i=0,j=0;
    get_nxt(t);
    while(i<slen){
        if(j==-1||s[i]==t[j]){//如果相等，则继续比较后面的字符
            i++;
            j++;
        }
        else
            j=nxt[j]; //j回退到nxt[j]
        if(j==tlen){ //匹配成功
            printf("%d\n",i-tlen+1);
            j=nxt[j];//不允许重叠，j从0重新开始，如果允许重叠，j=nex[j]
        }
    }
}

int main(){
    scanf("%s%s",s,t);
    slen = strlen(s);
    tlen = strlen(t);
    KMP(s,t);
    for(int i=1;i<=tlen;i++)
        printf("%d ",nxt[i]);
    return 0;
}
```

搜索

DFS

DFS确定连通块的数量（或者是连通块的最长路径）

- 注意边界条件，DFS标号

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;
using namespace std;
int n,m,cnt,id[102][102],maxans,ans;
char mapp[102][102];
void dfs(int x, int y, int num, char sym) {
    if(x < 1 ||y < 1 || x > n || y > m)
        return;
    id[x][y] = num;
    if(x+1<=n && mapp[x+1][y]==sym&& id[x+1][y] == 0) {
        dfs(x+1,y,num,sym);
        ans++;
    }
    if(x-1>=1 && mapp[x-1][y]==sym&& id[x-1][y] == 0) {
        dfs(x-1,y,num,sym);
        ans++;
    }
    if(y+1<=m && mapp[x][y+1]==sym&& id[x][y+1] == 0) {
        dfs(x,y+1,num,sym);
        ans++;
    }
    if(y-1>=1 && mapp[x][y-1]==sym&& id[x][y-1] == 0) {
        dfs(x,y-1,num,sym);
        ans++;
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    while(cin>>n>>m)
    {
        ms(mapp,0);
        ms(id,0);
        for(int i = 1; i<= n; i++)
        {
            for(int j = 1; j<=m; j++)
            {
                cin >> mapp[i][j];
            }
        }
        cnt = 1;
        for(int i = 1; i<= n; i++)
        {
            for(int j = 1; j<=m; j++)
            {
                if(id[i][j]==0)
                {
//                  putchar('\n');
//                  putchar('\n');
```

```
                    ans = 1;
                    dfs(i,j,cnt,mapp[i][j]);
                    //maxans = max(maxans,ans);
                    cnt++;
//                  for(int i = 1; i<= n; i++)
//                  {
//                      for(int j = 1; j<=m; j++)
//                      {
//                          printf("%d ",id[i][j]);
//                      }
//                      putchar('\n');
//                  }
//                  putchar('\n');
                }
            }
        }
        cout<<maxans<<endl;
    }
}
```

DFS判断环路

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;
using namespace std;
int n,m;
vector<int> g[maxn];
int color[maxn],last;
bool hasCycle = false;
void dfs(int root) {
    color[root] = 1;
    for (auto child : g[root]) {
        if (color[child] == 1 && child != last) {
            hasCycle = true;
            break;
        }
        else if (color[child] == 0) {
            last = root;
            dfs(child);
        }
    }
    color[root] = 2;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    last = -1;
    for(int i=1; i<=m; i++) {
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
```

```
    }
    dfs(1);
    cout << hasCycle;
    return 0;
}
```