目录

[一、DP 1](#_Toc498701402)

[1. 数位dp 1](#_Toc498701403)

[2．插头DP 2](#_Toc498701404)

[3.AC自动机上DP 11](#_Toc498701405)

[二、KD-tree 17](#_Toc498701406)

# 一、DP

## 数位dp

**/\***

**以不出现49的个数为例**

**\*/**

ll dp[30][2];//第几位 上一位是不是4

int dig[30];//最大范围的每一位数 下标从1开始 下标为1表示个位的数

ll DP(int wei,bool si,bool lim)//第几位 上一位是否为4 是否仍在边界限制（e.g. 范围568 到此时前两位为56 即有限制）

{

if(!wei)return 1;//此为到0位不需要特别判断的直接返回1（个数）

if(!lim&&dp[wei][si]!=-1)return dp[wei][si];//之前计算过直接返回

int da=9;//该位最大的数值

ll re=0;

if(lim)da=dig[wei];//若有边界限制 上界取为边界

for(int i=0;i<=da;i++)

{

if(si&&i==9)continue;//不能出现49

re+=DP(wei-1,i==4,lim&&(i==da));

}

if(!lim)dp[wei][si]=re;//若无边界限制 可以更新该位的值 （若有限制的话 此值随数的变化而变 无需记忆）

return re;

}

ll cal(ll x)

{

int wei=0;

while(x)//求出每一位的数码

{

dig[++wei]=x%10;x/=10;

}

return DP(wei,0,1);

}

int t;

ll n;

int main()

{

memset(dp,-1,sizeof(dp));//dp数组初始化为-1

scanf("%d",&t);

while(t--)

{

cin>>n;

cout<<(n-cal(n)+1LL)<<"\n";

}

return 0;

}

## 2．插头DP

**/\***

**第一种（不完整版）**

**HDU 1693**

**题意：在n\*m的矩阵中，有些格子有树，没有树的格子不能到达，**

**找一条或多条回路，吃完所有的树，求有多少中方法。**

**\*/**

const int HASH=10007;

const int STATE=1000010;

const int MAXD=15;

int N,M;

int code[MAXD],maze[MAXD][MAXD];

struct HASHMAP

{

int head[HASH],next[STATE],state[STATE],size;//链表形式 head通过取模进行了一种索引 提高查询效率

long long f[STATE];//到某状态的个数

void init()

{

size=0;memset(head,-1,sizeof(head));

}

void push(int st,long long ans)//达到第i种状态的方式加ans个

{

int i,h=st%HASH;//h为哈希过后的编码 作为索引

for(i=head[h];i!=-1;i=next[i])

if(st==state[i])//若成功找到

{

f[i]+=ans;

return;

}

//未找到，需自己创建

f[size]=ans;//达到该状态的方法数即为ans

state[size]=st;//其具体编码

next[size]=head[h];

head[h]=size++;

}

}hm[2];//滚动数组，节省空间

/\*

对st进行解码，将结果保存在code数组中，其长度为m

\*/

void decode(int \*code,int m,int st)

{

int i;

for(i=m;i>=0;i--)

{

code[i]=st&1;

st>>=1;

}

}

/\*

将当前的长度为m的code数组进行编码

\*/

int encode(int \*code,int m)

{

int i,st=0;

for(i=0;i<=m;i++)

{

st<<=1;st|=code[i];

}

return st;

}

/\*

可作为通用的读入函数

注意需要在右侧、下策边界设一圈代表不可“到达”的数

\*/

void init()

{

int i,j;

scanf("%d%d",&N,&M);

for(i=1;i<=N;i++)

for(j=1;j<=M;j++)

scanf("%d",&maze[i][j]);

for(int i=1;i<=N;i++)maze[i][M+1]=0;

for(int i=1;i<=M;i++)maze[N+1][i]=0;

}

/\*

换行的时候移位

\*/

void shift(int \*code,int m)

{

int i;

for(i=m;i>0;i--)code[i]=code[i-1];

code[0]=0;

}

/\*

处理可以放的格子

i行，j列，当前滚动数组的状态

\*/

void dpblank(int i,int j,int cur)

{

int k,left,up;

for(k=0;k<hm[cur].size;k++)

{

decode(code,M,hm[cur].state[k]);

left=code[j-1];//左侧是否有连

up=code[j];//上侧是否有连

if(left&&up)//11 -> 00

{

code[j-1]=code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

else if(left||up)//01 或 10

{

if(maze[i][j+1])//可以向右侧连 并且并不涉及换行 无需shift

{

code[j-1]=0;code[j]=1;

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

if(maze[i+1][j])

{

code[j-1]=1;code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

else//00 -> 11 必须两个都可以连

{

if(maze[i][j+1]&&maze[i+1][j])

{

code[j]=code[j-1]=1;

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

}

}

/\*

不能放的格子

i行，j列，当前滚动数组的状态

\*/

void dpblock(int i,int j,int cur)

{

int k;

for(k=0;k<hm[cur].size;k++)

{

decode(code,M,hm[cur].state[k]);

code[j-1]=code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

void solve()

{

int i,j,cur=0;

long long ans=0;

hm[cur].init();

hm[cur].push(0,1);//dp前初始化 仅状态0的个数为1（即当下）

for(i=1;i<=N;i++)

for(j=1;j<=M;j++)

{

hm[cur^1].init();//滚动数组的初始化

if(maze[i][j])dpblank(i,j,cur);

else dpblock(i,j,cur);

cur^=1;

}

for(i=0;i<hm[cur].size;i++)

ans+=hm[cur].f[i];

printf("There are %I64d ways to eat the trees.\n",ans);

}

int main()

{

int T;

int iCase=0;

scanf("%d",&T);

while(T--)

{

iCase++;

printf("Case %d: ",iCase);

init();

solve();

}

return 0;

}

**/\***

**完整版插头DP模版**

**URAL1519**

**题目简意：**

**给你一个m \* n的棋盘，有的格子是障碍，问共有多少条回路使得经过每个非障碍格子恰好一次．m, n ≤ 12。**

**\*/**

const int HASH=30007;

const int STATE=1000010;

const int MAXD=15;

int N,M,ex,ey;//ex\ey非必需

int code[MAXD],maze[MAXD][MAXD];

int ch[MAXD];

struct HASHMAP

{

int head[HASH],next[STATE],size;//链表形式 head通过取模进行了一种索引 提高查询效率

long long f[STATE],state[STATE];//到某状态的个数

void init()

{

size=0;memset(head,-1,sizeof(head));

}

void push(long long st,long long ans)//达到第i种状态的方式加ans个

{

int i,h=st%HASH;//h为哈希过后的编码 作为索引

for(i=head[h];i!=-1;i=next[i])

if(st==state[i])//若成功找到

{

f[i]+=ans;

return;

}

//未找到，需自己创建

f[size]=ans;//达到该状态的方法数即为ans

state[size]=st;//其具体编码

next[size]=head[h];

head[h]=size++;

}

}hm[2];//滚动数组，节省空间

/\*

对st进行解码，将结果保存在code数组中，其长度为m

\*/

void decode(int \*code,int m,long long st)

{

int i;

for(i=m;i>=0;i--)

{

code[i]=st&7;

st>>=3;

}

}

/\*

将当前的长度为m的code数组进行编码

\*/

long long encode(int \*code,int m)

{

int i;

int cnt=1;

memset(ch,-1,sizeof(ch));

ch[0]=0;

ll st=0;

for(i=0;i<=m;i++)

{

if(ch[code[i]]==-1)ch[code[i]]=cnt++;

code[i]=ch[code[i]];

st<<=3;

st|=code[i];

}

return st;

}

/\*

可作为通用的读入函数

注意需要在右侧、下策边界设一圈代表不可“到达”的数

\*/

char lin[20];

/\*

读入数据，将状态图记录在maze中

\*/

void init()

{

ex=ey=0;

memset(maze,0,sizeof(maze));

for(int i=1;i<=N;i++)

{

scanf("%s",lin);

for(int j=0;j<M;j++)

if(lin[j]=='.'){maze[i][j+1]=1;ex=i;ey=j+1;}

}

}

/\*

换行的时候移位

不想改变code数组 可以采用 encode(code,j==M?M-1:M);的方式

\*/

void shift(int \*code,int m)

{

int i;

for(i=m;i>0;i--)code[i]=code[i-1];

code[0]=0;

}

/\*

处理可以放的格子

i行，j列，当前滚动数组的状态

\*/

void dpblank(int i,int j,int cur)

{

int k,left,up;

for(k=0;k<hm[cur].size;k++)

{

decode(code,M,hm[cur].state[k]);

left=code[j-1];//左侧是否有连 及其连通编号

up=code[j];//上侧是否有连 及其连通编号

if(left&&up)//11 -> 00

{

if(left==up)//左、上已经连通

{

if(i==ex&&j==ey)

{

code[j-1]=code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

else//左、上并未连通 则合并

{

code[j-1]=code[j]=0;

for(int s=0;s<=M;s++)

if(code[s]==up)code[s]=left;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

else if(left||up)//01 或 10

{

int t=left?left:up;

if(maze[i][j+1])//可以向右侧连 并且并不涉及换行 无需shift

{

code[j-1]=0;code[j]=t;

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

if(maze[i+1][j])

{

code[j-1]=t;code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

else//00 -> 11 必须两个都可以连

{

if(maze[i][j+1]&&maze[i+1][j])

{

code[j]=code[j-1]=13;

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

}

}

/\*

不能放的格子

i行，j列，当前滚动数组的状态

\*/

void dpblock(int i,int j,int cur)

{

int k;

for(k=0;k<hm[cur].size;k++)

{

decode(code,M,hm[cur].state[k]);

code[j-1]=code[j]=0;

if(j==M)shift(code,M);

hm[cur^1].push(encode(code,M),hm[cur].f[k]);

}

}

void solve()

{

int i,j,cur=0;

long long ans=0;

hm[cur].init();

hm[cur].push(0,1);//dp前初始化 仅状态0的个数为1（即当下）

for(i=1;i<=N;i++)

for(j=1;j<=M;j++)

{

hm[cur^1].init();//滚动数组的初始化

if(maze[i][j])dpblank(i,j,cur);

else dpblock(i,j,cur);

cur^=1;

}

for(i=0;i<hm[cur].size;i++)

ans+=hm[cur].f[i];

printf("%I64d\n",ans);

}

int main()

{

while(~scanf("%d%d",&N,&M))

{

init();

if(!ex)

printf("0\n");

else solve();

}

return 0;

}

## 3.AC自动机上DP

**/\***

**HDU 2825**

**题意：**

**给 m 个单词构成的集合，统计所有长度为 n 的串中，包含至少 k 个单词的方案数。**

**\*/**

const int SIGMA\_SIZE=26;

const int MAXNODE=110;

struct AhoCorasickAutomata

{

int ch[MAXNODE][SIGMA\_SIZE];//Trie树

int f[MAXNODE];//fail函数

int val[MAXNODE];//每个字符串的结尾结点都有非0的val

int num;//trie树编号（亦为包含根结点的当前size）

//初始化

void init()

{

num=1;

memset(ch[0],-1,sizeof(ch[0]));

}

//返回字符对应编号

int idx(char c)

{

return c-'a';

}

//插入权值为v的字符串

void insert(char \*s,int v)

{

int u=0,n=strlen(s);

for(int i=0;i<n;i++)

{

int c=idx(s[i]);

if(ch[u][c]==-1)

{

memset(ch[num],-1,sizeof(ch[num]));

val[num]=0;

ch[u][c]=num++;

}

u=ch[u][c];

}

val[u]|=v;

}

//计算fail函数

void getFail()

{

queue <int> que;

f[0]=0;

for(int c=0;c<SIGMA\_SIZE;c++)

{

int u=ch[0][c];

if(u!=-1)

{

f[u]=0;que.push(u);

}

else ch[0][c]=0;

}

while(!que.empty())

{

int r=que.front();que.pop();

if(val[f[r]])

val[r]|=val[f[r]];

for(int c=0;c<SIGMA\_SIZE;c++)

{

int u=ch[r][c];

if(u==-1){ch[r][c]=ch[f[r]][c];continue;}

que.push(u);

int v=f[r];

// while(v&&ch[v][c]==-1)//类似kmp的过程

// v=f[v];

f[u]=ch[v][c];

}

}

}

}AC;

int n,m,k;

char tem[20];

ll dp[26][105][1024];

ll num[1024],an;

//vector<int> to[105];

int main()

{

num[0]=0;

for(int i=1;i<1024;i++)num[i]=num[i>>1]+(i&1);

while(scanf("%d%d%d",&n,&m,&k)&&n)

{

int tot=(1<<m);

AC.init();

an=0;

for(int i=1;i<=m;i++){scanf("%s",tem);AC.insert(tem,1<<(i-1));}

AC.getFail();

for(int i=0;i<=n;i++)

for(int j=0;j<=AC.num;j++)

for(int s=0;s<=tot;s++)dp[i][j][s]=0;

dp[0][0][0]=1;

for(int i=0;i<n;i++)//步数

for(int j=0;j<AC.num;j++)//当前位置

for(int s=0;s<tot;s++)//当前状态

{

if(!dp[i][j][s])continue;//无当前状态

for(int k=0;k<26;k++)

{

// (dp[i+1][AC.ch[j][k]][s|AC.val[AC.ch[j][k]]]+=dp[i][j][s])%MOD;

dp[i+1][AC.ch[j][k]][s|AC.val[AC.ch[j][k]]]+=dp[i][j][s];

dp[i+1][AC.ch[j][k]][s|AC.val[AC.ch[j][k]]]%=MOD;

}

}

for(int s=0;s<tot;s++)

{

if(num[s]<k)continue;

for(int i=0;i<AC.num;i++)

an=(an+dp[n][i][s])%MOD;

}

printf("%lld\n",an);

}

return 0;

}

**/\***

**ZOJ 3228**

**题目大意：首先给你一下母串，长度不超过10^5，然后有 N（10^5） 次查询，**

**每次查询有两种命令，0或者1，然后加一个子串，询问母串里面有多少个子串，**

**0表示可以重复，1表示不可以重复。**

**\*/**

int cnt[MAX][2];

int las[MAX];

const int SIGMA\_SIZE=26;

const int MAXNODE=610000;

struct AhoCorasickAutomata

{

int ch[MAXNODE][SIGMA\_SIZE];//Trie树

int f[MAXNODE];//fail函数

int val[MAXNODE];//每个字符串的结尾结点都有非0的val

int last[MAXNODE];//输出链表的下一个单词结点

int dep[MAXNODE];

int num;//trie树编号（亦为包含根结点的当前size）

//初始化

void init()

{

num=1;

memset(ch[0],-1,sizeof(ch[0]));

}

//返回字符对应编号

int idx(char c)

{

return c-'a';

}

//插入权值为v的字符串

int insert(char \*s,int& v)

{

int u=0,n=strlen(s);

for(int i=0;i<n;i++)

{

int c=idx(s[i]);

if(ch[u][c]==-1)

{

memset(ch[num],-1,sizeof(ch[num]));

val[num]=0;

dep[num]=dep[u]+1;

ch[u][c]=num++;

}

u=ch[u][c];

}

if(!val[u])

val[u]=++v;

return val[u];

}

//递归打印以结点j结尾的所有字符串

void print(int j,int pos)

{

if(j)

{

cnt[val[j]][0]++;

if(las[val[j]]==-1||las[val[j]]+dep[j]<=pos)

{

las[val[j]]=pos;

cnt[val[j]][1]++;

}

print(last[j],pos);

}

}

//计算fail函数

void getFail()

{

queue <int> que;

f[0]=0;

for(int c=0;c<SIGMA\_SIZE;c++)

{

int u=ch[0][c];

if(u!=-1)

{

f[u]=0;que.push(u);last[u]=0;

}

else

ch[0][c]=0;

}

while(!que.empty())

{

int r=que.front();que.pop();

// if(val[f[r]]&&)val[r]

for(int c=0;c<SIGMA\_SIZE;c++)

{

int u=ch[r][c];

// if(u==-1)continue;

if(u==-1){ch[r][c]=ch[f[r]][c];continue;}

que.push(u);

int v=f[r];

while(v&&!ch[v][c])//类似kmp的过程

v=f[v];

f[u]=ch[v][c];

last[u]=val[f[u]]?f[u]:last[f[u]];

}

}

}

//在T中中模板

int find(char \*T)

{

int n=strlen(T);

int u=0;//当前结点编号,初始为根结点

for(int i=0;i<n;i++)//文本串当前指针

{

int c=idx(T[i]);

// while(u&&!ch[u][c])u=f[u];//顺着失配指针走，直到可以匹配或回到根节点

u=ch[u][c];

if(val[u])

print(u,i);

else if(last[u])//明白了 如果当前恰为某串末尾 则直接print从其开始 不然从当前后缀作为某串的位置开始print

print(last[u],i);

}

}

}AC;

char a[MAX];

int n,num;

int kind[MAX],id[MAX],Case;

char tem[30];

int main()

{

while(~scanf("%s",a))

{

AC.init();

num=0;

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

scanf("%d",&kind[i]);

scanf("%s",tem);

id[i]=AC.insert(tem,num);

}

AC.getFail();

for(int i=0;i<=num;i++)

las[i]=-1;

for(int i=0;i<=num;i++)

for(int j=0;j<=1;j++)cnt[i][j]=0;

AC.find(a);

printf("Case %d\n",++Case);

for(int i=1;i<=n;i++)

printf("%d\n",cnt[id[i]][kind[i]]);

printf("\n");

}

}

# 二、KD-tree

/\*

BZOJ 4066

你有一个N\*N的棋盘，每个格子内有一个整数，初始时的时候全部为0，现在需要维护两种操作：

命令 参数限制 内容

1 x y A 1<=x,y<=N，A是正整数 将格子x,y里的数字加上A

2 x1 y1 x2 y2 1<=x1<= x2<=N

1<=y1<= y2<=N 输出x1 y1 x2 y2这个矩形内的数字和

3 无 终止程序

Input

输入文件第一行一个正整数N。

接下来每行一个操作。每条命令除第一个数字之外，

均要异或上一次输出的答案last\_ans，初始时last\_ans=0。

Output

对于每个2操作，输出一个对应的答案。

\*/

#include<map>

#include<cmath>

#include<queue>

#include<cstdio>

#include<vector>

#include<cstring>

#include<cstdlib>

#include<iostream>

#include<algorithm>

#define inf 1000000000

#define mod 1000000007

#define ll long long

#define eps 1e-12

using namespace std;

ll read()

{

ll x=0,f=1;char ch=getchar();

while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}

while(ch>='0'&&ch<='9'){x=x\*10+ch-'0';ch=getchar();}

return x\*f;

}

int n,D;

ll lastans;

struct P{

int d[2],mx[2],mn[2],v,l,r;

ll sum;

int &operator[](int x){

return d[x];

}

friend bool operator==(P a,P b){

return a.d[0]==b.d[0]&&a.d[1]==b.d[1];

}

friend bool operator<(P a,P b){

return a[D]<b[D];

}

}p[200005];

bool in(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2)

{

return x1<=X1&&X2<=x2&&y1<=Y1&&Y2<=y2;

}

bool out(int x1,int y1,int x2,int y2,int X1,int Y1,int X2,int Y2)

{

return x1>X2||x2<X1||y1>Y2||y2<Y1;

}

struct data{

P t[200005],now;

int rt,cnt;

void update(int k){

int l=t[k].l,r=t[k].r;

for(int i=0;i<2;i++)

{

t[k].mn[i]=t[k].mx[i]=t[k][i];

if(l)t[k].mn[i]=min(t[k].mn[i],t[l].mn[i]);

if(l)t[k].mx[i]=max(t[k].mx[i],t[l].mx[i]);

if(r)t[k].mn[i]=min(t[k].mn[i],t[r].mn[i]);

if(r)t[k].mx[i]=max(t[k].mx[i],t[r].mx[i]);

}

t[k].sum=t[k].v+t[l].sum+t[r].sum;

}

void insert(int &k,bool D){

if(!k)

{

k=++cnt;

t[k][0]=t[k].mn[0]=t[k].mx[0]=now[0];

t[k][1]=t[k].mn[1]=t[k].mx[1]=now[1];

}

if(now==t[k])

{

t[k].v+=now.v,t[k].sum+=now.v;

return;

}

if(now[D]<t[k][D])insert(t[k].l,D^1);

else insert(t[k].r,D^1);

update(k);

}

ll query(int k,int x1,int y1,int x2,int y2){

if(!k)return 0;

ll tmp=0;

if(in(x1,y1,x2,y2,t[k].mn[0],t[k].mn[1],t[k].mx[0],t[k].mx[1]))return t[k].sum;

if(out(x1,y1,x2,y2,t[k].mn[0],t[k].mn[1],t[k].mx[0],t[k].mx[1]))return 0;

if(in(x1,y1,x2,y2,t[k][0],t[k][1],t[k][0],t[k][1]))tmp+=t[k].v;

tmp+=query(t[k].l,x1,y1,x2,y2)+query(t[k].r,x1,y1,x2,y2);

return tmp;

}

int rebuild(int l,int r,bool f){

if(l>r)return 0;

int mid=(l+r)>>1;

D=f;nth\_element(p+l,p+mid,p+r+1);

t[mid]=p[mid];

t[mid].l=rebuild(l,mid-1,f^1);

t[mid].r=rebuild(mid+1,r,f^1);

update(mid);

return mid;

}

}T;

int main()

{

n=read();

int opt,x,y,x2,y2,A,m=10000;

while(1)

{

opt=read();if(opt==3)break;

x=read()^lastans;y=read()^lastans;

if(opt==1)

{

A=read()^lastans;T.now[0]=x;T.now[1]=y;

T.now.v=T.now.sum=A;

T.insert(T.rt,0);

if(T.cnt==m)

{

for(int i=1;i<=T.cnt;i++)p[i]=T.t[i];

T.rt=T.rebuild(1,T.cnt,0);m+=10000;

}

}

if(opt==2)

{

x2=read()^lastans;y2=read()^lastans;

lastans=T.query(T.rt,x,y,x2,y2);

printf("%lld\n",lastans);

}

}

return 0;

}